aws

開發人員指南

Amazon Timestream

Amazon Timestream: 開發人員指南

Table of Contents

什麼是適用於 的 Amazon Timestream LiveAnalytics？

適用於 的 Amazon Timestream LiveAnalytics 是快速、可擴展、完全受管、專用的時間序列資料庫，可讓您輕鬆每天儲存和分析數兆個時間序列資料點。的 Timestream 透過將最近的資料保留在記憶體中，並根據使用者定義的政策將歷史資料移至成本最佳化的儲存層， LiveAnalytics 節省您管理時間序列資料生命週期的時間和成本。 LiveAnalytics專用查詢引擎的 Timestream 可讓您一起存取和分析近期和歷史資料，而不必指定其位置。的 Amazon Timestream LiveAnalytics 具有內建的時間序列分析函數，可協助您近乎即時地識別資料中的趨勢和模式。的 Timestream LiveAnalytics 是無伺服器，並自動擴展或縮減，以調整容量和效能。由於您不需要管理基礎基礎設施，因此您可以專注於最佳化和建置應用程式。

的 Timestream LiveAnalytics 也與資料收集、視覺化和機器學習常用的服務整合。您可以將資料傳送至 Amazon Timestream，以便 LiveAnalytics 使用 AWS IoT Core、Amazon Kinesis MSK、Amazon 和開放原始碼 Telegraf。您可以透過 使用 Amazon QuickSight、Grafana 和商業智慧工具視覺化資料 JDBC。您也可以使用 Amazon SageMaker 搭配 Timestream for LiveAnalytics 進行機器學習。

LiveAnalytics 關鍵優勢的時間串流

Amazon Timestream 的主要優點 LiveAnalytics 為：

- 使用自動擴展的無伺服器 - 使用適用於 的 Amazon Timestream LiveAnalytics，無需管理伺服器，也無需佈建容量。隨著應用程式的需求變更， 的 Timestream LiveAnalytics 會自動擴展以調整容量。
- 資料生命週期管理 - Amazon Timestream for 可 LiveAnalytics 簡化資料生命週期管理的複雜程序。它提供儲存分層，具有用於最近資料的記憶體存放區和用於歷史資料的磁性存放區。Amazon Timestream 會根據使用者可設定的政策，自動將資料從記憶體存放區傳輸到磁性存放區。
- 簡化資料存取 - 使用適用於 的 Amazon Timestream LiveAnalytics，您不再需要使用不同的工具來存取近期和歷史資料。適用於 LiveAnalytics的 Amazon Timestream 專用查詢引擎可透明地存取和合併跨儲存層的資料，而無需指定資料位置。
- 專為時間序列打造 - 您可以使用 快速分析時間序列資料SQL，並具有內建時間序列函數，以進行平滑化、近似化和插補。的 Timestream LiveAnalytics 也支援進階彙總、視窗函數和複雜的資料類型，例如陣列和資料列。
- 一律加密 - 適用於 的 Amazon Timestream LiveAnalytics 可確保您的時間序列資料一律加密，無論是靜態或傳輸中。適用於 的 Amazon Timestream LiveAnalytics 也可讓您指定 AWS KMS 客戶受管金鑰 （CMK），以加密磁性存放區中的資料。

- 高可用性 - Amazon Timestream 透過自動複寫資料並在單一 AWS 區域中至少 3 個不同的可用區域分配資源，確保寫入和讀取請求的高可用性。如需詳細資訊，請參閱 Timestream Service Level Agreement。
- 耐久性 - Amazon Timestream 透過自動複寫單一 AWS 區域中不同可用區域的記憶體和磁性存放區資料，來確保資料的耐久性。在確認您的寫入請求為完成之前，所有資料都會寫入磁碟。

LiveAnalytics 使用案例的時間串流

Timestream 的不斷增長的使用案例清單範例 LiveAnalytics 包括：

- 監控指標以改善應用程式的效能和可用性。
- 工業遙測的儲存和分析，以簡化設備管理和維護。
- 追蹤使用者與應用程式的互動一段時間。
- IoT 感應器資料的儲存和分析。

Timestream for 入門 LiveAnalytics

建議您一開始先閱讀下列各節：

- 教學課程 - 建立填入範例資料集的資料庫，並執行範例查詢。
- 適用於 LiveAnalytics 概念的 Amazon Timestream - 了解 LiveAnalytics 概念的基本 Timestream。
- 存取 的 Timestream LiveAnalytics - 了解如何存取 Timestream 以 LiveAnalytics 使用主控台 AWS CLI，或 API。
- 配額 - 了解您可以佈建之 LiveAnalytics 元件的 Timestream 數量配額。

若要了解如何快速開始為 的 Timestream 開發應用程式 LiveAnalytics，請參閱以下內容：

- 使用 AWS SDKs
- 查詢語言參考

運作方式

下列各節提供 Amazon Timestream for Live Analytics 服務元件及其互動方式的概觀。

閱讀本簡介後，請參閱存取 的 Timestream LiveAnalytics章節，了解如何使用主控台或 AWS CLI存取 Timestream for Live AnalyticsSDKs。

主題

適用於 LiveAnalytics 概念的 Amazon Timestream

時間序列資料是在時間間隔內記錄的資料點序列。這種類型的資料用於測量隨時間變化的事件。範例如下。

- 一段時間內的股票價格
- 一段時間內的溫度測量
- CPU EC2執行個體隨時間的利用率

使用時間序列資料時，每個資料點都包含時間戳記、一或多個屬性，以及隨時間變化的事件。此資料可用來深入了解應用程式的效能和運作狀態、偵測異常，以及識別最佳化機會。例如， DevOps 工程人員可能想要檢視測量基礎設施效能指標變更的資料。製造商可能想要追蹤 IoT 感應器資料，以測量設施中設備的變化。線上行銷人員可能會想要分析點擊串流資料，以擷取使用者隨著時間導覽網站的方式。由於時間序列資料是從極高磁碟區中的多個來源產生，因此需要近乎即時地以符合成本效益的方式收集，因此需要有助於組織和分析資料的高效儲存。

以下是 的 Timestream 關鍵概念 LiveAnalytics。

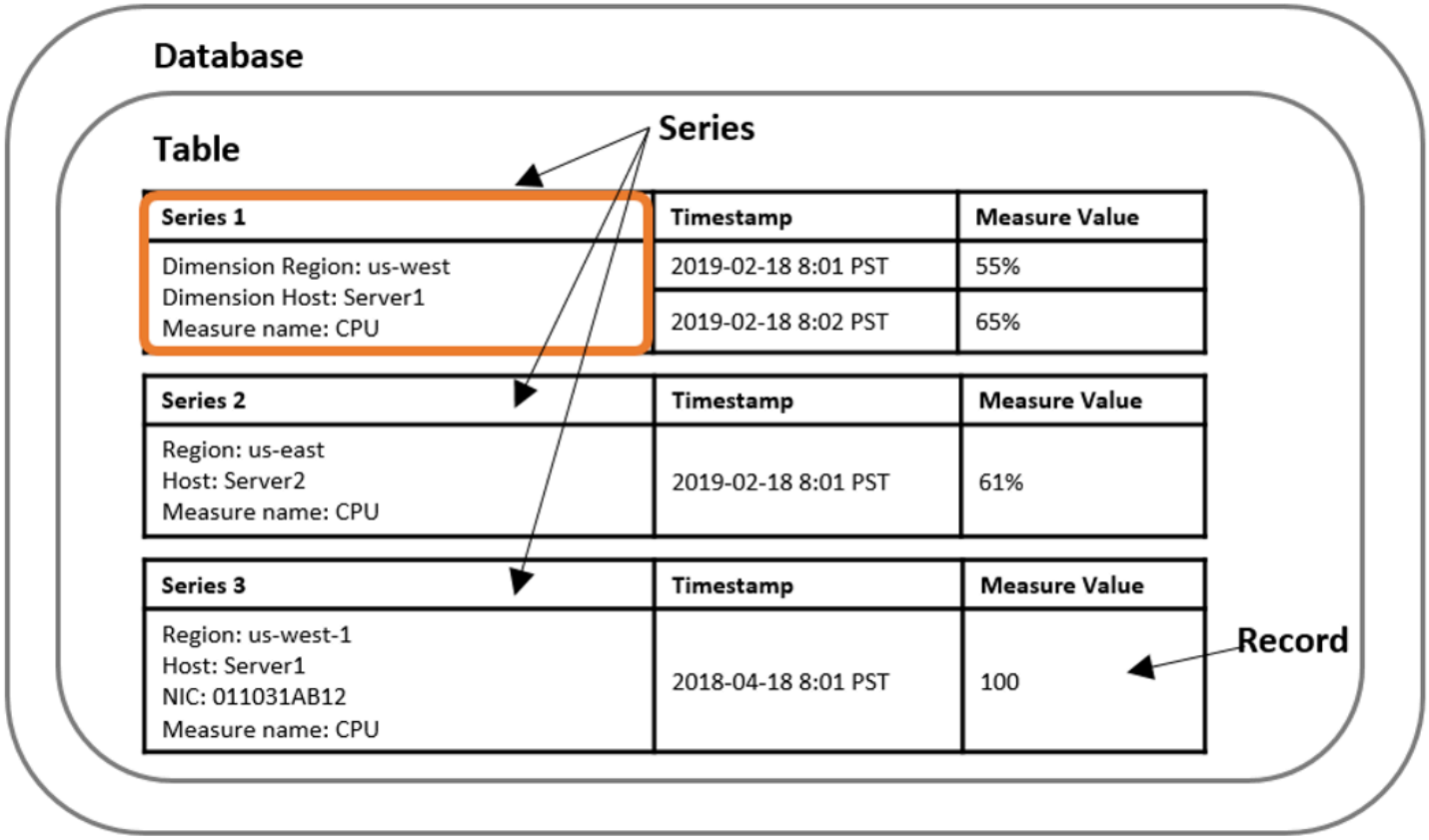
- 時間序列 - 在時間間隔內記錄的一或多個資料點 （ 或記錄 ） 的序列。 範例包括一段時間內的庫存價格、一段時間內的EC2執行個體CPU或記憶體使用率，以及一段時間內的 IoT 感應器溫度/壓力讀數。
- 記錄 - 時間序列中的單一資料點。
- Dimension - 描述時間序列中繼資料的屬性。維度由維度名稱和維度值組成。請考量下列範例：

  - 將股票交換視為維度時，維度名稱為「股票交換」，維度值為「NYSE」
  - 將 AWS 區域視為維度時，維度名稱為「區域」，維度值為「us-east-1」
  - 對於 IoT 感應器，維度名稱為「裝置 ID」，維度值為「12345」
- 測量 - 由記錄測量的實際值。 例如股價、 CPU或 記憶體使用率，以及溫度或濕度讀數。指標由指標名稱和指標值組成。請考量下列範例：
  - 對於股票價格，指標名稱為「股票價格」，指標值是某個時間點的實際股票價格。
  - 對於CPU使用率，測量名稱為「CPU使用率」，而測量值為實際CPU使用率。

  可在 Timestream 中將 的 測量建模 LiveAnalytics 為多測量或單一測量記錄。如需詳細資訊，請參閱多測量記錄與單一測量記錄。
- 時間戳記 - 指示何時為指定記錄收集量值。的 Timestream LiveAnalytics 支援奈秒精細度的時間戳記。
- 表 - 一組相關時間序列的容器。
- 資料庫 - 資料表的頂層容器。

LiveAnalytics 概念的 Timestream 摘要

資料庫包含 0 個以上的資料表 。每個資料表包含 0 個或更多時間序列 。每次序列都包含一段指定時間間隔內的記錄，以指定的精細度 表示。每次序列都可以使用其中繼資料或維度 、其資料或量值 及其時間戳記 來描述。

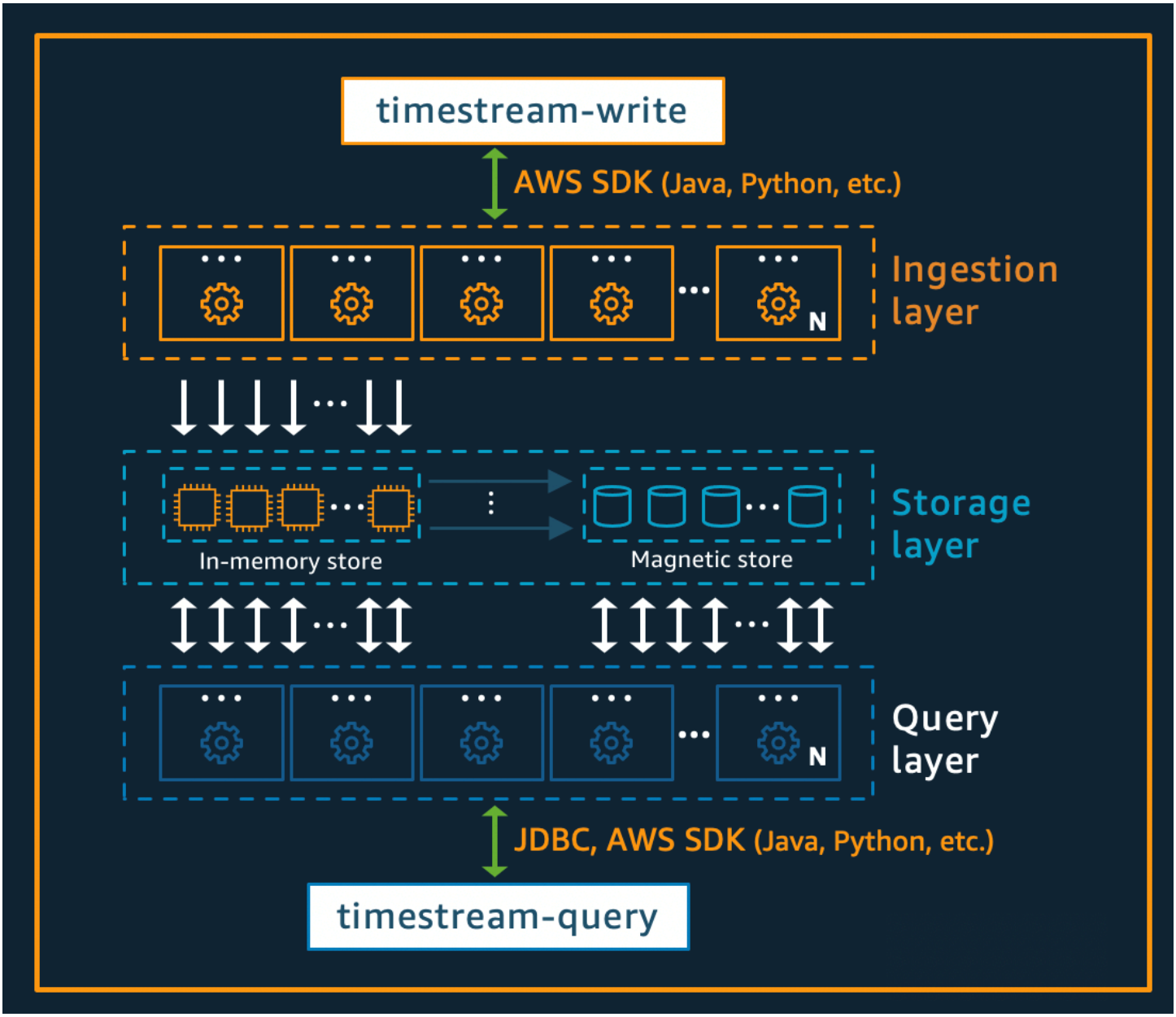


架構

Amazon Timestream for Live Analytics 從頭開始設計，以大規模收集、儲存和處理時間序列資料。其無伺服器架構支援可獨立擴展的完全解耦資料擷取、儲存和查詢處理系統。此設計可簡化每個子系統，讓您更輕鬆地實現堅定的可靠性、消除擴展瓶頸，並減少相關系統故障的機會。隨著系統擴展，每個因素都變得更加重要。

主題

- 寫入架構
- 儲存架構
- 查詢架構
- 行動架構



寫入架構

寫入時間序列資料時，Amazon Timestream for Live Analytics 會將資料表、分割區的寫入路由至處理高輸送量資料寫入的容錯記憶體存放區執行個體。記憶體存放區反過來在獨立的儲存系統中實現耐用性，該儲存系統會在三個可用區域（ ）中複寫資料AZs。複寫是以數量為基礎，因此節點或整個 AZ 的遺失不會中斷寫入可用性。在近乎即時的時間內，其他記憶體內儲存節點會同步至資料，以提供查詢。讀取器複本節點AZs也會跨越，以確保高讀取可用性。

Timestream for Live Analytics 支援將資料直接寫入磁性存放區，以用於產生較低輸送量延遲到達資料的應用程式。延遲到達資料是指時間戳記早於目前時間的資料。與記憶體存放區中的高輸送量寫入類似，寫入磁性存放區的資料會複寫為三個，AZs而複寫是以數量為基礎。

無論是將資料寫入記憶體或磁性儲存，Timestream for Live Analytics 都會在資料寫入儲存體之前，自動編製索引和分割資料。單一 Timestream for Live Analytics 資料表可能會有數百個、數千個或甚至數百萬個分割區。個別分割區不會直接彼此通訊，也不會共用任何資料（共用無結構）。而是透過高可用性的分割區追蹤和索引服務來追蹤資料表的分割區。這提供了另一個專為將系統中的故障影響降至最低而設計的疑慮分離，並降低了關聯故障的可能性。

儲存架構

當資料存放在 Timestream for Live Analytics 中時，資料會根據以資料寫入的內容屬性，依時間順序以及跨時間進行組織。除了時間之外，具有分隔「空間」的分割方案對於大規模擴展時間序列系統非常重要。這是因為大多數時間序列資料是在目前時間或前後寫入。因此，僅按時間分割在分發寫入流量或允許在查詢時間有效刪除資料方面沒有什麼好的作用。這對極端擴展時間序列處理很重要，它允許 Timestream for Live Analytics 以無伺服器方式擴展比其他領先系統更高的數量級。產生的分割區稱為「並列」，因為它們代表二維空間的分割（其設計為類似大小）。Live Analytics 資料表的時間串流一開始是單一分割區（動態磚），然後視需要在空間維度中分割。當動態磚達到特定大小時，它們接著會在時間維度中分割，以便在資料大小增加時實現更好的讀取平行處理。

Timestream for Live Analytics 的設計是為了自動管理時間序列資料的生命週期。Timestream for Live Analytics 提供兩個資料存放區：記憶體內存放區和符合成本效益的磁性存放區。它也支援設定資料表層級政策，以在各存放區之間自動傳輸資料。傳入的高輸送量資料會寫入記憶體存放區，其中的資料會針對寫入進行最佳化，以及針對目前時間執行的讀取，以為儀表板和警示類型查詢提供動力。當寫入、警示和儀表板需求的主要時間範圍已過時，可讓資料自動從記憶體存放區流向磁性存放區，以最佳化成本。Timestream for Live Analytics 允許為此在記憶體存放區上設定資料保留政策。延遲抵達的資料寫入會直接寫入磁性存放區。

一旦資料在磁性存放區中可用（由於記憶體存放區保留期過期或直接寫入磁性存放區），就會將其重新組織為針對大型磁碟區資料讀取進行高度最佳化的格式。磁性存放區也具有資料保留政策，如果資料超過其實用性的時間閾值，則可以設定該政策。當資料超過為磁性存放區保留政策定義的時間範圍時，會自動將其移除。因此，使用 Timestream for Live Analytics 時，除了某些組態之外，資料生命週期管理會無縫地在幕後進行。

查詢架構

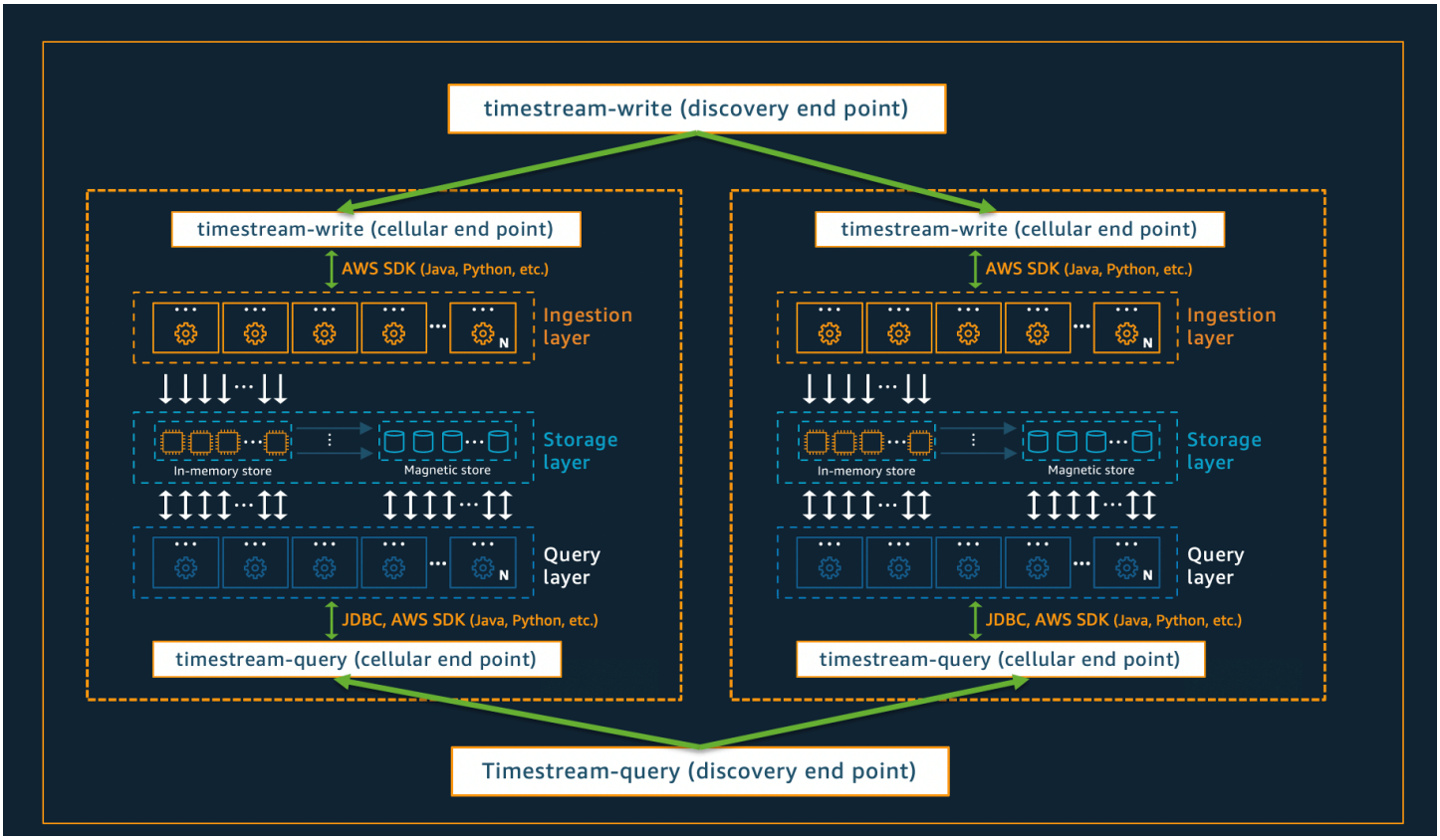
Live Analytics SQL 查詢的時間串流會以文法表示，該文法具有時間序列特定支援（時間序列特定資料類型和函數）的延伸，因此對於已熟悉 的開發人員而言，學習曲線很容易SQL。然後，查詢會由適應

性分散式查詢引擎處理，該引擎會使用動態磚追蹤和索引服務的中繼資料，在發出查詢時無縫存取和合併跨資料存放區的資料。這使得 體驗與客戶產生良好共鳴，因為它會將許多 Rube Goldberg 複雜性分解為簡單且熟悉的資料庫抽象。

查詢由專用工作者群執行，其中註冊執行指定查詢的工作者數量取決於查詢複雜性和資料大小。透過大量平行處理，在查詢執行期機群和系統的儲存機群上，可實現對大型資料集的複雜查詢效能。快速有效地分析大量資料的能力是 Timestream for Live Analytics 的最大優勢之一。執行超過 TB 或甚至 PB 資料的單一查詢，可能會有數千台機器同時處理。

行動架構

為了確保 Timestream for Live Analytics 可以為您的應用程式提供幾乎無限的擴展，同時確保 99.99% 的可用性，系統也使用行動架構設計。Timestream for Live Analytics 不會將系統整套擴展，而是將其分割成多個較小的自我複本，稱為儲存格 。這允許完整大規模測試儲存格，並防止一個儲存格中的系統問題影響指定區域中任何其他儲存格中的活動。雖然 Timestream for Live Analytics 旨在支援每個區域的多個儲存格，但請考慮下列虛構案例，其中一個區域中有 2 個儲存格。



在上述案例中，資料擷取和查詢請求會先由探索端點分別處理，以進行資料擷取和查詢。然後，探索端點會識別包含客戶資料的儲存格，並將請求導向至該儲存格的適當擷取或查詢端點。使用 時SDKs，系統會透明地為您處理這些端點管理任務。

> **Note**
>
> 將VPC端點與 Timestream for Live Analytics 搭配使用，或直接存取 Timestream for Live Analytics RESTAPI的操作時，您將需要直接與行動端點互動。如需如何執行此操作的指引，請參閱VPC端點以取得如何設定VPC端點的指示，以及端點探索模式以取得直接叫用RESTAPI操作的指示。

寫入

您可以從連接的裝置、IT 系統和工業設備收集時間序列資料，並將其寫入 Timestream for Live Analytics。Timestream for Live Analytics 可讓您在時間序列屬於相同資料表時，從單一時間序列和/或單一寫入請求中多個序列的資料點寫入資料點。為了您的方便，Timestream for Live Analytics 為您提供彈性結構描述，可根據您調用寫入資料庫時指定的測量值的維度名稱和資料類型，自動偵測 Timestream for Live Analytics 資料表的資料欄名稱和資料類型。您也可以將批次資料寫入 Timestream for Live Analytics。

> **Note**
>
> Timestream for Live Analytics 支援最終的讀取一致性語義。這表示當您在將一批資料寫入 Timestream for Live Analytics 後立即查詢資料時，查詢結果可能不會反映最近完成的寫入操作的結果。結果也可能包含一些過時的資料。同樣地，在寫入具有一或多個新維度的時間序列資料時，查詢可以在短時間內傳回部分資料欄子集。如果您在短時間內重複這些查詢請求，結果應該會傳回最新的資料。

您可以使用 AWS SDKs、 或透過 、AWS CLI、、AWS LambdaAWS IoT CoreAmazon Managed Service for Apache FlinkAmazon Kinesis、 Amazon MSK和 寫入資料開放原始碼 Telegraf。

主題

- 資料類型
- 沒有前期結構描述定義
- 寫入資料 （ 插入 和 upserts ）
- 讀取的最終一致性
- 批次寫入 WriteRecords API
- 批次載入

- 在操作和批次載入之間 WriteRecords API進行選擇

資料類型

Timestream for Live Analytics 支援下列寫入資料類型。

資料類型	描述
BIGINT	代表 64 位元已簽署整數。
BOOLEAN	表示邏輯的兩個真相值，即 true 和 false。
DOUBLE	64 位元可變精度，實作適用於二進位浮點運算IEEE的標準 754。 **Note** 和InfinityNaN雙值的查詢語言函數可用於查詢。但您無法將這些值寫入 Timestream。
VARCHAR	具有選用長度上限的變數長度字元資料。上限為 2 KB。
MULTI	多測量記錄的資料類型。此資料類型包含類型 BIGINT、、VARCHAR、 BOOLEAN DOUBLE和 的一或多個量值TIMESTAMP 。
TIMESTAMP	代表在 中使用奈秒精度時間的時間執行個體UTC，追蹤自 Unix 時間以來的時間。此資料類型目前僅支援多量值記錄 （即在類型 的量值內MULTI）。 *YYYY*-*MM*-*DD* *hh*:*mm*:*ss*.*sssssssss* 將 範圍內的支援時間戳記寫入 1970-01-01 00:00:00.000000000 2262-04-11 23:47:16.854775807 。

沒有前期結構描述定義

將資料傳送至 Amazon Timestream for Live Analytics 之前，您必須使用 AWS Management Console、Timestream for Live Analytics SDKs或 Timestream for Live Analytics API操作建立資料庫和

資料表。如需詳細資訊，請參閱 建立 資料庫 和 建立資料表。建立資料表時，您不需要預先定義結構描述。Amazon Timestream for Live Analytics 會根據傳送的資料點的量值和維度自動偵測結構描述，因此您不再需要離線變更結構描述，以適應快速變化的時間序列資料。

寫入資料 （ 插入 和 upserts ）

Amazon Timestream for Live Analytics 中的寫入操作可讓您插入和更新資料。根據預設，Amazon Timestream for Live Analytics 中的寫入會遵循第一個寫入器的語意，其中資料會儲存為僅附加，而重複的記錄會遭到拒絕。雖然第一個寫入器的語義符合許多時間序列應用程式的需求，但在某些情況下，應用程式需要以不容錯的方式更新現有記錄，和/或使用最後一個寫入器寫入資料則會獲得語義，其中具有最高版本的記錄會儲存在服務中。為了解決這些案例，Amazon Timestream for Live Analytics 提供升級資料的功能。Upsert 是一種操作，當記錄不存在時，將記錄插入系統，或在記錄存在時更新記錄。更新記錄時，會以不容錯過的方式更新記錄。

沒有要刪除的記錄層級操作。但資料表和資料庫可以刪除。

將資料寫入記憶體存放區和磁性存放區

Amazon Timestream for Live Analytics 提供將資料直接寫入記憶體存放區和磁性存放區的功能。記憶體存放區已針對高輸送量資料寫入進行最佳化，而磁性存放區已針對延遲到達資料的低輸送量寫入進行最佳化。

延遲到達資料是指時間戳記早於目前時間且超出記憶體存放區保留期的資料。您必須透過啟用資料表的磁性存放區寫入，明確啟用將延遲到達資料寫入至磁性存放區的功能。此外，`MagneticStoreRejectedDataLocation` 會在建立資料表時定義。若要寫入磁性存放區， 的呼叫者`WriteRecords`必須具有在建立資料表`MagneticStoreRejectedDataLocation`期間在 中指定的 S3 儲存貯體的`S3:PutObject`許可。如需詳細資訊，請參閱 CreateTable、 WriteRecords和 PutObject。

使用單一測量記錄和多測量記錄寫入資料

Amazon Timestream for Live Analytics 提供使用兩種記錄類型寫入資料的功能，即單一測量記錄和多測量記錄。

單一測量記錄

單一量值記錄可讓您為每個記錄傳送單一量值。使用此格式將資料傳送至 Timestream for Live Analytics 時，Timestream for Live Analytics 會為每個記錄建立一個資料表列。這表示，如果裝置發出 4 個指標，且每個指標都是以單一測量記錄傳送，則 Live Analytics 的 Timestream 將在資料表中建立 4 列來存放此資料，而且每列都會重複裝置屬性。當您想要從應用程式監控單一指標，或應用程式不會同時發出多個指標時，建議使用此格式。

多量值記錄

使用多量值記錄，您可以將多個量值存放在單一資料表列中，而不是儲存每個資料表列的一個量值。因此，多測量記錄可讓您將現有資料從關聯式資料庫遷移至 Amazon Timestream for Live Analytics，且變更最少。

您也可以在單一寫入請求中批次處理比單一測量記錄更多的資料。這會增加資料寫入輸送量和效能，並降低資料寫入的成本。這是因為在寫入請求中批次處理更多資料，可讓 Amazon Timestream for Live Analytics 在單一寫入請求中識別更多可重複的資料（如適用），並僅針對重複資料收取一次費用。

主題

- 多測量記錄
- 使用過去或未來的時間戳記寫入資料

多測量記錄

透過多測量記錄，您可以將時間序列資料以更精簡的格式儲存在記憶體和磁性儲存中，這有助於降低資料儲存成本。此外，精簡的資料儲存本身有助於撰寫更簡單的查詢以進行資料擷取、改善查詢效能，並降低查詢成本。

此外，多測量記錄也支援在時間序列記錄中儲存多個時間戳記的TIMESTAMP資料類型。TIMESTAMP多測量記錄中的屬性支援未來或過去的時間戳記。因此，多測量記錄有助於改善效能、成本和查詢簡單性，並為存放不同類型的相關測量提供更多彈性。

優勢

以下是使用多測量記錄的好處。

- 效能和成本 – 多量值記錄可讓您在單一寫入請求中寫入多個時間序列量值。這會增加寫入輸送量，並降低寫入成本。透過多測量記錄，您可以更精簡的方式儲存資料，這有助於降低資料儲存成本。多測量記錄的精簡資料儲存會導致查詢處理的資料較少。這旨在改善整體查詢效能，並協助降低查詢成本。
- 查詢簡單 – 使用多量值記錄，您不需要在查詢中寫入複雜的常見資料表表達式（CTEs），即可讀取具有相同時間戳記的多個量值。這是因為量值會以資料欄形式儲存在單一資料表列中。因此，多測量記錄可寫入更簡單的查詢。
- 資料建模彈性 – 您可以使用TIMESTAMP資料類型和多測量記錄，將未來的時間戳記寫入Timestream for Live Analytics。除了記錄中的時間欄位之外，多測量記錄還可以具有TIMESTAMP資料類型的多個屬性。TIMESTAMP 屬性在多量值記錄中，可以在未來或過去有時間戳記，並且行為

類似於時間欄位，但 Timestream for Live Analytics 不會在多量值記錄中的類型值TIMESTAMP上編製索引。

使用案例

您可以針對在任何指定時間從相同裝置產生多個測量的任何時間序列應用程式，使用多測量記錄。以下是一些範例應用程式。

- 影片串流平台，可在指定時間產生數百個指標。
- 產生測量結果的醫療設備，例如血氧水準、心率和脈波。
- 產生指標、溫度和天氣感應器的石油裝備等工業設備。
- 使用一或多個微服務建構的其他應用程式。

範例：監控影片串流應用程式的效能和運作狀態

請考慮在 200 個EC2執行個體上執行的影片串流應用程式。您想要使用 Amazon Timestream for Live Analytics 來儲存和分析從應用程式發出的指標，以便您可以了解應用程式的效能和運作狀態、快速識別異常、解決問題並探索最佳化機會。

我們將使用單一測量記錄和多測量記錄來模擬此案例，然後比較/對比這兩種方法。對於每種方法，我們都會做出下列假設。

- 每個EC2執行個體都會每秒發出四個量值（video_startup_time、rebuffering_ratio、Video_playback_failures 和 average_frame_rate）和四個維度（device_id、Device_type、os_version 和 region）。
- 您想要將 6 小時的資料儲存在記憶體存放區，並將 6 個月的資料儲存在磁性存放區。
- 為了識別異常，您設定了 10 個每分鐘執行的查詢，以識別過去幾分鐘內的任何異常活動。您也建立了具有八個小工具的儀表板，可顯示過去 6 小時的資料，以便您可以有效地監控應用程式。此儀表板可在任何指定時間由五個使用者存取，並且每小時自動重新整理一次。

使用單一量值記錄

資料建模：使用單一量值記錄，我們將為四個量值（影片啟動時間、重新緩衝比率、影片播放失敗和平均影格速率）中的每一個量值建立一個記錄。每個記錄都會有四個維度（device_id、device_type、os_version 和 region）和時間戳記。

寫入 ：當您將資料寫入 Amazon Timestream for Live Analytics 時，記錄的建構方式如下。

```
public void writeRecords() {
    System.out.println("Writing records");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();

    final Dimension device_id = new
 Dimension().withName("device_id").withValue("12345678");
    final Dimension device_type = new
 Dimension().withName("device_type").withValue("iPhone 11");
    final Dimension os_version = new
 Dimension().withName("os_version").withValue("14.8");
    final Dimension region = new Dimension().withName("region").withValue("us-east-1");

    dimensions.add(device_id);
    dimensions.add(device_type);
    dimensions.add(os_version);
    dimensions.add(region);

    Record videoStartupTime = new Record()
        .withDimensions(dimensions)
        .withMeasureName("video_startup_time")
        .withMeasureValue("200")
        .withMeasureValueType(MeasureValueType.BIGINT)
        .withTime(String.valueOf(time));
    Record rebufferingRatio = new Record()
        .withDimensions(dimensions)
        .withMeasureName("rebuffering_ratio")
        .withMeasureValue("0.5")
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));
    Record videoPlaybackFailures = new Record()
        .withDimensions(dimensions)
        .withMeasureName("video_playback_failures")
        .withMeasureValue("0")
        .withMeasureValueType(MeasureValueType.BIGINT)
        .withTime(String.valueOf(time));
    Record averageFrameRate = new Record()
        .withDimensions(dimensions)
        .withMeasureName("average_frame_rate")
        .withMeasureValue("0.5")
```

```
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));

    records.add(videoStartupTime);
    records.add(rebufferingRatio);
    records.add(videoPlaybackFailures);
    records.add(averageFrameRate);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ": "
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

當您儲存單一測量記錄時，資料會以邏輯方式表示，如下所示。

時間	device_id	device_type	os_version	region	measure_name	measure_value::bigint	measure_value::double
2021-09-07 21:48:44.0000000	12345678	iPhone 11	14.8	us-east-1	video_startup_time	200	

時間	device_id	device_type	os_version	region	measure_name	measure_value::bigint	measure_value::double
2021-09-07 21:48:44.0000000	12345678	iPhone 11	14.8	us-east-1	rebuffering_ratio		0.5
2021-09-07 21:48:44.0000000	12345678	iPhone 11	14.8	us-east-1	Video_playback_failures	0	
2021-09-07 21:48:44.0000000	12345678	iPhone 11	14.8	us-east-1	average_frame_rate		0.85
2021-09-07 21:53:44.0000000	12345678	iPhone 11	14.8	us-east-1	video_startup_time	500	
2021-09-07 21:53:44.0000000	12345678	iPhone 11	14.8	us-east-1	rebuffering_ratio		1.5

時間	device_id	device_type	os_version	region	measure_name	measure_value::bigint	measure_value::double
2021-09-07 21:53:44.0000000	12345678	iPhone 11	14.8	us-east-1	Video_playback_failures	10	
2021-09-07 21:53:44.0000000	12345678	iPhone 11	14.8	us-east-1	average_frame_rate		0.2

查詢 ：您可以撰寫查詢，以擷取過去 15 分鐘內收到相同時間戳記的所有資料點，如下所示。

```
with cte_video_startup_time as ( SELECT time, device_id, device_type, os_version,
 region, measure_value::bigint as video_startup_time FROM table where time >= ago(15m)
 and measure_name=”video_startup_time”),
cte_rebuffering_ratio as ( SELECT time, device_id, device_type, os_version, region,
 measure_value::double as rebuffering_ratio FROM table where time >= ago(15m) and
 measure_name=”rebuffering_ratio”),
cte_video_playback_failures as ( SELECT time, device_id, device_type, os_version,
 region, measure_value::bigint as video_playback_failures FROM table where time >=
 ago(15m) and measure_name=”video_playback_failures”),
cte_average_frame_rate as ( SELECT time, device_id, device_type, os_version, region,
 measure_value::double as average_frame_rate FROM table where time >= ago(15m) and
 measure_name=”average_frame_rate”)
SELECT a.time, a.device_id, a.os_version, a.region, a.video_startup_time,
 b.rebuffering_ratio, c.video_playback_failures, d.average_frame_rate FROM
 cte_video_startup_time a, cte_buffering_ratio b, cte_video_playback_failures c,
 cte_average_frame_rate d WHERE
a.time = b.time AND a.device_id = b.device_id AND a.os_version = b.os_version AND
 a.region=b.region AND
a.time = c.time AND a.device_id = c.device_id AND a.os_version = c.os_version AND
 a.region=c.region AND
```

```
a.time = d.time AND a.device_id = d.device_id AND a.os_version = d.os_version AND
 a.region=d.region
```

工作負載成本：此工作負載的估計成本為每月 373.23 美元，並具有單一測量記錄

使用多測量記錄

資料建模：使用多量值記錄，我們將建立一個記錄，其中包含所有四個量值（影片啟動時間、重新緩衝比率、影片播放失敗和平均影格速率）、所有四個維度（device_id、Device_type、os_version 和區域），以及時間戳記。

寫入 ：當您將資料寫入 Amazon Timestream for Live Analytics 時，記錄的建構方式如下。

```
public void writeRecords() {
    System.out.println("Writing records");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();

    final Dimension device_id = new
 Dimension().withName("device_id").withValue("12345678");
    final Dimension device_type = new
 Dimension().withName("device_type").withValue("iPhone 11");
    final Dimension os_version = new
 Dimension().withName("os_version").withValue("14.8");
    final Dimension region = new Dimension().withName("region").withValue("us-east-1");

    dimensions.add(device_id);
    dimensions.add(device_type);
    dimensions.add(os_version);
    dimensions.add(region);

    Record videoMetrics = new Record()
        .withDimensions(dimensions)
        .withMeasureName("video_metrics")
        .withTime(String.valueOf(time));
        .withMeasureValueType(MeasureValueType.MULTI)
        .withMeasureValues(
          new MeasureValue()
         .withName("video_startup_time")
         .withValue("0")
```

```
        .withValueType(MeasureValueType.BIGINT),
        new MeasureValue()
  .withName("rebuffering_ratio")
        .withValue("0.5")
        .withType(MeasureValueType.DOUBLE),
         new MeasureValue()
        .withName("video_playback_failures")
        .withValue("0")
        .withValueType(MeasureValueType.BIGINT),
   new MeasureValue()
         .withName("average_frame_rate")
        .withValue("0.5")
        .withValueType(MeasureValueType.DOUBLE))

    records.add(videoMetrics);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ": "
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

當您儲存多測量記錄時，資料會以邏輯方式表示，如下所示。

時間	device_id	device_type	os_version	region	measure_name	video_startup_time	rebuffering_ratio	Video_playback_failures	average_frame_rate
2021-09-7 21：48：44 .00000	1234567	iPhone 11	14.8	us-east-1	Video_metrics	200	0.5	0	0.85
2021-09-7 21：53：44 .00000	1234567	iPhone 11	14.8	us-east-1	Video_metrics	500	1.5	10	0.2

查詢 ：您可以撰寫查詢，以擷取過去 15 分鐘內收到相同時間戳記的所有資料點，如下所示。

```
SELECT time, device_id, device_type, os_version, region, video_startup_time,
 rebuffering_ratio, video_playback_failures, average_frame_rate FROM table where time
 >= ago(15m)
```

工作負載成本 ：具有多測量記錄的工作負載成本估計為 127.43 美元。

> **Note**
>
> 在此情況下，使用多測量記錄可減少 2.5 倍的整體每月預估支出，資料寫入成本減少 3.3 倍，儲存成本減少 3.3 倍，查詢成本減少 1.2 倍。

使用過去或未來的時間戳記寫入資料

Timestream for Live Analytics 提供透過幾個不同的機制，在記憶體儲存保留時段之外寫入時間戳記的資料的功能。

- 磁性存放區寫入 – 您可以透過磁性存放區寫入，將延遲抵達的資料直接寫入磁性存放區。若要使用磁性存放區寫入，您必須先啟用資料表的磁性存放區寫入。然後，您可以使用與將資料寫入記憶體

存放區相同的機制，將資料擷取至資料表。Amazon Timestream for Live Analytics 會根據其時間戳記，自動將資料寫入磁性存放區。

> **Note**
>
> 與將資料寫入記憶體存放區不同，磁性存放區的 write-to-read延遲最長可達 6 小時，其中 write-to-read延遲在秒以下的範圍內。

- TIMESTAMP 量值的資料類型 – 您可以使用TIMESTAMP資料類型來存放過去、現在或未來的資料。除了記錄中的時間欄位之外，多測量記錄還可以具有TIMESTAMP資料類型的多個屬性。TIMESTAMP 屬性在多量值記錄中，可以有未來的時間戳記或過去的時間戳記，並且行為類似於時間欄位，但 Timestream for Live Analytics 不會在多量值記錄中的類型值TIMESTAMP上編製索引。

> **Note**
>
> TIMESTAMP 資料類型僅支援多測量記錄。

讀取的最終一致性

Timestream for Live Analytics 支援最終的讀取一致性語義。這表示當您在將一批資料寫入 Timestream for Live Analytics 後立即查詢資料時，查詢結果可能不會反映最近完成的寫入操作的結果。如果您在短時間內重複這些查詢請求，結果應該會傳回最新的資料。

批次寫入 WriteRecords API

Amazon Timestream for Live Analytics 可讓您在單一寫入請求中，從單一時間序列和/或多個序列的資料點寫入資料點。從效能和成本角度來看，在單一寫入操作中批次多個資料點是有利的。如需詳細資訊，請參閱計量和定價一節寫入中的 。

> **Note**
>
> 您對 Timestream for Live Analytics 的寫入請求可能會隨著 Timestream for Live Analytics 擴展而限流，以適應應用程式的資料擷取需求。如果您的應用程式遇到限流例外狀況，您必須繼續以相同（或更高）的輸送量傳送資料，以便 Timestream for Live Analytics 自動擴展到應用程式的需求。

批次載入

透過 Amazon Timestream 的批次載入 LiveAnalytics，您可以將存放在 Amazon S3 中的CSV檔案以批次方式擷取到 Timestream。透過這項新功能，您可以在 Timestream 中為 取得資料， LiveAnalytics 而不必依賴其他工具或編寫自訂程式碼。您可以使用批次載入，以彈性的等待時間回填資料，例如查詢或分析時不需要的資料。

您可以使用 AWS Management Console、 AWS CLI和 AWS 來建立批次載入任務SDKs。如需詳細資訊，請參閱搭配主控台使用批次載入、搭配 使用批次載入 AWS CLI及搭配 使用批次載入 AWS SDKs。

如需批次載入的詳細資訊，請參閱 在 Timestream 中使用 的批次載入 LiveAnalytics。

在操作和批次載入之間 WriteRecords API進行選擇

透過 WriteRecords API操作，您可以將串流時間序列資料寫入 Timestream， LiveAnalytics 因為系統會產生資料。透過使用 WriteRecords，您可以持續即時擷取單一資料點或較小的資料批次。Timestream for LiveAnalytics 為您提供彈性結構描述，可根據您調用寫入資料庫時指定的資料點的維度名稱和資料類型，自動偵測 LiveAnalytics 資料表的 Timestream 資料欄名稱和資料類型。

相反地，批次載入會使用您定義的資料模型 LiveAnalytics，將批次時間序列資料從來源檔案（CSV 檔案）強健擷取到 的 Timestream。將批次載入與來源檔案搭配使用的幾個範例，是透過概念 LiveAnalytics 驗證大量匯入時間序列資料，以評估 Timestream，從離線一段時間的 IoT 裝置大量匯入時間序列資料，以及將歷史時間序列資料從 Amazon S3 遷移至 的 Timestream LiveAnalytics。如需批次載入的相關資訊，請參閱 在 Timestream 中使用 的批次載入 LiveAnalytics。

這兩種解決方案都安全、可靠且效能良好。

使用 WriteRecords 時機：

- 每次請求串流較少量（少於 10 MB）的資料。
- 填入現有資料表。
- 從日誌串流擷取資料。
- 執行即時分析。
- 需要較低的延遲。

在下列情況下使用批次載入：

- 在CSV檔案中擷取大量源自 Amazon S3 的資料。如需限制的詳細資訊，請參閱 配額。

- 填入新資料表，例如在資料遷移的情況下。
- 使用歷史資料擴充資料庫 （擷取至新資料表）。
- 您有緩慢變更或完全不變更的來源資料。
- 您有彈性的等待時間，因為批次載入任務可能處於待定狀態，直到資源可用為止，特別是當您載入大量資料時。批次載入適合不需要隨時可用於查詢或分析的資料，以增加清晰度。

儲存

Live Analytics 的 Timestream 會儲存和組織您的時間序列資料，以最佳化查詢處理時間並降低儲存成本。它提供資料儲存分層，並支援兩個儲存層：記憶體存放區和磁性存放區。記憶體存放區已針對高輸送量資料寫入和快速 point-in-time查詢進行最佳化。磁性存放區已針對低輸送量的延遲取得資料寫入、長期資料儲存和快速分析查詢進行最佳化。

Timestream for Live Analytics 透過在單一 的不同可用區域中自動複寫記憶體和磁性存放區資料，確保資料的耐久性 AWS 區域。在確認您的寫入請求為完成之前，所有資料都會寫入磁碟。

Timestream for Live Analytics 可讓您設定保留政策，將資料從記憶體存放區移至磁性存放區。當資料達到設定的值時，Timestream for Live Analytics 會自動將資料移至磁性存放區。您也可以在磁性存放區上設定保留值。當資料過期離開磁性存放區時，會將其永久刪除。

例如，請考慮您設定記憶體存放區以存放一週值的資料，以及磁性存放區以存放 1 年值資料的案例。資料使用與資料點相關聯的時間戳記來計算時間。當記憶體存放區中的資料一週後，會自動移至磁性存放區。然後會在磁性存放區中保留一年。當資料變成一年時，會從 Timestream for Live Analytics 中刪除。記憶體儲存體和磁性儲存體的保留值會累積定義資料將儲存在 Timestream for Live Analytics 中的時間量。這表示在上述情況下，從資料到達時起，資料會儲存在 Timestream for Live Analytics 中，總共 1 年 1 週。

> **Note**
>
> 當您升級記憶體或磁性存放區的保留期間時，保留變更會從該時間點開始生效。例如，如果記憶體儲存體的保留期設定為 2 小時，然後透過更新資料表保留政策將 變更為 24 小時，則記憶體儲存體將能夠保存 24 小時的資料，但在進行此變更的 22 小時後，將填入 24 小時的資料。Timestream for Live Analytics 不會從磁性存放區擷取資料以填入記憶體存放區。

為了確保時間序列資料的安全性，Timestream for Live Analytics 中的資料預設一律會加密。這適用於傳輸中和靜態的資料。此外，Timestream for Live Analytics 可讓您使用客戶受管金鑰來保護磁性存放區中的資料。如需客戶受管金鑰的詳細資訊，請參閱 AWS KMS keys。

查詢

透過 Timestream for Live Analytics，您可以輕鬆存放和分析 的指標 DevOps、適用於 IoT 應用程式的感應器資料，以及用於設備維護的工業遙測資料，以及許多其他使用案例。Timestream for Live Analytics 中的專用自適應查詢引擎可讓您使用單一SQL陳述式跨儲存層存取資料。它可以透明地存取和合併跨儲存層的資料，而無需您指定資料位置。您可以使用 SQL 查詢 Timestream for Live Analytics 中的資料，從一或多個資料表擷取時間序列資料。您可以存取資料庫和資料表的中繼資料資訊。Timestream for Live Analytics SQL也支援時間序列分析的內建函數。如需其他詳細資訊，請參閱查詢語言參考參考。

Timestream for Live Analytics 設計為具有完全解耦的資料擷取、儲存和查詢架構，其中每個元件都可以獨立於其他元件進行擴展（允許它為應用程式的需求提供幾乎無限的擴展）。這表示當您的應用程式每天傳送數百 TB 的資料或執行處理少量或大量資料的數百萬查詢時，即時分析的 Timestream 不會「提示」。隨著資料隨時間增加，Timestream for Live Analytics 中的查詢延遲大部分保持不變。這是因為 Timestream for Live Analytics 查詢架構可以利用大量平行處理來處理較大的資料磁碟區，並自動擴展以符合應用程式的查詢輸送量需求。

資料模型

Timestream 支援兩個查詢資料模型：平面模型和時間序列模型。

> **Note**
>
> Timestream 中的資料是使用平面模型儲存，它是查詢資料的預設模型。時間序列模型是一種查詢時間概念，用於時間序列分析。

- 平面模型
- 時間序列模型

平面模型

平面模型是 Timestream 的預設查詢資料模型。它以表格格式表示時間序列資料。維度名稱、時間、量值名稱和量值會顯示為資料欄。資料表中的每一列都是對應於時間序列內特定時間的測量的原子資料點。時間串流資料庫、資料表和資料欄有一些命名限制。這些會在 中說明服務限制。

下表顯示 Timestream 儲存資料的方式，當資料作為單一測量記錄傳送時，代表EC2執行個體的使用CPU率、記憶體使用率和網路活動。在此情況下，維度是IDsEC2執行個體的區域、可用區域、虛擬私有雲端和執行個體。這些措施是EC2執行個體的使用CPU率、記憶體使用率和傳入的網路資料。

資料欄區域、az、vpc 和 instance_id 包含維度值。資料欄時間包含每個記錄的時間戳記。資料欄 measure_name 包含 cpu-utilization、meory_utilization 和 network_bytes_in 表示的量值名稱。資料欄 measure_value：：double 包含以倍數發出的測量（例如CPU使用率和記憶體使用率）。資料欄 measure_value：：bigint 包含以整數發出的測量，例如傳入的網路資料。

時間	region	az	vpc	instance_id	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19：00：00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	35.0	null
2019-12-04 19：00：01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	38.2	null
2019-12-04 19：00：02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	45.3	null
2019-12-04 19：00：00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	memory_utilization	54.9	null
2019-12-04 19：00：01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	memory_utilization	42.6	null
2019-12-04 19：00：02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	memory_utilization	33.3	null

時間	region	az	vpc	instance_id	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19：00：00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	network_bytes	34，400	null
2019-12-04 19：00：01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	network_bytes	1,500	null
2019-12-04 19：00：02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	network_bytes	6,000	null

下表顯示 Timestream 如何儲存資料，當資料作為多測量記錄傳送時，代表EC2執行個體CPU的使用率、記憶體使用率和網路活動的說明性範例。

時間	region	az	vpc	instance_id	measure_ame	cpu_utilization	memory_ilization	network_bytes
2019-12-04 19：00：00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcde0	指標	35.0	54.9	34，400
2019-12-04 19：00：01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcde0	指標	38.2	42.6	1,500

時間	region	az	vpc	instance_id	measure_name	cpu_utilization	memory_utilization	network_bytes
2019-12-04 19：00：02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcde0	指標	45.3	33.3	6,600

時間序列模型

時間序列模型是用於時間序列分析的查詢時間建構。它以（時間、測量值）對的順序表示資料。Timestream 支援時間序列函數，例如插補，可讓您填補資料中的差距。若要使用這些函數，您必須使用 create_time_series 等函數將資料轉換為時間序列模型。如需更多詳細資訊查詢語言參考，請參閱 。

使用較早的EC2執行個體範例，以下是以時脈表示的CPU使用率資料。

region	az	vpc	instance_id	cpu_utilization
us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	【{時間：2019-12-04 19：00：00.000000000，值：35}，{時間：2019-12-04 19：00：01.000000000，值：38.2}，{時間：2019-12-04 19：00：02.000000000，值：45.3}】

排程查詢

Amazon Timestream for Live Analytics 中的排程查詢功能是完全受管、無伺服器和可擴展的解決方案，用於計算和儲存彙總、彙總和其他形式的預先處理資料，通常用於支援操作儀表板、業務報告、臨

時分析和其他應用程式。排程查詢可讓即時分析更具效能且符合成本效益，因此您可以從資料中衍生其他洞見，並可以繼續做出更好的業務決策。

如需排程查詢的詳細資訊，請參閱 在 Timestream 中使用 的排程查詢 LiveAnalytics。

Timestream Compute Unit （ TCU ）

Amazon Timestream for Live Analytics 會測量針對 Timestream 運算單位 （ ） 中查詢需求的配置給您的運算容量TCU。一個由 4 vCPUs 和 16 GB 記憶體TCU組成。當您在 Timestream for Live Analytics 中執行查詢時，服務會根據查詢的複雜性和處理的資料量TCUs，依需求進行配置。查詢消耗TCUs的數量決定相關聯的成本。

> **Note**
>
> 2024 年 4 月 29 日後加入服務的所有 AWS 帳戶 項目，都會預設為使用 TCUs進行查詢定價。

在本主題中

MaxQuery TCU

此設定會指定服務在任何時間點用來服務查詢的運算單位數量上限。若要執行查詢，您必須將最小容量設定為 4 TCUs。您可以設定 4 的TCUs倍數上限，例如 4、8、16、32 等。您只需為工作負載使用的運算資源付費。例如，如果您將最大值設定為 TCUs 128，但持續僅使用 8 TCUs。您只需在使用 8 的期間支付費用TCUs。`MaxQueryTCU` 您帳戶中的預設值設定為 200。您可以使用 或 搭配 AWS Management Console 或 UpdateAccountSettingsAPI操作 AWS SDK，`MaxQueryTCU`從 4 調整為 1000 AWS CLI。

建議您MaxQueryTCU為 帳戶設定 。設定TCU上限有助於控制成本，方法是限制服務可用於查詢工作負載的運算單位數量。這可讓您更好地預測和管理查詢支出。

的帳單 TCU

每個 TCU 都會以每小時為單位計費，以每秒的精細程度計費，並持續至少 30 秒。這些運算單位的使用單位為 TCU小時。

當您執行查詢時，系統會向您收取查詢執行期間TCUs使用的 費用，以TCU小時為單位。例如：

- 您的工作負載使用 20 TCUs 達 3 小時。您需支付 60 TCU小時 （ 20 TCUs x 3 小時 ） 的費用。
- 您的工作負載會使用 10 TCUs 進行 30 分鐘，然後在接下來 30 分鐘TCUs使用 20 分鐘。您需支付 15 TCU小時 （ 10 TCUs x 0.5 小時 + 20 TCUs x 0.5 小時 ） 的費用。

每小時定價因 TCU而異 AWS 區域。如需其他詳細資訊，請參閱 Amazon Timestream 定價。隨著工作負載的增加，服務會自動將運算容量擴展至指定的TCU上限 （ MaxQueryTCU ），以維持一致的效能。此MaxQueryTCU設定可用作服務可擴展的運算容量上限。此設定可協助您控制運算資源的數量，進而控制成本。

設定 TCU

當您加入服務時，每個 AWS 帳戶 的預設MaxQueryTCU限制為 200。您可以隨時使用 AWS Management Console 或 UpdateAccountSettingsAPI操作搭配 AWS SDK或 ，視需要更新此限制 AWS CLI。

如果您不確定要設定的值，請監控帳戶的QueryTCU指標。此指標可在 AWS Management Console 和 Amazon 中使用 CloudWatch。此指標可讓您深入了解每分鐘TCUs使用的最大數量。根據歷史資料和您對未來成長的估算，設定 MaxQueryTCU 以適應用量的尖峰。我們建議您的頭頂至少比尖峰用量TCUs高 4-16。例如，如果QueryTCU過去 30 天內的峰值為 128，建議您MaxQueryTCU將 設定為 132 到 144。

估計所需的運算單位

運算單位可以同時處理查詢。若要判斷所需的運算單位數量，請考慮下表中的一般準則：

並行查詢	TCUs
7	4
14	8

並行查詢	TCUs
21	12

> **Note**
>
> - 這些是一般準則，所需的實際運算單位數量取決於幾個因素，例如：
>   - 查詢的有效並行。
>   - 查詢模式。
>   - 掃描的分割區數量。
>   - 其他工作負載特有的特性。
> - 本指南適用於最近幾分鐘到一小時資料掃描的查詢，並遵守 Timestream 查詢最佳實務和資料建模指南。
> - 視需要監控應用程式的效能和QueryTCU指標，以調整運算單位。

何時增加 MaxQueryTCU

在MaxQueryTCU下列情況下，您應該考慮增加 ：

- 您的峰值查詢消耗即將接近或達到目前設定的查詢上限 TCU。我們建議將查詢上限設定為TCU至少比峰值消耗TCUs高 4-16。
- 您的查詢正在傳回超過訊息 MaxQueryTCU的 4xx 錯誤。如果您預期工作負載的計劃增加，請重新檢視並TCU相應地調整設定的最大查詢。

何時減少 MaxQueryTCU

在下列情況下，您應該考慮減少 MaxQueryTCU ：

- 您的工作負載具有可預測且穩定的使用模式，而且您充分了解運算使用需求。將查詢上限降TCU至 TCU 4-16 內，超過峰值消耗量，有助於防止意外使用和成本。您可以使用 UpdateAccountSettingsAPI操作修改 值。
- 由於應用程式或使用者行為模式的變更，您的工作負載尖峰用量已隨著時間減少。降低上限TCU有助於降低意外成本。

> **Note**
>
> 根據您目前的用量，減少TCU上限變更可能需要最多 24 小時才能生效。您只會因為查詢實際使用的 TCUs而收到帳單。除非工作負載TCUs使用它們，否則具有更高的查詢TCU上限不會影響您的成本。

使用 CloudWatch 指標監控用量

若要監控您的TCU用量，Timestream for Live Analytics 提供下列 CloudWatch 指標：`QueryTCU`。此指標指定一分鐘內使用的運算單位數量，每分鐘發出。您可以選擇監控一分鐘內TCUs使用的最大值和最小值。您也可以在此指標上設定警示，以即時追蹤查詢成本。

了解運算單位用量的變化

查詢所需的運算資源數量可以根據多個參數增加或減少。例如，使用即時和分析查詢的資料磁碟區、資料擷取模式、查詢延遲、查詢形狀、查詢效率和查詢組合。這些參數可能會導致工作負載所需的TCU單位較高或較低。在未變更這些參數的穩定狀態下，您可能會觀察到工作負載所需的運算單位數量減少。因此，這可以降低您的每月成本。

此外，如果工作負載或資料中的任何這些參數變更，所需的運算單位數量可能會增加。當 Timestream 收到查詢時，根據查詢存取的資料分割區，Timestream 會決定運算資源的數量，以有效處理查詢。

Timestream 會根據您的擷取和查詢存取模式，定期最佳化資料配置。Timestream 透過將存取不足的分割區合併為單一分割區，或將熱分割區分割為多個分割區來實現效能，來執行最佳化。因此，相同查詢使用的運算容量可能會在不同的時間點略有不同。

> **選擇加入以使用查詢的TCU定價**
>
> 身為現有使用者，您可以進行一次性選擇加入，以使用 TCUs 來改善成本管理和移除每個查詢計量的最小位元組。您可以使用 AWS Management Console 或 UpdateAccountSettingsAPI操作搭配 AWS SDK或 來選擇加入 AWS CLI。在 API操作中，將 `QueryPricingModel` 參數設定為 `COMPUTE_UNITS`。
>
> 選擇使用運算型定價模型是不可逆的變更。

存取 的 Timestream LiveAnalytics

您可以使用 LiveAnalytics 主控台CLI或 存取 TimestreamAPI。如需存取 的 Timestream 的相關資訊 LiveAnalytics，請檢閱下列內容：

主題

- 註冊 AWS 帳戶
- 建立具有管理存取權的使用者
- 提供 Timestream 以供 LiveAnalytics 存取
- 授與程式設計存取權

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟以建立 。

若要註冊 AWS 帳戶

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

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

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

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時前往 https://aws.amazon.com/ 並選擇我的帳戶 來檢視目前的帳戶活動和管理您的帳戶。

建立具有管理存取權的使用者

註冊 後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center並建立管理使用者，這樣您就不會將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

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

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

2. 為您的根使用者開啟多重要素驗證 （MFA）。

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

建立具有管理存取權的使用者

1. 啟用IAM身分中心。

   如需指示，請參閱 AWS IAM Identity Center 使用者指南中的啟用 AWS IAM Identity Center。

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

   如需使用 IAM Identity Center 目錄 作為身分來源的教學課程，請參閱 AWS IAM Identity Center 使用者指南 中的使用 設定使用者存取權 IAM Identity Center 目錄。

以具有管理存取權的使用者身分登入

- 若要使用 IAM Identity Center 使用者登入，請使用您建立 IAM Identity Center 使用者時URL傳送到您電子郵件地址的登入。

  如需使用 IAM Identity Center 使用者登入的協助，請參閱 AWS 登入 使用者指南 中的登入 AWS 存取入口網站。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立遵循套用最低權限許可最佳實務的許可集。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的建立許可集。

2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的新增群組。

提供 Timestream 以供 LiveAnalytics 存取

存取 的 Timestream 所需的許可 LiveAnalytics 已授予管理員。對於其他使用者，您應該使用下列政策授予他們 Timestream LiveAnalytics 存取權：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "timestream:*",
        "kms:DescribeKey",
        "kms:CreateGrant",
        "kms:Decrypt",
        "dbqms:CreateFavoriteQuery",
        "dbqms:DescribeFavoriteQueries",
        "dbqms:UpdateFavoriteQuery",
        "dbqms:DeleteFavoriteQueries",
        "dbqms:GetQueryString",
        "dbqms:CreateQueryHistory",
        "dbqms:UpdateQueryHistory",
        "dbqms:DeleteQueryHistory",
        "dbqms:DescribeQueryHistory",
        "s3:ListAllMyBuckets"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 如需 的相關資訊dbqms，請參閱資料庫查詢中繼資料服務 的動作、資源和條件金鑰。如需詳細資訊，kms請參閱 Key Management Service 的動作、資源和條件金鑰。 AWS

授與程式設計存取權

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

若要授與使用者程式設計存取權，請選擇下列其中一個選項。

哪個使用者需要程式設計存取權？	到	By
人力身分 （在 IAM Identity Center 中管理的使用者）	使用暫時憑證簽署對 AWS CLI AWS SDKs、 或 的程式設計請求 AWS APIs。	請依照您要使用的介面所提供的指示操作。 • 對於 AWS CLI，請參閱 使用者指南 中的設定 AWS CLI 要使用 AWS IAM Identity Center的 。 AWS Command Line Interface • 如需 AWS SDKs、工具和 AWS APIs，請參閱 AWS SDKs和 工具參考指南 中的IAM身分中心身分驗證。
IAM	使用暫時憑證簽署對 AWS CLI AWS SDKs、 或 的程式設計請求 AWS APIs。	請遵循 IAM 使用者指南 中的將臨時憑證與 AWS 資源搭配使用中的指示。
IAM	(不建議使用) 使用長期憑證簽署對 AWS CLI AWS SDKs、 或 的程式設計請求 AWS APIs。	請依照您要使用的介面所提供的指示操作。 • 對於 AWS CLI，請參閱 AWS Command Line Interface 使用者指南 中的使用IAM使用者憑證進行驗證。 • 如需 AWS SDKs 和 工具，請參閱 AWS SDKs和 工具參考指南 中的使用長期憑證進行身分驗證。 • 對於 AWS APIs，請參閱 IAM 使用者指南 中的管理IAM使用者的存取金鑰。

使用主控台

您可以使用 Timestream Live Analytics 的 AWS 管理主控台來建立、編輯、刪除、描述和列出資料庫和資料表。您也可以使用 主控台執行查詢。

主題

教學課程

本教學課程示範如何建立填入範例資料集的資料庫，並執行範例查詢。本教學課程中使用的範例資料集經常出現在 IoT 和 DevOps 情境中。IoT 資料集包含時間序列資料，例如卡車的速度、位置和負載，以簡化機群管理並識別最佳化機會。 DevOps 資料集包含EC2執行個體指標，例如 CPU、網路和記憶體使用率，以改善應用程式效能和可用性。以下是本節中說明的影片教學課程

請依照下列步驟建立填入範例資料集的資料庫，並使用 AWS 主控台執行範例查詢。

1. 開啟AWS 主控台 。
2. 在導覽窗格中，選擇資料庫
3. 按一下建立資料庫 。
4. 在建立資料庫頁面上，輸入下列內容：
    - 選擇組態：選取範例資料庫 。
    - 名稱 — 輸入您選擇的資料庫名稱。
    - 選擇範例資料集：選取 IoT 和 DevOps。
    - 按一下建立資料庫，以建立包含兩個資料表的資料庫：IoT 和 DevOps 填入的範例資料。

5. 在導覽窗格中，選擇查詢編輯器
6. 從頂端功能表中選取範例查詢。
7. 按一下其中一個範例查詢。這將帶您回到查詢編輯器，其中的編輯器會填入範例查詢。
8. 按一下執行以執行查詢，並查看查詢結果。

建立 資料庫

請依照下列步驟，使用 AWS 主控台建立資料庫。

1. 開啟AWS 主控台 。
2. 在導覽窗格中，選擇資料庫
3. 按一下建立資料庫 。
4. 在建立資料庫頁面上，輸入以下內容。
   - 選擇組態 — 選擇標準資料庫 。
   - 名稱 — 輸入您選擇的資料庫名稱。
   - 加密：選擇KMS金鑰或使用預設選項，如果尚未存在金鑰，Timestream Live Analytics 會在您的帳戶中建立KMS金鑰。
5. 按一下建立資料庫以建立資料庫。

建立資料表

請依照下列步驟，使用 AWS 主控台建立資料表。

1. 開啟AWS 主控台 。
2. 在導覽窗格中，選擇資料表
3. 按一下建立資料表 。
4. 在建立資料表頁面上，輸入以下內容。
   - 資料庫名稱 — 選取在 中建立的資料庫名稱建立 資料庫。
   - 資料表名稱 — 輸入您選擇的資料表名稱。
   - 記憶體存放區保留 — 指定您希望在記憶體存放區中保留資料的時間長度。記憶體存放區會處理傳入的資料，包括延遲到達的資料 （時間戳記早於目前時間的資料），並針對快速 point-in-time查詢進行最佳化。

- 磁性存放區保留 — 指定您要在磁性存放區中保留資料的時間長度。磁性存放區適用於長期儲存，並針對快速分析查詢進行最佳化。

5. 按一下建立資料表 。

執行查詢

請依照下列步驟，使用 AWS 主控台執行查詢。

1. 開啟AWS 主控台 。
2. 在導覽窗格中，選擇查詢編輯器
3. 在左窗格中，選取在 中建立的資料庫建立 資料庫。
4. 在左窗格中，選取在 中建立的資料庫建立資料表。
5. 在查詢編輯器中，您可以執行查詢。若要查看資料表中最新的 10 列，請執行：

```
SELECT * FROM <database_name>.<table_name> ORDER BY time DESC LIMIT 10
```

6. （選用）開啟 Enable Insights 以取得查詢效率的洞見。

建立排程查詢

請依照下列步驟，使用 AWS 主控台建立排程查詢。

1. 開啟AWS 主控台 。
2. 在導覽窗格中，選擇排程查詢 。
3. 按一下建立排程查詢 。
4. 在查詢名稱和目的地資料表區段中，輸入以下內容。

   - 名稱 — 輸入查詢名稱。
   - 資料庫名稱 — 選取在 中建立的資料庫名稱建立 資料庫。
   - 資料表名稱 — 選取在 中建立的資料表名稱建立資料表。

5. 在查詢陳述式區段中，輸入有效的查詢陳述式。然後按一下驗證查詢 。
6. 從目的地資料表模型 中，定義任何未定義屬性的模型。您可以使用視覺化建置器或 JSON。
7. 在執行排程區段中，選擇固定速率或 Chron 表達式 。如需時間表達式，請參閱排程查詢的排程表達式，以取得排程表達式的詳細資訊。
8. 在SNS主題區段中，輸入將用於通知SNS的主題。

9. 在錯誤日誌報告區段中，輸入將用於報告錯誤的 S3 位置。

   選擇 Encryption key type (加密金鑰類型)。

10. 在AWS KMS金鑰 的安全設定區段中，選擇金鑰類型 AWS KMS。

    輸入 Timestream LiveAnalytics 用來執行排程查詢IAM的角色。如需角色所需許可和信任關係的詳細資訊，請參閱IAM排程查詢的政策範例。

11. 按一下建立排程查詢 。

刪除排程查詢

請依照下列步驟，使用 AWS 主控台刪除或停用排程查詢。

1. 開啟AWS 主控台 。
2. 在導覽窗格中，選擇排程查詢
3. 選取在 中建立的排程查詢建立排程查詢。
4. 選取動作。
5. 選擇停用或刪除 。
6. 如果您選取刪除，請確認動作，然後選取刪除 。

刪除資料表

請依照下列步驟，使用 AWS 主控台刪除資料庫。

1. 開啟AWS 主控台 。
2. 在導覽窗格中，選擇資料表
3. 選取您在 中建立的資料表建立資料表。
4. 按一下 Delete (刪除)。
5. 在確認方塊中輸入刪除。

刪除資料庫

請依照下列步驟，使用 AWS 主控台刪除資料庫：

1. 開啟AWS 主控台 。
2. 在導覽窗格中，選擇資料庫

3. 選取您在建立資料庫 中建立的資料庫。
4. 按一下 Delete (刪除)。
5. 在確認方塊中輸入刪除。

編輯資料表

請依照下列步驟，使用 AWS 主控台編輯資料表。

1. 開啟AWS 主控台 。
2. 在導覽窗格中，選擇資料表
3. 選取您在 中建立的資料表建立資料表。
4. 按一下編輯
5. 編輯資料表詳細資訊並儲存。

   - 記憶體存放區保留 — 指定您希望在記憶體存放區中保留資料的時間長度。記憶體存放區會處理傳入的資料，包括延遲到達的資料 （ 時間戳記早於目前時間的資料 ），並針對快速 point-in-time查詢進行最佳化。
   - 磁性存放區保留 — 指定您希望在磁性存放區中保留資料的時間長度。磁性存放區適用於長期儲存，並針對快速分析查詢進行最佳化。

編輯資料庫

請依照下列步驟，使用 AWS 主控台編輯資料庫。

1. 開啟AWS 主控台 。
2. 在導覽窗格中，選擇資料庫
3. 選取您在建立資料庫 中建立的資料庫。
4. 按一下編輯
5. 編輯資料庫詳細資訊並儲存。

LiveAnalytics 使用 存取 Amazon Timestream AWS CLI

您可以使用 AWS Command Line Interface （ AWS CLI ） 從命令列控制多個 AWS 服務，並透過指令碼自動化。您可以使用 AWS CLI 進行臨時操作。您也可以使用它在公用程式指令碼中嵌入 Amazon Timestream 進行 LiveAnalytics 操作。

您必須先設定程式設計存取 LiveAnalytics，才能將 AWS CLI 與 Timestream 搭配使用。如需詳細資訊，請參閱授與程式設計存取權。

如需 LiveAnalytics API查詢時間串流可用的所有命令的完整清單 AWS CLI，請參閱 AWS CLI 命令參考 。

如需 API中可用於 LiveAnalytics 寫入時間串流的所有命令的完整清單 AWS CLI，請參閱AWS CLI 命令參考 。

主題

- 下載和設定 AWS CLI
- 將 AWS CLI 與 Timestream 搭配使用 LiveAnalytics

下載和設定 AWS CLI

AWS CLI 會在 Windows、macOS 或 Linux 上執行。若要下載、安裝和設定它，請遵循下列步驟：

1. 在 AWS CLI https：//http://aws.amazon.com/cli 下載 。
2. 請遵循 AWS Command Line Interface 使用者指南 中的安裝 AWS CLI 和設定 的指示。 AWS CLI

將 AWS CLI 與 Timestream 搭配使用 LiveAnalytics

命令列格式包含 LiveAnalytics 操作名稱的 Amazon Timestream，後面接著該操作的參數。除了 之外， AWS CLI 還支援參數值的短期語法JSON。

使用 `help` 列出 Timestream 中 的所有可用命令 LiveAnalytics。例如：

```
aws timestream-write help
```

```
aws timestream-query help
```

您也可以使用 `help` 描述特定命令，並進一步了解其用量：

```
aws timestream-write create-database help
```

例如，若要建立資料庫：

```
aws timestream-write create-database --database-name myFirstDatabase
```

若要建立已啟用磁性存放區寫入的資料表：

```
aws timestream-write create-table \
--database-name metricsdb \
--table-name metrics \
--magnetic-store-write-properties "{\"EnableMagneticStoreWrites\": true}"
```

若要使用單一測量記錄寫入資料：

```
aws timestream-write write-records \
--database-name metricsdb \
--table-name metrics \
--common-attributes "{\"Dimensions\":[{\"Name\":\"asset_id\", \"Value\":\"100\"}],
 \"Time\":\"1631051324000\",\"TimeUnit\":\"MILLISECONDS\"}" \
--records "[{\"MeasureName\":\"temperature\", \"MeasureValueType\":\"DOUBLE\",
\"MeasureValue\":\"30\"},{\"MeasureName\":\"windspeed\", \"MeasureValueType\":\"DOUBLE
\",\"MeasureValue\":\"7\"},{\"MeasureName\":\"humidity\", \"MeasureValueType\":\"DOUBLE
\",\"MeasureValue\":\"15\"},{\"MeasureName\":\"brightness\", \"MeasureValueType\":
\"DOUBLE\",\"MeasureValue\":\"17\"}]"
```

若要使用多測量記錄寫入資料：

```
# wide model helper method to create Multi-measure records
function ingest_multi_measure_records {
  epoch=`date +%s`
  epoch+=$i

  # multi-measure records
  aws timestream-write write-records \
  --database-name $src_db_wide \
  --table-name $src_tbl_wide \
  --common-attributes "{\"Dimensions\":[{\"Name\":\"device_id\", \
              \"Value\":\"12345678\"},\
            {\"Name\":\"device_type\", \"Value\":\"iPhone\"}, \
            {\"Name\":\"os_version\", \"Value\":\"14.8\"}, \
            {\"Name\":\"region\", \"Value\":\"us-east-1\"} ], \
            \"Time\":\"$epoch\",\"TimeUnit\":\"MILLISECONDS\"}" \
--records "[{\"MeasureName\":\"video_metrics\", \"MeasureValueType\":\"MULTI\", \
  \"MeasureValues\": \
  [{\"Name\":\"video_startup_time\",\"Value\":\"0\",\"Type\":\"BIGINT\"}, \
  {\"Name\":\"rebuffering_ratio\",\"Value\":\"0.5\",\"Type\":\"DOUBLE\"}, \
  {\"Name\":\"video_playback_failures\",\"Value\":\"0\",\"Type\":\"BIGINT\"}, \
```

```
  {\"Name\":\"average_frame_rate\",\"Value\":\"0.5\",\"Type\":\"DOUBLE\"}]}]" \
--endpoint-url $ingest_endpoint \
  --region  $region
}

# create 5 records
for i in {100..105};
  do ingest_multi_measure_records $i;
done
```

查詢資料表：

```
aws timestream-query query \
--query-string "SELECT time, device_id, device_type, os_version,
region, video_startup_time, rebuffering_ratio, video_playback_failures, \
average_frame_rate \
FROM metricsdb.metrics \
where time >= ago (15m)"
```

若要建立排程查詢：

```
aws timestream-query create-scheduled-query \
  --name scheduled_query_name \
  --query-string "select bin(time, 1m) as time, \
          avg(measure_value::double) as avg_cpu, min(measure_value::double) as min_cpu,
 region \
          from $src_db.$src_tbl where measure_name = 'cpu' \
          and time BETWEEN @scheduled_runtime - (interval '5' minute)  AND
 @scheduled_runtime \
          group by region, bin(time, 1m)" \
  --schedule-configuration "{\"ScheduleExpression\":\"$cron_exp\"}" \
  --notification-configuration "{\"SnsConfiguration\":{\"TopicArn\":\"$sns_topic_arn
\"}}" \
  --scheduled-query-execution-role-arn "arn:aws:iam::452360119086:role/
TimestreamSQExecutionRole" \
  --target-configuration "{\"TimestreamConfiguration\":{\
          \"DatabaseName\": \"$dest_db\",\
          \"TableName\": \"$dest_tbl\",\
          \"TimeColumn\":\"time\",\
          \"DimensionMappings\":[{\
            \"Name\": \"region\", \"DimensionValueType\": \"VARCHAR\"
          }],\
          \"MultiMeasureMappings\":{\
```

```
            \"TargetMultiMeasureName\": \"mma_name\",
            \"MultiMeasureAttributeMappings\":[{\
              \"SourceColumn\": \"avg_cpu\", \"MeasureValueType\": \"DOUBLE\",
\"TargetMultiMeasureAttributeName\": \"target_avg_cpu\"
            },\
            { \
              \"SourceColumn\": \"min_cpu\", \"MeasureValueType\": \"DOUBLE\",
\"TargetMultiMeasureAttributeName\": \"target_min_cpu\"
            }] \
          }\
          }}" \
 --error-report-configuration "{\"S3Configuration\": {\
       \"BucketName\": \"$s3_err_bucket\",\
       \"ObjectKeyPrefix\": \"scherrors\",\
       \"EncryptionOption\": \"SSE_S3\"\
       }\
     }"
```

使用 API

除了 之外SDKs， 的 Amazon Timestream 透過端點探索模式 LiveAnalytics 提供直接RESTAPI存取。端點探索模式及其使用案例如下所述。

端點探索模式

由於 Timestream Live Analytics SDKs的設計旨在透明地使用服務架構，包括服務端點的管理和映射，因此建議您將 SDKs用於大多數應用程式。不過，有些執行個體需要使用 Timestream 進行 LiveAnalytics RESTAPI端點探索模式：

- 您正在將VPC端點（AWS PrivateLink）與 Timestream 搭配使用 LiveAnalytics
- 您的應用程式使用尚未SDK支援的程式設計語言
- 您需要更好地控制用戶端實作

本節包含端點探索模式如何運作、如何實作端點探索模式以及用量備註的相關資訊。選取以下主題以進一步了解。

主題

- 端點探索模式的運作方式
- 實作端點探索模式

端點探索模式的運作方式

Timestream 使用行動架構建置，以確保更好的擴展和流量隔離屬性。由於每個客戶帳戶都對應至區域中的特定儲存格，因此您的應用程式必須使用帳戶已對應至的正確儲存格特定端點。使用 時SDKs，系統會透明地為您處理此映射，您不需要管理儲存格特定的端點。不過，當直接存取 REST 時API，您需要自行管理和對應正確的端點。此程序是端點探索模式 ，如下所述：

1. 端點探索模式從呼叫DescribeEndpoints動作開始（如 DescribeEndpoints 一節中所述）。
2. 端點應快取和重複使用，以傳回 time-to-live （TTL） 值 （ ） 指定的時間長度CachePeriodInMinutes。API 然後，可以在 的持續時間內呼叫 Timestream Live AnalyticsTTL。
3. TTL 過期後， DescribeEndpoints 應該對 進行新的呼叫，以重新整理端點 （換言之，從步驟 1 重新開始）。

> **Note**
>
> 該DescribeEndpoints動作的語法、參數和其他用量資訊如API參考 所述。請注意，DescribeEndpoints動作可透過 使用SDKs，且每個 的動作都相同。

如需端點探索模式的實作，請參閱 實作端點探索模式。

實作端點探索模式

若要實作端點探索模式，請選擇 API（寫入或查詢）、建立DescribeEndpoints請求，並在傳回TTL值（s）的持續時間內使用傳回的端點。實作程序說明如下。

> **Note**
>
> 確保您熟悉 用量備註 。

實作程序

1. 使用 DescribeEndpoints請求，取得API您要對 進行呼叫的 端點 （寫入或查詢 ）。

a. 使用下列兩個端點之一，為 建立DescribeEndpoints對應至目標 API（寫入或查詢 ）的請求。請求沒有輸入參數。請務必閱讀下列備註。

寫入 SDK：

```
ingest.timestream.<region>.amazonaws.com
```

查詢SDK：

```
query.timestream.<region>.amazonaws.com
```

區域的範例CLI呼叫us-east-1如下。

```
REGION_ENDPOINT="https://query.timestream.us-east-1.amazonaws.com"
REGION=us-east-1
aws timestream-write describe-endpoints \
--endpoint-url $REGION_ENDPOINT \
--region $REGION
```

> **Note**
>
> HTTP 「主機」標頭也必須包含API端點。如果未填入標頭，則請求會失敗。這是所有 HTTP/1.1 請求的標準要求。如果您使用支援 1.1 或更新版本的程式HTTP庫，程式HTTP庫應該會自動為您填入標頭。

> **Note**
>
> 替代 *<region>* 具有正在發出請求之區域的區域識別符，例如 us-east-1

b. 剖析擷取端點（s）和快取TTL值（s）的回應。回應是一或多個Endpoint物件的陣列。每個Endpoint物件都包含端點地址（Address）和該端點（ ）TTL的CachePeriodInMinutes。

2. 快取端點，最多可達指定的 TTL。
3. TTL 過期時，從實作的步驟 1 重新開始擷取新的端點。

端點探索模式的使用備註

- DescribeEndpoints 動作是 Timestream Live Analytics 區域端點唯一能辨識的動作。
- 回應包含端點清單，用於進行 Timestream Live Analytics API呼叫。
- 成功回應時，清單中至少應該有一個端點。如果清單中有一個以上的端點，則其中任何一個端點都同樣可用於API呼叫，而且呼叫者可以選擇隨機使用的端點。
- 除了端點DNS的地址之外，清單中的每個端點都會指定一個存留時間（TTL），允許使用以分鐘為單位指定的端點。
- 端點應快取和重複使用，以傳回TTL值指定的時間長度（以分鐘為單位）。TTL 過期後，DescribeEndpoints應該對 進行新的呼叫，以重新整理要使用的端點，因為端點在TTL過期後將無法再運作。

使用 AWS SDKs

您可以使用 存取 Amazon Timestream AWS SDKs。Timestream 支援SDKs每種語言兩種：即寫入SDK和查詢SDK。寫入SDK用於執行CRUD操作，並將時間序列資料插入 Timestream。查詢SDK用於查詢儲存在 Timestream 中的現有時間序列資料。

完成SDK您選擇的必要先決條件後，您就可以開始使用 程式碼範例。

主題

- Java
- Java v2
- Go
- Python
- Node.js
- .NET

Java

若要開始使用 Java 1.0 SDK 和 Amazon Timestream，請完成先決條件，如下所述。

完成 Java 的必要先決條件後SDK，您就可以開始使用 程式碼範例。

必要條件

開始使用 Java 之前，您必須執行下列動作：

1. 請遵循 中的 AWS 設定指示存取 的 Timestream LiveAnalytics。
2. 下載並安裝下列項目，以設定 Java 開發環境：
   - Java SE 開發套件 8（例如 Amazon Corretto 8）。
   - Java IDE（例如 Eclipse IntelliJ）。

   如需詳細資訊，請參閱 入門 AWS SDK for Java
3. 設定您的 AWS 憑證和開發區域：
   - 設定您的 AWS 安全憑證，以便與 搭配使用 AWS SDK for Java。
   - 設定您的 AWS 區域以決定 LiveAnalytics 端點的預設 Timestream。

使用 Apache Maven

您可以使用 Apache Maven 來設定和建置 AWS SDK for Java 專案。

> **Note**
>
> 若要使用 Apache Maven，請確定您的 Java SDK和執行期為 1.8 或更高版本。

您可以如搭配 Apache Maven 使用 所述，將 設定為 AWS SDK Maven 相依性。 SDK

您可以使用下列命令執行編譯和執行原始程式碼：

```
mvn clean compile
mvn exec:java -Dexec.mainClass=<your source code Main class>
```

> **Note**
>
> `<your source code Main class>` 是 Java 原始碼主類別的路徑。

設定您的 AWS 憑證

AWS SDK for Java 需要您在執行階段提供 AWS 憑證給應用程式。本指南中的程式碼範例假設您正在使用 AWS 憑證檔案，如 AWS SDK for Java 開發人員指南 中的設定 AWS 憑證和開發區域中所述。

以下是名為 的 AWS 登入資料檔案範例~/.aws/credentials，其中的波浪字元（~）代表您的主目錄。

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Java v2

若要開始使用 Java 2.0 SDK 和 Amazon Timestream，請完成先決條件，如下所述。

完成 Java 2.0 的必要先決條件後SDK，您就可以開始使用 程式碼範例。

必要條件

開始使用 Java 之前，您必須執行下列動作：

1. 請遵循 中的 AWS 設定指示存取 的 Timestream LiveAnalytics。
2. 您可以如搭配 Apache Maven 使用 所述，將 設定為 AWS SDK Maven 相依性。 SDK
3. 下載並安裝下列項目，以設定 Java 開發環境：

   - Java SE 開發套件 8（例如 Amazon Corretto 8）。
   - Java IDE（例如 Eclipse IntelliJ）。

     如需詳細資訊，請參閱 入門 AWS SDK for Java

使用 Apache Maven

您可以使用 Apache Maven 來設定和建置 AWS SDK for Java 專案。

> **Note**
>
> 若要使用 Apache Maven，請確定您的 Java SDK和執行期為 1.8 或更高版本。

您可以如搭配 Apache Maven 使用 所述，將 設定為 AWS SDK Maven 相依性。 SDK 此處說明 pom.xml 檔案所需的變更。

您可以使用下列命令執行編譯和執行原始程式碼：

```
mvn clean compile
```

```
mvn exec:java -Dexec.mainClass=<your source code Main class>
```

> **Note**
>
> `<your source code Main class>` 是 Java 原始碼主類別的路徑。

Go

若要開始使用 Go SDK 和 Amazon Timestream，請完成先決條件，如下所述。

完成 Go 的必要先決條件之後SDK，您就可以開始使用 程式碼範例。

必要條件

1. 下載 GO SDK 1.14。
2. 設定 GO SDK。
3. 建構您的用戶端 。

Python

若要開始使用 Python SDK 和 Amazon Timestream，請完成先決條件，如下所述。

完成 Python 的必要先決條件之後SDK，您就可以開始使用 程式碼範例。

必要條件

若要使用 Python，請依照此處的指示安裝和設定 Boto3。

Node.js

若要開始使用 Node.js SDK 和 Amazon Timestream，請完成先決條件，如下所述。

完成 Node.js 的必要先決條件後SDK，您就可以開始使用 程式碼範例。

必要條件

開始使用 Node.js 之前，您必須執行下列動作：

1. 安裝 Node.js 。
2. 為 AWS SDK 安裝 JavaScript。

.NET

若要開始使用 .NET SDK 和 Amazon Timestream，請完成先決條件，如下所述。

完成 NET的必要先決條件後SDK，您就可以開始使用 程式碼範例。

必要條件

開始使用 之前NET，請先安裝所需的 NuGet 套件，並執行下列命令，以確保 AWSSDK.Core 版本為 3.3.107 或更新版本：

```
dotnet add package AWSSDK.Core
dotnet add package AWSSDK.TimestreamWrite
dotnet add package AWSSDK.TimestreamQuery
```

開始使用

本節包含入門 Amazon Timestream Live Analytics 的教學課程，以及設定全功能範例應用程式的指示。您可以選擇下列其中一個連結，以開始使用教學課程或範例應用程式。

主題

- 教學課程
- 範例應用程式

教學課程

本教學課程示範如何建立填入範例資料集的資料庫，並執行範例查詢。本教學課程中使用的範例資料集經常出現在 IoT 和 DevOps 案例中。IoT 資料集包含時間序列資料，例如卡車的速度、位置和負載，以簡化機群管理並識別最佳化機會。 DevOps 資料集包含EC2執行個體指標，例如 CPU、網路和記憶體使用率，以改善應用程式效能和可用性。以下是本節中說明的影片教學課程。

請依照下列步驟建立填入範例資料集的資料庫，並使用 AWS 主控台執行範例查詢：

使用主控台

請依照下列步驟建立填入範例資料集的資料庫，並使用 AWS 主控台執行範例查詢：

1. 開啟AWS 主控台 。
2. 在導覽窗格中，選擇資料庫

3. 按一下建立資料庫 。
4. 在建立資料庫頁面上，輸入下列內容：
   - 選擇組態 — 選擇範例資料庫 。
   - 名稱 — 輸入您選擇的資料庫名稱。

     > **Note**
     >
     > 使用範例資料集建立資料庫後，若要使用主控台中可用的範例查詢，您可以調整查詢中參考的資料庫名稱，以符合您在此處輸入的資料庫名稱。每個範例資料集和時間序列記錄類型組合都有範例查詢。

   - 選擇範例資料集：選取 IoT 和 DevOps。
   - 選擇時間序列記錄的類型 — 選擇多量值記錄 。
   - 按一下建立資料庫，以建立包含填入範例資料的兩個資料表的資料庫。具有多測量記錄的範例資料集的資料表名稱為 `DevOpsMulti`和 `IoTMulti`。具有單一測量記錄的範例資料集的資料表名稱為 `DevOps`和 `IoT`。
5. 在導覽窗格中，選擇查詢編輯器
6. 從頂端功能表中選取範例查詢。
7. 在建立範例資料庫時，按一下所選資料集的其中一個範例查詢。這將帶您回到查詢編輯器，並將編輯器填入範例查詢。
8. 調整範例查詢的資料庫名稱。
9. 按一下執行以執行查詢，並查看查詢結果。

使用 SDKs

Timestream Live Analytics 提供全功能範例應用程式，示範如何建立資料庫和資料表、將 ~126K 列的範例資料填入資料表，以及執行範例查詢。範例應用程式適用於 GitHub Java、Python、Node.js、Go 和 NET。

1. 按照 的指示複製 GitHub 儲存庫 Timestream Live Analytics 範例應用程式 GitHub。
2. 依照 中所述的指示，設定 AWS SDK以連線至 使用 AWS SDKs Amazon Timestream Live Analytics。
3. 使用下列指示編譯和執行範例應用程式：
   - Java 範例應用程式 的指示。

- Java v2 範例應用程式 的指示。
- Go 範例應用程式 的指示。
- Python 範例應用程式 的指示。
- Node.js 範例應用程式 的說明。
- .NET 範例應用程式 的指示。

範例應用程式

Timestream 隨附全功能範例應用程式，示範如何建立資料庫和資料表、在資料表中填入約 126K 列的範例資料，以及執行範例查詢。請依照下列步驟，以任何支援的語言開始使用範例應用程式：

Java

1. 按照 的指示複製範例應用程式的 GitHub 儲存庫 TimestreamGitHub。 LiveAnalytics
2. 設定 AWS SDK 以連線至 Timestream，以 LiveAnalytics 遵循入門中所述的指示Java。
3. 按照此處說明執行 Java 範例應用程式

Java v2

1. 按照 的指示複製範例應用程式的 GitHub 儲存庫 TimestreamGitHub。 LiveAnalytics
2. 將 AWS SDK設定為連線至 Amazon Timestream，以 LiveAnalytics 遵循入門 中所述的指示Java v2。
3. 按照此處說明執行 Java 2.0 範例應用程式

Go

1. 按照 的指示複製範例應用程式的 GitHub 儲存庫 TimestreamGitHub。 LiveAnalytics
2. 設定 AWS SDK以連線至 Amazon Timestream，以 LiveAnalytics 遵循入門 中所述的指示Go。
3. 按照此處說明執行 Go 範例應用程式

Python

1. 按照 的指示複製範例應用程式的 GitHub 儲存庫 TimestreamGitHub。 LiveAnalytics
2. 設定 AWS SDK以連線至 Amazon Timestream，以 LiveAnalytics 遵循中所述的指示Python。

3. 按照此處說明執行 Python 範例應用程式

Node.js

1. 按照 的指示複製範例應用程式的 GitHub 儲存庫 TimestreamGitHub。 LiveAnalytics
2. 將 AWS SDK設定為連線至 Amazon Timestream，以 LiveAnalytics 遵循入門 中所述的指示Node.js。
3. 按照此處說明執行 Node.js 範例應用程式

.NET

1. 按照 的指示複製範例應用程式的 GitHub 儲存庫 TimestreamGitHub。 LiveAnalytics
2. 設定 AWS SDK以連線至 Amazon Timestream，以 LiveAnalytics 遵循入門 中所述的指示.NET。
3. 按照此處說明執行 .NET 範例應用程式

程式碼範例

您可以使用 存取 Amazon Timestream AWS SDKs。Timestream 支援SDKs每種語言兩種：即寫入SDK和查詢SDK。寫入SDK用於執行CRUD操作，並將時間序列資料插入 Timestream。查詢SDK用於查詢儲存在 Timestream 中的現有時間序列資料。從下列清單中選擇主題以取得更多詳細資訊，包括每個支援 的程式碼範例SDKs。

主題

- 寫入SDK用戶端
- 查詢SDK用戶端
- 建立資料庫
- 描述資料庫
- 更新資料庫
- 刪除資料庫
- 列出資料庫
- 建立資料表
- 描述資料表
- 更新資料表

- 刪除資料表
- 列出資料表
- 寫入資料（插入和 upserts）
- 執行查詢
- 執行UNLOAD查詢
- 取消查詢
- 建立批次載入任務
- 描述批次載入任務
- 列出批次載入任務
- 恢復批次載入任務
- 建立排程查詢
- 列出排程查詢
- 描述排程查詢
- 執行排程查詢
- 更新排程查詢
- 刪除排程查詢

寫入SDK用戶端

您可以使用下列程式碼片段為寫入 建立 Timestream 用戶端SDK。寫入SDK用於執行CRUD操作，並將時間序列資料插入 Timestream。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
    private static AmazonTimestreamWrite buildWriteClient() {
        final ClientConfiguration clientConfiguration = new ClientConfiguration()
                .withMaxConnections(5000)
                .withRequestTimeout(20 * 1000)
                .withMaxErrorRetry(10);
```

```
        return AmazonTimestreamWriteClientBuilder
                .standard()
                .withRegion("us-east-1")
                .withClientConfiguration(clientConfiguration)
                .build();
    }
```

Java v2

```
    private static TimestreamWriteClient buildWriteClient() {
        ApacheHttpClient.Builder httpClientBuilder =
                ApacheHttpClient.builder();
        httpClientBuilder.maxConnections(5000);

        RetryPolicy.Builder retryPolicy =
                RetryPolicy.builder();
        retryPolicy.numRetries(10);

        ClientOverrideConfiguration.Builder overrideConfig =
                ClientOverrideConfiguration.builder();
        overrideConfig.apiCallAttemptTimeout(Duration.ofSeconds(20));
        overrideConfig.retryPolicy(retryPolicy.build());

        return TimestreamWriteClient.builder()
                .httpClientBuilder(httpClientBuilder)
                .overrideConfiguration(overrideConfig.build())
                .region(Region.US_EAST_1)
                .build();
    }
```

Go

```
tr := &http.Transport{
        ResponseHeaderTimeout: 20 * time.Second,
        // Using DefaultTransport values for other parameters: https://golang.org/
pkg/net/http/#RoundTripper
        Proxy: http.ProxyFromEnvironment,
        DialContext: (&net.Dialer{
            KeepAlive: 30 * time.Second,
            DualStack: true,
            Timeout:   30 * time.Second,
        }).DialContext,
```

```
        MaxIdleConns:          100,
        IdleConnTimeout:       90 * time.Second,
        TLSHandshakeTimeout:   10 * time.Second,
        ExpectContinueTimeout: 1 * time.Second,
    }

    // So client makes HTTP/2 requests
    http2.ConfigureTransport(tr)

    sess, err := session.NewSession(&aws.Config{ Region: aws.String("us-east-1"),
 MaxRetries: aws.Int(10), HTTPClient: &http.Client{ Transport: tr }})
    writeSvc := timestreamwrite.New(sess)
```

Python

```
write_client = session.client('timestream-write', config=Config(read_timeout=20,
 max_pool_connections = 5000, retries={'max_attempts': 10}))
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

此處顯示額外的命令匯入。建立用戶端不需要CreateDatabaseCommand匯入。

```
import { TimestreamWriteClient, CreateDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });
```

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
var https = require('https');
var agent = new https.Agent({
    maxSockets: 5000
});
writeClient = new AWS.TimestreamWrite({
        maxRetries: 10,
        httpOptions: {
            timeout: 20000,
            agent: agent
        }
```

```
    });
```

.NET

```
var writeClientConfig = new AmazonTimestreamWriteConfig
{
    RegionEndpoint = RegionEndpoint.USEast1,
    Timeout = TimeSpan.FromSeconds(20),
    MaxErrorRetry = 10
};

var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
```

我們建議您使用下列組態。

- 將SDK重試計數設定為 10。
- 請使用 SDK DEFAULT_BACKOFF_STRATEGY。
- RequestTimeout 設定為 20 秒。
- 將最大連線設定為 5000 或更高。

查詢SDK用戶端

您可以使用下列程式碼片段來建立查詢 的 Timestream 用戶端SDK。查詢SDK用於查詢儲存在 Timestream 中的現有時間序列資料。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
    private static AmazonTimestreamQuery buildQueryClient() {
        AmazonTimestreamQuery client =
AmazonTimestreamQueryClient.builder().withRegion("us-east-1").build();
        return client;
    }
```

Java v2

```
    private static TimestreamQueryClient buildQueryClient() {
        return TimestreamQueryClient.builder()
                .region(Region.US_EAST_1)
                .build();
    }
```

Go

```
sess, err := session.NewSession(&aws.Config{Region: aws.String("us-east-1")})
```

Python

```
query_client = session.client('timestream-query')
```

Node.js

下列程式碼片段用於 AWS SDK JavaScript v3。如需有關如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Query Client - ，AWS SDK for JavaScript v3。

此處顯示額外的命令匯入。建立用戶端不需要QueryCommand匯入。

```
import { TimestreamQueryClient, QueryCommand } from "@aws-sdk/client-timestream-
query";
const queryClient = new TimestreamQueryClient({ region: "us-east-1" });
```

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。其以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
queryClient = new AWS.TimestreamQuery();
```

.NET

```
var queryClientConfig = new AmazonTimestreamQueryConfig
{
    RegionEndpoint = RegionEndpoint.USEast1
};

var queryClient = new AmazonTimestreamQueryClient(queryClientConfig);
```

建立資料庫

您可以使用下列程式碼片段來建立資料庫。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
  public void createDatabase() {
       System.out.println("Creating database");
       CreateDatabaseRequest request = new CreateDatabaseRequest();
       request.setDatabaseName(DATABASE_NAME);
       try {
           amazonTimestreamWrite.createDatabase(request);
           System.out.println("Database [" + DATABASE_NAME + "] created
successfully");
       } catch (ConflictException e) {
           System.out.println("Database [" + DATABASE_NAME + "] exists. Skipping
database creation");
       }
   }
```

Java v2

```
   public void createDatabase() {
       System.out.println("Creating database");
       CreateDatabaseRequest request =
CreateDatabaseRequest.builder().databaseName(DATABASE_NAME).build();
       try {
           timestreamWriteClient.createDatabase(request);
           System.out.println("Database [" + DATABASE_NAME + "] created
successfully");
       } catch (ConflictException e) {
           System.out.println("Database [" + DATABASE_NAME + "] exists. Skipping
database creation");
       }
   }
```

Go

```
// Create database.
    createDatabaseInput := &timestreamwrite.CreateDatabaseInput{
        DatabaseName: aws.String(*databaseName),
    }

    _, err = writeSvc.CreateDatabase(createDatabaseInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Database successfully created")
    }

    fmt.Println("Describing the database, hit enter to continue")
```

Python

```
    def create_database(self):
        print("Creating Database")
        try:
            self.client.create_database(DatabaseName=Constant.DATABASE_NAME)
            print("Database [%s] created successfully." % Constant.DATABASE_NAME)
        except self.client.exceptions.ConflictException:
            print("Database [%s] exists. Skipping database creation" %
 Constant.DATABASE_NAME)
        except Exception as err:
            print("Create database failed:", err)
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

另請參閱類別 CreateDatabaseCommand 和 CreateDatabase。

```
import { TimestreamWriteClient, CreateDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
```

```
    DatabaseName: "testDbFromNode"
};

const command = new CreateDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Database ${data.Database.DatabaseName} created successfully`);
} catch (error) {
    if (error.code === 'ConflictException') {
        console.log(`Database ${params.DatabaseName} already exists. Skipping
 creation.`);
    } else {
        console.log("Error creating database", error);
    }
}
```

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function createDatabase() {
    console.log("Creating Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME
    };

    const promise = writeClient.createDatabase(params).promise();

    await promise.then(
        (data) => {
            console.log(`Database ${data.Database.DatabaseName} created
 successfully`);
        },
        (err) => {
            if (err.code === 'ConflictException') {
                console.log(`Database ${params.DatabaseName} already exists.
 Skipping creation.`);
            } else {
                console.log("Error creating database", err);
            }
        }
    );
}
```

.NET

```
        public async Task CreateDatabase()
        {
            Console.WriteLine("Creating Database");

            try
            {
                var createDatabaseRequest = new CreateDatabaseRequest
                {
                    DatabaseName = Constants.DATABASE_NAME
                };
                CreateDatabaseResponse response = await
writeClient.CreateDatabaseAsync(createDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} created");
            }
            catch (ConflictException)
            {
                Console.WriteLine("Database already exists.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Create database failed:" + e.ToString());
            }

        }
```

描述資料庫

您可以使用下列程式碼片段來取得新建立資料庫屬性的相關資訊。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
    public void describeDatabase() {
        System.out.println("Describing database");
```

```
        final DescribeDatabaseRequest describeDatabaseRequest = new
 DescribeDatabaseRequest();
        describeDatabaseRequest.setDatabaseName(DATABASE_NAME);
        try {
            DescribeDatabaseResult result =
 amazonTimestreamWrite.describeDatabase(describeDatabaseRequest);
            final Database databaseRecord = result.getDatabase();
            final String databaseId = databaseRecord.getArn();
            System.out.println("Database " + DATABASE_NAME + " has id " +
 databaseId);
        } catch (final Exception e) {
            System.out.println("Database doesn't exist = " + e);
            throw e;
        }
    }
```

Java v2

```
    public void describeDatabase() {
        System.out.println("Describing database");
        final DescribeDatabaseRequest describeDatabaseRequest =
 DescribeDatabaseRequest.builder()
                .databaseName(DATABASE_NAME).build();
        try {
            DescribeDatabaseResponse response =
 timestreamWriteClient.describeDatabase(describeDatabaseRequest);
            final Database databaseRecord = response.database();
            final String databaseId = databaseRecord.arn();
            System.out.println("Database " + DATABASE_NAME + " has id " +
 databaseId);
        } catch (final Exception e) {
            System.out.println("Database doesn't exist = " + e);
            throw e;
        }
    }
```

Go

```
describeDatabaseOutput, err := writeSvc.DescribeDatabase(describeDatabaseInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
```

```
    } else {
        fmt.Println("Describe database is successful, below is the output:")
        fmt.Println(describeDatabaseOutput)
    }
```

Python

```
    def describe_database(self):
        print("Describing database")
        try:
            result =
 self.client.describe_database(DatabaseName=Constant.DATABASE_NAME)
            print("Database [%s] has id [%s]" % (Constant.DATABASE_NAME,
 result['Database']['Arn']))
        except self.client.exceptions.ResourceNotFoundException:
            print("Database doesn't exist")
        except Exception as err:
            print("Describe database failed:", err)
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

另請參閱類別 DescribeDatabaseCommand 和 DescribeDatabase。

```
import { TimestreamWriteClient, DescribeDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode"
};

const command = new DescribeDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Database ${data.Database.DatabaseName} has id
 ${data.Database.Arn}`);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Database doesn't exist.");
```

```
    } else {
        console.log("Describe database failed.", error);
        throw error;
    }
}
```

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function describeDatabase () {
    console.log("Describing Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME
    };

    const promise = writeClient.describeDatabase(params).promise();

    await promise.then(
        (data) => {
            console.log(`Database ${data.Database.DatabaseName} has id
 ${data.Database.Arn}`);
        },
        (err) => {
            if (err.code === 'ResourceNotFoundException') {
                console.log("Database doesn't exist.");
            } else {
                console.log("Describe database failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DescribeDatabase()
        {
            Console.WriteLine("Describing Database");

            try
            {
                var describeDatabaseRequest = new DescribeDatabaseRequest
                {
```

```
                    DatabaseName = Constants.DATABASE_NAME
                };
                DescribeDatabaseResponse response = await
 writeClient.DescribeDatabaseAsync(describeDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} has id:
{response.Database.Arn}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Database does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Describe database failed:" + e.ToString());
            }

        }
```

更新資料庫

您可以使用下列程式碼片段來更新資料庫。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
    public void updateDatabase(String kmsId) {
        System.out.println("Updating kmsId to " + kmsId);
        UpdateDatabaseRequest request = new UpdateDatabaseRequest();
        request.setDatabaseName(DATABASE_NAME);
        request.setKmsKeyId(kmsId);
        try {
            UpdateDatabaseResult result =
 amazonTimestreamWrite.updateDatabase(request);
            System.out.println("Update Database complete");
        } catch (final ValidationException e) {
            System.out.println("Update database failed:");
```

```
            e.printStackTrace();
        } catch (final ResourceNotFoundException e) {
            System.out.println("Database " + DATABASE_NAME + " doesn't exist = " +
e);
        } catch (final Exception e) {
            System.out.println("Could not update Database " + DATABASE_NAME + " = "
+ e);
            throw e;
        }
    }
```

Java v2

```
    public void updateDatabase(String kmsKeyId) {

        if (kmsKeyId == null) {
            System.out.println("Skipping UpdateDatabase because KmsKeyId was not
given");
            return;
        }

        System.out.println("Updating database");

        UpdateDatabaseRequest request = UpdateDatabaseRequest.builder()
                .databaseName(DATABASE_NAME)
                .kmsKeyId(kmsKeyId)
                .build();
        try {
            timestreamWriteClient.updateDatabase(request);
            System.out.println("Database [" + DATABASE_NAME + "] updated
successfully with kmsKeyId " + kmsKeyId);
        } catch (ResourceNotFoundException e) {
            System.out.println("Database [" + DATABASE_NAME + "] does not exist.
Skipping UpdateDatabase");
        } catch (Exception e) {
            System.out.println("UpdateDatabase failed: " + e);
        }
    }
```

Go

```
// Update Database.
        updateDatabaseInput := &timestreamwrite.UpdateDatabaseInput {
```

```
            DatabaseName: aws.String(*databaseName),
            KmsKeyId: aws.String(*kmsKeyId),
        }

        updateDatabaseOutput, err := writeSvc.UpdateDatabase(updateDatabaseInput)

        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
            fmt.Println("Update database is successful, below is the output:")
            fmt.Println(updateDatabaseOutput)
        }
```

Python

```
    def update_database(self, kms_id):
        print("Updating database")
        try:
            result =
self.client.update_database(DatabaseName=Constant.DATABASE_NAME, KmsKeyId=kms_id)
            print("Database [%s] was updated to use kms [%s] successfully" %
(Constant.DATABASE_NAME,

result['Database']['KmsKeyId']))
        except self.client.exceptions.ResourceNotFoundException:
            print("Database doesn't exist")
        except Exception as err:
            print("Update database failed:", err)
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

另請參閱類別 UpdateDatabaseCommand 和 UpdateDatabase。

```
import { TimestreamWriteClient, UpdateDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });
let updatedKmsKeyId = "<updatedKmsKeyId>";

const params = {
```

```
    DatabaseName: "testDbFromNode",
    KmsKeyId: updatedKmsKeyId
};

const command = new UpdateDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Database ${data.Database.DatabaseName} updated kmsKeyId to
 ${updatedKmsKeyId}`);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Database doesn't exist.");
    } else {
        console.log("Update database failed.", error);
    }
}
```

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function updateDatabase(updatedKmsKeyId) {

    if (updatedKmsKeyId === undefined) {
        console.log("Skipping UpdateDatabase; KmsKeyId was not given");
        return;
    }
    console.log("Updating Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        KmsKeyId: updatedKmsKeyId
    }

    const promise = writeClient.updateDatabase(params).promise();

    await promise.then(
        (data) => {
            console.log(`Database ${data.Database.DatabaseName} updated kmsKeyId to
 ${updatedKmsKeyId}`);
        },
        (err) => {
            if (err.code === 'ResourceNotFoundException') {
                console.log("Database doesn't exist.");
```

```
            } else {
                console.log("Update database failed.", err);
            }
        }
    );
}
```

.NET

```
        public async Task UpdateDatabase(String updatedKmsKeyId)
        {
            Console.WriteLine("Updating Database");

            try
            {
                var updateDatabaseRequest = new UpdateDatabaseRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    KmsKeyId = updatedKmsKeyId
                };
                UpdateDatabaseResponse response = await
 writeClient.UpdateDatabaseAsync(updateDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} updated with
 KmsKeyId {updatedKmsKeyId}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Database does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Update database failed: " + e.ToString());
            }

        }

        private void PrintDatabases(List<Database> databases)
        {
            foreach (Database database in databases)
                Console.WriteLine($"Database:{database.DatabaseName}");
        }
```

刪除資料庫

您可以使用下列程式碼片段來刪除資料庫。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
   public void deleteDatabase() {
       System.out.println("Deleting database");
       final DeleteDatabaseRequest deleteDatabaseRequest = new
DeleteDatabaseRequest();
       deleteDatabaseRequest.setDatabaseName(DATABASE_NAME);
       try {
           DeleteDatabaseResult result =
                   amazonTimestreamWrite.deleteDatabase(deleteDatabaseRequest);
           System.out.println("Delete database status: " +
result.getSdkHttpMetadata().getHttpStatusCode());
       } catch (final ResourceNotFoundException e) {
           System.out.println("Database " + DATABASE_NAME + " doesn't exist = " +
e);
           throw e;
       } catch (final Exception e) {
           System.out.println("Could not delete Database " + DATABASE_NAME + " = "
+ e);
           throw e;
       }
   }
```

Java v2

```
   public void deleteDatabase() {
       System.out.println("Deleting database");
       final DeleteDatabaseRequest deleteDatabaseRequest = new
DeleteDatabaseRequest();
       deleteDatabaseRequest.setDatabaseName(DATABASE_NAME);
       try {
```

```
            DeleteDatabaseResult result =
                    amazonTimestreamWrite.deleteDatabase(deleteDatabaseRequest);
            System.out.println("Delete database status: " +
 result.getSdkHttpMetadata().getHttpStatusCode());
        } catch (final ResourceNotFoundException e) {
            System.out.println("Database " + DATABASE_NAME + " doesn't exist = " +
 e);
            throw e;
        } catch (final Exception e) {
            System.out.println("Could not delete Database " + DATABASE_NAME + " = "
 + e);
            throw e;
        }
    }
```

Go

```
deleteDatabaseInput := &timestreamwrite.DeleteDatabaseInput{
        DatabaseName:   aws.String(*databaseName),
    }

    _, err = writeSvc.DeleteDatabase(deleteDatabaseInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Database deleted:", *databaseName)
    }
```

Python

```
    def delete_database(self):
        print("Deleting Database")
        try:
            result =
 self.client.delete_database(DatabaseName=Constant.DATABASE_NAME)
            print("Delete database status [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
        except self.client.exceptions.ResourceNotFoundException:
            print("database [%s] doesn't exist" % Constant.DATABASE_NAME)
        except Exception as err:
            print("Delete database failed:", err)
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

另請參閱類別 DeleteDatabaseCommand 和 DeleteDatabase。

```
import { TimestreamWriteClient, DeleteDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode"
};

const command = new DeleteDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Deleted database");
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log(`Database ${params.DatabaseName} doesn't exists.`);
    } else {
        console.log("Delete database failed.", error);
        throw error;
    }
}
```

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function deleteDatabase() {
    console.log("Deleting Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME
    };

    const promise = writeClient.deleteDatabase(params).promise();

    await promise.then(
        function (data) {
            console.log("Deleted database");
```

```
         },
        function(err) {
            if (err.code === 'ResourceNotFoundException') {
                console.log(`Database ${params.DatabaseName} doesn't exists.`);
            } else {
                console.log("Delete database failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DeleteDatabase()
        {
            Console.WriteLine("Deleting database");
            try
            {
                var deleteDatabaseRequest = new DeleteDatabaseRequest
                {
                    DatabaseName = Constants.DATABASE_NAME
                };
                DeleteDatabaseResponse response = await
 writeClient.DeleteDatabaseAsync(deleteDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} delete
 request status:{response.HttpStatusCode}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine($"Database {Constants.DATABASE_NAME} does not
 exists");
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception while deleting database:" +
 e.ToString());
            }
        }
```

列出資料庫

您可以使用下列程式碼片段來列出資料庫。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
   public void listDatabases() {
       System.out.println("Listing databases");
       ListDatabasesRequest request = new ListDatabasesRequest();
       ListDatabasesResult result = amazonTimestreamWrite.listDatabases(request);
       final List<Database> databases = result.getDatabases();
       printDatabases(databases);

       String nextToken = result.getNextToken();
       while (nextToken != null && !nextToken.isEmpty()) {
           request.setNextToken(nextToken);
           ListDatabasesResult nextResult =
amazonTimestreamWrite.listDatabases(request);
           final List<Database> nextDatabases = nextResult.getDatabases();
           printDatabases(nextDatabases);
           nextToken = nextResult.getNextToken();
       }
   }

   private void printDatabases(List<Database> databases) {
       for (Database db : databases) {
           System.out.println(db.getDatabaseName());
       }
   }
```

Java v2

```
   public void listDatabases() {
       System.out.println("Listing databases");
       ListDatabasesRequest request =
ListDatabasesRequest.builder().maxResults(2).build();
```

```
        ListDatabasesIterable listDatabasesIterable =
 timestreamWriteClient.listDatabasesPaginator(request);
        for(ListDatabasesResponse listDatabasesResponse : listDatabasesIterable) {
            final List<Database> databases = listDatabasesResponse.databases();
            databases.forEach(database ->
 System.out.println(database.databaseName()));
        }
    }
```

Go

```
// List databases.
    listDatabasesMaxResult := int64(15)

    listDatabasesInput := &timestreamwrite.ListDatabasesInput{
        MaxResults: &listDatabasesMaxResult,
    }

    listDatabasesOutput, err := writeSvc.ListDatabases(listDatabasesInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("List databases is successful, below is the output:")
        fmt.Println(listDatabasesOutput)
    }
```

Python

```
    def list_databases(self):
        print("Listing databases")
        try:
            result = self.client.list_databases(MaxResults=5)
            self._print_databases(result['Databases'])
            next_token = result.get('NextToken', None)
            while next_token:
                result = self.client.list_databases(NextToken=next_token,
 MaxResults=5)
                self._print_databases(result['Databases'])
                next_token = result.get('NextToken', None)
        except Exception as err:
            print("List databases failed:", err)
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

另請參閱類別 ListDatabasesCommand 和 ListDatabases。

```
import { TimestreamWriteClient, ListDatabasesCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    MaxResults: 15
};

const command = new ListDatabasesCommand(params);

getDatabasesList(null);

async function getDatabasesList(nextToken) {
    if (nextToken) {
        params.NextToken = nextToken;
    }

    try {
        const data = await writeClient.send(command);

        data.Databases.forEach(function (database) {
            console.log(database.DatabaseName);
        });

        if (data.NextToken) {
            return getDatabasesList(data.NextToken);
        }
    } catch (error) {
        console.log("Error while listing databases", error);
    }
}
```

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function listDatabases() {
```

```
    console.log("Listing databases:");
    const databases = await getDatabasesList(null);
    databases.forEach(function(database){
        console.log(database.DatabaseName);
    });
}

function getDatabasesList(nextToken, databases = []) {
    var params = {
        MaxResults: 15
    };

    if(nextToken) {
        params.NextToken = nextToken;
    }

    return writeClient.listDatabases(params).promise()
        .then(
            (data) => {
                databases.push.apply(databases, data.Databases);
                if (data.NextToken) {
                    return getDatabasesList(data.NextToken, databases);
                } else {
                    return databases;
                }
            },
            (err) => {
                console.log("Error while listing databases", err);
            });
}
```

.NET

```
        public async Task ListDatabases()
        {
            Console.WriteLine("Listing Databases");

            try
            {
                var listDatabasesRequest = new ListDatabasesRequest
                {
                    MaxResults = 5
                };
```

```
                ListDatabasesResponse response = await
writeClient.ListDatabasesAsync(listDatabasesRequest);
                PrintDatabases(response.Databases);
                var nextToken = response.NextToken;
                while (nextToken != null)
                {
                    listDatabasesRequest.NextToken = nextToken;
                    response = await
writeClient.ListDatabasesAsync(listDatabasesRequest);
                    PrintDatabases(response.Databases);
                    nextToken = response.NextToken;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("List database failed:" + e.ToString());
            }

        }
```

建立資料表

主題

- 記憶體存放區寫入
- 磁性存放區寫入

記憶體存放區寫入

您可以使用下列程式碼片段來建立停用磁性存放區寫入的資料表，因此您只能將資料寫入記憶體存放區保留時段。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
   public void createTable() {
       System.out.println("Creating table");
       CreateTableRequest createTableRequest = new CreateTableRequest();
       createTableRequest.setDatabaseName(DATABASE_NAME);
       createTableRequest.setTableName(TABLE_NAME);
       final RetentionProperties retentionProperties = new RetentionProperties()
               .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
               .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);
       createTableRequest.setRetentionProperties(retentionProperties);

       try {
           amazonTimestreamWrite.createTable(createTableRequest);
           System.out.println("Table [" + TABLE_NAME + "] successfully created.");
       } catch (ConflictException e) {
           System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
DATABASE_NAME + "] . Skipping database creation");
       }
   }
```

Java v2

```
   public void createTable() {
       System.out.println("Creating table");

       final RetentionProperties retentionProperties =
RetentionProperties.builder()
               .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
               .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS).build();
       final CreateTableRequest createTableRequest = CreateTableRequest.builder()

.databaseName(DATABASE_NAME).tableName(TABLE_NAME).retentionProperties(retentionProperties)

       try {
           timestreamWriteClient.createTable(createTableRequest);
           System.out.println("Table [" + TABLE_NAME + "] successfully created.");
       } catch (ConflictException e) {
           System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
DATABASE_NAME + "] . Skipping database creation");
       }
   }
```

Go

```
// Create table.
    createTableInput := &timestreamwrite.CreateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    }
    _, err = writeSvc.CreateTable(createTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Create table is successful")
    }
```

Python

```
    def create_table(self):
        print("Creating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': Constant.HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': Constant.CT_TTL_DAYS
        }
        try:
            self.client.create_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                                     RetentionProperties=retention_properties)
            print("Table [%s] successfully created." % Constant.TABLE_NAME)
        except self.client.exceptions.ConflictException:
            print("Table [%s] exists on database [%s]. Skipping table creation" % (
                Constant.TABLE_NAME, Constant.DATABASE_NAME))
        except Exception as err:
            print("Create table failed:", err)
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

另請參閱類別 CreateTableCommand 和 CreateTable。

```
import { TimestreamWriteClient, CreateTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode",
    RetentionProperties: {
        MemoryStoreRetentionPeriodInHours: 24,
        MagneticStoreRetentionPeriodInDays: 365
    }
};

const command = new CreateTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Table ${data.Table.TableName} created successfully`);
} catch (error) {
    if (error.code === 'ConflictException') {
        console.log(`Table ${params.TableName} already exists on db
 ${params.DatabaseName}. Skipping creation.`);
    } else {
        console.log("Error creating table. ", error);
        throw error;
    }
}
```

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function createTable() {
    console.log("Creating Table");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME,
        RetentionProperties: {
            MemoryStoreRetentionPeriodInHours: constants.HT_TTL_HOURS,
            MagneticStoreRetentionPeriodInDays: constants.CT_TTL_DAYS
        }
    };

    const promise = writeClient.createTable(params).promise();
```

```
    await promise.then(
        (data) => {
            console.log(`Table ${data.Table.TableName} created successfully`);
        },
        (err) => {
            if (err.code === 'ConflictException') {
                console.log(`Table ${params.TableName} already exists on db
 ${params.DatabaseName}. Skipping creation.`);
            } else {
                console.log("Error creating table. ", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task CreateTable()
        {
            Console.WriteLine("Creating Table");

            try
            {
                var createTableRequest = new CreateTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME,
                    RetentionProperties = new RetentionProperties
                    {
                        MagneticStoreRetentionPeriodInDays = Constants.CT_TTL_DAYS,
                        MemoryStoreRetentionPeriodInHours = Constants.HT_TTL_HOURS
                    }
                };
                CreateTableResponse response = await
 writeClient.CreateTableAsync(createTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} created");
            }
            catch (ConflictException)
            {
                Console.WriteLine("Table already exists.");
            }
```

```
            catch (Exception e)
            {
                Console.WriteLine("Create table failed:" + e.ToString());
            }

        }
```

磁性存放區寫入

您可以使用下列程式碼片段來建立已啟用磁性存放區寫入的資料表。透過磁性儲存寫入，您可以將資料寫入記憶體儲存保留時段和磁性儲存保留時段。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
   public void createTable(String databaseName, String tableName) {
       System.out.println("Creating table");
       CreateTableRequest createTableRequest = new CreateTableRequest();
       createTableRequest.setDatabaseName(databaseName);
       createTableRequest.setTableName(tableName);
       final RetentionProperties retentionProperties = new RetentionProperties()
               .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
               .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);
       createTableRequest.setRetentionProperties(retentionProperties);
       // Enable MagneticStoreWrite
       final MagneticStoreWriteProperties magneticStoreWriteProperties = new
MagneticStoreWriteProperties()
               .withEnableMagneticStoreWrites(true);

createTableRequest.setMagneticStoreWriteProperties(magneticStoreWriteProperties);
       try {
           amazonTimestreamWrite.createTable(createTableRequest);
           System.out.println("Table [" + tableName + "] successfully created.");
       } catch (ConflictException e) {
           System.out.println("Table [" + tableName + "] exists on database [" +
databaseName + "] . Skipping table creation");
```

```
            //We do not throw exception here, we use the existing table instead
        }
    }
```

Java v2

```
    public void createTable(String databaseName, String tableName) {
        System.out.println("Creating table");

        // Enable MagneticStoreWrite
        final MagneticStoreWriteProperties magneticStoreWriteProperties =
                MagneticStoreWriteProperties.builder()
                        .enableMagneticStoreWrites(true)
                        .build();

        CreateTableRequest createTableRequest =
                CreateTableRequest.builder()
                        .databaseName(databaseName)
                        .tableName(tableName)
                        .retentionProperties(RetentionProperties.builder()
                                .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                                .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS)
                                .build())
                        .magneticStoreWriteProperties(magneticStoreWriteProperties)
                        .build();
        try {
            timestreamWriteClient.createTable(createTableRequest);
            System.out.println("Table [" + tableName + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + tableName + "] exists in database [" +
 databaseName + "] . Skipping table creation");
        }
    }
```

Go

```
// Create table.
    createTableInput := &timestreamwrite.CreateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    // Enable MagneticStoreWrite
        MagneticStoreWriteProperties: &timestreamwrite.MagneticStoreWriteProperties{
            EnableMagneticStoreWrites: aws.Bool(true),
```

```
            },
      }
    _, err = writeSvc.CreateTable(createTableInput)
```

Python

```
    def create_table(self):
        print("Creating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': Constant.HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': Constant.CT_TTL_DAYS
        }
        magnetic_store_write_properties = {
            'EnableMagneticStoreWrites': True
        }
        try:
            self.client.create_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                                     RetentionProperties=retention_properties,

 MagneticStoreWriteProperties=magnetic_store_write_properties)
            print("Table [%s] successfully created." % Constant.TABLE_NAME)
        except self.client.exceptions.ConflictException:
            print("Table [%s] exists on database [%s]. Skipping table creation" % (
                Constant.TABLE_NAME, Constant.DATABASE_NAME))
        except Exception as err:
            print("Create table failed:", err)
```

Node.js

```
async function createTable() {
    console.log("Creating Table");

    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME,
        RetentionProperties: {
            MemoryStoreRetentionPeriodInHours: constants.HT_TTL_HOURS,
            MagneticStoreRetentionPeriodInDays: constants.CT_TTL_DAYS
        },
        MagneticStoreWriteProperties: {
            EnableMagneticStoreWrites: true
        }
```

```
    };

    const promise = writeClient.createTable(params).promise();

    await promise.then(
        (data) => {
            console.log(`Table ${data.Table.TableName} created successfully`);
        },
        (err) => {
            if (err.code === 'ConflictException') {
                console.log(`Table ${params.TableName} already exists on db
 ${params.DatabaseName}. Skipping creation.`);
            } else {
                console.log("Error creating table. ", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task CreateTable()
        {
            Console.WriteLine("Creating Table");

            try
            {
                var createTableRequest = new CreateTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME,
                    RetentionProperties = new RetentionProperties
                    {
                        MagneticStoreRetentionPeriodInDays = Constants.CT_TTL_DAYS,
                        MemoryStoreRetentionPeriodInHours = Constants.HT_TTL_HOURS
                    },
                    // Enable MagneticStoreWrite
                    MagneticStoreWriteProperties = new MagneticStoreWriteProperties
                    {
                        EnableMagneticStoreWrites = true,
                    }
                };
```

```
                CreateTableResponse response = await
writeClient.CreateTableAsync(createTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} created");
            }
            catch (ConflictException)
            {
                Console.WriteLine("Table already exists.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Create table failed:" + e.ToString());
            }

        }
```

描述資料表

您可以使用下列程式碼片段來取得資料表屬性的相關資訊。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
   public void describeTable() {
        System.out.println("Describing table");
        final DescribeTableRequest describeTableRequest = new
DescribeTableRequest();
        describeTableRequest.setDatabaseName(DATABASE_NAME);
        describeTableRequest.setTableName(TABLE_NAME);
        try {
            DescribeTableResult result =
amazonTimestreamWrite.describeTable(describeTableRequest);
            String tableId = result.getTable().getArn();
            System.out.println("Table " + TABLE_NAME + " has id " + tableId);
        } catch (final Exception e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
```

```
        }
    }
```

Java v2

```
    public void describeTable() {
        System.out.println("Describing table");
        final DescribeTableRequest describeTableRequest =
 DescribeTableRequest.builder()
                .databaseName(DATABASE_NAME).tableName(TABLE_NAME).build();
        try {
            DescribeTableResponse response =
 timestreamWriteClient.describeTable(describeTableRequest);
            String tableId = response.table().arn();
            System.out.println("Table " + TABLE_NAME + " has id " + tableId);
        } catch (final Exception e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        }
    }
```

Go

```
// Describe table.
    describeTableInput := &timestreamwrite.DescribeTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    }
    describeTableOutput, err := writeSvc.DescribeTable(describeTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Describe table is successful, below is the output:")
        fmt.Println(describeTableOutput)
    }
```

Python

```
    def describe_table(self):
        print("Describing table")
        try:
```

```
            result = self.client.describe_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME)
            print("Table [%s] has id [%s]" % (Constant.TABLE_NAME, result['Table']
['Arn']))
        except self.client.exceptions.ResourceNotFoundException:
            print("Table doesn't exist")
        except Exception as err:
            print("Describe table failed:", err)
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

另請參閱類別 DescribeTableCommand 和 DescribeTable。

```
import { TimestreamWriteClient, DescribeTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode"
};

const command = new DescribeTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Table ${data.Table.TableName} has id ${data.Table.Arn}`);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Table or Database doesn't exist.");
    } else {
        console.log("Describe table failed.", error);
        throw error;
    }
}
```

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function describeTable() {
```

```
    console.log("Describing Table");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME
    };

    const promise = writeClient.describeTable(params).promise();

    await promise.then(
        (data) => {
            console.log(`Table ${data.Table.TableName} has id ${data.Table.Arn}`);
        },
        (err) => {
            if (err.code === 'ResourceNotFoundException') {
                console.log("Table or Database doesn't exists.");
            } else {
                console.log("Describe table failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DescribeTable()
        {
            Console.WriteLine("Describing Table");

            try
            {
                var describeTableRequest = new DescribeTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME
                };
                DescribeTableResponse response = await
 writeClient.DescribeTableAsync(describeTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} has id:
{response.Table.Arn}");
            }
            catch (ResourceNotFoundException)
            {
```

```
                Console.WriteLine("Table does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Describe table failed:" + e.ToString());
            }

        }
```

更新資料表

您可以使用下列程式碼片段來更新資料表。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
    public void updateTable() {
        System.out.println("Updating table");
        UpdateTableRequest updateTableRequest = new UpdateTableRequest();
        updateTableRequest.setDatabaseName(DATABASE_NAME);
        updateTableRequest.setTableName(TABLE_NAME);

        final RetentionProperties retentionProperties = new RetentionProperties()
                .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);

        updateTableRequest.setRetentionProperties(retentionProperties);

        amazonTimestreamWrite.updateTable(updateTableRequest);
        System.out.println("Table updated");
    }
```

Java v2

```
    public void updateTable() {
        System.out.println("Updating table");
```

```
        final RetentionProperties retentionProperties =
 RetentionProperties.builder()
                .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS).build();
        final UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()

 .databaseName(DATABASE_NAME).tableName(TABLE_NAME).retentionProperties(retentionProperties)

        timestreamWriteClient.updateTable(updateTableRequest);
        System.out.println("Table updated");
    }
```

Go

```
// Update table.
    magneticStoreRetentionPeriodInDays := int64(7 * 365)
    memoryStoreRetentionPeriodInHours := int64(24)

    updateTableInput := &timestreamwrite.UpdateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
        RetentionProperties: &timestreamwrite.RetentionProperties{
            MagneticStoreRetentionPeriodInDays: &magneticStoreRetentionPeriodInDays,
            MemoryStoreRetentionPeriodInHours:  &memoryStoreRetentionPeriodInHours,
        },
    }
    updateTableOutput, err := writeSvc.UpdateTable(updateTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Update table is successful, below is the output:")
        fmt.Println(updateTableOutput)
    }
```

Python

```
    def update_table(self):
        print("Updating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': Constant.HT_TTL_HOURS,
```

```
            'MagneticStoreRetentionPeriodInDays': Constant.CT_TTL_DAYS
        }
        try:
            self.client.update_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                                     RetentionProperties=retention_properties)
            print("Table updated.")
        except Exception as err:
            print("Update table failed:", err)
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

另請參閱類別 UpdateTableCommand 和 UpdateTable。

```
import { TimestreamWriteClient, UpdateTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode",
    RetentionProperties: {
        MemoryStoreRetentionPeriodInHours: 24,
        MagneticStoreRetentionPeriodInDays: 180
    }
};

const command = new UpdateTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Table updated")
} catch (error) {
    console.log("Error updating table. ", error);
}
```

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function updateTable() {
```

```
    console.log("Updating Table");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME,
        RetentionProperties: {
            MemoryStoreRetentionPeriodInHours: constants.HT_TTL_HOURS,
            MagneticStoreRetentionPeriodInDays: constants.CT_TTL_DAYS
        }
    };

    const promise = writeClient.updateTable(params).promise();

    await promise.then(
        (data) => {
            console.log("Table updated")
        },
        (err) => {
            console.log("Error updating table. ", err);
            throw err;
        }
    );
}
```

.NET

```
        public async Task UpdateTable()
        {
            Console.WriteLine("Updating Table");

            try
            {
                var updateTableRequest = new UpdateTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME,
                    RetentionProperties = new RetentionProperties
                    {
                        MagneticStoreRetentionPeriodInDays = Constants.CT_TTL_DAYS,
                        MemoryStoreRetentionPeriodInHours = Constants.HT_TTL_HOURS
                    }
                };
                UpdateTableResponse response = await
 writeClient.UpdateTableAsync(updateTableRequest);
```

```
                Console.WriteLine($"Table {Constants.TABLE_NAME} updated");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Table does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Update table failed:" + e.ToString());
            }

        }
```

刪除資料表

您可以使用下列程式碼片段來刪除資料表。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
   public void deleteTable() {
       System.out.println("Deleting table");
       final DeleteTableRequest deleteTableRequest = new DeleteTableRequest();
       deleteTableRequest.setDatabaseName(DATABASE_NAME);
       deleteTableRequest.setTableName(TABLE_NAME);
       try {
           DeleteTableResult result =
                   amazonTimestreamWrite.deleteTable(deleteTableRequest);
           System.out.println("Delete table status: " +
result.getSdkHttpMetadata().getHttpStatusCode());
       } catch (final ResourceNotFoundException e) {
           System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
           throw e;
       } catch (final Exception e) {
           System.out.println("Could not delete table " + TABLE_NAME + " = " + e);
           throw e;
```

```
        }
    }
```

Java v2

```
    public void deleteTable() {
        System.out.println("Deleting table");
        final DeleteTableRequest deleteTableRequest = DeleteTableRequest.builder()
                .databaseName(DATABASE_NAME).tableName(TABLE_NAME).build();
        try {
            DeleteTableResponse response =
                    timestreamWriteClient.deleteTable(deleteTableRequest);
            System.out.println("Delete table status: " +
 response.sdkHttpResponse().statusCode());
        } catch (final ResourceNotFoundException e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        } catch (final Exception e) {
            System.out.println("Could not delete table " + TABLE_NAME + " = " + e);
            throw e;
        }
    }
```

Go

```
deleteTableInput := &timestreamwrite.DeleteTableInput{
        DatabaseName:   aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    }
    _, err = writeSvc.DeleteTable(deleteTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Table deleted", *tableName)
    }
```

Python

```
    def delete_table(self):
        print("Deleting Table")
```

```
        try:
            result = self.client.delete_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME)
            print("Delete table status [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
        except self.client.exceptions.ResourceNotFoundException:
            print("Table [%s] doesn't exist" % Constant.TABLE_NAME)
        except Exception as err:
            print("Delete table failed:", err)
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

另請參閱類別 DeleteTableCommand 和 DeleteTable。

```
import { TimestreamWriteClient, DeleteTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode"
};

const command = new DeleteTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Deleted table");
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log(`Table ${params.TableName} or Database ${params.DatabaseName}
 doesn't exist.`);
    } else {
        console.log("Delete table failed.", error);
        throw error;
    }
}
```

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function deleteTable() {
    console.log("Deleting Table");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME
    };

    const promise = writeClient.deleteTable(params).promise();

    await promise.then(
        function (data) {
            console.log("Deleted table");
        },
        function(err) {
            if (err.code === 'ResourceNotFoundException') {
                console.log(`Table ${params.TableName} or Database
 ${params.DatabaseName} doesn't exists.`);
            } else {
                console.log("Delete table failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DeleteTable()
        {
            Console.WriteLine("Deleting table");
            try
            {
                var deleteTableRequest = new DeleteTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME
                };
                DeleteTableResponse response = await
 writeClient.DeleteTableAsync(deleteTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} delete request
 status: {response.HttpStatusCode}");
            }
            catch (ResourceNotFoundException)
```

```
            {
                Console.WriteLine($"Table {Constants.TABLE_NAME} does not exists");
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception while deleting table:" + e.ToString());
            }
        }
```

列出資料表

您可以使用下列程式碼片段來列出資料表。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
    public void listTables() {
        System.out.println("Listing tables");
        ListTablesRequest request = new ListTablesRequest();
        request.setDatabaseName(DATABASE_NAME);
        ListTablesResult result = amazonTimestreamWrite.listTables(request);
        printTables(result.getTables());

        String nextToken = result.getNextToken();
        while (nextToken != null && !nextToken.isEmpty()) {
            request.setNextToken(nextToken);
            ListTablesResult nextResult = amazonTimestreamWrite.listTables(request);

            printTables(nextResult.getTables());
            nextToken = nextResult.getNextToken();
        }
    }

     private void printTables(List<Table> tables) {
        for (Table table : tables) {
            System.out.println(table.getTableName());
```

```
        }
    }
```

Java v2

```
    public void listTables() {
        System.out.println("Listing tables");
        ListTablesRequest request =
 ListTablesRequest.builder().databaseName(DATABASE_NAME).maxResults(2).build();
        ListTablesIterable listTablesIterable =
 timestreamWriteClient.listTablesPaginator(request);
        for(ListTablesResponse listTablesResponse : listTablesIterable) {
            final List<Table> tables = listTablesResponse.tables();
            tables.forEach(table -> System.out.println(table.tableName()));
        }
    }
```

Go

```
listTablesMaxResult := int64(15)

    listTablesInput := &timestreamwrite.ListTablesInput{
        DatabaseName: aws.String(*databaseName),
        MaxResults:   &listTablesMaxResult,
    }
    listTablesOutput, err := writeSvc.ListTables(listTablesInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("List tables is successful, below is the output:")
        fmt.Println(listTablesOutput)
    }
```

Python

```
    def list_tables(self):
        print("Listing tables")
        try:
            result = self.client.list_tables(DatabaseName=Constant.DATABASE_NAME,
 MaxResults=5)
```

```
            self.__print_tables(result['Tables'])
            next_token = result.get('NextToken', None)
            while next_token:
                result =
 self.client.list_tables(DatabaseName=Constant.DATABASE_NAME,
                                                    NextToken=next_token, MaxResults=5)
                self.__print_tables(result['Tables'])
                next_token = result.get('NextToken', None)
        except Exception as err:
            print("List tables failed:", err)
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

另請參閱類別 ListTablesCommand 和 ListTables。

```
import { TimestreamWriteClient, ListTablesCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    MaxResults: 15
};

const command = new ListTablesCommand(params);

getTablesList(null);

async function getTablesList(nextToken) {
    if (nextToken) {
        params.NextToken = nextToken;
    }

    try {
        const data = await writeClient.send(command);

        data.Tables.forEach(function (table) {
            console.log(table.TableName);
        });

        if (data.NextToken) {
```

```
            return getTablesList(data.NextToken);
        }
    } catch (error) {
        console.log("Error while listing tables", error);
    }
}
```

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function listTables() {
    console.log("Listing tables:");
    const tables = await getTablesList(null);
    tables.forEach(function(table){
        console.log(table.TableName);
    });
}

function getTablesList(nextToken, tables = []) {
    var params = {
        DatabaseName: constants.DATABASE_NAME,
        MaxResults: 15
    };

    if(nextToken) {
        params.NextToken = nextToken;
    }

    return writeClient.listTables(params).promise()
        .then(
            (data) => {
                tables.push.apply(tables, data.Tables);
                if (data.NextToken) {
                    return getTablesList(data.NextToken, tables);
                } else {
                    return tables;
                }
            },
            (err) => {
                console.log("Error while listing databases", err);
            });
}
```

.NET

```
        public async Task ListTables()
        {
            Console.WriteLine("Listing Tables");

            try
            {
                var listTablesRequest = new ListTablesRequest
                {
                    MaxResults = 5,
                    DatabaseName = Constants.DATABASE_NAME
                };
                ListTablesResponse response = await
writeClient.ListTablesAsync(listTablesRequest);
                PrintTables(response.Tables);
                string nextToken = response.NextToken;
                while (nextToken != null)
                {
                    listTablesRequest.NextToken = nextToken;
                    response = await writeClient.ListTablesAsync(listTablesRequest);
                    PrintTables(response.Tables);
                    nextToken = response.NextToken;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("List table failed:" + e.ToString());
            }

        }

        private void PrintTables(List<Table> tables)
        {
            foreach (Table table in tables)
                Console.WriteLine($"Table: {table.TableName}");
        }
```

寫入資料（插入和 upserts）

主題

- 寫入批次記錄
- 使用常見屬性寫入記錄批次
- 提升記錄
- 多測量屬性範例
- 處理寫入失敗

寫入批次記錄

您可以使用下列程式碼片段將資料寫入 Amazon Timestream 資料表。以批次方式寫入資料有助於最佳化寫入成本。如需更多資訊，請參閱計算寫入次數。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
  public void writeRecords() {
    System.out.println("Writing records");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue("us-
east-1");
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record cpuUtilization = new Record()
        .withDimensions(dimensions)
        .withMeasureName("cpu_utilization")
        .withMeasureValue("13.5")
```

```
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));
    Record memoryUtilization = new Record()
        .withDimensions(dimensions)
        .withMeasureName("memory_utilization")
        .withMeasureValue("40")
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ":
"
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Java v2

```
 public void writeRecords() {
    System.out.println("Writing records");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
```

```
    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = Dimension.builder().name("region").value("us-
east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
 Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record cpuUtilization = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .measureName("cpu_utilization")
        .measureValue("13.5")
        .time(String.valueOf(time)).build();

    Record memoryUtilization = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .measureName("memory_utilization")
        .measureValue("40")
        .time(String.valueOf(time)).build();

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME).tableName(TABLE_NAME).records(records).build();

    try {
      WriteRecordsResponse writeRecordsResponse =
 timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
 writeRecordsResponse.sdkHttpResponse().statusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.rejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.recordIndex() + ": "
            + rejectedRecord.reason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
```

```
      System.out.println("Error: " + e);
    }
  }
```

Go

```
now := time.Now()
currentTimeInSeconds := now.Unix()
writeRecordsInput := &timestreamwrite.WriteRecordsInput{
  DatabaseName: aws.String(*databaseName),
  TableName:  aws.String(*tableName),
  Records: []*timestreamwrite.Record{
    &timestreamwrite.Record{
      Dimensions: []*timestreamwrite.Dimension{
        &timestreamwrite.Dimension{
          Name:  aws.String("region"),
          Value: aws.String("us-east-1"),
        },
        &timestreamwrite.Dimension{
          Name:  aws.String("az"),
          Value: aws.String("az1"),
        },
        &timestreamwrite.Dimension{
          Name:  aws.String("hostname"),
          Value: aws.String("host1"),
        },
      },
      MeasureName:    aws.String("cpu_utilization"),
      MeasureValue:   aws.String("13.5"),
      MeasureValueType: aws.String("DOUBLE"),
      Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
      TimeUnit:  aws.String("SECONDS"),
    },
    &timestreamwrite.Record{
      Dimensions: []*timestreamwrite.Dimension{
        &timestreamwrite.Dimension{
          Name:  aws.String("region"),
          Value: aws.String("us-east-1"),
        },
        &timestreamwrite.Dimension{
          Name:  aws.String("az"),
          Value: aws.String("az1"),
        },
```

```
        &timestreamwrite.Dimension{
          Name:  aws.String("hostname"),
          Value: aws.String("host1"),
        },
      },
      MeasureName:    aws.String("memory_utilization"),
      MeasureValue:   aws.String("40"),
      MeasureValueType: aws.String("DOUBLE"),
      Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
      TimeUnit:  aws.String("SECONDS"),
    },
  },
}

_, err = writeSvc.WriteRecords(writeRecordsInput)

if err != nil {
  fmt.Println("Error:")
  fmt.Println(err)
} else {
  fmt.Println("Write records is successful")
}
```

Python

```
  def write_records(self):
    print("Writing records")
    current_time = self._current_milli_time()

    dimensions = [
      {'Name': 'region', 'Value': 'us-east-1'},
      {'Name': 'az', 'Value': 'az1'},
      {'Name': 'hostname', 'Value': 'host1'}
    ]

    cpu_utilization = {
      'Dimensions': dimensions,
      'MeasureName': 'cpu_utilization',
      'MeasureValue': '13.5',
      'MeasureValueType': 'DOUBLE',
      'Time': current_time
    }
```

```
    memory_utilization = {
      'Dimensions': dimensions,
      'MeasureName': 'memory_utilization',
      'MeasureValue': '40',
      'MeasureValueType': 'DOUBLE',
      'Time': current_time
    }

    records = [cpu_utilization, memory_utilization]

    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                         Records=records, CommonAttributes={})
      print("WriteRecords Status: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

  @staticmethod
  def _print_rejected_records_exceptions(err):
    print("RejectedRecords: ", err)
    for rr in err.response["RejectedRecords"]:
      print("Rejected Index " + str(rr["RecordIndex"]) + ": " + rr["Reason"])
      if "ExistingVersion" in rr:
        print("Rejected record existing version: ", rr["ExistingVersion"])

  @staticmethod
  def _current_milli_time():
    return str(int(round(time.time() * 1000)))
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function writeRecords() {
  console.log("Writing records");
  const currentTime = Date.now().toString(); // Unix time in milliseconds

  const dimensions = [
```

```
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
  ];

  const cpuUtilization = {
    'Dimensions': dimensions,
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '13.5',
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString()
  };

  const memoryUtilization = {
    'Dimensions': dimensions,
    'MeasureName': 'memory_utilization',
    'MeasureValue': '40',
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString()
  };

  const records = [cpuUtilization, memoryUtilization];

  const params = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: records
  };

  const request = writeClient.writeRecords(params);

  await request.promise().then(
    (data) => {
      console.log("Write records successful");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        const responsePayload =
JSON.parse(request.response.httpResponse.body.toString());
        console.log("RejectedRecords: ", responsePayload.RejectedRecords);
        console.log("Other records were written successfully. ");
      }
    }
```

```
  );
}
```

.NET

```
   public async Task WriteRecords()
   {
     Console.WriteLine("Writing records");

     DateTimeOffset now = DateTimeOffset.UtcNow;
     string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

     List<Dimension> dimensions = new List<Dimension>{
       new Dimension { Name = "region", Value = "us-east-1" },
       new Dimension { Name = "az", Value = "az1" },
       new Dimension { Name = "hostname", Value = "host1" }
     };

     var cpuUtilization = new Record
     {
       Dimensions = dimensions,
       MeasureName = "cpu_utilization",
       MeasureValue = "13.6",
       MeasureValueType = MeasureValueType.DOUBLE,
       Time = currentTimeString
     };

     var memoryUtilization = new Record
     {
       Dimensions = dimensions,
       MeasureName = "memory_utilization",
       MeasureValue = "40",
       MeasureValueType = MeasureValueType.DOUBLE,
       Time = currentTimeString
     };


     List<Record> records = new List<Record> {
       cpuUtilization,
       memoryUtilization
     };

     try
```

```
    {
      var writeRecordsRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = records
      };
      WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
      Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      Console.WriteLine("RejectedRecordsException:" + e.ToString());
      foreach (RejectedRecord rr in e.RejectedRecords) {
        Console.WriteLine("RecordIndex " + rr.RecordIndex + " : " + rr.Reason);
      }
      Console.WriteLine("Other records were written successfully. ");
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }
  }
```

使用常見屬性寫入記錄批次

如果您的時間序列資料在許多資料點之間具有共同的量值和/或維度，您也可以使用下列最佳化版本的 writeRecords API，將資料插入 的 Timestream LiveAnalytics。搭配批次處理使用常見屬性可進一步最佳化寫入成本，如 中所述計算寫入次數。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
public void writeRecordsWithCommonAttributes() {
  System.out.println("Writing records with extracting common attributes");
```

```
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue("us-
east-1");
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
        .withDimensions(dimensions)
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));

    Record cpuUtilization = new Record()
        .withMeasureName("cpu_utilization")
        .withMeasureValue("13.5");
    Record memoryUtilization = new Record()
        .withMeasureName("memory_utilization")
        .withMeasureValue("40");

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes);
    writeRecordsRequest.setRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
 amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("writeRecordsWithCommonAttributes Status: " +
 writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
```

```
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ":
 "
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Java v2

```
  public void writeRecordsWithCommonAttributes() {
    System.out.println("Writing records with extracting common attributes");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = Dimension.builder().name("region").value("us-
east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
 Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .time(String.valueOf(time)).build();

    Record cpuUtilization = Record.builder()
        .measureName("cpu_utilization")
        .measureValue("13.5").build();
    Record memoryUtilization = Record.builder()
        .measureName("memory_utilization")
        .measureValue("40").build();

    records.add(cpuUtilization);
    records.add(memoryUtilization);
```

```
   WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
       .databaseName(DATABASE_NAME)
       .tableName(TABLE_NAME)
       .commonAttributes(commonAttributes)
       .records(records).build();

   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("writeRecordsWithCommonAttributes Status: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     System.out.println("RejectedRecords: " + e);
     for (RejectedRecord rejectedRecord : e.rejectedRecords()) {
       System.out.println("Rejected Index " + rejectedRecord.recordIndex() + ": "
           + rejectedRecord.reason());
     }
     System.out.println("Other records were written successfully. ");
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }
 }
```

Go

```
now = time.Now()
currentTimeInSeconds = now.Unix()
writeRecordsCommonAttributesInput := &timestreamwrite.WriteRecordsInput{
 DatabaseName: aws.String(*databaseName),
 TableName:  aws.String(*tableName),
 CommonAttributes: &timestreamwrite.Record{
  Dimensions: []*timestreamwrite.Dimension{
   &timestreamwrite.Dimension{
    Name:  aws.String("region"),
    Value: aws.String("us-east-1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("az"),
    Value: aws.String("az1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("hostname"),
```

```
    Value: aws.String("host1"),
   },
  },
  MeasureValueType: aws.String("DOUBLE"),
  Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
  TimeUnit:     aws.String("SECONDS"),
 },
 Records: []*timestreamwrite.Record{
  &timestreamwrite.Record{
   MeasureName:  aws.String("cpu_utilization"),
   MeasureValue: aws.String("13.5"),
  },
  &timestreamwrite.Record{
   MeasureName:  aws.String("memory_utilization"),
   MeasureValue: aws.String("40"),
  },
 },
}

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Ingest records is successful")
}
```

Python

```
  def write_records_with_common_attributes(self):
    print("Writing records extracting common attributes")
    current_time = self._current_milli_time()

    dimensions = [
      {'Name': 'region', 'Value': 'us-east-1'},
      {'Name': 'az', 'Value': 'az1'},
      {'Name': 'hostname', 'Value': 'host1'}
    ]

    common_attributes = {
      'Dimensions': dimensions,
      'MeasureValueType': 'DOUBLE',
```

```
      'Time': current_time
    }

    cpu_utilization = {
      'MeasureName': 'cpu_utilization',
      'MeasureValue': '13.5'
    }

    memory_utilization = {
      'MeasureName': 'memory_utilization',
      'MeasureValue': '40'
    }

    records = [cpu_utilization, memory_utilization]

    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                         Records=records, CommonAttributes=common_attributes)
      print("WriteRecords Status: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

  @staticmethod
  def _print_rejected_records_exceptions(err):
    print("RejectedRecords: ", err)
    for rr in err.response["RejectedRecords"]:
      print("Rejected Index " + str(rr["RecordIndex"]) + ": " + rr["Reason"])
      if "ExistingVersion" in rr:
        print("Rejected record existing version: ", rr["ExistingVersion"])

  @staticmethod
  def _current_milli_time():
    return str(int(round(time.time() * 1000)))
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function writeRecordsWithCommonAttributes() {
  console.log("Writing records with common attributes");
  const currentTime = Date.now().toString(); // Unix time in milliseconds

  const dimensions = [
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
  ];

  const commonAttributes = {
    'Dimensions': dimensions,
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString()
  };

  const cpuUtilization = {
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '13.5'
  };

  const memoryUtilization = {
    'MeasureName': 'memory_utilization',
    'MeasureValue': '40'
  };

  const records = [cpuUtilization, memoryUtilization];

  const params = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: records,
    CommonAttributes: commonAttributes
  };

  const request = writeClient.writeRecords(params);

  await request.promise().then(
    (data) => {
      console.log("Write records successful");
    },
    (err) => {
      console.log("Error writing records:", err);
```

```
      if (err.code === 'RejectedRecordsException') {
        const responsePayload =
 JSON.parse(request.response.httpResponse.body.toString());
        console.log("RejectedRecords: ", responsePayload.RejectedRecords);
        console.log("Other records were written successfully. ");
      }
    }
  );
}
```

.NET

```
  public async Task WriteRecordsWithCommonAttributes()
  {
    Console.WriteLine("Writing records with common attributes");

    DateTimeOffset now = DateTimeOffset.UtcNow;
    string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

    List<Dimension> dimensions = new List<Dimension>{
      new Dimension { Name = "region", Value = "us-east-1" },
      new Dimension { Name = "az", Value = "az1" },
      new Dimension { Name = "hostname", Value = "host1" }
    };

    var commonAttributes = new Record
    {
      Dimensions = dimensions,
      MeasureValueType = MeasureValueType.DOUBLE,
      Time = currentTimeString
    };

    var cpuUtilization = new Record
    {
      MeasureName = "cpu_utilization",
      MeasureValue = "13.6"
    };

    var memoryUtilization = new Record
    {
      MeasureName = "memory_utilization",
      MeasureValue = "40"
    };
```

```
   List<Record> records = new List<Record>();
   records.Add(cpuUtilization);
   records.Add(memoryUtilization);

   try
   {
     var writeRecordsRequest = new WriteRecordsRequest
     {
       DatabaseName = Constants.DATABASE_NAME,
       TableName = Constants.TABLE_NAME,
       Records = records,
       CommonAttributes = commonAttributes
     };
     WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
     Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
   }
   catch (RejectedRecordsException e) {
     Console.WriteLine("RejectedRecordsException:" + e.ToString());
     foreach (RejectedRecord rr in e.RejectedRecords) {
       Console.WriteLine("RecordIndex " + rr.RecordIndex + " : " + rr.Reason);
     }
     Console.WriteLine("Other records were written successfully. ");
   }
   catch (Exception e)
   {
     Console.WriteLine("Write records failure:" + e.ToString());
   }
 }
```

提升記錄

雖然 Amazon Timestream 中的預設寫入會依照第一個寫入器的語義來獲勝，其中資料僅以附加方式儲存，而重複記錄會被拒絕，但有些應用程式需要使用最後一個寫入器將資料寫入 Amazon Timestream，以獲勝語義來獲勝，其中具有最高版本的記錄會儲存在系統中。還有需要更新現有記錄的應用程式。為了解決這些案例，Amazon Timestream 提供升級資料的功能。Upsert 是一種操作，當記錄不存在時，將記錄插入系統，或在記錄存在時更新記錄。

您可以在傳送WriteRecords請求時，將 包含在記錄定義Version中，以升級記錄。Amazon Timestream 會將記錄與最高 的記錄一起存放Version。下面的程式碼範例示範如何升級資料：

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
  public void writeRecordsWithUpsert() {
    System.out.println("Writing records with upsert");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    // To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue("us-
east-1");
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
        .withDimensions(dimensions)
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time))
        .withVersion(version);

    Record cpuUtilization = new Record()
        .withMeasureName("cpu_utilization")
        .withMeasureValue("13.5");
    Record memoryUtilization = new Record()
        .withMeasureName("memory_utilization")
```

```
        .withMeasureValue("40");

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes);
    writeRecordsRequest.setRecords(records);

    // write records for first time
    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status for first time: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }

    // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status for retry: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }

    // upsert with lower version, this would fail because a higher version is
required to update the measure value.
    version -= 1;
    commonAttributes.setVersion(version);

    cpuUtilization.setMeasureValue("14.5");
    memoryUtilization.setMeasureValue("50");
```

```
   List<Record> upsertedRecords = new ArrayList<>();
   upsertedRecords.add(cpuUtilization);
   upsertedRecords.add(memoryUtilization);

   WriteRecordsRequest writeRecordsUpsertRequest = new WriteRecordsRequest()
       .withDatabaseName(DATABASE_NAME)
       .withTableName(TABLE_NAME)
       .withCommonAttributes(commonAttributes);
   writeRecordsUpsertRequest.setRecords(upsertedRecords);

   try {
     WriteRecordsResult writeRecordsUpsertResult =
amazonTimestreamWrite.writeRecords(writeRecordsUpsertRequest);
     System.out.println("WriteRecords Status for upsert with lower version: " +
writeRecordsUpsertResult.getSdkHttpMetadata().getHttpStatusCode());
   } catch (RejectedRecordsException e) {
     System.out.println("WriteRecords Status for upsert with lower version: ");
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }

   // upsert with higher version as new data in generated
   version = System.currentTimeMillis();
   commonAttributes.setVersion(version);

   writeRecordsUpsertRequest = new WriteRecordsRequest()
       .withDatabaseName(DATABASE_NAME)
       .withTableName(TABLE_NAME)
       .withCommonAttributes(commonAttributes);
   writeRecordsUpsertRequest.setRecords(upsertedRecords);

   try {
     WriteRecordsResult writeRecordsUpsertResult =
amazonTimestreamWrite.writeRecords(writeRecordsUpsertRequest);
     System.out.println("WriteRecords Status for upsert with higher version: " +
writeRecordsUpsertResult.getSdkHttpMetadata().getHttpStatusCode());
   } catch (RejectedRecordsException e) {
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }
 }
```

Java v2

```
  public void writeRecordsWithUpsert() {
    System.out.println("Writing records with upsert");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    // To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = Dimension.builder().name("region").value("us-
east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
 Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .time(String.valueOf(time))
        .version(version)
        .build();

    Record cpuUtilization = Record.builder()
        .measureName("cpu_utilization")
        .measureValue("13.5").build();
    Record memoryUtilization = Record.builder()
        .measureName("memory_utilization")
        .measureValue("40").build();

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(records).build();
```

```
   // write records for first time
   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("WriteRecords Status for first time: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }

   // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("WriteRecords Status for retry: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }

   // upsert with lower version, this would fail because a higher version is
required to update the measure value.
   version -= 1;
   commonAttributes = Record.builder()
       .dimensions(dimensions)
       .measureValueType(MeasureValueType.DOUBLE)
       .time(String.valueOf(time))
       .version(version)
       .build();

   cpuUtilization = Record.builder()
       .measureName("cpu_utilization")
       .measureValue("14.5").build();
   memoryUtilization = Record.builder()
       .measureName("memory_utilization")
       .measureValue("50").build();

   List<Record> upsertedRecords = new ArrayList<>();
```

```
    upsertedRecords.add(cpuUtilization);
    upsertedRecords.add(memoryUtilization);

    WriteRecordsRequest writeRecordsUpsertRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(upsertedRecords).build();

    try {
      WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsUpsertRequest);
      System.out.println("WriteRecords Status for upsert with lower version: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("WriteRecords Status for upsert with lower version: ");
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }

    // upsert with higher version as new data in generated
    version = System.currentTimeMillis();
    commonAttributes = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .time(String.valueOf(time))
        .version(version)
        .build();

    writeRecordsUpsertRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(upsertedRecords).build();

    try {
      WriteRecordsResponse writeRecordsUpsertResponse =
timestreamWriteClient.writeRecords(writeRecordsUpsertRequest);
      System.out.println("WriteRecords Status for upsert with higher version: " +
writeRecordsUpsertResponse.sdkHttpResponse().statusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
```

```
      System.out.println("Error: " + e);
    }
  }
```

Go

```
// Below code will ingest and upsert cpu_utilization and memory_utilization metric
 for a host on
// region=us-east-1, az=az1, and hostname=host1
fmt.Println("Ingesting records and set version as currentTimeInMills, hit enter to
 continue")
reader.ReadString('\n')

// Get current time in seconds.
now = time.Now()
currentTimeInSeconds = now.Unix()
// To achieve upsert (last writer wins) semantic, one example is to use current time
 as the version if you are writing directly from the data source
version := time.Now().Round(time.Millisecond).UnixNano() / 1e6   // set version as
 currentTimeInMills

writeRecordsCommonAttributesUpsertInput := &timestreamwrite.WriteRecordsInput{
 DatabaseName: aws.String(*databaseName),
 TableName:  aws.String(*tableName),
 CommonAttributes: &timestreamwrite.Record{
  Dimensions: []*timestreamwrite.Dimension{
   &timestreamwrite.Dimension{
    Name:  aws.String("region"),
    Value: aws.String("us-east-1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("az"),
    Value: aws.String("az1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("hostname"),
    Value: aws.String("host1"),
   },
  },
  MeasureValueType: aws.String("DOUBLE"),
  Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
  TimeUnit:  aws.String("SECONDS"),
  Version:      &version,
```

```
 },
 Records: []*timestreamwrite.Record{
  &timestreamwrite.Record{
   MeasureName:  aws.String("cpu_utilization"),
   MeasureValue: aws.String("13.5"),
  },
  &timestreamwrite.Record{
   MeasureName:  aws.String("memory_utilization"),
   MeasureValue: aws.String("40"),
  },
 },
}

// write records for first time
_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Frist-time write records is successful")
}

fmt.Println("Retry same writeRecordsRequest with same records and versions. Because
 writeRecords API is idempotent, this will success. hit enter to continue")
reader.ReadString('\n')

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Retry write records for same request is successful")
}

fmt.Println("Upsert with lower version, this would fail because a higher version is
 required to update the measure value. hit enter to continue")
reader.ReadString('\n')
version -= 1
writeRecordsCommonAttributesUpsertInput.CommonAttributes.Version = &version

updated_cpu_utilization := &timestreamwrite.Record{
 MeasureName:    aws.String("cpu_utilization"),
```

```
 MeasureValue:   aws.String("14.5"),
}
updated_memory_utilization := &timestreamwrite.Record{
 MeasureName:    aws.String("memory_utilization"),
 MeasureValue:   aws.String("50"),
}


writeRecordsCommonAttributesUpsertInput.Records = []*timestreamwrite.Record{
 updated_cpu_utilization,
 updated_memory_utilization,
}

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Write records with lower version is successful")
}

fmt.Println("Upsert with higher version as new data in generated, this would
 success. hit enter to continue")
reader.ReadString('\n')

version = time.Now().Round(time.Millisecond).UnixNano() / 1e6  // set version as
 currentTimeInMills
writeRecordsCommonAttributesUpsertInput.CommonAttributes.Version = &version

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Write records with higher version is successful")
}
```

Python

```
  def write_records_with_upsert(self):
    print("Writing records with upsert")
```

```
    current_time = self._current_milli_time()
    # To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
    version = int(self._current_milli_time())

    dimensions = [
          {'Name': 'region', 'Value': 'us-east-1'},
          {'Name': 'az', 'Value': 'az1'},
          {'Name': 'hostname', 'Value': 'host1'}
        ]

    common_attributes = {
      'Dimensions': dimensions,
      'MeasureValueType': 'DOUBLE',
      'Time': current_time,
      'Version': version
    }

    cpu_utilization = {
      'MeasureName': 'cpu_utilization',
      'MeasureValue': '13.5'
    }

    memory_utilization = {
      'MeasureName': 'memory_utilization',
      'MeasureValue': '40'
    }

    records = [cpu_utilization, memory_utilization]

    # write records for first time
    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                          Records=records, CommonAttributes=common_attributes)
      print("WriteRecords Status for first time: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

    # Successfully retry same writeRecordsRequest with same records and versions,
 because writeRecords API is idempotent.
```

```
    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                          Records=records, CommonAttributes=common_attributes)
      print("WriteRecords Status for retry: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

    # upsert with lower version, this would fail because a higher version is
 required to update the measure value.
    version -= 1
    common_attributes["Version"] = version

    cpu_utilization["MeasureValue"] = '14.5'
    memory_utilization["MeasureValue"] = '50'

    upsertedRecords = [cpu_utilization, memory_utilization]

    try:
      upsertedResult =
 self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                            Records=upsertedRecords,
 CommonAttributes=common_attributes)
      print("WriteRecords Status for upsert with lower version: [%s]" %
 upsertedResult['ResponseMetadata']['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)


    # upsert with higher version as new data is generated
    version = int(self._current_milli_time())
    common_attributes["Version"] = version

    try:
      upsertedResult =
 self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
```

```
                                Records=upsertedRecords,
 CommonAttributes=common_attributes)
      print("WriteRecords Upsert Status: [%s]" % upsertedResult['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

  @staticmethod
  def _current_milli_time():
    return str(int(round(time.time() * 1000)))
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function writeRecordsWithUpsert() {
  console.log("Writing records with upsert");
  const currentTime = Date.now().toString(); // Unix time in milliseconds
  // To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
  let version = Date.now();

  const dimensions = [
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
  ];

  const commonAttributes = {
    'Dimensions': dimensions,
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString(),
    'Version': version
  };

  const cpuUtilization = {
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '13.5'
  };
```

```
  const memoryUtilization = {
    'MeasureName': 'memory_utilization',
    'MeasureValue': '40'
  };

  const records = [cpuUtilization, memoryUtilization];

  const params = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: records,
    CommonAttributes: commonAttributes
  };

  const request = writeClient.writeRecords(params);

  // write records for first time
  await request.promise().then(
    (data) => {
      console.log("Write records successful for first time.");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        printRejectedRecordsException(request);
      }
    }
  );

  // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
  await request.promise().then(
    (data) => {
      console.log("Write records successful for retry.");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        printRejectedRecordsException(request);
      }
    }
  );
```

```
 // upsert with lower version, this would fail because a higher version is required
to update the measure value.
 version--;

 const commonAttributesWithLowerVersion = {
   'Dimensions': dimensions,
   'MeasureValueType': 'DOUBLE',
   'Time': currentTime.toString(),
   'Version': version
 };

 const updatedCpuUtilization = {
   'MeasureName': 'cpu_utilization',
   'MeasureValue': '14.5'
 };

 const updatedMemoryUtilization = {
   'MeasureName': 'memory_utilization',
   'MeasureValue': '50'
 };

 const upsertedRecords = [updatedCpuUtilization, updatedMemoryUtilization];

 const upsertedParamsWithLowerVersion = {
   DatabaseName: constants.DATABASE_NAME,
   TableName: constants.TABLE_NAME,
   Records: upsertedRecords,
   CommonAttributes: commonAttributesWithLowerVersion
 };

 const upsertRequestWithLowerVersion =
writeClient.writeRecords(upsertedParamsWithLowerVersion);

 await upsertRequestWithLowerVersion.promise().then(
   (data) => {
     console.log("Write records for upsert with lower version successful");
   },
   (err) => {
     console.log("Error writing records:", err);
     if (err.code === 'RejectedRecordsException') {
       printRejectedRecordsException(upsertRequestWithLowerVersion);
     }
   }
 );
```

```
  // upsert with higher version as new data in generated
  version = Date.now();

  const commonAttributesWithHigherVersion = {
    'Dimensions': dimensions,
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString(),
    'Version': version
  };

  const upsertedParamsWithHigherVerion = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: upsertedRecords,
    CommonAttributes: commonAttributesWithHigherVersion
  };

  const upsertRequestWithHigherVersion =
 writeClient.writeRecords(upsertedParamsWithHigherVerion);

  await upsertRequestWithHigherVersion.promise().then(
    (data) => {
      console.log("Write records upsert successful with higher version");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        printRejectedRecordsException(upsertedParamsWithHigherVerion);
      }
    }
  );

}
```

.NET

```
  public async Task WriteRecordsWithUpsert()
  {
    Console.WriteLine("Writing records with upsert");

    DateTimeOffset now = DateTimeOffset.UtcNow;
    string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();
```

```
   // To achieve upsert (last writer wins) semantic, one example is to use current
time as the version if you are writing directly from the data source
   long version = now.ToUnixTimeMilliseconds();

   List<Dimension> dimensions = new List<Dimension>{
     new Dimension { Name = "region", Value = "us-east-1" },
     new Dimension { Name = "az", Value = "az1" },
     new Dimension { Name = "hostname", Value = "host1" }
   };

   var commonAttributes = new Record
   {
     Dimensions = dimensions,
     MeasureValueType = MeasureValueType.DOUBLE,
     Time = currentTimeString,
     Version = version
   };

   var cpuUtilization = new Record
   {
     MeasureName = "cpu_utilization",
     MeasureValue = "13.6"
   };

   var memoryUtilization = new Record
   {
     MeasureName = "memory_utilization",
     MeasureValue = "40"
   };


   List<Record> records = new List<Record>();
   records.Add(cpuUtilization);
   records.Add(memoryUtilization);

   // write records for first time
   try
   {
     var writeRecordsRequest = new WriteRecordsRequest
     {
       DatabaseName = Constants.DATABASE_NAME,
       TableName = Constants.TABLE_NAME,
       Records = records,
       CommonAttributes = commonAttributes
```

```
      };
      WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
      Console.WriteLine($"WriteRecords Status for first time:
{response.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      PrintRejectedRecordsException(e);
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }

    // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
    try
    {
      var writeRecordsRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = records,
        CommonAttributes = commonAttributes
      };
      WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
      Console.WriteLine($"WriteRecords Status for retry:
{response.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      PrintRejectedRecordsException(e);
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }

    // upsert with lower version, this would fail because a higher version is
required to update the measure value.
    version--;
    Type recordType = typeof(Record);
    recordType.GetProperty("Version").SetValue(commonAttributes, version);
    recordType.GetProperty("MeasureValue").SetValue(cpuUtilization, "14.6");
```

```
    recordType.GetProperty("MeasureValue").SetValue(memoryUtilization, "50");

    List<Record> upsertedRecords = new List<Record> {
      cpuUtilization,
      memoryUtilization
    };

    try
    {
      var writeRecordsUpsertRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = upsertedRecords,
        CommonAttributes = commonAttributes
      };
      WriteRecordsResponse upsertResponse = await
writeClient.WriteRecordsAsync(writeRecordsUpsertRequest);
      Console.WriteLine($"WriteRecords Status for upsert with lower version:
{upsertResponse.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      PrintRejectedRecordsException(e);
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }

    // upsert with higher version as new data in generated
    now = DateTimeOffset.UtcNow;
    version = now.ToUnixTimeMilliseconds();
    recordType.GetProperty("Version").SetValue(commonAttributes, version);

    try
    {
      var writeRecordsUpsertRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = upsertedRecords,
        CommonAttributes = commonAttributes
      };
```

```
     WriteRecordsResponse upsertResponse = await
writeClient.WriteRecordsAsync(writeRecordsUpsertRequest);
     Console.WriteLine($"WriteRecords Status for upsert with higher version:
{upsertResponse.HttpStatusCode.ToString()}");
   }
   catch (RejectedRecordsException e) {
     PrintRejectedRecordsException(e);
   }
   catch (Exception e)
   {
     Console.WriteLine("Write records failure:" + e.ToString());
   }
 }
```

多測量屬性範例

此範例說明寫入多模態屬性。當您追蹤的裝置或應用程式在相同的時間戳記發出多個指標或事件時，多測量屬性很有用。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
package com.amazonaws.services.timestream;

import static com.amazonaws.services.timestream.Main.DATABASE_NAME;
import static com.amazonaws.services.timestream.Main.REGION;
import static com.amazonaws.services.timestream.Main.TABLE_NAME;

import java.util.ArrayList;
import java.util.List;

import com.amazonaws.services.timestreamwrite.AmazonTimestreamWrite;
import com.amazonaws.services.timestreamwrite.model.Dimension;
import com.amazonaws.services.timestreamwrite.model.MeasureValue;
import com.amazonaws.services.timestreamwrite.model.MeasureValueType;
import com.amazonaws.services.timestreamwrite.model.Record;
```

```
import com.amazonaws.services.timestreamwrite.model.RejectedRecordsException;
import com.amazonaws.services.timestreamwrite.model.WriteRecordsRequest;
import com.amazonaws.services.timestreamwrite.model.WriteRecordsResult;


public class multimeasureAttributeExample {
  AmazonTimestreamWrite timestreamWriteClient;

  public multimeasureAttributeExample(AmazonTimestreamWrite client) {
    this.timestreamWriteClient = client;
  }

  public void writeRecordsMultiMeasureValueSingleRecord() {
    System.out.println("Writing records with multi value attributes");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue(REGION);
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
        .withDimensions(dimensions)
        .withTime(String.valueOf(time))
        .withVersion(version);

    MeasureValue cpuUtilization = new MeasureValue()
        .withName("cpu_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("13.5");
    MeasureValue memoryUtilization = new MeasureValue()
        .withName("memory_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("40");
    Record computationalResources = new Record()
        .withMeasureName("cpu_memory")
```

```
        .withMeasureValues(cpuUtilization, memoryUtilization)
        .withMeasureValueType(MeasureValueType.MULTI);

    records.add(computationalResources);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes)
        .withRecords(records);

    // write records for first time
    try {
      WriteRecordsResult writeRecordResult =
timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println(
          "WriteRecords Status for multi value attributes: " + writeRecordResult
              .getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }

  public void writeRecordsMultiMeasureValueMultipleRecords() {
    System.out.println(
        "Writing records with multi value attributes mixture type");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue(REGION);
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
```

```
        .withDimensions(dimensions)
        .withTime(String.valueOf(time))
        .withVersion(version);

    MeasureValue cpuUtilization = new MeasureValue()
        .withName("cpu_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("13");
    MeasureValue memoryUtilization =new MeasureValue()
        .withName("memory_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("40");
    MeasureValue activeCores = new MeasureValue()
        .withName("active_cores")
        .withType(MeasureValueType.BIGINT)
        .withValue("4");


    Record computationalResources = new Record()
        .withMeasureName("computational_utilization")
        .withMeasureValues(cpuUtilization, memoryUtilization, activeCores)
        .withMeasureValueType(MeasureValueType.MULTI);

    records.add(computationalResources);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes)
        .withRecords(records);

    // write records for first time
    try {
      WriteRecordsResult writeRecordResult =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println(
         "WriteRecords Status for multi value attributes: " + writeRecordResult
             .getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

```
  private void printRejectedRecordsException(RejectedRecordsException e) {
    System.out.println("RejectedRecords: " + e);
    e.getRejectedRecords().forEach(System.out::println);
  }
}
```

Java v2

```
package com.amazonaws.services.timestream;

import java.util.ArrayList;
import java.util.List;

import software.amazon.awssdk.services.timestreamwrite.TimestreamWriteClient;
import software.amazon.awssdk.services.timestreamwrite.model.Dimension;
import software.amazon.awssdk.services.timestreamwrite.model.MeasureValue;
import software.amazon.awssdk.services.timestreamwrite.model.MeasureValueType;
import software.amazon.awssdk.services.timestreamwrite.model.Record;
import
 software.amazon.awssdk.services.timestreamwrite.model.RejectedRecordsException;
import software.amazon.awssdk.services.timestreamwrite.model.WriteRecordsRequest;
import software.amazon.awssdk.services.timestreamwrite.model.WriteRecordsResponse;

import static com.amazonaws.services.timestream.Main.DATABASE_NAME;
import static com.amazonaws.services.timestream.Main.TABLE_NAME;


public class multimeasureAttributeExample {

  TimestreamWriteClient timestreamWriteClient;

  public multimeasureAttributeExample(TimestreamWriteClient client) {
    this.timestreamWriteClient = client;
  }

  public void writeRecordsMultiMeasureValueSingleRecord() {
    System.out.println("Writing records with multi value attributes");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();
```

```
    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region =
        Dimension.builder().name("region").value("us-east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
        Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = Record.builder()
        .dimensions(dimensions)
        .time(String.valueOf(time))
        .version(version)
        .build();

    MeasureValue cpuUtilization = MeasureValue.builder()
        .name("cpu_utilization")
        .type(MeasureValueType.DOUBLE)
        .value("13.5").build();
    MeasureValue memoryUtilization = MeasureValue.builder()
        .name("memory_utilization")
        .type(MeasureValueType.DOUBLE)
        .value("40").build();
    Record computationalResources = Record
        .builder()
        .measureName("cpu_memory")
        .measureValues(cpuUtilization, memoryUtilization)
        .measureValueType(MeasureValueType.MULTI)
        .build();

    records.add(computationalResources);

    WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(records).build();

    // write records for first time
    try {
      WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
```

```
      System.out.println(
          "WriteRecords Status for multi value attributes: " + writeRecordsResponse
              .sdkHttpResponse()
              .statusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }

  public void writeRecordsMultiMeasureValueMultipleRecords() {
    System.out.println(
        "Writing records with multi value attributes mixture type");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region =
        Dimension.builder().name("region").value("us-east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
        Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = Record.builder()
        .dimensions(dimensions)
        .time(String.valueOf(time))
        .version(version)
        .build();

    MeasureValue cpuUtilization = MeasureValue.builder()
        .name("cpu_utilization")
        .type(MeasureValueType.DOUBLE)
        .value("13.5").build();
    MeasureValue memoryUtilization = MeasureValue.builder()
        .name("memory_utilization")
        .type(MeasureValueType.DOUBLE)
        .value("40").build();
```

```
    MeasureValue activeCores = MeasureValue.builder()
        .name("active_cores")
        .type(MeasureValueType.BIGINT)
        .value("4").build();


    Record computationalResources = Record
        .builder()
        .measureName("computational_utilization")
        .measureValues(cpuUtilization, memoryUtilization, activeCores)
        .measureValueType(MeasureValueType.MULTI)
        .build();

    records.add(computationalResources);

    WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(records).build();

    // write records for first time
    try {
      WriteRecordsResponse writeRecordsResponse =
 timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println(
          "WriteRecords Status for multi value attributes: " + writeRecordsResponse
              .sdkHttpResponse()
              .statusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }

  private void printRejectedRecordsException(RejectedRecordsException e) {
    System.out.println("RejectedRecords: " + e);
    e.rejectedRecords().forEach(System.out::println);
  }
}
```

Go

```
now := time.Now()
currentTimeInSeconds := now.Unix()
writeRecordsInput := &timestreamwrite.WriteRecordsInput{
  DatabaseName: aws.String(*databaseName),
  TableName:  aws.String(*tableName),
  Records: []*timestreamwrite.Record{
  &timestreamwrite.Record{
    Dimensions: []*timestreamwrite.Dimension{
    &timestreamwrite.Dimension{
      Name:  aws.String("region"),
      Value: aws.String("us-east-1"),
    },
    &timestreamwrite.Dimension{
      Name:  aws.String("az"),
      Value: aws.String("az1"),
    },
    &timestreamwrite.Dimension{
      Name:  aws.String("hostname"),
      Value: aws.String("host1"),
    },
    },
    MeasureName:  aws.String("metrics"),
    MeasureValueType: aws.String("MULTI"),
    Time:     aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
    TimeUnit:  aws.String("SECONDS"),
    MeasureValues: []*timestreamwrite.MeasureValue{
    &timestreamwrite.MeasureValue{
      Name:  aws.String("cpu_utilization"),
      Value: aws.String("13.5"),
      Type:  aws.String("DOUBLE"),
    },
    &timestreamwrite.MeasureValue{
      Name:  aws.String("memory_utilization"),
      Value: aws.String("40"),
      Type:  aws.String("DOUBLE"),
    },
    },
  },
  },
}

_, err = writeSvc.WriteRecords(writeRecordsInput)
```

```
  if err != nil {
    fmt.Println("Error:")
    fmt.Println(err)
  } else {
    fmt.Println("Write records is successful")
  }
```

Python

```
import time
import boto3
import psutil
import os

from botocore.config import Config

DATABASE_NAME = os.environ['DATABASE_NAME']
TABLE_NAME = os.environ['TABLE_NAME']

COUNTRY = "UK"
CITY = "London"
HOSTNAME = "MyHostname" # You can make it dynamic using socket.gethostname()

INTERVAL = 1 # Seconds

def prepare_common_attributes():
  common_attributes = {
    'Dimensions': [
      {'Name': 'country', 'Value': COUNTRY},
      {'Name': 'city', 'Value': CITY},
      {'Name': 'hostname', 'Value': HOSTNAME}
    ],
    'MeasureName': 'utilization',
    'MeasureValueType': 'MULTI'
  }
  return common_attributes


def prepare_record(current_time):
  record = {
    'Time': str(current_time),
    'MeasureValues': []
```

```
  }
  return record


def prepare_measure(measure_name, measure_value):
  measure = {
    'Name': measure_name,
    'Value': str(measure_value),
    'Type': 'DOUBLE'
  }
  return measure


def write_records(records, common_attributes):
  try:
    result = write_client.write_records(DatabaseName=DATABASE_NAME,
                                        TableName=TABLE_NAME,
                                        CommonAttributes=common_attributes,
                                        Records=records)
    status = result['ResponseMetadata']['HTTPStatusCode']
    print("Processed %d records. WriteRecords HTTPStatusCode: %s" %
        (len(records), status))
  except Exception as err:
    print("Error:", err)


if __name__ == '__main__':

  print("writing data to database {} table {}".format(
    DATABASE_NAME, TABLE_NAME))

  session = boto3.Session()
  write_client = session.client('timestream-write', config=Config(
    read_timeout=20, max_pool_connections=5000, retries={'max_attempts': 10}))
  query_client = session.client('timestream-query') # Not used

  common_attributes = prepare_common_attributes()

  records = []

  while True:

    current_time = int(time.time() * 1000)
    cpu_utilization = psutil.cpu_percent()
```

```
    memory_utilization = psutil.virtual_memory().percent
    swap_utilization = psutil.swap_memory().percent
    disk_utilization = psutil.disk_usage('/').percent

    record = prepare_record(current_time)
    record['MeasureValues'].append(prepare_measure('cpu', cpu_utilization))
    record['MeasureValues'].append(prepare_measure('memory', memory_utilization))
    record['MeasureValues'].append(prepare_measure('swap', swap_utilization))
    record['MeasureValues'].append(prepare_measure('disk', disk_utilization))

    records.append(record)

    print("records {} - cpu {} - memory {} - swap {} - disk {}".format(
      len(records), cpu_utilization, memory_utilization,
      swap_utilization, disk_utilization))

    if len(records) == 100:
      write_records(records, common_attributes)
      records = []

    time.sleep(INTERVAL)
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function writeRecords() {
  console.log("Writing records");
  const currentTime = Date.now().toString(); // Unix time in milliseconds

  const dimensions = [
  {'Name': 'region', 'Value': 'us-east-1'},
  {'Name': 'az', 'Value': 'az1'},
  {'Name': 'hostname', 'Value': 'host1'}
  ];

  const record = {
  'Dimensions': dimensions,
  'MeasureName': 'metrics',
  'MeasureValues': [
    {
      'Name': 'cpu_utilization',
```

```
        'Value': '40',
        'Type': 'DOUBLE',
      },
      {
        'Name': 'memory_utilization',
        'Value': '13.5',
        'Type': 'DOUBLE',
      },
      ],
      'MeasureValueType': 'MULTI',
      'Time': currentTime.toString()
    }

    const records = [record];

    const params = {
    DatabaseName: 'DatabaseName',
    TableName: 'TableName',
    Records: records
    };

    const response = await writeClient.writeRecords(params);

    console.log(response);
  }
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
  static class MultiMeasureValueConstants
  {
    public const string MultiMeasureValueSampleDb = "multiMeasureValueSampleDb";
    public const string MultiMeasureValueSampleTable =
 "multiMeasureValueSampleTable";
  }
```

```
public class MultiValueAttributesExample
{
  private readonly AmazonTimestreamWriteClient writeClient;

  public MultiValueAttributesExample(AmazonTimestreamWriteClient writeClient)
  {
    this.writeClient = writeClient;
  }

  public async Task WriteRecordsMultiMeasureValueSingleRecord()
  {
    Console.WriteLine("Writing records with multi value attributes");

    DateTimeOffset now = DateTimeOffset.UtcNow;
    string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

    List<Dimension> dimensions = new List<Dimension>{
      new Dimension { Name = "region", Value = "us-east-1" },
      new Dimension { Name = "az", Value = "az1" },
      new Dimension { Name = "hostname", Value = "host1" }
    };

    var commonAttributes = new Record
    {
      Dimensions = dimensions,
      Time = currentTimeString
    };

    var cpuUtilization = new MeasureValue
    {
      Name = "cpu_utilization",
      Value = "13.6",
      Type = "DOUBLE"
    };

    var memoryUtilization = new MeasureValue
    {
      Name = "memory_utilization",
      Value = "40",
      Type = "DOUBLE"
    };

    var computationalRecord = new Record
```

```
      {
        MeasureName = "cpu_memory",
        MeasureValues = new List<MeasureValue> {cpuUtilization, memoryUtilization},
        MeasureValueType = "MULTI"
      };


      List<Record> records = new List<Record>();
      records.Add(computationalRecord);

      try
      {
        var writeRecordsRequest = new WriteRecordsRequest
        {
          DatabaseName = MultiMeasureValueConstants.MultiMeasureValueSampleDb,
          TableName = MultiMeasureValueConstants.MultiMeasureValueSampleTable,
          Records = records,
          CommonAttributes = commonAttributes
        };
        WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
        Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
      }
      catch (Exception e)
      {
        Console.WriteLine("Write records failure:" + e.ToString());
      }
    }

    public async Task WriteRecordsMultiMeasureValueMultipleRecords()
    {
      Console.WriteLine("Writing records with multi value attributes mixture type");

      DateTimeOffset now = DateTimeOffset.UtcNow;
      string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

      List<Dimension> dimensions = new List<Dimension>{
        new Dimension { Name = "region", Value = "us-east-1" },
        new Dimension { Name = "az", Value = "az1" },
        new Dimension { Name = "hostname", Value = "host1" }
      };

      var commonAttributes = new Record
```

```
      {
        Dimensions = dimensions,
        Time = currentTimeString
      };

      var cpuUtilization = new MeasureValue
      {
        Name = "cpu_utilization",
        Value = "13.6",
        Type = "DOUBLE"
      };

      var memoryUtilization = new MeasureValue
      {
        Name = "memory_utilization",
        Value = "40",
        Type = "DOUBLE"
      };

      var activeCores = new MeasureValue
      {
        Name = "active_cores",
        Value = "4",
        Type = "BIGINT"
      };

      var computationalRecord = new Record
      {
        MeasureName = "computational_utilization",
        MeasureValues = new List<MeasureValue> {cpuUtilization, memoryUtilization,
activeCores},
        MeasureValueType = "MULTI"
      };

      var aliveRecord = new Record
      {
        MeasureName = "is_healthy",
        MeasureValue = "true",
        MeasureValueType = "BOOLEAN"
      };

      List<Record> records = new List<Record>();
      records.Add(computationalRecord);
      records.Add(aliveRecord);
```

```
      try
      {
        var writeRecordsRequest = new WriteRecordsRequest
        {
          DatabaseName = MultiMeasureValueConstants.MultiMeasureValueSampleDb,
          TableName = MultiMeasureValueConstants.MultiMeasureValueSampleTable,
          Records = records,
          CommonAttributes = commonAttributes
        };
        WriteRecordsResponse response = await
 writeClient.WriteRecordsAsync(writeRecordsRequest);
        Console.WriteLine($"Write records status code:
 {response.HttpStatusCode.ToString()}");
      }
      catch (Exception e)
      {
        Console.WriteLine("Write records failure:" + e.ToString());
      }
    }
  }
}
```

處理寫入失敗

Amazon Timestream 中的寫入可能會因下列一或多個原因失敗：

- 有些具有時間戳記的記錄位於記憶體存放區的保留期間之外。
- 有些記錄包含超過 Timestream 定義限制的維度和/或量值。
- Amazon Timestream 偵測到重複的記錄。當有多個具有相同維度、時間戳記和量值名稱的記錄，但：
  - 測量值不同。
  - 請求中沒有版本，或新記錄中的版本值等於或低於現有值。如果 Amazon Timestream 因此原因而拒絕資料， 中的 `ExistingVersion` 欄位`RejectedRecords`將包含 Amazon Timestream 中存放的記錄目前版本。若要強制更新，您可以將記錄集的 版本重新傳送請求至大於 的值`ExistingVersion`。

如需有關錯誤和拒絕記錄的詳細資訊，請參閱錯誤和 RejectedRecord。

如果您的應用程式在嘗試將記錄寫入 Timestream `RejectedRecordsException`時收到 ，您可以剖析被拒絕的記錄，以進一步了解寫入失敗，如下所示。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
 try {
   WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
   System.out.println("WriteRecords Status: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
 } catch (RejectedRecordsException e) {
   System.out.println("RejectedRecords: " + e);
   for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
     System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ": "
         + rejectedRecord.getReason());
   }
   System.out.println("Other records were written successfully. ");
 } catch (Exception e) {
   System.out.println("Error: " + e);
 }
```

Java v2

```
   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("writeRecordsWithCommonAttributes Status: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     System.out.println("RejectedRecords: " + e);
     for (RejectedRecord rejectedRecord : e.rejectedRecords()) {
       System.out.println("Rejected Index " + rejectedRecord.recordIndex() + ": "
           + rejectedRecord.reason());
     }
     System.out.println("Other records were written successfully. ");
   } catch (Exception e) {
```

```
      System.out.println("Error: " + e);
    }
```

Go

```
_, err = writeSvc.WriteRecords(writeRecordsInput)

if err != nil {
  fmt.Println("Error:")
  fmt.Println(err)
} else {
  fmt.Println("Write records is successful")
}
```

Python

```
try:
  result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME, Records=records, CommonAttributes=common_attributes)
  print("WriteRecords Status: [%s]" % result['ResponseMetadata']['HTTPStatusCode'])
except self.client.exceptions.RejectedRecordsException as err:
  print("RejectedRecords: ", err)
  for rr in err.response["RejectedRecords"]:
    print("Rejected Index " + str(rr["RecordIndex"]) + ": " + rr["Reason"])
  print("Other records were written successfully. ")
except Exception as err:
  print("Error:", err)
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
await request.promise().then(
    (data) => {
      console.log("Write records successful");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        const responsePayload =
 JSON.parse(request.response.httpResponse.body.toString());
```

```
        console.log("RejectedRecords: ", responsePayload.RejectedRecords);
        console.log("Other records were written successfully. ");
      }
    }
  );
```

.NET

```
 try
 {
   var writeRecordsRequest = new WriteRecordsRequest
   {
     DatabaseName = Constants.DATABASE_NAME,
     TableName = Constants.TABLE_NAME,
     Records = records,
     CommonAttributes = commonAttributes
   };
   WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
   Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
 }
 catch (RejectedRecordsException e) {
   Console.WriteLine("RejectedRecordsException:" + e.ToString());
   foreach (RejectedRecord rr in e.RejectedRecords) {
     Console.WriteLine("RecordIndex " + rr.RecordIndex + " : " + rr.Reason);
   }
   Console.WriteLine("Other records were written successfully. ");
 }
 catch (Exception e)
 {
   Console.WriteLine("Write records failure:" + e.ToString());
 }
```

執行查詢

主題

- 編碼結果
- 剖析結果集
- 存取查詢狀態

編碼結果

當您執行查詢時，Timestream 會以分頁方式傳回結果集，以最佳化應用程式的回應能力。下面的程式碼片段顯示如何瀏覽結果集。您必須循環瀏覽所有結果集頁面，直到您遇到 Null 值為止。分頁權杖會在 Timestream 為 發出 3 小時後過期 LiveAnalytics。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
   private void runQuery(String queryString) {
       try {
           QueryRequest queryRequest = new QueryRequest();
           queryRequest.setQueryString(queryString);
           QueryResult queryResult = queryClient.query(queryRequest);
           while (true) {
               parseQueryResult(queryResult);
               if (queryResult.getNextToken() == null) {
                   break;
               }
               queryRequest.setNextToken(queryResult.getNextToken());
               queryResult = queryClient.query(queryRequest);
           }
       } catch (Exception e) {
           // Some queries might fail with 500 if the result of a sequence function
has more than 10000 entries
           e.printStackTrace();
       }
   }
```

Java v2

```
   private void runQuery(String queryString) {
       try {
           QueryRequest queryRequest =
QueryRequest.builder().queryString(queryString).build();
           final QueryIterable queryResponseIterator =
timestreamQueryClient.queryPaginator(queryRequest);
```

```
            for(QueryResponse queryResponse : queryResponseIterator) {
                parseQueryResult(queryResponse);
            }
        } catch (Exception e) {
            // Some queries might fail with 500 if the result of a sequence function
 has more than 10000 entries
            e.printStackTrace();
        }
    }
```

Go

```
func runQuery(queryPtr *string, querySvc *timestreamquery.TimestreamQuery, f
 *os.File) {
    queryInput := &timestreamquery.QueryInput{
        QueryString: aws.String(*queryPtr),
    }
    fmt.Println("QueryInput:")
    fmt.Println(queryInput)
    // execute the query
    err := querySvc.QueryPages(queryInput,
        func(page *timestreamquery.QueryOutput, lastPage bool) bool {
            // process query response
            queryStatus := page.QueryStatus
            fmt.Println("Current query status:", queryStatus)
            // query response metadata
            // includes column names and types
            metadata := page.ColumnInfo
            // fmt.Println("Metadata:")
            fmt.Println(metadata)
            header := ""
            for i := 0; i < len(metadata); i++ {
                header += *metadata[i].Name
                if i != len(metadata)-1 {
                    header += ", "
                }
            }
            write(f, header)

            // query response data
            fmt.Println("Data:")
            // process rows
            rows := page.Rows
```

```
            for i := 0; i < len(rows); i++ {
                data := rows[i].Data
                value := processRowType(data, metadata)
                fmt.Println(value)
                write(f, value)
            }
            fmt.Println("Number of rows:", len(page.Rows))
            return true
        })
    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    }
}
```

Python

```
    def run_query(self, query_string):
        try:
            page_iterator = self.paginator.paginate(QueryString=query_string)
            for page in page_iterator:
                self._parse_query_result(page)
        except Exception as err:
            print("Exception while running query:", err)
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function getAllRows(query, nextToken) {
    const params = {
        QueryString: query
    };

    if (nextToken) {
        params.NextToken = nextToken;
    }

    await queryClient.query(params).promise()
        .then(
            (response) => {
                parseQueryResult(response);
```

```
                if (response.NextToken) {
                    getAllRows(query, response.NextToken);
                }
            },
            (err) => {
                console.error("Error while querying:", err);
            });
}
```

.NET

```
        private async Task RunQueryAsync(string queryString)
        {
            try
            {
                QueryRequest queryRequest = new QueryRequest();
                queryRequest.QueryString = queryString;
                QueryResponse queryResponse = await
 queryClient.QueryAsync(queryRequest);
                while (true)
                {
                    ParseQueryResult(queryResponse);
                    if (queryResponse.NextToken == null)
                    {
                        break;
                    }
                    queryRequest.NextToken = queryResponse.NextToken;
                    queryResponse = await queryClient.QueryAsync(queryRequest);
                }
            } catch(Exception e)
            {
                // Some queries might fail with 500 if the result of a sequence
 function has more than 10000 entries
                Console.WriteLine(e.ToString());
            }
        }
```

剖析結果集

您可以使用下列程式碼片段從結果集中擷取資料。查詢完成後，最多可以存取 24 小時的查詢結果。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
   private static final DateTimeFormatter TIMESTAMP_FORMATTER =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss.SSSSSSSSS");
   private static final DateTimeFormatter DATE_FORMATTER =
DateTimeFormatter.ofPattern("yyyy-MM-dd");
   private static final DateTimeFormatter TIME_FORMATTER =
DateTimeFormatter.ofPattern("HH:mm:ss.SSSSSSSSS");

   private static final long ONE_GB_IN_BYTES = 1073741824L;

   private void parseQueryResult(QueryResult response) {
       final QueryStatus currentStatusOfQuery = queryResult.getQueryStatus();

       System.out.println("Query progress so far: " +
currentStatusOfQuery.getProgressPercentage() + "%");

       double bytesScannedSoFar = ((double)
currentStatusOfQuery.getCumulativeBytesScanned() / ONE_GB_IN_BYTES);
       System.out.println("Bytes scanned so far: " + bytesScannedSoFar + " GB");

       double bytesMeteredSoFar = ((double)
currentStatusOfQuery.getCumulativeBytesMetered() / ONE_GB_IN_BYTES);
       System.out.println("Bytes metered so far: " + bytesMeteredSoFar + " GB");

       List<ColumnInfo> columnInfo = response.getColumnInfo();
       List<Row> rows = response.getRows();

       System.out.println("Metadata: " + columnInfo);
       System.out.println("Data: ");

       // iterate every row
       for (Row row : rows) {
           System.out.println(parseRow(columnInfo, row));
       }
   }
```

```
    private String parseRow(List<ColumnInfo> columnInfo, Row row) {
        List<Datum> data = row.getData();
        List<String> rowOutput = new ArrayList<>();
        // iterate every column per row
        for (int j = 0; j < data.size(); j++) {
            ColumnInfo info = columnInfo.get(j);
            Datum datum = data.get(j);
            rowOutput.add(parseDatum(info, datum));
        }
        return String.format("{%s}",
rowOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseDatum(ColumnInfo info, Datum datum) {
        if (datum.isNullValue() != null && datum.isNullValue()) {
            return info.getName() + "=" + "NULL";
        }
        Type columnType = info.getType();
        // If the column is of TimeSeries Type
        if (columnType.getTimeSeriesMeasureValueColumnInfo() != null) {
            return parseTimeSeries(info, datum);
        }
        // If the column is of Array Type
        else if (columnType.getArrayColumnInfo() != null) {
            List<Datum> arrayValues = datum.getArrayValue();
            return info.getName() + "=" +
parseArray(info.getType().getArrayColumnInfo(), arrayValues);
        }
        // If the column is of Row Type
        else if (columnType.getRowColumnInfo() != null) {
            List<ColumnInfo> rowColumnInfo = info.getType().getRowColumnInfo();
            Row rowValues = datum.getRowValue();
            return parseRow(rowColumnInfo, rowValues);
        }
        // If the column is of Scalar Type
        else {
            return parseScalarType(info, datum);
        }
    }

    private String parseTimeSeries(ColumnInfo info, Datum datum) {
        List<String> timeSeriesOutput = new ArrayList<>();
        for (TimeSeriesDataPoint dataPoint : datum.getTimeSeriesValue()) {
            timeSeriesOutput.add("{time=" + dataPoint.getTime() + ", value=" +
```

```
                    parseDatum(info.getType().getTimeSeriesMeasureValueColumnInfo(),
dataPoint.getValue()) + "}");
        }
        return String.format("[%s]",
timeSeriesOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseScalarType(ColumnInfo info, Datum datum) {
        switch (ScalarType.fromValue(info.getType().getScalarType())) {
            case VARCHAR:
                return parseColumnName(info) + datum.getScalarValue();
            case BIGINT:
                Long longValue = Long.valueOf(datum.getScalarValue());
                return parseColumnName(info) + longValue;
            case INTEGER:
                Integer intValue = Integer.valueOf(datum.getScalarValue());
                return parseColumnName(info) + intValue;
            case BOOLEAN:
                Boolean booleanValue = Boolean.valueOf(datum.getScalarValue());
                return parseColumnName(info) + booleanValue;
            case DOUBLE:
                Double doubleValue = Double.valueOf(datum.getScalarValue());
                return parseColumnName(info) + doubleValue;
            case TIMESTAMP:
                return parseColumnName(info) +
LocalDateTime.parse(datum.getScalarValue(), TIMESTAMP_FORMATTER);
            case DATE:
                return parseColumnName(info) +
LocalDate.parse(datum.getScalarValue(), DATE_FORMATTER);
            case TIME:
                return parseColumnName(info) +
LocalTime.parse(datum.getScalarValue(), TIME_FORMATTER);
            case INTERVAL_DAY_TO_SECOND:
            case INTERVAL_YEAR_TO_MONTH:
                return parseColumnName(info) + datum.getScalarValue();
            case UNKNOWN:
                return parseColumnName(info) + datum.getScalarValue();
            default:
                throw new IllegalArgumentException("Given type is not valid: " +
info.getType().getScalarType());
        }
    }

    private String parseColumnName(ColumnInfo info) {
```

```
        return info.getName() == null ? "" : info.getName() + "=";
    }

    private String parseArray(ColumnInfo arrayColumnInfo, List<Datum> arrayValues) {
        List<String> arrayOutput = new ArrayList<>();
        for (Datum datum : arrayValues) {
            arrayOutput.add(parseDatum(arrayColumnInfo, datum));
        }
        return String.format("[%s]",
arrayOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }
```

Java v2

```
    private static final long ONE_GB_IN_BYTES = 1073741824L;

    private void parseQueryResult(QueryResponse response) {
        final QueryStatus currentStatusOfQuery = response.queryStatus();

        System.out.println("Query progress so far: " +
currentStatusOfQuery.progressPercentage() + "%");

        double bytesScannedSoFar = ((double)
currentStatusOfQuery.cumulativeBytesScanned() / ONE_GB_IN_BYTES);
        System.out.println("Bytes scanned so far: " + bytesScannedSoFar + " GB");

        double bytesMeteredSoFar = ((double)
currentStatusOfQuery.cumulativeBytesMetered() / ONE_GB_IN_BYTES);
        System.out.println("Bytes metered so far: " + bytesMeteredSoFar + " GB");

        List<ColumnInfo> columnInfo = response.columnInfo();
        List<Row> rows = response.rows();

        System.out.println("Metadata: " + columnInfo);
        System.out.println("Data: ");

        // iterate every row
        for (Row row : rows) {
            System.out.println(parseRow(columnInfo, row));
        }
    }

    private String parseRow(List<ColumnInfo> columnInfo, Row row) {
```

```
        List<Datum> data = row.data();
        List<String> rowOutput = new ArrayList<>();
        // iterate every column per row
        for (int j = 0; j < data.size(); j++) {
            ColumnInfo info = columnInfo.get(j);
            Datum datum = data.get(j);
            rowOutput.add(parseDatum(info, datum));
        }
        return String.format("{%s}",
rowOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseDatum(ColumnInfo info, Datum datum) {
        if (datum.nullValue() != null && datum.nullValue()) {
            return info.name() + "=" + "NULL";
        }
        Type columnType = info.type();
        // If the column is of TimeSeries Type
        if (columnType.timeSeriesMeasureValueColumnInfo() != null) {
            return parseTimeSeries(info, datum);
        }
        // If the column is of Array Type
        else if (columnType.arrayColumnInfo() != null) {
            List<Datum> arrayValues = datum.arrayValue();
            return info.name() + "=" + parseArray(info.type().arrayColumnInfo(),
arrayValues);
        }
        // If the column is of Row Type
        else if (columnType.rowColumnInfo() != null &&
columnType.rowColumnInfo().size() > 0) {
            List<ColumnInfo> rowColumnInfo = info.type().rowColumnInfo();
            Row rowValues = datum.rowValue();
            return parseRow(rowColumnInfo, rowValues);
        }
        // If the column is of Scalar Type
        else {
            return parseScalarType(info, datum);
        }
    }

    private String parseTimeSeries(ColumnInfo info, Datum datum) {
        List<String> timeSeriesOutput = new ArrayList<>();
        for (TimeSeriesDataPoint dataPoint : datum.timeSeriesValue()) {
            timeSeriesOutput.add("{time=" + dataPoint.time() + ", value=" +
```

```
                    parseDatum(info.type().timeSeriesMeasureValueColumnInfo(),
 dataPoint.value()) + "}");
        }
        return String.format("[%s]",
 timeSeriesOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseScalarType(ColumnInfo info, Datum datum) {
        return parseColumnName(info) + datum.scalarValue();
    }

    private String parseColumnName(ColumnInfo info) {
        return info.name() == null ? "" : info.name() + "=";
    }

    private String parseArray(ColumnInfo arrayColumnInfo, List<Datum> arrayValues) {
        List<String> arrayOutput = new ArrayList<>();
        for (Datum datum : arrayValues) {
            arrayOutput.add(parseDatum(arrayColumnInfo, datum));
        }
        return String.format("[%s]",
 arrayOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }
```

Go

```
func processScalarType(data *timestreamquery.Datum) string {
    return *data.ScalarValue
}

func processTimeSeriesType(data []*timestreamquery.TimeSeriesDataPoint, columnInfo
 *timestreamquery.ColumnInfo) string {
    value := ""
    for k := 0; k < len(data); k++ {
        time := data[k].Time
        value += *time + ":"
        if columnInfo.Type.ScalarType != nil {
            value += processScalarType(data[k].Value)
        } else if columnInfo.Type.ArrayColumnInfo != nil {
            value += processArrayType(data[k].Value.ArrayValue,
 columnInfo.Type.ArrayColumnInfo)
        } else if columnInfo.Type.RowColumnInfo != nil {
```

```
            value += processRowType(data[k].Value.RowValue.Data,
 columnInfo.Type.RowColumnInfo)
        } else {
            fail("Bad data type")
        }
        if k != len(data)-1 {
            value += ", "
        }
    }
    return value
}

func processArrayType(datumList []*timestreamquery.Datum, columnInfo
 *timestreamquery.ColumnInfo) string {
    value := ""
    for k := 0; k < len(datumList); k++ {
        if columnInfo.Type.ScalarType != nil {
            value += processScalarType(datumList[k])
        } else if columnInfo.Type.TimeSeriesMeasureValueColumnInfo != nil {
            value += processTimeSeriesType(datumList[k].TimeSeriesValue,
 columnInfo.Type.TimeSeriesMeasureValueColumnInfo)
        } else if columnInfo.Type.ArrayColumnInfo != nil {
            value += "["
            value += processArrayType(datumList[k].ArrayValue,
 columnInfo.Type.ArrayColumnInfo)
            value += "]"
        } else if columnInfo.Type.RowColumnInfo != nil {
            value += "["
            value += processRowType(datumList[k].RowValue.Data,
 columnInfo.Type.RowColumnInfo)
            value += "]"
        } else {
            fail("Bad column type")
        }

        if k != len(datumList)-1 {
            value += ", "
        }
    }
    return value
}

func processRowType(data []*timestreamquery.Datum, metadata
 []*timestreamquery.ColumnInfo) string {
```

```
    value := ""
    for j := 0; j < len(data); j++ {
        if metadata[j].Type.ScalarType != nil {
            // process simple data types
            value += processScalarType(data[j])
        } else if metadata[j].Type.TimeSeriesMeasureValueColumnInfo != nil {
            // fmt.Println("Timeseries measure value column info")
            // fmt.Println(metadata[j].Type.TimeSeriesMeasureValueColumnInfo.Type)
            datapointList := data[j].TimeSeriesValue
            value += "["
            value += processTimeSeriesType(datapointList,
 metadata[j].Type.TimeSeriesMeasureValueColumnInfo)
            value += "]"
        } else if metadata[j].Type.ArrayColumnInfo != nil {
            columnInfo := metadata[j].Type.ArrayColumnInfo
            // fmt.Println("Array column info")
            // fmt.Println(columnInfo)
            datumList := data[j].ArrayValue
            value += "["
            value += processArrayType(datumList, columnInfo)
            value += "]"
        } else if metadata[j].Type.RowColumnInfo != nil {
            columnInfo := metadata[j].Type.RowColumnInfo
            datumList := data[j].RowValue.Data
            value += "["
            value += processRowType(datumList, columnInfo)
            value += "]"
        } else {
            panic("Bad column type")
        }
        // comma seperated column values
        if j != len(data)-1 {
            value += ", "
        }
    }
    return value
}
```

Python

```
    def _parse_query_result(self, query_result):
        query_status = query_result["QueryStatus"]
```

```
        progress_percentage = query_status["ProgressPercentage"]
        print(f"Query progress so far: {progress_percentage}%")

        bytes_scanned = float(query_status["CumulativeBytesScanned"]) /
 ONE_GB_IN_BYTES
        print(f"Data scanned so far: {bytes_scanned} GB")

        bytes_metered = float(query_status["CumulativeBytesMetered"]) /
 ONE_GB_IN_BYTES
        print(f"Data metered so far: {bytes_metered} GB")

        column_info = query_result['ColumnInfo']

        print("Metadata: %s" % column_info)
        print("Data: ")
        for row in query_result['Rows']:
            print(self._parse_row(column_info, row))

    def _parse_row(self, column_info, row):
        data = row['Data']
        row_output = []
        for j in range(len(data)):
            info = column_info[j]
            datum = data[j]
            row_output.append(self._parse_datum(info, datum))

        return "{%s}" % str(row_output)

    def _parse_datum(self, info, datum):
        if datum.get('NullValue', False):
            return "%s=NULL" % info['Name'],

        column_type = info['Type']

        # If the column is of TimeSeries Type
        if 'TimeSeriesMeasureValueColumnInfo' in column_type:
            return self._parse_time_series(info, datum)

        # If the column is of Array Type
        elif 'ArrayColumnInfo' in column_type:
            array_values = datum['ArrayValue']
            return "%s=%s" % (info['Name'], self._parse_array(info['Type']
['ArrayColumnInfo'], array_values))
```

```
        # If the column is of Row Type
        elif 'RowColumnInfo' in column_type:
            row_column_info = info['Type']['RowColumnInfo']
            row_values = datum['RowValue']
            return self._parse_row(row_column_info, row_values)

        # If the column is of Scalar Type
        else:
            return self._parse_column_name(info) + datum['ScalarValue']

    def _parse_time_series(self, info, datum):
        time_series_output = []
        for data_point in datum['TimeSeriesValue']:
            time_series_output.append("{time=%s, value=%s}"
                                      % (data_point['Time'],
                                         self._parse_datum(info['Type']
['TimeSeriesMeasureValueColumnInfo'],
                                                           data_point['Value'])))
        return "[%s]" % str(time_series_output)

    def _parse_array(self, array_column_info, array_values):
        array_output = []
        for datum in array_values:
            array_output.append(self._parse_datum(array_column_info, datum))

        return "[%s]" % str(array_output)

    @staticmethod
    def _parse_column_name(info):
        if 'Name' in info:
            return info['Name'] + "="
        else:
            return ""
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
function parseQueryResult(response) {
    const queryStatus = response.QueryStatus;
    console.log("Current query status: " + JSON.stringify(queryStatus));
```

```
    const columnInfo = response.ColumnInfo;
    const rows = response.Rows;

    console.log("Metadata: " + JSON.stringify(columnInfo));
    console.log("Data: ");

    rows.forEach(function (row) {
        console.log(parseRow(columnInfo, row));
    });
}

function parseRow(columnInfo, row) {
    const data = row.Data;
    const rowOutput = [];

    var i;
    for ( i = 0; i < data.length; i++ ) {
        info = columnInfo[i];
        datum = data[i];
        rowOutput.push(parseDatum(info, datum));
    }

    return `{${rowOutput.join(", ")}}`
}

function parseDatum(info, datum) {
    if (datum.NullValue != null && datum.NullValue === true) {
        return `${info.Name}=NULL`;
    }

    const columnType = info.Type;

    // If the column is of TimeSeries Type
    if (columnType.TimeSeriesMeasureValueColumnInfo != null) {
        return parseTimeSeries(info, datum);
    }
    // If the column is of Array Type
    else if (columnType.ArrayColumnInfo != null) {
        const arrayValues = datum.ArrayValue;
        return `${info.Name}=${parseArray(info.Type.ArrayColumnInfo, arrayValues)}`;
    }
    // If the column is of Row Type
    else if (columnType.RowColumnInfo != null) {
        const rowColumnInfo = info.Type.RowColumnInfo;
```

```
        const rowValues = datum.RowValue;
        return parseRow(rowColumnInfo, rowValues);
    }
    // If the column is of Scalar Type
    else {
        return parseScalarType(info, datum);
    }
}

function parseTimeSeries(info, datum) {
    const timeSeriesOutput = [];
    datum.TimeSeriesValue.forEach(function (dataPoint) {
        timeSeriesOutput.push(`{time=${dataPoint.Time}, value=
${parseDatum(info.Type.TimeSeriesMeasureValueColumnInfo, dataPoint.Value)}}`)
    });

    return `[${timeSeriesOutput.join(", ")}]`
}

function parseScalarType(info, datum) {
    return parseColumnName(info) + datum.ScalarValue;
}

function parseColumnName(info) {
    return info.Name == null ? "" : `${info.Name}=`;
}

function parseArray(arrayColumnInfo, arrayValues) {
    const arrayOutput = [];
    arrayValues.forEach(function (datum) {
        arrayOutput.push(parseDatum(arrayColumnInfo, datum));
    });
    return `[${arrayOutput.join(", ")}]`
}
```

.NET

```
        private void ParseQueryResult(QueryResponse response)
        {
            List<ColumnInfo> columnInfo = response.ColumnInfo;
            var options = new JsonSerializerOptions
            {
                IgnoreNullValues = true
```

```
            };
            List<String> columnInfoStrings = columnInfo.ConvertAll(x =>
JsonSerializer.Serialize(x, options));
            List<Row> rows = response.Rows;

            QueryStatus queryStatus = response.QueryStatus;
            Console.WriteLine("Current Query status:" +
JsonSerializer.Serialize(queryStatus, options));

            Console.WriteLine("Metadata:" + string.Join(",", columnInfoStrings));
            Console.WriteLine("Data:");

            foreach (Row row in rows)
            {
                Console.WriteLine(ParseRow(columnInfo, row));
            }
        }

        private string ParseRow(List<ColumnInfo> columnInfo, Row row)
        {
            List<Datum> data = row.Data;
            List<string> rowOutput = new List<string>();
            for (int j = 0; j < data.Count; j++)
            {
                ColumnInfo info = columnInfo[j];
                Datum datum = data[j];
                rowOutput.Add(ParseDatum(info, datum));
            }
            return $"{{{string.Join(",", rowOutput)}}}";
        }

        private string ParseDatum(ColumnInfo info, Datum datum)
        {
            if (datum.NullValue)
            {
                return $"{info.Name}=NULL";
            }

            Amazon.TimestreamQuery.Model.Type columnType = info.Type;
            if (columnType.TimeSeriesMeasureValueColumnInfo != null)
            {
                return ParseTimeSeries(info, datum);
            }
            else if (columnType.ArrayColumnInfo != null)
```

```
            {
                List<Datum> arrayValues = datum.ArrayValue;
                return $"{info.Name}={ParseArray(info.Type.ArrayColumnInfo,
arrayValues)}";
            }
            else if (columnType.RowColumnInfo != null &&
columnType.RowColumnInfo.Count > 0)
            {
                List<ColumnInfo> rowColumnInfo = info.Type.RowColumnInfo;
                Row rowValue = datum.RowValue;
                return ParseRow(rowColumnInfo, rowValue);
            }
            else
            {
                return ParseScalarType(info, datum);
            }
        }

        private string ParseTimeSeries(ColumnInfo info, Datum datum)
        {
            var timeseriesString = datum.TimeSeriesValue
                .Select(value => $"{{time={value.Time},
value={ParseDatum(info.Type.TimeSeriesMeasureValueColumnInfo, value.Value)}}}")
                .Aggregate((current, next) => current + "," + next);

            return $"[{timeseriesString}]";
        }

        private string ParseScalarType(ColumnInfo info, Datum datum)
        {
            return ParseColumnName(info) + datum.ScalarValue;
        }

        private string ParseColumnName(ColumnInfo info)
        {
            return info.Name == null ? "" : (info.Name + "=");
        }

        private string ParseArray(ColumnInfo arrayColumnInfo, List<Datum>
arrayValues)
        {
            return $"[{arrayValues.Select(value => ParseDatum(arrayColumnInfo,
value)).Aggregate((current, next) => current + "," + next)}]";
```

```
        }
```

存取查詢狀態

您可以透過 存取查詢狀態QueryResponse，其中包含查詢進度、查詢掃描的位元組，以及查詢計量的位元組等相關資訊。bytesMetered 和 bytesScanned值會累積並持續更新，同時分頁查詢結果。您可以使用此資訊來了解個別查詢掃描的位元組，也可以使用這些資訊做出特定決策。例如，假設掃描的查詢價格為每 GB 0.01 美元，您可能想要取消每次查詢超過 25 美元的查詢，或 X GB。下面的程式碼片段顯示如何完成此操作。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
   private static final long ONE_GB_IN_BYTES = 1073741824L;
   private static final double QUERY_COST_PER_GB_IN_DOLLARS = 0.01; // Assuming the
price of query is $0.01 per GB

   public void cancelQueryBasedOnQueryStatus() {
       System.out.println("Starting query: " + SELECT_ALL_QUERY);
       QueryRequest queryRequest = new QueryRequest();
       queryRequest.setQueryString(SELECT_ALL_QUERY);
       QueryResult queryResult = queryClient.query(queryRequest);

       while (true) {
           final QueryStatus currentStatusOfQuery = queryResult.getQueryStatus();
           System.out.println("Query progress so far: " +
currentStatusOfQuery.getProgressPercentage() + "%");
           double bytesMeteredSoFar = ((double)
currentStatusOfQuery.getCumulativeBytesMetered() / ONE_GB_IN_BYTES);
           System.out.println("Bytes metered so far: " + bytesMeteredSoFar + "
GB");
           // Cancel query if its costing more than 1 cent
           if (bytesMeteredSoFar * QUERY_COST_PER_GB_IN_DOLLARS > 0.01) {
               cancelQuery(queryResult);
               break;
           }
```

```
            if (queryResult.getNextToken() == null) {
                break;
            }
            queryRequest.setNextToken(queryResult.getNextToken());
            queryResult = queryClient.query(queryRequest);
        }
    }
```

Java v2

```
    private static final long ONE_GB_IN_BYTES = 1073741824L;
    private static final double QUERY_COST_PER_GB_IN_DOLLARS = 0.01; // Assuming the
price of query is $0.01 per GB

    public void cancelQueryBasedOnQueryStatus() {
        System.out.println("Starting query: " + SELECT_ALL_QUERY);
        QueryRequest queryRequest =
QueryRequest.builder().queryString(SELECT_ALL_QUERY).build();

        final QueryIterable queryResponseIterator =
timestreamQueryClient.queryPaginator(queryRequest);
        for(QueryResponse queryResponse : queryResponseIterator) {
            final QueryStatus currentStatusOfQuery = queryResponse.queryStatus();
            System.out.println("Query progress so far: " +
currentStatusOfQuery.progressPercentage() + "%");
            double bytesMeteredSoFar = ((double)
currentStatusOfQuery.cumulativeBytesMetered() / ONE_GB_IN_BYTES);
            System.out.println("Bytes metered so far: " + bytesMeteredSoFar + "GB");
            // Cancel query if its costing more than 1 cent
            if (bytesMeteredSoFar * QUERY_COST_PER_GB_IN_DOLLARS > 0.01) {
                cancelQuery(queryResponse);
                break;
            }
        }
    }
```

Go

```
const OneGbInBytes = 1073741824
// Assuming the price of query is $0.01 per GB
const QueryCostPerGbInDollars = 0.01
```

```
func cancelQueryBasedOnQueryStatus(queryPtr *string, querySvc
 *timestreamquery.TimestreamQuery, f *os.File) {
    queryInput := &timestreamquery.QueryInput{
        QueryString: aws.String(*queryPtr),
    }
    fmt.Println("QueryInput:")
    fmt.Println(queryInput)
    // execute the query
    err := querySvc.QueryPages(queryInput,
        func(page *timestreamquery.QueryOutput, lastPage bool) bool {
            // process query response
            queryStatus := page.QueryStatus
            fmt.Println("Current query status:", queryStatus)
            bytes_metered := float64(*queryStatus.CumulativeBytesMetered) /
 float64(ONE_GB_IN_BYTES)
            if bytes_metered * QUERY_COST_PER_GB_IN_DOLLARS > 0.01 {
                cancelQuery(page, querySvc)
                return true
            }
            // query response metadata
            // includes column names and types
            metadata := page.ColumnInfo
            // fmt.Println("Metadata:")
            fmt.Println(metadata)
            header := ""
            for i := 0; i < len(metadata); i++ {
                header += *metadata[i].Name
                if i != len(metadata)-1 {
                    header += ", "
                }
            }
            write(f, header)

            // query response data
            fmt.Println("Data:")
            // process rows
            rows := page.Rows
            for i := 0; i < len(rows); i++ {
                data := rows[i].Data
                value := processRowType(data, metadata)
                fmt.Println(value)
                write(f, value)
            }
            fmt.Println("Number of rows:", len(page.Rows))
```

```
            return true
        })
    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    }
}
```

Python

```
ONE_GB_IN_BYTES = 1073741824
# Assuming the price of query is $0.01 per GB
QUERY_COST_PER_GB_IN_DOLLARS = 0.01

    def cancel_query_based_on_query_status(self):
        try:
            print("Starting query: " + self.SELECT_ALL)
            page_iterator = self.paginator.paginate(QueryString=self.SELECT_ALL)
            for page in page_iterator:
                query_status = page["QueryStatus"]
                progress_percentage = query_status["ProgressPercentage"]
                print("Query progress so far: " + str(progress_percentage) + "%")
                bytes_metered = query_status["CumulativeBytesMetered"] /
 self.ONE_GB_IN_BYTES
                print("Bytes Metered so far: " + str(bytes_metered) + " GB")
                if bytes_metered * self.QUERY_COST_PER_GB_IN_DOLLARS > 0.01:
                    self.cancel_query_for(page)
                    break
        except Exception as err:
            print("Exception while running query:", err)
            traceback.print_exc(file=sys.stderr)
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
function parseQueryResult(response) {
    const queryStatus = response.QueryStatus;
    console.log("Current query status: " + JSON.stringify(queryStatus));

    const columnInfo = response.ColumnInfo;
    const rows = response.Rows;
```

```
    console.log("Metadata: " + JSON.stringify(columnInfo));
    console.log("Data: ");

    rows.forEach(function (row) {
        console.log(parseRow(columnInfo, row));
    });
}

function parseRow(columnInfo, row) {
    const data = row.Data;
    const rowOutput = [];

    var i;
    for ( i = 0; i < data.length; i++ ) {
        info = columnInfo[i];
        datum = data[i];
        rowOutput.push(parseDatum(info, datum));
    }

    return `{${rowOutput.join(", ")}}`
}

function parseDatum(info, datum) {
    if (datum.NullValue != null && datum.NullValue === true) {
        return `${info.Name}=NULL`;
    }

    const columnType = info.Type;

    // If the column is of TimeSeries Type
    if (columnType.TimeSeriesMeasureValueColumnInfo != null) {
        return parseTimeSeries(info, datum);
    }
    // If the column is of Array Type
    else if (columnType.ArrayColumnInfo != null) {
        const arrayValues = datum.ArrayValue;
        return `${info.Name}=${parseArray(info.Type.ArrayColumnInfo, arrayValues)}`;
    }
    // If the column is of Row Type
    else if (columnType.RowColumnInfo != null) {
        const rowColumnInfo = info.Type.RowColumnInfo;
        const rowValues = datum.RowValue;
        return parseRow(rowColumnInfo, rowValues);
```

```
    }
    // If the column is of Scalar Type
    else {
        return parseScalarType(info, datum);
    }
}

function parseTimeSeries(info, datum) {
    const timeSeriesOutput = [];
    datum.TimeSeriesValue.forEach(function (dataPoint) {
        timeSeriesOutput.push(`{time=${dataPoint.Time}, value=
${parseDatum(info.Type.TimeSeriesMeasureValueColumnInfo, dataPoint.Value)}}`)
    });

    return `[${timeSeriesOutput.join(", ")}]`
}

function parseScalarType(info, datum) {
    return parseColumnName(info) + datum.ScalarValue;
}

function parseColumnName(info) {
    return info.Name == null ? "" : `${info.Name}=`;
}

function parseArray(arrayColumnInfo, arrayValues) {
    const arrayOutput = [];
    arrayValues.forEach(function (datum) {
        arrayOutput.push(parseDatum(arrayColumnInfo, datum));
    });
    return `[${arrayOutput.join(", ")}]`
}
```

.NET

```
private static readonly long ONE_GB_IN_BYTES = 1073741824L;
private static readonly double QUERY_COST_PER_GB_IN_DOLLARS = 0.01; // Assuming the
 price of query is $0.01 per GB

private async Task CancelQueryBasedOnQueryStatus(string queryString)
{
    try
    {
```

```
        QueryRequest queryRequest = new QueryRequest();
        queryRequest.QueryString = queryString;
        QueryResponse queryResponse = await queryClient.QueryAsync(queryRequest);
        while (true)
        {
            QueryStatus queryStatus = queryResponse.QueryStatus;
            double bytesMeteredSoFar = ((double)
 queryStatus.CumulativeBytesMetered / ONE_GB_IN_BYTES);
            // Cancel query if its costing more than 1 cent
            if (bytesMeteredSoFar * QUERY_COST_PER_GB_IN_DOLLARS > 0.01)
            {
                await CancelQuery(queryResponse);
                break;
            }

            ParseQueryResult(queryResponse);
            if (queryResponse.NextToken == null)
            {
                break;
            }
            queryRequest.NextToken = queryResponse.NextToken;
            queryResponse = await queryClient.QueryAsync(queryRequest);
       }
    } catch(Exception e)
    {
        // Some queries might fail with 500 if the result of a sequence function has
 more than 10000 entries
        Console.WriteLine(e.ToString());
    }
}
```

如需有關如何取消查詢的其他詳細資訊，請參閱 取消查詢。

執行UNLOAD查詢

下列程式碼範例會呼叫UNLOAD查詢。如需 UNLOAD 的資訊，請參閱「使用 UNLOAD 將查詢結果從 Timestream 匯出至 S3 LiveAnalytics」。如需UNLOAD查詢的範例，請參閱 UNLOAD 從 Timestream 的 使用案例範例 LiveAnalytics。

主題

- 建置和執行UNLOAD查詢

- 剖析UNLOAD回應，並取得資料列計數、資訊清單連結和中繼資料連結
- 讀取和剖析資訊清單內容
- 讀取和剖析中繼資料內容

建置和執行UNLOAD查詢

Java

```
// When you have a SELECT like below

String QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time,
 query, quantity, product_id, channel FROM "
        + DATABASE_NAME + "." + UNLOAD_TABLE_NAME
        + " WHERE time BETWEEN ago(2d) AND now()";

// You can construct UNLOAD query as follows
UnloadQuery unloadQuery = UnloadQuery.builder()
        .selectQuery(QUERY_1)
        .bucketName("timestream-sample-<region>-<accountId>")
        .resultsPrefix("without_partition")
        .format(CSV)
        .compression(UnloadQuery.Compression.GZIP)
        .build();
QueryResult unloadResult = runQuery(unloadQuery.getUnloadQuery());

// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination
    private QueryResult runQuery(String queryString) {
        QueryResult queryResult = null;
        try {
            QueryRequest queryRequest = new QueryRequest();
            queryRequest.setQueryString(queryString);
            queryResult = queryClient.query(queryRequest);
            while (true) {
                parseQueryResult(queryResult);
                if (queryResult.getNextToken() == null) {
                    break;
                }
                queryRequest.setNextToken(queryResult.getNextToken());
                queryResult = queryClient.query(queryRequest);
            }
```

```
        } catch (Exception e) {
            // Some queries might fail with 500 if the result of a sequence function
 has more than 10000 entries
            e.printStackTrace();
        }
        return queryResult;
    }

// Utility that helps to construct UNLOAD query

@Builder
static class UnloadQuery {
    private String selectQuery;
    private String bucketName;
    private String resultsPrefix;
    private Format format;
    private Compression compression;
    private EncryptionType encryptionType;
    private List<String> partitionColumns;
    private String kmsKey;
    private Character csvFieldDelimiter;
    private Character csvEscapeCharacter;

    public String getUnloadQuery() {
        String destination = constructDestination();
        String withClause = constructOptionalParameters();
        return String.format("UNLOAD (%s) TO '%s' %s", selectQuery, destination,
 withClause);
    }

    private String constructDestination() {
        return "s3://" + this.bucketName + "/" + this.resultsPrefix + "/";
    }

    private String constructOptionalParameters() {
        boolean isOptionalParametersPresent = Objects.nonNull(format)
                || Objects.nonNull(compression)
                || Objects.nonNull(encryptionType)
                || Objects.nonNull(partitionColumns)
                || Objects.nonNull(kmsKey)
                || Objects.nonNull(csvFieldDelimiter)
                || Objects.nonNull(csvEscapeCharacter);

        String withClause = "";
```

```
        if (isOptionalParametersPresent) {
            StringJoiner optionalParameters = new StringJoiner(",");
            if (Objects.nonNull(format)) {
                optionalParameters.add("format = '" + format + "'");
            }
            if (Objects.nonNull(compression)) {
                optionalParameters.add("compression = '" + compression + "'");
            }
            if (Objects.nonNull(encryptionType)) {
                optionalParameters.add("encryption = '" + encryptionType + "'");
            }
            if (Objects.nonNull(kmsKey)) {
                optionalParameters.add("kms_key = '" + kmsKey + "'");
            }
            if (Objects.nonNull(csvFieldDelimiter)) {
                optionalParameters.add("field_delimiter = '" + csvFieldDelimiter +
"'");
            }
            if (Objects.nonNull(csvEscapeCharacter)) {
                optionalParameters.add("escaped_by = '" + csvEscapeCharacter + "'");
            }
            if (Objects.nonNull(partitionColumns) && !partitionColumns.isEmpty()) {
                final StringJoiner partitionedByList = new StringJoiner(",");
                partitionColumns.forEach(column -> partitionedByList.add("'" +
column + "'"));
                optionalParameters.add(String.format("partitioned_by = ARRAY[%s]",
partitionedByList));
            }
            withClause = String.format("WITH (%s)", optionalParameters);
        }
        return withClause;
    }

    public enum Format {
        CSV, PARQUET
    }

    public enum Compression {
        GZIP, NONE
    }

    public enum EncryptionType {
        SSE_S3, SSE_KMS
    }
```

```
    @Override
    public String toString() {
        return getUnloadQuery();
    }
}
```

Java v2

```
// When you have a SELECT like below

String QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time,
 query, quantity, product_id, channel FROM "
        + DATABASE_NAME + "." + UNLOAD_TABLE_NAME
        + " WHERE time BETWEEN ago(2d) AND now()";

//You can construct UNLOAD query as follows
UnloadQuery unloadQuery = UnloadQuery.builder()
        .selectQuery(QUERY_1)
        .bucketName("timestream-sample-<region>-<accountId>")
        .resultsPrefix("without_partition")
        .format(CSV)
        .compression(UnloadQuery.Compression.GZIP)
        .build();

QueryResponse unloadResponse = runQuery(unloadQuery.getUnloadQuery());


// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination
private QueryResponse runQuery(String queryString) {
   QueryResponse finalResponse = null;
    try {
        QueryRequest queryRequest =
 QueryRequest.builder().queryString(queryString).build();
        final QueryIterable queryResponseIterator =
 timestreamQueryClient.queryPaginator(queryRequest);
        for(QueryResponse queryResponse : queryResponseIterator) {
            parseQueryResult(queryResponse);
           finalResponse = queryResponse;
        }
    } catch (Exception e) {
```

```
        // Some queries might fail with 500 if the result of a sequence function has
 more than 10000 entries
        e.printStackTrace();
    }
   return finalResponse;
}

// Utility that helps to construct UNLOAD query
@Builder
static class UnloadQuery {
    private String selectQuery;
    private String bucketName;
    private String resultsPrefix;
    private Format format;
    private Compression compression;
    private EncryptionType encryptionType;
    private List<String> partitionColumns;
    private String kmsKey;
    private Character csvFieldDelimiter;
    private Character csvEscapeCharacter;

    public String getUnloadQuery() {
        String destination = constructDestination();
        String withClause = constructOptionalParameters();
        return String.format("UNLOAD (%s) TO '%s' %s", selectQuery, destination,
 withClause);
    }

    private String constructDestination() {
        return "s3://" + this.bucketName + "/" + this.resultsPrefix + "/";
    }

    private String constructOptionalParameters() {
        boolean isOptionalParametersPresent = Objects.nonNull(format)
                || Objects.nonNull(compression)
                || Objects.nonNull(encryptionType)
                || Objects.nonNull(partitionColumns)
                || Objects.nonNull(kmsKey)
                || Objects.nonNull(csvFieldDelimiter)
                || Objects.nonNull(csvEscapeCharacter);

        String withClause = "";
        if (isOptionalParametersPresent) {
            StringJoiner optionalParameters = new StringJoiner(",");
```

```
            if (Objects.nonNull(format)) {
                optionalParameters.add("format = '" + format + "'");
            }
            if (Objects.nonNull(compression)) {
                optionalParameters.add("compression = '" + compression + "'");
            }
            if (Objects.nonNull(encryptionType)) {
                optionalParameters.add("encryption = '" + encryptionType + "'");
            }
            if (Objects.nonNull(kmsKey)) {
                optionalParameters.add("kms_key = '" + kmsKey + "'");
            }
            if (Objects.nonNull(csvFieldDelimiter)) {
                optionalParameters.add("field_delimiter = '" + csvFieldDelimiter +
"'");
            }
            if (Objects.nonNull(csvEscapeCharacter)) {
                optionalParameters.add("escaped_by = '" + csvEscapeCharacter + "'");
            }
            if (Objects.nonNull(partitionColumns) && !partitionColumns.isEmpty()) {
                final StringJoiner partitionedByList = new StringJoiner(",");
                partitionColumns.forEach(column -> partitionedByList.add("'" +
column + "'"));
                optionalParameters.add(String.format("partitioned_by = ARRAY[%s]",
partitionedByList));
            }
            withClause = String.format("WITH (%s)", optionalParameters);
        }
        return withClause;
    }

    public enum Format {
        CSV, PARQUET
    }

    public enum Compression {
        GZIP, NONE
    }

    public enum EncryptionType {
        SSE_S3, SSE_KMS
    }

    @Override
```

```
    public String toString() {
        return getUnloadQuery();
    }
}
```

Go

```
// When you have a SELECT like below
var Query = "SELECT user_id, ip_address, event, session_id, measure_name, time,
 query, quantity, product_id, channel FROM "
+ *databaseName + "." + *tableName + " WHERE time BETWEEN ago(2d) AND now()"

// You can construct UNLOAD query as follows
var unloadQuery = UnloadQuery{
    Query: "SELECT user_id, ip_address, session_id, measure_name, time, query,
 quantity, product_id, channel, event FROM " + *databaseName + "." + *tableName +
    " WHERE time BETWEEN ago(2d) AND now()",
    Partitioned_by: []string{},
    Compression: "GZIP",
    Format: "CSV",
    S3Location: bucketName,
    ResultPrefix: "without_partition",
}


// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination

queryInput := &timestreamquery.QueryInput{
    QueryString: build_query(unloadQuery),
}

err := querySvc.QueryPages(queryInput,
    func(page *timestreamquery.QueryOutput, lastPage bool) bool {
        if (lastPage) {
            var response = parseQueryResult(page)
            var unloadFiles = getManifestAndMetadataFiles(s3Svc, response)
            displayColumns(unloadFiles, unloadQuery.Partitioned_by)
            displayResults(s3Svc, unloadFiles)
        }
        return true
    })
```

```
if err != nil {
    fmt.Println("Error:")
    fmt.Println(err)
}

// Utility that helps to construct UNLOAD query
type UnloadQuery struct {
    Query string
    Partitioned_by []string
    Format string
    S3Location string
    ResultPrefix string
    Compression string
}

func build_query(unload_query UnloadQuery) *string {
    var query_results_s3_path = "'s3://" + unload_query.S3Location + "/" +
 unload_query.ResultPrefix + "/'"
    var query = "UNLOAD(" + unload_query.Query + ") TO " + query_results_s3_path + "
 WITH ( "
    if (len(unload_query.Partitioned_by) > 0) {
        query = query + "partitioned_by=ARRAY["
        for i, column := range unload_query.Partitioned_by {
            if i == 0 {
                query = query + "'" + column + "'"
            } else {
                query = query + ",'" + column + "'"
            }
        }
        query = query + "],"
    }
    query = query + " format='" + unload_query.Format + "', "
    query = query + "  compression='" + unload_query.Compression + "')"
    fmt.Println(query)
    return aws.String(query)
}
```

Python

```
# When you have a SELECT like below
QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time, query,
 quantity, product_id, channel FROM "
```

```
        + database_name + "." + table_name + " WHERE time BETWEEN ago(2d) AND now()"
# You can construct UNLOAD query as follows
UNLOAD_QUERY_1 = UnloadQuery(QUERY_1, "timestream-sample-<region>-<accountId>",
 "without_partition", "CSV", "GZIP", "")

# Run UNLOAD query (Similar to how you run SELECT query)
# https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination
def run_query(self, query_string):
    try:
        page_iterator = self.paginator.paginate(QueryString=UNLOAD_QUERY_1)
    except Exception as err:
        print("Exception while running query:", err)

# Utility that helps to construct UNLOAD query
class UnloadQuery:
    def __init__(self, query, s3_bucket_location, results_prefix, format,
 compression , partition_by):
        self.query = query
        self.s3_bucket_location = s3_bucket_location
        self.results_prefix = results_prefix
        self.format = format
        self.compression = compression
        self.partition_by = partition_by

    def build_query(self):
        query_results_s3_path = "'s3://" + self.s3_bucket_location + "/" +
 self.results_prefix + "/'"
        unload_query = "UNLOAD("
        unload_query = unload_query + self.query
        unload_query = unload_query + ") "
        unload_query = unload_query + " TO " + query_results_s3_path
        unload_query = unload_query + " WITH ( "

        if(len(self.partition_by) > 0) :
            unload_query = unload_query + " partitioned_by = ARRAY" +
 str(self.partition_by) + ","

        unload_query = unload_query + " format='" + self.format  + "', "
        unload_query = unload_query + "  compression='" + self.compression + "')"

        return unload_query
```

Node.js

```
// When you have a SELECT like below
QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time, query,
 quantity, product_id, channel FROM "
        + database_name + "." + table_name + " WHERE time BETWEEN ago(2d) AND now()"
// You can construct UNLOAD query as follows
UNLOAD_QUERY_1 = new UnloadQuery(QUERY_1, "timestream-sample-<region>-<accountId>",
 "without_partition", "CSV", "GZIP", "")


// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination

async runQuery(query = UNLOAD_QUERY_1, nextToken) {
    const params = new QueryCommand({
        QueryString: query
    });

    if (nextToken) {
        params.NextToken = nextToken;
    }

    await queryClient.send(params).then(
            (response) => {
                if (response.NextToken) {
                    runQuery(queryClient, query, response.NextToken);
                } else {
                    await parseAndDisplayResults(response);
                }
            },
            (err) => {
                console.error("Error while querying:", err);
            });
}


class UnloadQuery {
    constructor(query, s3_bucket_location, results_prefix, format, compression ,
 partition_by) {
        this.query = query;
        this.s3_bucket_location = s3_bucket_location
        this.results_prefix = results_prefix
```

```
        this.format = format
        this.compression = compression
        this.partition_by = partition_by
    }

    buildQuery() {
        const query_results_s3_path = "'s3://" + this.s3_bucket_location + "/" +
 this.results_prefix + "/'"
        let unload_query = "UNLOAD("
        unload_query = unload_query + this.query
        unload_query = unload_query + ") "
        unload_query = unload_query + " TO " + query_results_s3_path
        unload_query = unload_query + " WITH ( "

        if(this.partition_by.length > 0) {
            let partitionBy = ""
            this.partition_by.forEach((str, i) => {
                partitionBy = partitionBy + (i ? ",'" : "'") + str + "'"
            })
            unload_query = unload_query + " partitioned_by = ARRAY[" + partitionBy +
 "],"
        }
        unload_query = unload_query + " format='" + this.format  + "', "
        unload_query = unload_query + "  compression='" + this.compression + "')"

        return unload_query
    }
}
```

剖析UNLOAD回應，並取得資料列計數、資訊清單連結和中繼資料連結

Java

```
// Parsing UNLOAD query response is similar to how you parse SELECT query response:
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing

// But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
// (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

public UnloadResponse parseResult(QueryResult queryResult) {
    Map<String, String> outputMap = new HashMap<>();
```

```
    for (int i = 0; i < queryResult.getColumnInfo().size(); i++) {
        outputMap.put(queryResult.getColumnInfo().get(i).getName(),
                queryResult.getRows().get(0).getData().get(i).getScalarValue());

    }
    return new UnloadResponse(outputMap);
}

@Getter
class UnloadResponse {
    private final String metadataFile;
    private final String manifestFile;
    private final int rows;

    public UnloadResponse(Map<String, String> unloadResponse) {
        this.metadataFile = unloadResponse.get("metadataFile");
        this.manifestFile = unloadResponse.get("manifestFile");
        this.rows = Integer.parseInt(unloadResponse.get("rows"));
    }
}
```

Java v2

```
// Parsing UNLOAD query response is similar to how you parse SELECT query response:
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing

// But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
// (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

public UnloadResponse parseResult(QueryResponse queryResponse) {
    Map<String, String> outputMap = new HashMap<>();
    for (int i = 0; i < queryResponse.columnInfo().size(); i++) {
        outputMap.put(queryResponse.columnInfo().get(i).name(),
                queryResponse.rows().get(0).data().get(i).scalarValue());

    }
    return new UnloadResponse(outputMap);
}

@Getter
class UnloadResponse {
    private final String metadataFile;
```

```
    private final String manifestFile;
    private final int rows;

    public UnloadResponse(Map<String, String> unloadResponse) {
        this.metadataFile = unloadResponse.get("metadataFile");
        this.manifestFile = unloadResponse.get("manifestFile");
        this.rows = Integer.parseInt(unloadResponse.get("rows"));
    }
}
```

Go

```
// Parsing UNLOAD query response is similar to how you parse SELECT query response:
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing

// But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
// (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

func parseQueryResult(queryOutput *timestreamquery.QueryOutput) map[string]string {
    var columnInfo = queryOutput.ColumnInfo;
    fmt.Println("ColumnInfo", columnInfo)
    fmt.Println("QueryId", queryOutput.QueryId)
    fmt.Println("QueryStatus", queryOutput.QueryStatus)
    return parseResponse(columnInfo, queryOutput.Rows[0])
}

func parseResponse(columnInfo []*timestreamquery.ColumnInfo, row
 *timestreamquery.Row) map[string]string {
    var datum = row.Data
    response := make(map[string]string)
    for i, column := range columnInfo {
        response[*column.Name] = *datum[i].ScalarValue
    }
    return response
}
```

Python

```
# Parsing UNLOAD query response is similar to how you parse SELECT query response:
# https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing
```

```
# But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
# (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

for page in page_iterator:
  last_page = page
response = self._parse_unload_query_result(last_page)

def _parse_unload_query_result(self, query_result):
    column_info = query_result['ColumnInfo']

    print("ColumnInfo: %s" % column_info)
    print("QueryId: %s" % query_result['QueryId'])
    print("QueryStatus:%s" % query_result['QueryStatus'])
    return self.parse_unload_response(column_info, query_result['Rows'][0])

def parse_unload_response(self, column_info, row):
    response = {}
    data = row['Data']
    for i, column in enumerate(column_info):
        response[column['Name']] = data[i]['ScalarValue']
   print("Rows: %s" % response['rows'])
   print("Metadata File location: %s" % response['metadataFile'])
   print("Manifest File location: %s" % response['manifestFile'])
   return response
```

Node.js

```
# Parsing UNLOAD query response is similar to how you parse SELECT query response:
# https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing

# But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
# (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

async parseAndDisplayResults(data, query) {
    const columnInfo = data['ColumnInfo'];
    console.log("ColumnInfo:", columnInfo)
    console.log("QueryId: %s", data['QueryId'])
    console.log("QueryStatus:", data['QueryStatus'])
    await this.parseResponse(columnInfo, data['Rows'][0], query)
}

async parseResponse(columnInfo, row, query) {
```

```
    let response = {}
    const data = row['Data']
    columnInfo.forEach((column, i) => {
        response[column['Name']] = data[i]['ScalarValue']
    })

   console.log("Manifest file", response['manifestFile']);
   console.log("Metadata file", response['metadataFile']);

   return response
}
```

讀取和剖析資訊清單內容

Java

```
// Read and parse manifest content
public UnloadManifest getUnloadManifest(UnloadResponse unloadResponse) throws
 IOException {
   AmazonS3URI s3URI = new AmazonS3URI(unloadResponse.getManifestFile());
    S3Object s3Object = s3Client.getObject(s3URI.getBucket(), s3URI.getKey());
    String manifestFileContent = new
 String(IOUtils.toByteArray(s3Object.getObjectContent()), StandardCharsets.UTF_8);
    return new Gson().fromJson(manifestFileContent, UnloadManifest.class);
}

class UnloadManifest {
    @Getter
    public class FileMetadata {
        long content_length_in_bytes;
        long row_count;
    }

    @Getter
    public class ResultFile {
        String url;
        FileMetadata file_metadata;
    }

    @Getter
    public class QueryMetadata {
        long total_content_length_in_bytes;
```

```
        long total_row_count;
        String result_format;
        String result_version;
    }

    @Getter
    public class Author {
        String name;
        String manifest_file_version;
    }

    @Getter
    private List<ResultFile> result_files;
    @Getter
    private QueryMetadata query_metadata;
    @Getter
    private Author author;
}
```

Java v2

```
// Read and parse manifest content
public UnloadManifest getUnloadManifest(UnloadResponse unloadResponse) throws
 URISyntaxException {
    // Space needs to encoded to use S3 parseUri function
    S3Uri s3Uri =
 s3Utilities.parseUri(URI.create(unloadResponse.getManifestFile().replace(" ",
 "%20")));
    ResponseBytes<GetObjectResponse> objectBytes =
 s3Client.getObjectAsBytes(GetObjectRequest.builder()
            .bucket(s3Uri.bucket().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getManifestFile(), "Invalid S3 URI")))
            .key(s3Uri.key().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getManifestFile(), "Invalid S3 URI")))
            .build());
    String manifestFileContent = new String(objectBytes.asByteArray(),
 StandardCharsets.UTF_8);
    return new Gson().fromJson(manifestFileContent, UnloadManifest.class);
}

class UnloadManifest {
    @Getter
    public class FileMetadata {
```

```
        long content_length_in_bytes;
        long row_count;
    }

    @Getter
    public class ResultFile {
        String url;
        FileMetadata file_metadata;
    }

    @Getter
    public class QueryMetadata {
        long total_content_length_in_bytes;
        long total_row_count;
        String result_format;
        String result_version;
    }

    @Getter
    public class Author {
        String name;
        String manifest_file_version;
    }

    @Getter
    private List<ResultFile> result_files;
    @Getter
    private QueryMetadata query_metadata;
    @Getter
    private Author author;
}
```

Go

```
// Read and parse manifest content

func getManifestFile(s3Svc *s3.S3, response map[string]string) Manifest {
    var manifestBuf = getObject(s3Svc, response["manifestFile"])
    var manifest Manifest
    json.Unmarshal(manifestBuf.Bytes(), &manifest)
    return manifest
}
```

```
func getObject(s3Svc *s3.S3, s3Uri string) *bytes.Buffer {
    u,_ := url.Parse(s3Uri)
    getObjectInput := &s3.GetObjectInput{
        Key:    aws.String(u.Path),
        Bucket: aws.String(u.Host),
    }
    getObjectOutput, err := s3Svc.GetObject(getObjectInput)
    if err != nil {
        fmt.Println("Error: %s\n", err.Error())
    }
    buf := new(bytes.Buffer)
    buf.ReadFrom(getObjectOutput.Body)
    return buf
}

// Unload's Manifest structure

type Manifest struct {
    Author interface{}
    Query_metadata map[string]any
    Result_files  []struct {
        File_metadata interface{}
        Url string
    }
}}
```

Python

```
def __get_manifest_file(self, response):
    manifest = self.get_object(response['manifestFile']).read().decode('utf-8')
    parsed_manifest = json.loads(manifest)
    print("Manifest contents: \n%s" % parsed_manifest)

def get_object(self, uri):
    try:
        bucket, key = uri.replace("s3://", "").split("/", 1)
        s3_client = boto3.client('s3', region_name=<region>)
        response = s3_client.get_object(Bucket=bucket, Key=key)
        return response['Body']
    except Exception as err:
        print("Failed to get the object for URI:", uri)
        raise err
```

Node.js

```
// Read and parse manifest content

async getManifestFile(response) {
    let manifest;
    await this.getS3Object(response['manifestFile']).then(
        (data) => {
            manifest = JSON.parse(data);
        }
    );
    return manifest;
}

async getS3Object(uri) {
    const {bucketName, key} = this.getBucketAndKey(uri);
    const params = new GetObjectCommand({
        Bucket: bucketName,
        Key: key
    })
    const response = await this.s3Client.send(params);
    return await response.Body.transformToString();
}

getBucketAndKey(uri) {
    const [bucketName] = uri.replace("s3://", "").split("/", 1);
    const key = uri.replace("s3://", "").split('/').slice(1).join('/');
    return {bucketName, key};
}
```

讀取和剖析中繼資料內容

Java

```
// Read and parse metadata content
public UnloadMetadata getUnloadMetadata(UnloadResponse unloadResponse) throws
 IOException {
   AmazonS3URI s3URI = new AmazonS3URI(unloadResponse.getMetadataFile());
   S3Object s3Object = s3Client.getObject(s3URI.getBucket(), s3URI.getKey());
    String metadataFileContent = new
 String(IOUtils.toByteArray(s3Object.getObjectContent()), StandardCharsets.UTF_8);
    final Gson gson = new GsonBuilder()
```

```
            .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
            .create();
    return gson.fromJson(metadataFileContent, UnloadMetadata.class);
}

class UnloadMetadata {
    @JsonProperty("ColumnInfo")
    List<ColumnInfo> columnInfo;
    @JsonProperty("Author")
    Author author;

    @Data
    public class Author {
        @JsonProperty("Name")
        String name;
        @JsonProperty("MetadataFileVersion")
        String metadataFileVersion;
    }
}
```

Java v2

```
// Read and parse metadata content

public UnloadMetadata getUnloadMetadata(UnloadResponse unloadResponse) throws
 URISyntaxException {
   // Space needs to encoded to use S3 parseUri function
    S3Uri s3Uri =
 s3Utilities.parseUri(URI.create(unloadResponse.getMetadataFile().replace(" ",
 "%20")));
    ResponseBytes<GetObjectResponse> objectBytes =
 s3Client.getObjectAsBytes(GetObjectRequest.builder()
            .bucket(s3Uri.bucket().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getMetadataFile(), "Invalid S3 URI")))
            .key(s3Uri.key().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getMetadataFile(), "Invalid S3 URI")))
            .build());
    String metadataFileContent = new String(objectBytes.asByteArray(),
 StandardCharsets.UTF_8);
    final Gson gson = new GsonBuilder()
            .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
            .create();
    return gson.fromJson(metadataFileContent, UnloadMetadata.class);
```

```
}

class UnloadMetadata {
    @JsonProperty("ColumnInfo")
    List<ColumnInfo> columnInfo;
    @JsonProperty("Author")
    Author author;

    @Data
    public class Author {
        @JsonProperty("Name")
        String name;
        @JsonProperty("MetadataFileVersion")
        String metadataFileVersion;
    }
}
```

Go

```
// Read and parse metadata content

func getMetadataFile(s3Svc *s3.S3, response map[string]string) Metadata {
    var metadataBuf = getObject(s3Svc, response["metadataFile"])
    var metadata Metadata
    json.Unmarshal(metadataBuf.Bytes(), &metadata)
    return metadata
}

func getObject(s3Svc *s3.S3, s3Uri string) *bytes.Buffer {
    u,_ := url.Parse(s3Uri)
    getObjectInput := &s3.GetObjectInput{
        Key:    aws.String(u.Path),
        Bucket: aws.String(u.Host),
    }
    getObjectOutput, err := s3Svc.GetObject(getObjectInput)
    if err != nil {
        fmt.Println("Error: %s\n", err.Error())
    }
    buf := new(bytes.Buffer)
    buf.ReadFrom(getObjectOutput.Body)
    return buf
}
```

```
// Unload's Metadata structure

type Metadata struct {
    Author interface{}
    ColumnInfo []struct {
        Name string
        Type map[string]string
    }
}
```

Python

```
def __get_metadata_file(self, response):
    metadata = self.get_object(response['metadataFile']).read().decode('utf-8')
    parsed_metadata = json.loads(metadata)
   print("Metadata contents: \n%s" % parsed_metadata)

def get_object(self, uri):
    try:
        bucket, key = uri.replace("s3://", "").split("/", 1)
        s3_client = boto3.client('s3', region_name=<region>)
        response = s3_client.get_object(Bucket=bucket, Key=key)
        return response['Body']
    except Exception as err:
        print("Failed to get the object for URI:", uri)
        raise err
```

Node.js

```
// Read and parse metadata content
async getMetadataFile(response) {
    let metadata;
    await this.getS3Object(response['metadataFile']).then(
        (data) => {
            metadata = JSON.parse(data);
        }
    );
    return metadata;
}

async getS3Object(uri) {
```

```
    const {bucketName, key} = this.getBucketAndKey(uri);
    const params = new GetObjectCommand({
        Bucket: bucketName,
        Key: key
    })
    const response = await this.s3Client.send(params);
    return await response.Body.transformToString();
}

getBucketAndKey(uri) {
    const [bucketName] = uri.replace("s3://", "").split("/", 1);
    const key = uri.replace("s3://", "").split('/').slice(1).join('/');
    return {bucketName, key};
}
```

取消查詢

您可以使用下列程式碼片段來取消查詢。

> **Note**
>
> 這些程式碼片段是以 上的完整範例應用程式為基礎GitHub。如需如何開始使用範例應用程式的詳細資訊，請參閱 範例應用程式。

Java

```
    public void cancelQuery() {
        System.out.println("Starting query: " + SELECT_ALL_QUERY);
        QueryRequest queryRequest = new QueryRequest();
        queryRequest.setQueryString(SELECT_ALL_QUERY);
        QueryResult queryResult = queryClient.query(queryRequest);

        System.out.println("Cancelling the query: " + SELECT_ALL_QUERY);
        final CancelQueryRequest cancelQueryRequest = new CancelQueryRequest();
        cancelQueryRequest.setQueryId(queryResult.getQueryId());
        try {
            queryClient.cancelQuery(cancelQueryRequest);
            System.out.println("Query has been successfully cancelled");
        } catch (Exception e) {
```

```
            System.out.println("Could not cancel the query: " + SELECT_ALL_QUERY + "
 = " + e);
        }
    }
```

Java v2

```
    public void cancelQuery() {
        System.out.println("Starting query: " + SELECT_ALL_QUERY);
        QueryRequest queryRequest =
 QueryRequest.builder().queryString(SELECT_ALL_QUERY).build();
        QueryResponse queryResponse = timestreamQueryClient.query(queryRequest);

        System.out.println("Cancelling the query: " + SELECT_ALL_QUERY);
        final CancelQueryRequest cancelQueryRequest = CancelQueryRequest.builder()
                .queryId(queryResponse.queryId()).build();
        try {
            timestreamQueryClient.cancelQuery(cancelQueryRequest);
            System.out.println("Query has been successfully cancelled");
        } catch (Exception e) {
            System.out.println("Could not cancel the query: " + SELECT_ALL_QUERY + "
 = " + e);
        }
    }
```

Go

```
cancelQueryInput := &timestreamquery.CancelQueryInput{
      QueryId: aws.String(*queryOutput.QueryId),
  }

  fmt.Println("Submitting cancellation for the query")
  fmt.Println(cancelQueryInput)

  // submit the query
  cancelQueryOutput, err := querySvc.CancelQuery(cancelQueryInput)

  if err != nil {
      fmt.Println("Error:")
      fmt.Println(err)
  } else {
    fmt.Println("Query has been cancelled successfully")
    fmt.Println(cancelQueryOutput)
```

```
    }
```

Python

```
    def cancel_query(self):
        print("Starting query: " + self.SELECT_ALL)
        result = self.client.query(QueryString=self.SELECT_ALL)
        print("Cancelling query: " + self.SELECT_ALL)
        try:
            self.client.cancel_query(QueryId=result['QueryId'])
            print("Query has been successfully cancelled")
        except Exception as err:
            print("Cancelling query failed:", err)
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function tryCancelQuery() {
    const params = {
        QueryString: SELECT_ALL_QUERY
    };
    console.log(`Running query: ${SELECT_ALL_QUERY}`);

    await queryClient.query(params).promise()
        .then(
            async (response) => {
                await cancelQuery(response.QueryId);
            },
            (err) => {
                console.error("Error while executing select all query:", err);
            });
}

async function cancelQuery(queryId) {
    const cancelParams = {
        QueryId: queryId
    };
    console.log(`Sending cancellation for query: ${SELECT_ALL_QUERY}`);
    await queryClient.cancelQuery(cancelParams).promise()
        .then(
            (response) => {
```

```
                console.log("Query has been cancelled successfully");
            },
            (err) => {
                console.error("Error while cancelling select all:", err);
            });
}
```

.NET

```
        public async Task CancelQuery()
        {
            Console.WriteLine("Starting query: " + SELECT_ALL_QUERY);
            QueryRequest queryRequest = new QueryRequest();
            queryRequest.QueryString = SELECT_ALL_QUERY;
            QueryResponse queryResponse = await
 queryClient.QueryAsync(queryRequest);

            Console.WriteLine("Cancelling query: " + SELECT_ALL_QUERY);
            CancelQueryRequest cancelQueryRequest = new CancelQueryRequest();
            cancelQueryRequest.QueryId = queryResponse.QueryId;

            try
            {
                await queryClient.CancelQueryAsync(cancelQueryRequest);
                Console.WriteLine("Query has been successfully cancelled.");
            } catch(Exception e)
            {
                Console.WriteLine("Could not cancel the query: " + SELECT_ALL_QUERY
 + " = " + e);
            }
        }
```

建立批次載入任務

您可以使用下列程式碼片段來建立批次載入任務。

Java

```
package com.example.tryit;

import java.util.Arrays;
```

```
import
 software.amazon.awssdk.services.timestreamwrite.model.CreateBatchLoadTaskRequest;
import
 software.amazon.awssdk.services.timestreamwrite.model.CreateBatchLoadTaskResponse;
import software.amazon.awssdk.services.timestreamwrite.model.DataModel;
import software.amazon.awssdk.services.timestreamwrite.model.DataModelConfiguration;
import
 software.amazon.awssdk.services.timestreamwrite.model.DataSourceConfiguration;
import
 software.amazon.awssdk.services.timestreamwrite.model.DataSourceS3Configuration;
import software.amazon.awssdk.services.timestreamwrite.model.DimensionMapping;
import
 software.amazon.awssdk.services.timestreamwrite.model.MultiMeasureAttributeMapping;
import software.amazon.awssdk.services.timestreamwrite.model.MultiMeasureMappings;
import software.amazon.awssdk.services.timestreamwrite.model.ReportConfiguration;
import software.amazon.awssdk.services.timestreamwrite.model.ReportS3Configuration;
import software.amazon.awssdk.services.timestreamwrite.model.ScalarMeasureValueType;
import software.amazon.awssdk.services.timestreamwrite.model.TimeUnit;
import software.amazon.awssdk.services.timestreamwrite.TimestreamWriteClient;

public class BatchLoadExample {
    public static final String DATABASE_NAME = <database name>;
    public static final String TABLE_NAME = <table name>;
    public static final String INPUT_BUCKET = <S3 location>;
    public static final String INPUT_OBJECT_KEY_PREFIX = <CSV filename>;
    public static final String REPORT_BUCKET = <S3 location>;
    public static final long HT_TTL_HOURS = 24L;
    public static final long CT_TTL_DAYS = 7L;

    TimestreamWriteClient amazonTimestreamWrite;

    public BatchLoadExample(TimestreamWriteClient client) {
        this.amazonTimestreamWrite = client;
    }

    public String createBatchLoadTask() {
        System.out.println("Creating batch load task");

        CreateBatchLoadTaskRequest request = CreateBatchLoadTaskRequest.builder()
                .dataModelConfiguration(DataModelConfiguration.builder()
                        .dataModel(DataModel.builder()
                                .timeColumn("timestamp")
                                .timeUnit(TimeUnit.SECONDS)
                                .dimensionMappings(Arrays.asList(
```

```
                                        DimensionMapping.builder()
                                                .sourceColumn("vehicle")
                                                .build(),
                                        DimensionMapping.builder()
                                                .sourceColumn("registration")
                                                .destinationColumn("license")
                                                .build()))
                                .multiMeasureMappings(MultiMeasureMappings.builder()
                                        .targetMultiMeasureName("mva_measure_name")

.multiMeasureAttributeMappings(Arrays.asList(

MultiMeasureAttributeMapping.builder()
                                                                .sourceColumn("wgt")

.targetMultiMeasureAttributeName("weight")

.measureValueType(ScalarMeasureValueType.DOUBLE)
                                                                .build(),

MultiMeasureAttributeMapping.builder()
                                                                .sourceColumn("spd")

.targetMultiMeasureAttributeName("speed")

.measureValueType(ScalarMeasureValueType.DOUBLE)
                                                                .build(),

MultiMeasureAttributeMapping.builder()
                                                                .sourceColumn("fuel")

.measureValueType(ScalarMeasureValueType.DOUBLE)
                                                                .build(),

MultiMeasureAttributeMapping.builder()
                                                                .sourceColumn("miles")

.measureValueType(ScalarMeasureValueType.DOUBLE)
                                                                .build()))
                                        .build())
                                .build())
                        .build())
                .dataSourceConfiguration(DataSourceConfiguration.builder()
                        .dataSourceS3Configuration(
```

```
                                DataSourceS3Configuration.builder()
                                        .bucketName(INPUT_BUCKET)
                                        .objectKeyPrefix(INPUT_OBJECT_KEY_PREFIX)
                                        .build())
                                .dataFormat("CSV")
                                .build())
                        .reportConfiguration(ReportConfiguration.builder()
                                .reportS3Configuration(ReportS3Configuration.builder()
                                        .bucketName(REPORT_BUCKET)
                                        .build())
                                .build())
                        .targetDatabaseName(DATABASE_NAME)
                        .targetTableName(TABLE_NAME)
                        .build();
                try {
                    final CreateBatchLoadTaskResponse createBatchLoadTaskResponse =
 amazonTimestreamWrite.createBatchLoadTask(request);
                    String taskId = createBatchLoadTaskResponse.taskId();
                    System.out.println("Successfully created batch load task: " + taskId);
                    return taskId;
                } catch (Exception e) {
                    System.out.println("Failed to create batch load task: " + e);
                    throw e;
                }
            }
        }
```

Go

```
package main

import (
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite/types"
)

func main() {
```

```
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{})(aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID &&  region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &  aws.EndpointNotFoundError{}
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))

 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }

 client := timestreamwrite.NewFromConfig(cfg)

 response, err := client.CreateBatchLoadTask(context.TODO(), &
 timestreamwrite.CreateBatchLoadTaskInput{
            TargetDatabaseName: aws.String("BatchLoadExampleDatabase"),
            TargetTableName: aws.String("BatchLoadExampleTable"),
            RecordVersion: aws.Int64(1),
            DataModelConfiguration: &  types.DataModelConfiguration{
                DataModel: &  types.DataModel{
                    TimeColumn: aws.String("timestamp"),
                    TimeUnit: types.TimeUnitMilliseconds,
                    DimensionMappings: []types.DimensionMapping{
                        {
                            SourceColumn: aws.String("registration"),
                            DestinationColumn: aws.String("license"),
                        },
                    },
                    MultiMeasureMappings: &  types.MultiMeasureMappings{
                        TargetMultiMeasureName: aws.String("mva_measure_name"),
                        MultiMeasureAttributeMappings:
 []types.MultiMeasureAttributeMapping{
                            {
                                SourceColumn: aws.String("wgt"),
```

```
                                TargetMultiMeasureAttributeName:
 aws.String("weight"),
                                MeasureValueType:
 types.ScalarMeasureValueTypeDouble,
                            },
                            {
                                SourceColumn: aws.String("spd"),
                                TargetMultiMeasureAttributeName:
 aws.String("speed"),
                                MeasureValueType:
 types.ScalarMeasureValueTypeDouble,
                            },
                            {
                                SourceColumn: aws.String("fuel_consumption"),
                                TargetMultiMeasureAttributeName: aws.String("fuel"),
                                MeasureValueType:
 types.ScalarMeasureValueTypeDouble,
                            },
                        },
                    },
                },
            },
           },
           DataSourceConfiguration: &  types.DataSourceConfiguration{
                DataSourceS3Configuration: &  types.DataSourceS3Configuration{
                    BucketName: aws.String("test-batch-load-west-2"),
                    ObjectKeyPrefix: aws.String("sample.csv"),
                },
                DataFormat: types.BatchLoadDataFormatCsv,
           },
           ReportConfiguration: & types.ReportConfiguration{
                ReportS3Configuration: & types.ReportS3Configuration{
                    BucketName: aws.String("test-batch-load-report-west-2"),
                    EncryptionOption: types.S3EncryptionOptionSseS3,
                },
           },
 })

 fmt.Println(aws.ToString(response.TaskId))
}
```

Python

```
import boto3
```

```
from botocore.config import Config

INGEST_ENDPOINT = "<URL>"
REGION = "us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7
DATABASE_NAME = "<database name>"
TABLE_NAME = "<table name>"
INPUT_BUCKET_NAME = "<S3 location>"
INPUT_OBJECT_KEY_PREFIX = "<CSV file name>"
REPORT_BUCKET_NAME = "<S3 location>"


def create_batch_load_task(client, database_name, table_name, input_bucket_name,
 input_object_key_prefix, report_bucket_name):
    try:
        result = client.create_batch_load_task(TargetDatabaseName=database_name,
 TargetTableName=table_name,
                                               DataModelConfiguration={"DataModel":
 {
                                                   "TimeColumn": "timestamp",
                                                   "TimeUnit": "SECONDS",
                                                   "DimensionMappings": [
                                                       {
                                                           "SourceColumn": "vehicle"
                                                       },
                                                       {
                                                           "SourceColumn":
 "registration",
                                                           "DestinationColumn":
 "license"
                                                       }
                                                   ],
                                                   "MultiMeasureMappings": {
                                                       "TargetMultiMeasureName":
 "metrics",

 "MultiMeasureAttributeMappings": [
                                                           {
                                                               "SourceColumn":
 "wgt",
                                                               "MeasureValueType":
 "DOUBLE"
                                                           },
```

```
                                                        {
                                                            "SourceColumn":
"spd",
                                                            "MeasureValueType":
"DOUBLE"
                                                        },
                                                        {
                                                            "SourceColumn":
"fuel_consumption",

"TargetMultiMeasureAttributeName": "fuel",
                                                            "MeasureValueType":
"DOUBLE"
                                                        },
                                                        {
                                                            "SourceColumn":
"miles",
                                                            "MeasureValueType":
"DOUBLE"
                                                        }
                                                    ]}
                                            }
                                            },
                                            DataSourceConfiguration={
                                                "DataSourceS3Configuration": {
                                                    "BucketName":
input_bucket_name,
                                                    "ObjectKeyPrefix":
input_object_key_prefix
                                                },
                                                "DataFormat": "CSV"
                                            },
                                            ReportConfiguration={
                                                "ReportS3Configuration": {
                                                    "BucketName":
report_bucket_name,
                                                    "EncryptionOption": "SSE_S3"
                                                }
                                            }
                                            )

        task_id = result["TaskId"]
        print("Successfully created batch load task: ", task_id)
        return task_id
```

```
    except Exception as err:
        print("Create batch load task job failed:", err)
        return None


if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write',
                                  endpoint_url=INGEST_ENDPOINT, region_name=REGION,
                                  config=Config(read_timeout=20,
 max_pool_connections=5000, retries={'max_attempts': 10}))

    task_id = create_batch_load_task(write_client, DATABASE_NAME, TABLE_NAME,
                                     INPUT_BUCKET_NAME, INPUT_OBJECT_KEY_PREFIX,
 REPORT_BUCKET_NAME)
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

如需API詳細資訊，請參閱類別 CreateBatchLoadCommand 和 CreateBatchLoadTask。

```
import { TimestreamWriteClient, CreateBatchLoadTaskCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-west-2", endpoint:
 "https://gamma-ingest-cell3.timestream.us-west-2.amazonaws.com" });

const params = {
    TargetDatabaseName: "BatchLoadExampleDatabase",
    TargetTableName: "BatchLoadExampleTable",
 RecordVersion: 1,
 DataModelConfiguration: {
  DataModel: {
   TimeColumn: "timestamp",
            TimeUnit: "MILLISECONDS",
            DimensionMappings: [
                {
                    SourceColumn: "registration",
                    DestinationColumn: "license"
                }
            ],
            MultiMeasureMappings: {
```

```
                TargetMultiMeasureName: "mva_measure_name",
                MultiMeasureAttributeMappings: [
                    {
                        SourceColumn: "wgt",
                        TargetMultiMeasureAttributeName: "weight",
                        MeasureValueType: "DOUBLE"
                    },
                    {
                        SourceColumn: "spd",
                        TargetMultiMeasureAttributeName: "speed",
                        MeasureValueType: "DOUBLE"
                    },
                    {
                        SourceColumn: "fuel_consumption",
                        TargetMultiMeasureAttributeName: "fuel",
                        MeasureValueType: "DOUBLE"
                    }
                ]
            }
        }
    },
 DataSourceConfiguration: {
        DataSourceS3Configuration: {
            BucketName: "test-batch-load-west-2",
            ObjectKeyPrefix: "sample.csv"
        },
        DataFormat: "CSV"
    },
    ReportConfiguration: {
        ReportS3Configuration: {
            BucketName: "test-batch-load-report-west-2",
            EncryptionOption: "SSE_S3"
        }
    }
};

const command = new CreateBatchLoadTaskCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Created batch load task ` + data.TaskId);
} catch (error) {
 console.log("Error creating table. ", error);
 throw error;
```

```
}
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
    public class CreateBatchLoadTaskExample
    {
        public const string DATABASE_NAME = "<database name>";
        public const string TABLE_NAME = "<table name>";
        public const string INPUT_BUCKET = "<input bucket name>";
        public const string INPUT_OBJECT_KEY_PREFIX = "<CSV file name>";
        public const string REPORT_BUCKET = "<report bucket name>";
        public const long HT_TTL_HOURS = 24L;
        public const long CT_TTL_DAYS = 7L;
        private readonly AmazonTimestreamWriteClient writeClient;

        public CreateBatchLoadTaskExample(AmazonTimestreamWriteClient writeClient)
        {
            this.writeClient = writeClient;
        }

        public async Task CreateBatchLoadTask()
        {
            try
            {
                var createBatchLoadTaskRequest = new CreateBatchLoadTaskRequest
                {
                    DataModelConfiguration = new DataModelConfiguration
                    {
                        DataModel = new DataModel
                        {
                            TimeColumn = "timestamp",
                            TimeUnit = TimeUnit.SECONDS,
                            DimensionMappings = new List<DimensionMapping>()
                            {
```

```
                                new()
                                {
                                        SourceColumn = "vehicle"
                                },
                                new()
                                {
                                        SourceColumn = "registration",
                                        DestinationColumn = "license"
                                }
                            },
                            MultiMeasureMappings = new MultiMeasureMappings
                            {
                                TargetMultiMeasureName = "mva_measure_name",
                                MultiMeasureAttributeMappings = new
List<MultiMeasureAttributeMapping>()
                                {
                                        new()
                                        {
                                                SourceColumn = "wgt",
                                                TargetMultiMeasureAttributeName =
"weight",
                                                MeasureValueType =
ScalarMeasureValueType.DOUBLE
                                        },
                                        new()
                                        {
                                                SourceColumn = "spd",
                                                TargetMultiMeasureAttributeName =
"speed",
                                                MeasureValueType =
ScalarMeasureValueType.DOUBLE
                                        },
                                        new()
                                        {
                                                SourceColumn = "fuel",
                                                TargetMultiMeasureAttributeName =
"fuel",
                                                MeasureValueType =
ScalarMeasureValueType.DOUBLE
                                        },
                                        new()
                                        {
                                                SourceColumn = "miles",
```

```
                                                TargetMultiMeasureAttributeName =
 "miles",
                                                MeasureValueType =
 ScalarMeasureValueType.DOUBLE
                                            }
                                    }
                                }
                            }
                        },
                        DataSourceConfiguration = new DataSourceConfiguration
                        {
                            DataSourceS3Configuration = new DataSourceS3Configuration
                            {
                                BucketName = INPUT_BUCKET,
                                ObjectKeyPrefix = INPUT_OBJECT_KEY_PREFIX
                            },
                            DataFormat = "CSV"
                        },
                        ReportConfiguration = new ReportConfiguration
                        {
                            ReportS3Configuration = new ReportS3Configuration
                            {
                                BucketName = REPORT_BUCKET
                            }
                        },
                        TargetDatabaseName = DATABASE_NAME,
                        TargetTableName = TABLE_NAME
                    };

                    CreateBatchLoadTaskResponse response = await
 writeClient.CreateBatchLoadTaskAsync(createBatchLoadTaskRequest);
                    Console.WriteLine($"Task created: " + response.TaskId);
                }
                catch (Exception e)
                {
                    Console.WriteLine("Create batch load task failed:" + e.ToString());
                }
            }
        }
    }
```

```
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
```

```
using Amazon;
using Amazon.TimestreamQuery;
using System.Threading.Tasks;
using System;
using CommandLine;
static class Constants
{

}
namespace TimestreamDotNetSample
{
    class MainClass
    {
        public class Options
        {

        }
        public static void Main(string[] args)
        {
            Parser.Default.ParseArguments<Options>(args)
                .WithParsed<Options>(o => {
                    MainAsync().GetAwaiter().GetResult();
                });
        }

        static async Task MainAsync()
        {
            var writeClientConfig = new AmazonTimestreamWriteConfig
            {
                ServiceURL =  "<service URL>",
                Timeout = TimeSpan.FromSeconds(20),
                MaxErrorRetry = 10
            };

            var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
            var example = new CreateBatchLoadTaskExample(writeClient);
            await example.CreateBatchLoadTask();
        }
    }
}
```

描述批次載入任務

您可以使用下列程式碼片段來描述批次載入任務。

Java

```
    public void describeBatchLoadTask(String taskId) {
            final DescribeBatchLoadTaskResponse batchLoadTaskResponse =
 amazonTimestreamWrite

 .describeBatchLoadTask(DescribeBatchLoadTaskRequest.builder()
                                             .taskId(taskId)
                                             .build());

            System.out.println("Task id: " +
 batchLoadTaskResponse.batchLoadTaskDescription().taskId());
            System.out.println("Status: " +
 batchLoadTaskResponse.batchLoadTaskDescription().taskStatusAsString());
            System.out.println("Records processed: "
                               +
 batchLoadTaskResponse.batchLoadTaskDescription().progressReport().recordsProcessed());
    }
```

Go

```
package main

import (
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
)

func main() {
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID && region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
```

```
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &aws.EndpointNotFoundError{}
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))

 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }

 client := timestreamwrite.NewFromConfig(cfg)

 response, err := client.DescribeBatchLoadTask(context.TODO(),
 &timestreamwrite.DescribeBatchLoadTaskInput{
  TaskId: aws.String("<TaskId>"),
 })

 fmt.Println(aws.ToString(response.BatchLoadTaskDescription.TaskId))
}
```

Python

```
import boto3
from botocore.config import Config

INGEST_ENDPOINT="<url>"
REGION="us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7
TASK_ID = "<task id>"

def describe_batch_load_task(client, task_id):
    try:
        result = client.describe_batch_load_task(TaskId=task_id)
        print("Successfully described batch load task: ", result)
    except Exception as err:
        print("Describe batch load task job failed:", err)
```

```
if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write', \
        endpoint_url=INGEST_ENDPOINT, region_name=REGION, \
        config=Config(read_timeout=20, max_pool_connections = 5000,
 retries={'max_attempts': 10}))

    describe_batch_load_task(write_client, TASK_ID)
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

如需API詳細資訊，請參閱類別 DescribeBatchLoadCommand 和 DescribeBatchLoadTask。

```
import { TimestreamWriteClient, DescribeBatchLoadTaskCommand } from "@aws-sdk/
client-timestream-write";
const writeClient = new TimestreamWriteClient({ region: "<region>", endpoint:
 "<endpoint>" });

const params = {
    TaskId: "<TaskId>"
};

const command = new DescribeBatchLoadTaskCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Batch load task has id ` + data.BatchLoadTaskDescription.TaskId);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Batch load task doesn't exist.");
    } else {
        console.log("Describe batch load task failed.", error);
        throw error;
    }
}
```

.NET

```
using System;
```

```
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
    public class DescribeBatchLoadTaskExample
    {
        private readonly AmazonTimestreamWriteClient writeClient;

        public DescribeBatchLoadTaskExample(AmazonTimestreamWriteClient writeClient)
        {
            this.writeClient = writeClient;
        }

        public async Task DescribeBatchLoadTask(String taskId)
        {
            try
            {
                var describeBatchLoadTaskRequest = new DescribeBatchLoadTaskRequest
                {
                    TaskId = taskId
                };
                DescribeBatchLoadTaskResponse response = await
 writeClient.DescribeBatchLoadTaskAsync(describeBatchLoadTaskRequest);
                Console.WriteLine($"Task has id:
{response.BatchLoadTaskDescription.TaskId}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Batch load task does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Describe batch load task failed:" +
 e.ToString());
            }
        }
    }
}
```

```
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using Amazon;
using Amazon.TimestreamQuery;
using System.Threading.Tasks;
using System;
using CommandLine;
static class Constants
{

}
namespace TimestreamDotNetSample
{
    class MainClass
    {
        public class Options
        {

        }
        public static void Main(string[] args)
        {
            Parser.Default.ParseArguments<Options>(args)
                .WithParsed<Options>(o => {
                    MainAsync().GetAwaiter().GetResult();
                });
        }

        static async Task MainAsync()
        {
            var writeClientConfig = new AmazonTimestreamWriteConfig
            {
                ServiceURL =  "<service URL>",
                Timeout = TimeSpan.FromSeconds(20),
                MaxErrorRetry = 10
            };

            var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
            var example = new DescribeBatchLoadTaskExample(writeClient);
            await example.DescribeBatchLoadTask("<batch load task id>");
        }
    }
}
```

列出批次載入任務

您可以使用下列程式碼片段來列出批次載入任務。

Java

```
    public void listBatchLoadTasks() {
            final ListBatchLoadTasksResponse listBatchLoadTasksResponse =
 amazonTimestreamWrite
                            .listBatchLoadTasks(ListBatchLoadTasksRequest.builder()
                                    .maxResults(15)
                                    .build());

            for (BatchLoadTask batchLoadTask :
 listBatchLoadTasksResponse.batchLoadTasks()) {
                    System.out.println(batchLoadTask.taskId());
            }
    }
```

Go

```
package main

import (
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
)

func main() {
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID && region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &aws.EndpointNotFoundError{}
```

```
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))

 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }

 client := timestreamwrite.NewFromConfig(cfg)
 listBatchLoadTasksMaxResult := int32(15)

 response, err := client.ListBatchLoadTasks(context.TODO(),
 &timestreamwrite.ListBatchLoadTasksInput{
  MaxResults: &listBatchLoadTasksMaxResult,
 })

 for i, task := range response.BatchLoadTasks {
  fmt.Println(i, aws.ToString(task.TaskId))
 }
}
```

Python

```
import boto3
from botocore.config import Config

INGEST_ENDPOINT = "<url>"
REGION = "us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7


def print_batch_load_tasks(batch_load_tasks):
    for batch_load_task in batch_load_tasks:
        print(batch_load_task['TaskId'])


def list_batch_load_tasks(client):
    print("\nListing batch load tasks")
    try:
        response = client.list_batch_load_tasks(MaxResults=10)
```

```
        print_batch_load_tasks(response['BatchLoadTasks'])
        next_token = response.get('NextToken', None)
        while next_token:
            response = client.list_batch_load_tasks(
                NextToken=next_token, MaxResults=10)
            print_batch_load_tasks(response['BatchLoadTasks'])
            next_token = response.get('NextToken', None)
    except Exception as err:
        print("List batch load tasks failed:", err)
        raise err


if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write',
                                  endpoint_url=INGEST_ENDPOINT, region_name=REGION,
                                  config=Config(read_timeout=20,
 max_pool_connections=5000, retries={'max_attempts': 10}))

    list_batch_load_tasks(write_client)
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

如需API詳細資訊，請參閱類別 DescribeBatchLoadCommand 和 DescribeBatchLoadTask。

```
import { TimestreamWriteClient, ListBatchLoadTasksCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "<region>", endpoint:
 "<endpoint>" });

const params = {
    MaxResults: <15>
};

const command = new ListBatchLoadTasksCommand(params);

getBatchLoadTasksList(null);

async function getBatchLoadTasksList(nextToken) {
    if (nextToken) {
```

```
        params.NextToken = nextToken;
    }

    try {
        const data = await writeClient.send(command);

        data.BatchLoadTasks.forEach(function (task) {
            console.log(task.TaskId);
        });

        if (data.NextToken) {
            return getBatchLoadTasksList(data.NextToken);
        }
    } catch (error) {
        console.log("Error while listing batch load tasks", error);
    }
}
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
    public class ListBatchLoadTasksExample
    {
        private readonly AmazonTimestreamWriteClient writeClient;

        public ListBatchLoadTasksExample(AmazonTimestreamWriteClient writeClient)
        {
            this.writeClient = writeClient;
        }

        public async Task ListBatchLoadTasks()
        {
            Console.WriteLine("Listing batch load tasks");

            try
```

```
            {
                var listBatchLoadTasksRequest = new ListBatchLoadTasksRequest
                {
                    MaxResults = 15
                };

                ListBatchLoadTasksResponse response = await
 writeClient.ListBatchLoadTasksAsync(listBatchLoadTasksRequest);

                PrintBatchLoadTasks(response.BatchLoadTasks);
                var nextToken = response.NextToken;

                while (nextToken != null)
                {
                    listBatchLoadTasksRequest.NextToken = nextToken;
                    response = await
 writeClient.ListBatchLoadTasksAsync(listBatchLoadTasksRequest);
                    PrintBatchLoadTasks(response.BatchLoadTasks);
                    nextToken = response.NextToken;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("List batch load tasks failed:" + e.ToString());
            }
        }

        private void PrintBatchLoadTasks(List<BatchLoadTask> tasks)
        {
            foreach (BatchLoadTask task in tasks)
                Console.WriteLine($"Task:{task.TaskId}");
        }
    }
}
```

```
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using Amazon;
using Amazon.TimestreamQuery;
using System.Threading.Tasks;
using System;
using CommandLine;
static class Constants
```

```
{

}
namespace TimestreamDotNetSample
{
    class MainClass
    {
        public class Options
        {

        }
        public static void Main(string[] args)
        {
            Parser.Default.ParseArguments<Options>(args)
                .WithParsed<Options>(o => {
                    MainAsync().GetAwaiter().GetResult();
                });
        }

        static async Task MainAsync()
        {
            var writeClientConfig = new AmazonTimestreamWriteConfig
            {
                ServiceURL =  "<service URL>",
                Timeout = TimeSpan.FromSeconds(20),
                MaxErrorRetry = 10
            };

            var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
            var example = new ListBatchLoadTasksExample(writeClient);
            await example.ListBatchLoadTasks();
        }
    }
}
```

恢復批次載入任務

您可以使用下列程式碼片段來繼續批次載入任務。

Java

```
    public void resumeBatchLoadTask(String taskId) {
```

```
            try {
                    amazonTimestreamWrite

 .resumeBatchLoadTask(ResumeBatchLoadTaskRequest.builder()
                                                    .taskId(taskId)
                                                    .build());

                    System.out.println("Successfully resumed batch load task.");
            } catch (ValidationException validationException) {
                    System.out.println(validationException.getMessage());
            }
    }
```

Go

```
package main

import (
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
)

func main() {
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID && region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &aws.EndpointNotFoundError{}
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))
```

```
 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }

 client := timestreamwrite.NewFromConfig(cfg)

 response, err := client.ResumeBatchLoadTask(context.TODO(),
 &timestreamwrite.ResumeBatchLoadTaskInput{
  TaskId: aws.String("TaskId"),
 })

 if err != nil {
  fmt.Println("Error:")
  fmt.Println(err)
 } else {
  fmt.Println("Resume batch load task is successful")
  fmt.Println(response)
 }
}
```

Python

```
import boto3
from botocore.config import Config

INGEST_ENDPOINT="<url>"
REGION="us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7
TASK_ID = "<TaskId>"

def resume_batch_load_task(client, task_id):
    try:
        result = client.resume_batch_load_task(TaskId=task_id)
        print("Successfully resumed batch load task: ", result)
    except Exception as err:
        print("Resume batch load task failed:", err)


if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write', \
```

```
        endpoint_url=INGEST_ENDPOINT, region_name=REGION, \
        config=Config(read_timeout=20, max_pool_connections = 5000,
 retries={'max_attempts': 10}))

    resume_batch_load_task(write_client, TASK_ID)
```

Node.js

下列程式碼片段AWSSDK用於 JavaScript v3。如需如何安裝用戶端和用量的詳細資訊，請參閱 Timestream Write Client - AWS SDK for JavaScript v3。

如需API詳細資訊，請參閱類別 CreateBatchLoadCommand 和 CreateBatchLoadTask。

```
import { TimestreamWriteClient, ResumeBatchLoadTaskCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "<region>", endpoint:
 "<endpoint>" });

const params = {
    TaskId: "<TaskId>"
};

const command = new ResumeBatchLoadTaskCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Resumed batch load task");
} catch (error) {
    console.log("Resume batch load task failed.", error);
 throw error;
}
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
```

```
    public class ResumeBatchLoadTaskExample
    {
        private readonly AmazonTimestreamWriteClient writeClient;

        public ResumeBatchLoadTaskExample(AmazonTimestreamWriteClient writeClient)
        {
            this.writeClient = writeClient;
        }

        public async Task ResumeBatchLoadTask(String taskId)
        {
            try
            {
                var resumeBatchLoadTaskRequest = new ResumeBatchLoadTaskRequest
                {
                    TaskId = taskId
                };
                ResumeBatchLoadTaskResponse response = await
 writeClient.ResumeBatchLoadTaskAsync(resumeBatchLoadTaskRequest);
                Console.WriteLine("Successfully resumed batch load task.");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Batch load task does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Resume batch load task failed: " + e.ToString());
            }
        }
    }
}
```

建立排程查詢

您可以使用下列程式碼片段來建立具有多測量映射的排程查詢。

Java

```
public static String DATABASE_NAME = "devops_sample_application";
public static String TABLE_NAME = "host_metrics_sample_application";
public static String HOSTNAME = "host-24Gju";
```

```
public static String SQ_NAME = "daily-sample";
public static String SCHEDULE_EXPRESSION = "cron(0/2 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
public static String QUERY = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
"ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
"FROM " +  DATABASE_NAME + "." +  TABLE_NAME + " " +
"WHERE measure_name = 'metrics' " +
"AND hostname = '" + HOSTNAME + "' " +
"AND time > ago(2h) " +
"GROUP BY region, hostname, az, BIN(time, 15s) " +
"ORDER BY binned_timestamp ASC " +
"LIMIT 5";


public String createScheduledQuery(String topic_arn,
    String role_arn,
    String database_name,
    String table_name) {
    System.out.println("Creating Scheduled Query");

    List<Pair<String, MeasureValueType>> sourceColToMeasureValueTypes =
 Arrays.asList(
        Pair.of("avg_cpu_utilization", DOUBLE),
        Pair.of("p90_cpu_utilization", DOUBLE),
        Pair.of("p95_cpu_utilization", DOUBLE),
        Pair.of("p99_cpu_utilization", DOUBLE));

    CreateScheduledQueryRequest createScheduledQueryRequest = new
 CreateScheduledQueryRequest()
            .withName(SQ_NAME)
            .withQueryString(QUERY)
            .withScheduleConfiguration(new ScheduleConfiguration()
                    .withScheduleExpression(SCHEDULE_EXPRESSION))
            .withNotificationConfiguration(new NotificationConfiguration()
                    .withSnsConfiguration(new SnsConfiguration()
                            .withTopicArn(topic_arn)))
            .withTargetConfiguration(new
 TargetConfiguration().withTimestreamConfiguration(new TimestreamConfiguration()
```

```
                    .withDatabaseName(database_name)
                    .withTableName(table_name)
                    .withTimeColumn("binned_timestamp")
                    .withDimensionMappings(Arrays.asList(
                            new DimensionMapping()
                                    .withName("region")
                                    .withDimensionValueType("VARCHAR"),
                            new DimensionMapping()
                                    .withName("az")
                                    .withDimensionValueType("VARCHAR"),
                            new DimensionMapping()
                                    .withName("hostname")
                                    .withDimensionValueType("VARCHAR")
                    ))
                    .withMultiMeasureMappings(new MultiMeasureMappings()
                        .withTargetMultiMeasureName("multi-metrics")
                        .withMultiMeasureAttributeMappings(
                            sourceColToMeasureValueTypes.stream()
                            .map(pair -> new MultiMeasureAttributeMapping()
                                .withMeasureValueType(pair.getValue().name())
                                .withSourceColumn(pair.getKey()))
                            .collect(Collectors.toList()))))
            .withErrorReportConfiguration(new ErrorReportConfiguration()
                    .withS3Configuration(new S3Configuration()

 .withBucketName(timestreamDependencyHelper.getS3ErrorReportBucketName())))
            .withScheduledQueryExecutionRoleArn(role_arn);

    try {
        final CreateScheduledQueryResult createScheduledQueryResult =
 queryClient.createScheduledQuery(createScheduledQueryRequest);
        final String scheduledQueryArn = createScheduledQueryResult.getArn();
        System.out.println("Successfully created scheduled query : " +
 scheduledQueryArn);
        return scheduledQueryArn;
    }
    catch (Exception e) {
        System.out.println("Scheduled Query creation failed: " + e);
        throw e;
    }
}
```

Java v2

```
public static String DATABASE_NAME = "testJavaV2DB";
public static String TABLE_NAME = "testJavaV2Table";
public static String HOSTNAME = "host-24Gju";
public static String SQ_NAME = "daily-sample";
public static String SCHEDULE_EXPRESSION = "cron(0/2 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
public static String VALID_QUERY = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
"ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
"FROM " +  DATABASE_NAME + "." +  TABLE_NAME + " " +
"WHERE measure_name = 'metrics' " +
"AND hostname = '" + HOSTNAME + "' " +
"AND time > ago(2h) " +
"GROUP BY region, hostname, az, BIN(time, 15s) " +
"ORDER BY binned_timestamp ASC " +
"LIMIT 5";


private String createScheduledQueryHelper(String topicArn, String roleArn,
        String s3ErrorReportBucketName, String query,
        TargetConfiguration targetConfiguration) {
    System.out.println("Creating Scheduled Query");

    CreateScheduledQueryRequest createScheduledQueryRequest =
 CreateScheduledQueryRequest.builder()
            .name(SQ_NAME)
            .queryString(query)
            .scheduleConfiguration(ScheduleConfiguration.builder()
                    .scheduleExpression(SCHEDULE_EXPRESSION)
                    .build())
            .notificationConfiguration(NotificationConfiguration.builder()
                    .snsConfiguration(SnsConfiguration.builder()
                            .topicArn(topicArn)
                            .build())
                    .build())
            .targetConfiguration(targetConfiguration)
            .errorReportConfiguration(ErrorReportConfiguration.builder()
```

```
                    .s3Configuration(S3Configuration.builder()
                            .bucketName(s3ErrorReportBucketName)
                            .objectKeyPrefix(SCHEDULED_QUERY_EXAMPLE)
                            .build())
                    .build())
            .scheduledQueryExecutionRoleArn(roleArn)
            .build();

    try {
        final CreateScheduledQueryResponse response =
 queryClient.createScheduledQuery(createScheduledQueryRequest);
        final String scheduledQueryArn = response.arn();
        System.out.println("Successfully created scheduled query : " +
 scheduledQueryArn);
        return scheduledQueryArn;
    }
    catch (Exception e) {
        System.out.println("Scheduled Query creation failed: " + e);
        throw e;
    }
}

public String createScheduledQuery(String topicArn, String roleArn,
        String databaseName, String tableName, String s3ErrorReportBucketName) {
    List<Pair<String, MeasureValueType>> sourceColToMeasureValueTypes =
 Arrays.asList(
            Pair.of("avg_cpu_utilization", DOUBLE),
            Pair.of("p90_cpu_utilization", DOUBLE),
            Pair.of("p95_cpu_utilization", DOUBLE),
            Pair.of("p99_cpu_utilization", DOUBLE));

    TargetConfiguration targetConfiguration = TargetConfiguration.builder()
            .timestreamConfiguration(TimestreamConfiguration.builder()
            .databaseName(databaseName)
            .tableName(tableName)
            .timeColumn("binned_timestamp")
            .dimensionMappings(Arrays.asList(
                    DimensionMapping.builder()
                            .name("region")
                            .dimensionValueType("VARCHAR")
                            .build(),
                    DimensionMapping.builder()
                            .name("az")
                            .dimensionValueType("VARCHAR")
```

```
                            .build(),
                    DimensionMapping.builder()
                            .name("hostname")
                            .dimensionValueType("VARCHAR")
                            .build()
            ))
            .multiMeasureMappings(MultiMeasureMappings.builder()
                    .targetMultiMeasureName("multi-metrics")
                    .multiMeasureAttributeMappings(
                            sourceColToMeasureValueTypes.stream()
                                    .map(pair ->
 MultiMeasureAttributeMapping.builder()

 .measureValueType(pair.getValue().name())
                                            .sourceColumn(pair.getKey())
                                            .build())
                                    .collect(Collectors.toList()))
                    .build())
            .build())
            .build();

    return createScheduledQueryHelper(topicArn, roleArn, s3ErrorReportBucketName,
 VALID_QUERY, targetConfiguration);
}}
```

Go

```
SQ_ERROR_CONFIGURATION_S3_BUCKET_NAME_PREFIX = "sq-error-configuration-sample-s3-
bucket-"
HOSTNAME            = "host-24Gju"
SQ_NAME             = "daily-sample"
SCHEDULE_EXPRESSION = "cron(0/1 * * * ? *)"
QUERY               = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
    "ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
    "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
    "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
    "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
    "FROM %s.%s " +
    "WHERE measure_name = 'metrics' " +
    "AND hostname = '" + HOSTNAME + "' " +
    "AND time > ago(2h) " +
    "GROUP BY region, hostname, az, BIN(time, 15s) " +
```

```
    "ORDER BY binned_timestamp ASC " +
    "LIMIT 5"
s3BucketName = utils.SQ_ERROR_CONFIGURATION_S3_BUCKET_NAME_PREFIX +
 generateRandomStringWithSize(5)

func generateRandomStringWithSize(size int) string {
     rand.Seed(time.Now().UnixNano())
     alphaNumericList := []rune("abcdefghijklmnopqrstuvwxyz0123456789")
     randomPrefix := make([]rune, size)
     for i := range randomPrefix {
         randomPrefix[i] = alphaNumericList[rand.Intn(len(alphaNumericList))]
     }
     return string(randomPrefix)
 }

func (timestreamBuilder TimestreamBuilder) createScheduledQuery(topicArn string,
 roleArn string, s3ErrorReportBucketName string,
query string, targetConfiguration timestreamquery.TargetConfiguration) (string,
 error) {

createScheduledQueryInput := &timestreamquery.CreateScheduledQueryInput{
    Name:        aws.String(SQ_NAME),
    QueryString: aws.String(query),
    ScheduleConfiguration: &timestreamquery.ScheduleConfiguration{
        ScheduleExpression: aws.String(SCHEDULE_EXPRESSION),
    },
    NotificationConfiguration: &timestreamquery.NotificationConfiguration{
        SnsConfiguration: &timestreamquery.SnsConfiguration{
            TopicArn: aws.String(topicArn),
        },
    },
    TargetConfiguration: &targetConfiguration,
    ErrorReportConfiguration: &timestreamquery.ErrorReportConfiguration{
        S3Configuration: &timestreamquery.S3Configuration{
            BucketName: aws.String(s3ErrorReportBucketName),
        },
    },
    ScheduledQueryExecutionRoleArn: aws.String(roleArn),
}

createScheduledQueryOutput, err :=
 timestreamBuilder.QuerySvc.CreateScheduledQuery(createScheduledQueryInput)

if err != nil {
```

```
    fmt.Printf("Error: %s", err.Error())
} else {
    fmt.Println("createScheduledQueryResult is successful")
    return *createScheduledQueryOutput.Arn, nil
}
return "", err
}

 func (timestreamBuilder TimestreamBuilder) CreateValidScheduledQuery(topicArn
 string, roleArn string, s3ErrorReportBucketName string,
     sqDatabaseName string, sqTableName string, databaseName string, tableName
 string) (string, error) {

     targetConfiguration := timestreamquery.TargetConfiguration{
         TimestreamConfiguration: &timestreamquery.TimestreamConfiguration{
             DatabaseName: aws.String(sqDatabaseName),
             TableName:    aws.String(sqTableName),
             TimeColumn:   aws.String("binned_timestamp"),
             DimensionMappings: []*timestreamquery.DimensionMapping{
                 {
                     Name:               aws.String("region"),
                     DimensionValueType: aws.String("VARCHAR"),
                 },
                 {
                     Name:               aws.String("az"),
                     DimensionValueType: aws.String("VARCHAR"),
                 },
                 {
                     Name:               aws.String("hostname"),
                     DimensionValueType: aws.String("VARCHAR"),
                 },
             },
             MultiMeasureMappings: &timestreamquery.MultiMeasureMappings{
                 TargetMultiMeasureName: aws.String("multi-metrics"),
                 MultiMeasureAttributeMappings:
 []*timestreamquery.MultiMeasureAttributeMapping{
                     {
                         SourceColumn:     aws.String("avg_cpu_utilization"),
                         MeasureValueType:
 aws.String(timestreamquery.MeasureValueTypeDouble),
                     },
                     {
                         SourceColumn:     aws.String("p90_cpu_utilization"),
```

```
                        MeasureValueType:
aws.String(timestreamquery.MeasureValueTypeDouble),
                    },
                    {
                        SourceColumn:     aws.String("p95_cpu_utilization"),
                        MeasureValueType:
aws.String(timestreamquery.MeasureValueTypeDouble),
                    },
                    {
                        SourceColumn:     aws.String("p99_cpu_utilization"),
                        MeasureValueType:
aws.String(timestreamquery.MeasureValueTypeDouble),
                    },
                },
            },
        },
    }
    return timestreamBuilder.createScheduledQuery(topicArn, roleArn,
s3ErrorReportBucketName,
        fmt.Sprintf(QUERY, databaseName, tableName), targetConfiguration)
}
```

Python

```
HOSTNAME = "host-24Gju"
SQ_NAME = "daily-sample"
ERROR_BUCKET_NAME = "scheduledquerysamplerrorbucket" +
 ''.join([choice(ascii_lowercase) for _ in range(5)])
QUERY = \
    "SELECT region, az, hostname, BIN(time, 15s) AS binned_timestamp, " \
    "    ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " \
    "    ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, "
 \
    "    ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization,
 " \
    "    ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization "
 \
    "FROM " + database_name + "." + table_name + " " \
    "WHERE measure_name = 'metrics' " \
    "AND hostname = '" + self.HOSTNAME + "' " \
    "AND time > ago(2h) " \
    "GROUP BY region, hostname, az, BIN(time, 15s) " \
    "ORDER BY binned_timestamp ASC " \
```

```
    "LIMIT 5"

def create_scheduled_query_helper(self, topic_arn, role_arn, query,
 target_configuration):
    print("\nCreating Scheduled Query")
    schedule_configuration = {
        'ScheduleExpression': 'cron(0/2 * * * ? *)'
    }
    notification_configuration = {
        'SnsConfiguration': {
            'TopicArn': topic_arn
        }
    }
    error_report_configuration = {
        'S3Configuration': {
            'BucketName': ERROR_BUCKET_NAME
        }
    }

    try:
        create_scheduled_query_response = \
            query_client.create_scheduled_query(Name=self.SQ_NAME,
                 QueryString=query,
                 ScheduleConfiguration=schedule_configuration,
                 NotificationConfiguration=notification_configuration,
                 TargetConfiguration=target_configuration,
                 ScheduledQueryExecutionRoleArn=role_arn,
                 ErrorReportConfiguration=error_report_configuration
                 )
        print("Successfully created scheduled query : ",
 create_scheduled_query_response['Arn'])
        return create_scheduled_query_response['Arn']
    except Exception as err:
        print("Scheduled Query creation failed:", err)
        raise err

def create_valid_scheduled_query(self, topic_arn, role_arn):
    target_configuration = {
        'TimestreamConfiguration': {
            'DatabaseName': self.sq_database_name,
            'TableName': self.sq_table_name,
            'TimeColumn': 'binned_timestamp',
            'DimensionMappings': [
                {'Name': 'region', 'DimensionValueType': 'VARCHAR'},
```

```
                {'Name': 'az', 'DimensionValueType': 'VARCHAR'},
                {'Name': 'hostname', 'DimensionValueType': 'VARCHAR'}
            ],
            'MultiMeasureMappings': {
                'TargetMultiMeasureName': 'target_name',
                'MultiMeasureAttributeMappings': [
                    {'SourceColumn': 'avg_cpu_utilization', 'MeasureValueType':
 'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'avg_cpu_utilization'},
                    {'SourceColumn': 'p90_cpu_utilization', 'MeasureValueType':
 'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'p90_cpu_utilization'},
                    {'SourceColumn': 'p95_cpu_utilization', 'MeasureValueType':
 'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'p95_cpu_utilization'},
                    {'SourceColumn': 'p99_cpu_utilization', 'MeasureValueType':
 'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'p99_cpu_utilization'},
                ]
            }
        }
    }

    return self.create_scheduled_query_helper(topic_arn, role_arn, QUERY,
 target_configuration)
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
const DATABASE_NAME = 'devops_sample_application';
const TABLE_NAME = 'host_metrics_sample_application';
const SQ_DATABASE_NAME = 'sq_result_database';
const SQ_TABLE_NAME = 'sq_result_table';
const HOSTNAME = "host-24Gju";
const SQ_NAME = "daily-sample";
const SCHEDULE_EXPRESSION = "cron(0/1 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
const VALID_QUERY = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
```

```
    " ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
    " ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
    " ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
    " ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
    "FROM " + DATABASE_NAME + "." + TABLE_NAME + " " +
    "WHERE measure_name = 'metrics' " +
    " AND hostname = '" + HOSTNAME + "' " +
    " AND time > ago(2h) " +
    "GROUP BY region, hostname, az, BIN(time, 15s) " +
    "ORDER BY binned_timestamp ASC " +
    "LIMIT 5";

async function createScheduledQuery(topicArn, roleArn, s3ErrorReportBucketName) {
    console.log("Creating Valid Scheduled Query");
    const DimensionMappingList = [{
            'Name': 'region',
            'DimensionValueType': 'VARCHAR'
        },
        {
            'Name': 'az',
            'DimensionValueType': 'VARCHAR'
        },
        {
            'Name': 'hostname',
            'DimensionValueType': 'VARCHAR'
        }
    ];

    const MultiMeasureMappings = {
        TargetMultiMeasureName: "multi-metrics",
        MultiMeasureAttributeMappings: [{
                'SourceColumn': 'avg_cpu_utilization',
                'MeasureValueType': 'DOUBLE'
            },
            {
                'SourceColumn': 'p90_cpu_utilization',
                'MeasureValueType': 'DOUBLE'
            },
            {
                'SourceColumn': 'p95_cpu_utilization',
                'MeasureValueType': 'DOUBLE'
            },
            {
                'SourceColumn': 'p99_cpu_utilization',
```

```
                'MeasureValueType': 'DOUBLE'
            },
        ]
    }

    const timestreamConfiguration = {
        DatabaseName: SQ_DATABASE_NAME,
        TableName: SQ_TABLE_NAME,
        TimeColumn: "binned_timestamp",
        DimensionMappings: DimensionMappingList,
        MultiMeasureMappings: MultiMeasureMappings
    }

    const createScheduledQueryRequest = {
        Name: SQ_NAME,
        QueryString: VALID_QUERY,
        ScheduleConfiguration: {
            ScheduleExpression: SCHEDULE_EXPRESSION
        },
        NotificationConfiguration: {
            SnsConfiguration: {
                TopicArn: topicArn
            }
        },
        TargetConfiguration: {
            TimestreamConfiguration: timestreamConfiguration
        },
        ScheduledQueryExecutionRoleArn: roleArn,
        ErrorReportConfiguration: {
            S3Configuration: {
                BucketName: s3ErrorReportBucketName
            }
        }
    };
    try {
        const data = await
queryClient.createScheduledQuery(createScheduledQueryRequest).promise();
        console.log("Successfully created scheduled query: " + data.Arn);
        return data.Arn;
    } catch (err) {
        console.log("Scheduled Query creation failed: ", err);
        throw err;
    }
```

```
}
```

.NET

```
public const string Hostname = "host-24Gju";
public const string SqName = "timestream-sample";
public const string SqDatabaseName = "sq_result_database";
public const string SqTableName = "sq_result_table";

public const string ErrorConfigurationS3BucketNamePrefix = "error-configuration-
sample-s3-bucket-";
public const string ScheduleExpression = "cron(0/2 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
public const string ValidQuery = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
      "ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
      "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
      "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, "
 +
      "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
      "FROM " + Constants.DATABASE_NAME + "." + Constants.TABLE_NAME + " " +
      "WHERE measure_name = 'metrics' " +
      "AND hostname = '" + Hostname + "' " +
      "AND time > ago(2h) " +
      "GROUP BY region, hostname, az, BIN(time, 15s) " +
      "ORDER BY binned_timestamp ASC " +
      "LIMIT 5";

private async Task<String> CreateValidScheduledQuery(string topicArn, string
 roleArn,
             string databaseName, string tableName, string s3ErrorReportBucketName)
 {
     List<MultiMeasureAttributeMapping> sourceColToMeasureValueTypes =
         new List<MultiMeasureAttributeMapping>()
         {
             new()
             {
                 SourceColumn = "avg_cpu_utilization",
                 MeasureValueType = MeasureValueType.DOUBLE.Value
             },
             new()
```

```
            {
                SourceColumn = "p90_cpu_utilization",
                MeasureValueType = MeasureValueType.DOUBLE.Value
            },
            new()
            {
                SourceColumn = "p95_cpu_utilization",
                MeasureValueType = MeasureValueType.DOUBLE.Value
            },
            new()
            {
                SourceColumn = "p99_cpu_utilization",
                MeasureValueType = MeasureValueType.DOUBLE.Value
            }
        };

    TargetConfiguration targetConfiguration = new TargetConfiguration()
    {
        TimestreamConfiguration = new TimestreamConfiguration()
        {
            DatabaseName = databaseName,
            TableName = tableName,
            TimeColumn = "binned_timestamp",
            DimensionMappings = new List<DimensionMapping>()
            {
                new()
                {
                    Name = "region",
                    DimensionValueType = "VARCHAR"
                },
                new()
                {
                    Name = "az",
                    DimensionValueType = "VARCHAR"
                },
                new()
                {
                    Name = "hostname",
                    DimensionValueType = "VARCHAR"
                }
            },
            MultiMeasureMappings = new MultiMeasureMappings()
            {
                TargetMultiMeasureName = "multi-metrics",
```

```
                MultiMeasureAttributeMappings = sourceColToMeasureValueTypes
            }
        }
    };
    return await CreateScheduledQuery(topicArn, roleArn, s3ErrorReportBucketName,
        ScheduledQueryConstants.ValidQuery, targetConfiguration);
}

private async Task<String> CreateScheduledQuery(string topicArn, string roleArn,
            string s3ErrorReportBucketName, string query, TargetConfiguration
 targetConfiguration)
{
    try
    {
        Console.WriteLine("Creating Scheduled Query");
        CreateScheduledQueryResponse response = await
 _amazonTimestreamQuery.CreateScheduledQueryAsync(
            new CreateScheduledQueryRequest()
            {
                Name = ScheduledQueryConstants.SqName,
                QueryString = query,
                ScheduleConfiguration = new ScheduleConfiguration()
                {
                    ScheduleExpression = ScheduledQueryConstants.ScheduleExpression
                },
                NotificationConfiguration = new NotificationConfiguration()
                {
                    SnsConfiguration = new SnsConfiguration()
                    {
                        TopicArn = topicArn
                    }
                },
                TargetConfiguration = targetConfiguration,
                ErrorReportConfiguration = new ErrorReportConfiguration()
                {
                    S3Configuration = new S3Configuration()
                    {
                        BucketName = s3ErrorReportBucketName
                    }
                },
                ScheduledQueryExecutionRoleArn = roleArn
            });
        Console.WriteLine($"Successfully created scheduled query :
 {response.Arn}");
```

```
            return response.Arn;
        }
        catch (Exception e)
        {
            Console.WriteLine($"Scheduled Query creation failed: {e}");
            throw;
        }
    }
```

列出排程查詢

您可以使用下列程式碼片段來列出已排程的查詢。

Java

```
public void listScheduledQueries() {
    System.out.println("Listing Scheduled Query");
    try {
        String nextToken = null;
        List<String> scheduledQueries = new ArrayList<>();

        do {
            ListScheduledQueriesResult listScheduledQueriesResult =
                    queryClient.listScheduledQueries(new
 ListScheduledQueriesRequest()
                            .withNextToken(nextToken).withMaxResults(10));
            List<ScheduledQuery> scheduledQueryList =
 listScheduledQueriesResult.getScheduledQueries();

            printScheduledQuery(scheduledQueryList);
            nextToken = listScheduledQueriesResult.getNextToken();
        } while (nextToken != null);
    }
    catch (Exception e) {
        System.out.println("List Scheduled Query failed: " + e);
        throw e;
    }
}

public void printScheduledQuery(List<ScheduledQuery> scheduledQueryList) {
    for (ScheduledQuery scheduledQuery: scheduledQueryList) {
        System.out.println(scheduledQuery.getArn());
```

```
    }
}
```

Java v2

```
public void listScheduledQueries() {
    System.out.println("Listing Scheduled Query");
    try {
        String nextToken = null;

        do {
            ListScheduledQueriesResponse listScheduledQueriesResult =

 queryClient.listScheduledQueries(ListScheduledQueriesRequest.builder()
                            .nextToken(nextToken).maxResults(10)
                            .build());
            List<ScheduledQuery> scheduledQueryList =
 listScheduledQueriesResult.scheduledQueries();

            printScheduledQuery(scheduledQueryList);
            nextToken = listScheduledQueriesResult.nextToken();
        } while (nextToken != null);
    }
    catch (Exception e) {
        System.out.println("List Scheduled Query failed: " + e);
        throw e;
    }
}

public void printScheduledQuery(List<ScheduledQuery> scheduledQueryList) {
    for (ScheduledQuery scheduledQuery: scheduledQueryList) {
        System.out.println(scheduledQuery.arn());
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) ListScheduledQueries()
 ([]*timestreamquery.ScheduledQuery, error) {

     var nextToken *string = nil
     var scheduledQueries []*timestreamquery.ScheduledQuery
     for ok := true; ok; ok = nextToken != nil {
```

```
        listScheduledQueriesInput := &timestreamquery.ListScheduledQueriesInput{
            MaxResults: aws.Int64(15),
        }
        if nextToken != nil {
            listScheduledQueriesInput.NextToken = aws.String(*nextToken)
        }

        listScheduledQueriesOutput, err :=
 timestreamBuilder.QuerySvc.ListScheduledQueries(listScheduledQueriesInput)
        if err != nil {
            fmt.Printf("Error: %s", err.Error())
            return nil, err
        }
        scheduledQueries = append(scheduledQueries,
 listScheduledQueriesOutput.ScheduledQueries...)
        nextToken = listScheduledQueriesOutput.NextToken
    }
    return scheduledQueries, nil
}
```

Python

```
def list_scheduled_queries(self):
    print("\nListing Scheduled Queries")
    try:
        response = self.query_client.list_scheduled_queries(MaxResults=10)
        self.print_scheduled_queries(response['ScheduledQueries'])
        next_token = response.get('NextToken', None)
        while next_token:
            response =
 self.query_client.list_scheduled_queries(NextToken=next_token, MaxResults=10)
            self.print_scheduled_queries(response['ScheduledQueries'])
            next_token = response.get('NextToken', None)
    except Exception as err:
        print("List scheduled queries failed:", err)
        raise err

@staticmethod
def print_scheduled_queries(scheduled_queries):
    for scheduled_query in scheduled_queries:
        print(scheduled_query['Arn'])
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function listScheduledQueries() {
     console.log("Listing Scheduled Query");
     try {
         var nextToken = null;
         do {
             var params = {
                 MaxResults: 10,
                 NextToken: nextToken
             }
             var data = await queryClient.listScheduledQueries(params).promise();
             var scheduledQueryList = data.ScheduledQueries;
             printScheduledQuery(scheduledQueryList);
             nextToken = data.NextToken;
         }
         while (nextToken != null);
     }  catch (err) {
         console.log("List Scheduled Query failed: ", err);
         throw err;
     }
 }

 async function printScheduledQuery(scheduledQueryList) {
     scheduledQueryList.forEach(element => console.log(element.Arn));
 }
```

.NET

```
private async Task ListScheduledQueries()
 {
     try
     {
         Console.WriteLine("Listing Scheduled Query");
         string nextToken;
         do
         {
             ListScheduledQueriesResponse response =
                 await _amazonTimestreamQuery.ListScheduledQueriesAsync(new
 ListScheduledQueriesRequest());
```

```
            foreach (var scheduledQuery in response.ScheduledQueries)
            {
                Console.WriteLine($"{scheduledQuery.Arn}");
            }

            nextToken = response.NextToken;
        } while (nextToken != null);
    }
    catch (Exception e)
    {
        Console.WriteLine($"List Scheduled Query failed: {e}");
        throw;
    }
}
```

描述排程查詢

您可以使用下列程式碼片段來描述排程查詢。

Java

```
public void describeScheduledQueries(String scheduledQueryArn) {
    System.out.println("Describing Scheduled Query");
    try {
        DescribeScheduledQueryResult describeScheduledQueryResult =
 queryClient.describeScheduledQuery(new
 DescribeScheduledQueryRequest().withScheduledQueryArn(scheduledQueryArn));
        System.out.println(describeScheduledQueryResult);
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Describe Scheduled Query failed: " + e);
        throw e;
    }
}
```

Java v2

```
public void describeScheduledQueries(String scheduledQueryArn) {
```

```
    System.out.println("Describing Scheduled Query");
    try {
        DescribeScheduledQueryResponse describeScheduledQueryResult =

 queryClient.describeScheduledQuery(DescribeScheduledQueryRequest.builder()
                        .scheduledQueryArn(scheduledQueryArn)
                        .build());
        System.out.println(describeScheduledQueryResult);
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Describe Scheduled Query failed: " + e);
        throw e;
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) DescribeScheduledQuery(scheduledQueryArn
 string) error {

    describeScheduledQueryInput := &timestreamquery.DescribeScheduledQueryInput{
        ScheduledQueryArn: aws.String(scheduledQueryArn),
    }
    describeScheduledQueryOutput, err :=
 timestreamBuilder.QuerySvc.DescribeScheduledQuery(describeScheduledQueryInput)

    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case timestreamquery.ErrCodeResourceNotFoundException:
                fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
 aerr.Error())
            default:
                fmt.Printf("Error: %s", err.Error())
            }
        } else {
            fmt.Printf("Error: %s", aerr.Error())
        }
        return err
```

```
    } else {
        fmt.Println("DescribeScheduledQuery is successful, below is the output:")
        fmt.Println(describeScheduledQueryOutput.ScheduledQuery)
        return nil
    }
}
```

Python

```
def describe_scheduled_query(self, scheduled_query_arn):
    print("\nDescribing Scheduled Query")
    try:
        response =
 self.query_client.describe_scheduled_query(ScheduledQueryArn=scheduled_query_arn)
        if 'ScheduledQuery' in response:
            response = response['ScheduledQuery']
            for key in response:
                print("{} :{}".format(key, response[key]))
    except self.query_client.exceptions.ResourceNotFoundException as err:
        print("Scheduled Query doesn't exist")
        raise err
    except Exception as err:
        print("Scheduled Query describe failed:", err)
        raise err
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function describeScheduledQuery(scheduledQueryArn) {
    console.log("Describing Scheduled Query");
    var params = {
        ScheduledQueryArn: scheduledQueryArn
    }
    try {
        const data = await queryClient.describeScheduledQuery(params).promise();
        console.log(data.ScheduledQuery);
    } catch (err) {
        console.log("Describe Scheduled Query failed: ", err);
        throw err;
    }
}
```

.NET

```
private async Task DescribeScheduledQuery(string scheduledQueryArn)
 {
     try
     {
         Console.WriteLine("Describing Scheduled Query");
         DescribeScheduledQueryResponse response = await
 _amazonTimestreamQuery.DescribeScheduledQueryAsync(
             new DescribeScheduledQueryRequest()
             {
                 ScheduledQueryArn = scheduledQueryArn
             });

 Console.WriteLine($"{JsonConvert.SerializeObject(response.ScheduledQuery)}");
     }
     catch (ResourceNotFoundException e)
     {
         Console.WriteLine($"Scheduled Query doesn't exist: {e}");
         throw;
     }
     catch (Exception e)
     {
         Console.WriteLine($"Describe Scheduled Query failed: {e}");
         throw;
     }
 }
```

執行排程查詢

您可以使用下列程式碼片段來執行排程查詢。

Java

```
public void executeScheduledQueries(String scheduledQueryArn, Date invocationTime) {
    System.out.println("Executing Scheduled Query");
    try {
        ExecuteScheduledQueryResult executeScheduledQueryResult =
 queryClient.executeScheduledQuery(new ExecuteScheduledQueryRequest()
                .withScheduledQueryArn(scheduledQueryArn)
                .withInvocationTime(invocationTime)
        );
```

```
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Java v2

```
public void executeScheduledQuery(String scheduledQueryArn) {
    System.out.println("Executing Scheduled Query");
    try {
        ExecuteScheduledQueryResponse executeScheduledQueryResult =
 queryClient.executeScheduledQuery(ExecuteScheduledQueryRequest.builder()
                .scheduledQueryArn(scheduledQueryArn)
                .invocationTime(Instant.now())
                .build()
        );

        System.out.println("Execute ScheduledQuery response code: " +
 executeScheduledQueryResult.sdkHttpResponse().statusCode());

    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) ExecuteScheduledQuery(scheduledQueryArn
 string, invocationTime time.Time) error {
```

```
    executeScheduledQueryInput := &timestreamquery.ExecuteScheduledQueryInput{
        ScheduledQueryArn: aws.String(scheduledQueryArn),
        InvocationTime:    aws.Time(invocationTime),
    }
    executeScheduledQueryOutput, err :=
timestreamBuilder.QuerySvc.ExecuteScheduledQuery(executeScheduledQueryInput)

    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case timestreamquery.ErrCodeResourceNotFoundException:
                fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
aerr.Error())
            default:
                fmt.Printf("Error: %s", aerr.Error())
            }
        } else {
            fmt.Printf("Error: %s", err.Error())
        }
        return err
    } else {
        fmt.Println("ExecuteScheduledQuery is successful, below is the output:")
        fmt.Println(executeScheduledQueryOutput.GoString())
        return nil
    }
}
```

Python

```
def execute_scheduled_query(self, scheduled_query_arn, invocation_time):
    print("\nExecuting Scheduled Query")
    try:

 self.query_client.execute_scheduled_query(ScheduledQueryArn=scheduled_query_arn,
 InvocationTime=invocation_time)
        print("Successfully started executing scheduled query")
    except self.query_client.exceptions.ResourceNotFoundException as err:
        print("Scheduled Query doesn't exist")
        raise err
    except Exception as err:
        print("Scheduled Query execution failed:", err)
        raise err
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function executeScheduledQuery(scheduledQueryArn, invocationTime) {
     console.log("Executing Scheduled Query");
     var params = {
         ScheduledQueryArn: scheduledQueryArn,
         InvocationTime: invocationTime
     }
     try {
         await queryClient.executeScheduledQuery(params).promise();
     } catch (err) {
         console.log("Execute Scheduled Query failed: ", err);
         throw err;
     }
 }
```

.NET

```
private async Task ExecuteScheduledQuery(string scheduledQueryArn, DateTime
 invocationTime)
 {
     try
     {
         Console.WriteLine("Running Scheduled Query");
         await _amazonTimestreamQuery.ExecuteScheduledQueryAsync(new
 ExecuteScheduledQueryRequest()
         {
             ScheduledQueryArn = scheduledQueryArn,
             InvocationTime = invocationTime
         });
         Console.WriteLine("Successfully started manual run of scheduled query");
     }
     catch (ResourceNotFoundException e)
     {
         Console.WriteLine($"Scheduled Query doesn't exist: {e}");
         throw;
     }
     catch (Exception e)
     {
         Console.WriteLine($"Execute Scheduled Query failed: {e}");
```

```
            throw;
        }
    }
```

更新排程查詢

您可以使用下列程式碼片段來更新排程查詢。

Java

```
public void updateScheduledQueries(String scheduledQueryArn) {
    System.out.println("Updating Scheduled Query");
    try {
        queryClient.updateScheduledQuery(new UpdateScheduledQueryRequest()
                .withScheduledQueryArn(scheduledQueryArn)
                .withState(ScheduledQueryState.DISABLED));
        System.out.println("Successfully update scheduled query state");
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Java v2

```
public void updateScheduledQuery(String scheduledQueryArn, ScheduledQueryState
 state) {
    System.out.println("Updating Scheduled Query");
    try {
        queryClient.updateScheduledQuery(UpdateScheduledQueryRequest.builder()
                .scheduledQueryArn(scheduledQueryArn)
                .state(state)
                .build());
        System.out.println("Successfully update scheduled query state");
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
```

```
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) UpdateScheduledQuery(scheduledQueryArn
 string) error {

    updateScheduledQueryInput := &timestreamquery.UpdateScheduledQueryInput{
        ScheduledQueryArn: aws.String(scheduledQueryArn),
        State:             aws.String(timestreamquery.ScheduledQueryStateDisabled),
    }
    _, err :=
 timestreamBuilder.QuerySvc.UpdateScheduledQuery(updateScheduledQueryInput)

    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case timestreamquery.ErrCodeResourceNotFoundException:
                fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
 aerr.Error())
            default:
                fmt.Printf("Error: %s", aerr.Error())
            }
        } else {
            fmt.Printf("Error: %s", err.Error())
        }
        return err
    } else {
        fmt.Println("UpdateScheduledQuery is successful")
        return nil
    }
}
```

Python

```
def update_scheduled_query(self, scheduled_query_arn, state):
    print("\nUpdating Scheduled Query")
```

```
    try:

 self.query_client.update_scheduled_query(ScheduledQueryArn=scheduled_query_arn,
                                                    State=state)
        print("Successfully update scheduled query state to", state)
    except self.query_client.exceptions.ResourceNotFoundException as err:
        print("Scheduled Query doesn't exist")
        raise err
    except Exception as err:
        print("Scheduled Query deletion failed:", err)
        raise err
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function updateScheduledQueries(scheduledQueryArn) {
     console.log("Updating Scheduled Query");
     var params = {
         ScheduledQueryArn: scheduledQueryArn,
         State: "DISABLED"
     }
     try {
         await queryClient.updateScheduledQuery(params).promise();
         console.log("Successfully update scheduled query state");
     } catch (err) {
         console.log("Update Scheduled Query failed: ", err);
         throw err;
     }
 }
```

.NET

```
private async Task UpdateScheduledQuery(string scheduledQueryArn,
 ScheduledQueryState state)
 {
     try
     {
         Console.WriteLine("Updating Scheduled Query");
         await _amazonTimestreamQuery.UpdateScheduledQueryAsync(new
 UpdateScheduledQueryRequest()
         {
```

```
            ScheduledQueryArn = scheduledQueryArn,
            State = state
        });
        Console.WriteLine("Successfully update scheduled query state");
    }
    catch (ResourceNotFoundException e)
    {
        Console.WriteLine($"Scheduled Query doesn't exist: {e}");
        throw;
    }
    catch (Exception e)
    {
        Console.WriteLine($"Update Scheduled Query failed: {e}");
        throw;
    }
}
```

刪除排程查詢

您可以使用下列程式碼片段來刪除排程查詢。

Java

```
public void deleteScheduledQuery(String scheduledQueryArn) {
    System.out.println("Deleting Scheduled Query");

    try {
        queryClient.deleteScheduledQuery(new
 DeleteScheduledQueryRequest().withScheduledQueryArn(scheduledQueryArn));
        System.out.println("Successfully deleted scheduled query");
    }
    catch (Exception e) {
        System.out.println("Scheduled Query deletion failed: " + e);
    }
}
```

Java v2

```
public void deleteScheduledQuery(String scheduledQueryArn) {
    System.out.println("Deleting Scheduled Query");

    try {
```

```
        queryClient.deleteScheduledQuery(DeleteScheduledQueryRequest.builder()
                .scheduledQueryArn(scheduledQueryArn).build());
        System.out.println("Successfully deleted scheduled query");
    }
    catch (Exception e) {
        System.out.println("Scheduled Query deletion failed: " + e);
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) DeleteScheduledQuery(scheduledQueryArn
 string) error {

    deleteScheduledQueryInput := &timestreamquery.DeleteScheduledQueryInput{
        ScheduledQueryArn: aws.String(scheduledQueryArn),
    }
    _, err :=
 timestreamBuilder.QuerySvc.DeleteScheduledQuery(deleteScheduledQueryInput)

    if err != nil {
        fmt.Println("Error:")
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case timestreamquery.ErrCodeResourceNotFoundException:
                fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
 aerr.Error())
            default:
                fmt.Printf("Error: %s", aerr.Error())
            }
        } else {
            fmt.Printf("Error: %s", err.Error())
        }
        return err
    } else {
        fmt.Println("DeleteScheduledQuery is successful")
        return nil
    }
}
```

Python

```
def delete_scheduled_query(self, scheduled_query_arn):
```

```
    print("\nDeleting Scheduled Query")
    try:

 self.query_client.delete_scheduled_query(ScheduledQueryArn=scheduled_query_arn)
        print("Successfully deleted scheduled query :", scheduled_query_arn)
    except Exception as err:
        print("Scheduled Query deletion failed:", err)
        raise err
```

Node.js

下列程式碼片段使用 AWS SDK適用於 JavaScript V2 樣式的 。它以 Node.js 範例應用程式為基礎，適用於 上的 LiveAnalytics 應用程式 GitHub。

```
async function deleteScheduleQuery(scheduledQueryArn) {
     console.log("Deleting Scheduled Query");
     const params = {
         ScheduledQueryArn: scheduledQueryArn
     }
     try {
         await queryClient.deleteScheduledQuery(params).promise();
         console.log("Successfully deleted scheduled query");
     } catch (err) {
         console.log("Scheduled Query deletion failed: ", err);
     }
 }
```

.NET

```
private async Task DeleteScheduledQuery(string scheduledQueryArn)
 {
     try
     {
         Console.WriteLine("Deleting Scheduled Query");
         await _amazonTimestreamQuery.DeleteScheduledQueryAsync(new
 DeleteScheduledQueryRequest()
         {
             ScheduledQueryArn = scheduledQueryArn
         });
         Console.WriteLine($"Successfully deleted scheduled query :
 {scheduledQueryArn}");
     }
     catch (Exception e)
```

```
    {
        Console.WriteLine($"Scheduled Query deletion failed: {e}");
        throw;
    }
}
```

在 Timestream 中使用 的批次載入 LiveAnalytics

透過 Amazon Timestream 的批次載入 LiveAnalytics，您可以將存放在 Amazon S3 中的CSV檔案以批次方式擷取到 Timestream。透過這項新功能，您可以在 Timestream 中為 取得資料， LiveAnalytics 而不必依賴其他工具或編寫自訂程式碼。您可以使用批次載入，以彈性的等待時間回填資料，例如查詢或分析時不需要的資料。

您可以使用 AWS Management Console、 AWS CLI和 來建立批次載入任務 AWS SDKs。如需詳細資訊，請參閱搭配主控台使用批次載入、搭配 使用批次載入 AWS CLI及搭配 使用批次載入 AWS SDKs。

除了批次載入之外，您還可以使用 WriteRecords API操作同時寫入多個記錄。如需使用哪些項目的指引，請參閱 在操作和批次載入之間 WriteRecords API進行選擇。

主題

- Timestream 中的批次載入概念
- 批次載入先決條件
- 批次載入最佳實務
- 準備批次載入資料檔案
- 批次載入的資料模型映射
- 搭配主控台使用批次載入
- 搭配 使用批次載入 AWS CLI
- 搭配 使用批次載入 AWS SDKs
- 使用批次載入錯誤報告

Timestream 中的批次載入概念

檢閱下列概念，以進一步了解批次載入功能。

批次載入任務 – 在 Amazon Timestream 中定義來源資料和目的地的任務。您可以在建立批次載入任務時指定其他組態，例如資料模型。您可以透過 AWS Management Console、 AWS CLI和 建立批次載入任務 AWS SDKs。

匯入目的地 – Timestream 中的目的地資料庫和資料表。如需有關建立資料庫和資料表的資訊，請參閱建立 資料庫和 建立資料表。

資料來源 – 存放在 S3 儲存貯體中的來源CSV檔案。如需有關準備資料檔案的資訊，請參閱 準備批次載入資料檔案。如需有關 S3 定價的資訊，請參閱 Amazon S3 定價。

批次載入錯誤報告 – 儲存批次載入任務錯誤相關資訊的報告。您可以在批次載入任務中定義批次載入錯誤報告的 S3 位置。如需報告中的資訊，請參閱 使用批次載入錯誤報告。

資料模型映射 – 從 S3 位置的資料來源到 LiveAnalytics 資料表目標時間串流的時間、維度和量值的批次載入映射。如需詳細資訊，請參閱批次載入的資料模型映射。

批次載入先決條件

這是使用批次載入的先決條件清單。如需最佳實務做法，請參閱「批次載入最佳實務」。

- 批次載入來源資料會以具有標頭的CSV格式儲存在 Amazon S3 中。
- 對於每個 Amazon S3 來源儲存貯體，您必須在附加政策中擁有下列許可：

```
"s3:GetObject",
"s3:GetBucketAcl"
"s3:ListBucket"
```

同樣地，針對寫入報告的每個 Amazon S3 輸出儲存貯體，您必須在附加政策中擁有下列許可：

```
"s3:PutObject",
"s3:GetBucketAcl"
```

例如：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject",
        "s3:GetBucketAcl"
```

```
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::inputs-source-bucket-name-A”
        "arn:aws:s3:::inputs-source-bucket-name-B”
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "s3:PutObject",
        "s3:GetBucketAcl"
      ],
      "Resource": [
        "arn:aws:s3:::reports-output-bucket-name”
      ]
      "Effect": "Allow"
    }
  ]
}
```

- 的時間串流CSV會將資料模型中提供的資訊映射至CSV標頭，以 LiveAnalytics 剖析 。資料必須具有代表時間戳記的欄、至少一個維度欄和至少一個量值欄。
- 與批次載入搭配使用的 S3 儲存貯體必須位於與批次載入所用 LiveAnalytics 資料表的 Timestream 相同的 區域，且來自相同的 帳戶。
- `timestamp` 資料欄必須是長資料類型，代表自 Unix epoch 以來的時間。例如，時間戳記`2021-03-25T08:45:21Z`會以 表示`1616661921`。Timestream 支援時間戳記精確度的秒、毫秒、微秒和奈秒。使用查詢語言時，您可以在具有 等函數的格式之間轉換`to_unixtime`。如需詳細資訊，請參閱日期/時間函數。
- Timestream 支援維度值的字串資料類型。它支援長、雙、字串和布林資料類型來測量資料欄。

如需批次載入限制和配額，請參閱 批次載入。

批次載入最佳實務

遵循下列條件和建議時，批次載入效果最佳 （ 高輸送量 ）：

1. CSV 提交供擷取的檔案很小，特別是檔案大小為 100MB – 1GB，以改善平行處理和擷取速度。
2. 當批次載入進行中時，避免同時將資料擷取至相同的資料表 （ 例如，使用 WriteRecords API 操作或排程查詢 ）。這可能會導致限流，批次載入任務將會失敗。

3. 在批次載入任務執行時，請勿從批次載入中使用的 S3 儲存貯體新增、修改或移除檔案。
4. 請勿刪除或撤銷資料表或來源的許可，或報告已排程或進行中批次載入任務的 S3 儲存貯體。
5. 擷取具有高維度值組的資料時，請遵循 的指示分割多測量記錄的建議。
6. 請務必提交小型檔案來測試資料是否正確。無論正確性如何，提交批次載入的任何資料都會向您收取費用。如需定價的詳細資訊，請參閱 Amazon Timestream 定價。
7. 除非`ActiveMagneticStorePartitions`低於 250，否則請勿繼續批次載入任務。任務可能會受到調節並失敗。同時為相同的資料庫提交多個任務應減少數字。

以下是主控台最佳實務：

1. 僅將建置器用於更簡單的資料建模，該建模僅針對多量值記錄使用一個量值名稱。
2. 如需更複雜的資料建模，請使用 JSON。例如，當您在使用多量值記錄JSON時使用多個量值名稱時使用 。

如需 LiveAnalytics 最佳實務的其他 Timestream，請參閱 最佳實務。

準備批次載入資料檔案

來源資料檔案具有分隔符號分隔值。更具體的術語逗號分隔值（CSV）是通用的。有效的資料欄分隔符號包括逗號和管道。記錄會以新行分隔。檔案必須存放在 Amazon S3 中。當您建立新的批次載入任務時， 會ARN為 檔案指定來源資料的位置。檔案包含標頭。一欄代表時間戳記。至少有一個其他資料欄代表量值。

與批次載入搭配使用的 S3 儲存貯體必須與批次載入中使用的 LiveAnalytics 資料表的 Timestream 位於相同區域。提交批次載入任務後，請勿從批次載入中使用的 S3 儲存貯體新增或移除檔案。如需使用 S3 儲存貯體的相關資訊，請參閱 Amazon S3 入門。

> **Note**
>
> CSV 某些應用程式產生的檔案，例如 Excel，可能包含與預期編碼衝突的位元組順序標記（BOM）。 LiveAnalytics 批次載入任務的時間串流，該任務在以程式設計方式處理時參考具有BOM擲回錯誤CSV的檔案。若要避免這種情況，您可以移除 BOM，這是不可見的字元。例如，您可以從 Notepad++ 等應用程式儲存檔案，該應用程式可讓您指定新的編碼。您也可以使用程式設計選項來讀取第一行、從該行移除字元，以及將新值寫入檔案的第一行。
> 從 Excel 儲存時，有多個CSV選項。使用不同的CSV選項儲存可能會阻止所描述的問題。但您應該檢查結果，因為編碼的變更可能會影響某些字元。

CSV 格式參數

當您表示格式參數以其他方式保留的值時，可以使用逸出字元。例如，如果引號字元是雙引號，若要在資料中代表雙引號，請將逸出字元放在雙引號之前。

如需建立批次載入任務時何時指定這些項目的詳細資訊，請參閱 建立批次載入任務。

參數	選項
資料欄分隔符號	（逗號（','）| 管道（'|'）| 分號（';'）| 索引標籤（'/t'）| 空白空間（''））
逸出字元	無
引號字元	主控台：（雙引號（"）| 單引號（'））
Null 值	空白空間（''）
修剪空白空間	主控台：（否 | 是）

批次載入的資料模型映射

以下內容討論資料模型映射的結構描述，並提供 和 範例。

資料模型映射結構描述

`CreateBatchLoadTask` 請求語法和呼叫傳回的`BatchLoadTaskDescription`物件，以`DescribeBatchLoadTask`包含包含`DataModel`批次載入的 `DataModelConfiguration` 物件。`DataModel` 定義從以 S3 位置CSV格式儲存的來源資料到 LiveAnalytics 資料庫和資料表目標時間串流的映射。

`TimeColumn` 欄位指出要映射到 Timestream 中目的地資料表`time`資料欄的值的來源資料位置 LiveAnalytics。`TimeUnit` 指定 的單位`TimeColumn`，並且可以是`MILLISECONDS`、`MICROSECONDS`、 `SECONDS`或 之一`NANOSECONDS`。還有維度和量值的映射。維度映射由來源資料欄和目標欄位組成。

如需詳細資訊，請參閱 DimensionMapping。量值的映射有兩個選項，`MixedMeasureMappings`以及`MultiMeasureMappings`。

總而言之， DataModel包含從 S3 位置的資料來源到目標時間串流的 LiveAnalytics 對應，用於下列項目。

- 時間
- 維度
- 量值

如果可能，建議您將測量資料映射到 Timestream 中的多測量記錄 LiveAnalytics。如需多測量記錄優點的相關資訊，請參閱 多測量記錄。

如果來源資料中的多個量值儲存在一列中，您可以將這些多個量值映射到 Timestream 中的多量值記錄，以便 LiveAnalytics 使用 MultiMeasureMappings。如果有必須映射至單一測量記錄的值，您可以使用 MixedMeasureMappings。

MixedMeasureMappings MultiMeasureMappings兩者都包含 MultiMeasureAttributeMappings。無論是否需要單一測量記錄，都支援多測量記錄。

如果 的 Timestream 中只需要多量值目標記錄 LiveAnalytics，您可以在下列結構中定義量值映射。

```
CreateBatchLoadTask
    MeasureNameColumn
    MultiMeasureMappings
        TargetMultiMeasureName
        MultiMeasureAttributeMappings array
```

> **Note**
>
> 我們建議您MultiMeasureMappings盡可能使用 。

如果 的 Timestream 中需要單一量值目標記錄 LiveAnalytics，您可以在下列結構中定義量值映射。

```
CreateBatchLoadTask
    MeasureNameColumn
    MixedMeasureMappings array
        MixedMeasureMapping
            MeasureName
            MeasureValueType
            SourceColumn
```

```
            TargetMeasureName
            MultiMeasureAttributeMappings array
```

使用 時MultiMeasureMappings，陣列一律MultiMeasureAttributeMappings為必要項目。當您使用MixedMeasureMappings陣列時，如果 MeasureValueType MULTI是指定 的 MixedMeasureMapping，MultiMeasureAttributeMappings則該 需要 MixedMeasureMapping。否則， 會MeasureValueType指示單一量值記錄的量值類型。

無論哪種方式，都有MultiMeasureAttributeMapping可用的陣列。您可以定義對應，以多測量每個 中的記錄MultiMeasureAttributeMapping，如下所示：

SourceColumn

位於 Amazon S3 的來源資料中的資料欄。

TargetMultiMeasureAttributeName

目的地資料表中目標多量值名稱的名稱。MeasureNameColumn 未提供時，需要此輸入。MeasureNameColumn 如果提供 ，該資料欄的值會用作多量值名稱。

MeasureValueType

DOUBLE、BIGINTBOOLEAN、 VARCHAR或 之一TIMESTAMP。

具有MultiMeasureMappings範例的資料模型映射

此範例示範對應多測量記錄，這是偏好的方法，可將每個測量值存放在專用欄中。您可以在範例 下載CSV範例。 CSV範例具有下列標題，可對應至 Timestream 中 LiveAnalytics 資料表的目標欄。

- time
- measure_name
- region
- location
- hostname
- memory_utilization
- cpu_utilization

識別CSV檔案中的 time和 measure_name資料欄。在此情況下，這些會直接映射至相同名稱之 LiveAnalytics 資料表資料欄的 Timestream。

- time 對應至 time
- measure_name 映射至 measure_name（或您選擇的值）

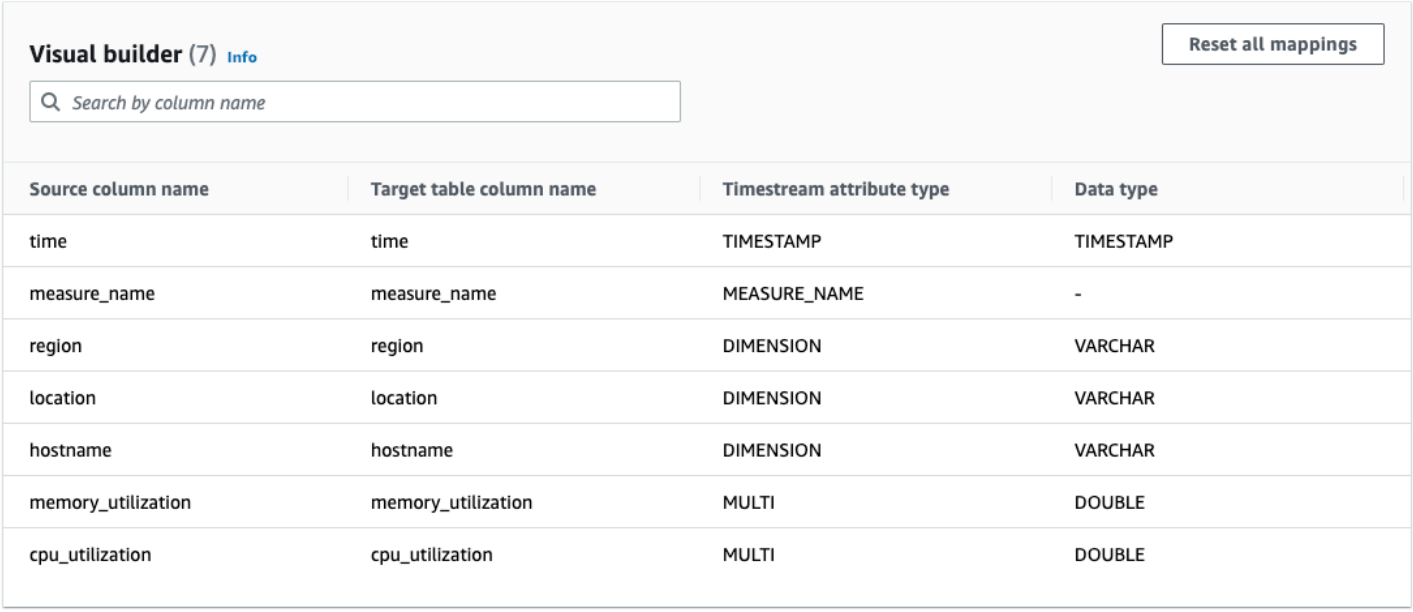
使用 時API，您可以在 TimeColumn time欄位中指定 ，並在 TimeUnit 欄位MILLISECONDS指定支援的時間單位值，例如 。這些對應至主控台中的來源資料欄名稱和時間戳記時間輸入。您可以使用MeasureNameColumn金鑰measure_name定義的 來分組或分割記錄。

在範例中，region、 location和 hostname是維度。維度會映射在DimensionMapping物件陣列中。

對於量值，該值TargetMultiMeasureAttributeName將成為 LiveAnalytics 資料表時間串流中的資料欄。您可以保留相同的名稱，例如在此範例中。或者，您可以指定新的 。 MeasureValueType是DOUBLE、BIGINT、VARCHAR、 BOOLEAN或 之一TIMESTAMP。

```
{
  "TimeColumn": "time",
  "TimeUnit": "MILLISECONDS",
  "DimensionMappings": [
    {
      "SourceColumn": "region",
      "DestinationColumn": "region"
    },
    {
      "SourceColumn": "location",
      "DestinationColumn": "location"
    },
    {
      "SourceColumn": "hostname",
      "DestinationColumn": "hostname"
    }
  ],
  "MeasureNameColumn": "measure_name",
  "MultiMeasureMappings": {
    "MultiMeasureAttributeMappings": [
      {
        "SourceColumn": "memory_utilization",
        "TargetMultiMeasureAttributeName": "memory_utilization",
        "MeasureValueType": "DOUBLE"
      },
      {
        "SourceColumn": "cpu_utilization",
        "TargetMultiMeasureAttributeName": "cpu_utilization",
```

```
        "MeasureValueType": "DOUBLE"
      }
    ]
  }
}
```



Visual builder (7) Info

Reset all mappings

Search by column name

Source column name	Target table column name	Timestream attribute type	Data type
time	time	TIMESTAMP	TIMESTAMP
measure_name	measure_name	MEASURE_NAME	-
region	region	DIMENSION	VARCHAR
location	location	DIMENSION	VARCHAR
hostname	hostname	DIMENSION	VARCHAR
memory_utilization	memory_utilization	MULTI	DOUBLE
cpu_utilization	cpu_utilization	MULTI	DOUBLE

具有MixedMeasureMappings範例的資料模型映射

建議您僅在需要映射至 Timestream 中 的單一測量記錄時，才使用此方法 LiveAnalytics。

搭配主控台使用批次載入

以下是搭配 使用批次載入的步驟 AWS Management Console。您可以在範例 下載CSV範例。 CSV

主題

- 存取批次載入
- 建立批次載入任務
- 恢復批次載入任務
- 使用視覺化建置器

存取批次載入

請依照下列步驟，使用 存取批次載入 AWS Management Console。

1. 開啟 Amazon Timestream 主控台 。
2. 在導覽窗格中，選擇 管理工具 ，然後選擇 批次載入任務 。
3. 從這裡，您可以檢視批次載入任務的清單，並深入探索指定的任務，以取得更多詳細資訊。您也可以建立和繼續任務。

建立批次載入任務

請依照下列步驟，使用 建立批次載入任務 AWS Management Console。

1. 開啟 Amazon Timestream 主控台 。
2. 在導覽窗格中，選擇 管理工具 ，然後選擇 批次載入任務 。
3. 選擇建立批次載入任務 。
4. 在匯入目的地 中，選擇下列項目。

   - 目標資料庫 – 選取在 中建立的資料庫名稱建立 資料庫。
   - 目標資料表 – 選取在 中建立的資料表名稱建立資料表。

   如有必要，您可以使用建立新資料表按鈕從此面板新增資料表。
5. 從資料來源 中的資料來源 S3 位置，選取存放來源資料的 S3 儲存貯體。 使用瀏覽 S3 按鈕來檢視作用中AWS帳戶可存取的 S3 資源，或輸入 S3 位置 URL。資料來源必須位於相同的區域。
6. 在檔案格式設定 （ 可擴展區段） 中，您可以使用預設設定來剖析輸入資料。您也可以選擇進階設定 。您可以從中選擇CSV格式參數 ，然後選擇參數來剖析輸入資料。如需有關這些參數的資訊，請參閱 CSV 格式參數。
7. 從設定資料模型映射 中，設定資料模型。如需其他資料模型指南，請參閱 批次載入的資料模型映射

   - 從資料模型映射 中，選擇映射組態輸入 ，然後選擇下列其中一項。
     - 視覺化建置器 – 若要以視覺化方式映射資料，請選擇 TargetMultiMeasureName或MeasureNameColumn。然後，從視覺化建置器 中，映射資料欄。

       將單一檔案選取為資料來源時，視覺化建置器會自動偵測來源資料欄標頭，並從資料來源CSV檔案載入。選擇屬性和資料類型以建立映射。

       如需使用視覺化建置器的資訊，請參閱 使用視覺化建置器。
     - JSON 編輯器 – 用於設定資料模型的自由格式JSON編輯器。如果您熟悉 的 Timestream，LiveAnalytics 並想要建置進階資料模型映射，請選擇此選項。

- JSON 檔案來自 S3 – 選取您儲存在 S3 中的JSON模型檔案。如果您已設定資料模型，並希望將其重複使用以進行其他批次載入，請選擇此選項。

8. 從錯誤日誌報告中的 S3 位置，選取將用於報告錯誤的 S3 位置。 如需有關如何使用此報告的資訊，請參閱 使用批次載入錯誤報告。
9. 針對加密金鑰類型 ，選擇下列其中一項。
   - Amazon S3-managed金鑰 （ SSE-S3 ） – Amazon S3 為您建立、管理和使用的加密金鑰。
   - AWS KMS key （ SSE-KMS ） – 受 AWS Key Management Service （ ） 保護的加密金鑰AWS KMS。
10. 選擇 Next (下一步)。
11. 在檢閱和建立頁面上，檢閱設定並視需要編輯。

> **Note**
>
> 建立任務後，您無法變更批次載入任務設定。任務完成時間會根據匯入的資料量而有所不同。

12. 選擇建立批次載入任務 。

恢復批次載入任務

當您選取狀態為「進度已停止」的批次載入任務，但仍然可以繼續時，系統會提示您繼續任務。當您檢視這些任務的詳細資訊時，也會有一個含有繼續任務按鈕的橫幅。可繼續任務具有「繼續依據」日期。在該日期過期後，任務將無法繼續。

使用視覺化建置器

您可以使用視覺化建置器，將存放在 S3 儲存貯體中的一或多個CSV檔案資料欄 （ 一或多個 ） 映射至 Timestream for LiveAnalytics Table 中的目的地資料欄。

> **Note**
>
> 您的角色將需要 檔案的`SelectObjectContent`許可。如果沒有這樣做，您將需要手動新增和刪除資料欄。

自動載入來源資料欄模式

如果您只指定一個儲存貯體， 的 Timestream LiveAnalytics 可以自動掃描來源CSV檔案，以取得資料欄名稱。沒有現有映射時，您可以選擇匯入來源資料欄 。

1. 使用從映射組態輸入設定中選取的視覺化建置器選項，設定時間戳記時間輸入。`Milliseconds`是預設設定。
2. 按一下載入來源資料欄按鈕，匯入來源資料檔案中的資料欄標頭。資料表會填入資料來源檔案中的來源資料欄標頭名稱。
3. 選擇每個來源資料欄的目標資料表資料欄名稱 、時間流屬性類型 和資料類型。

   如需這些資料欄和可能值的詳細資訊，請參閱 映射欄位。
4. 使用 drag-to-fill功能一次設定多個資料欄的值。

手動新增來源資料欄

如果您使用的是儲存貯體或CSV字首，而不是單一 CSV，則可以使用新增資料欄映射和刪除資料欄映射按鈕，從視覺化編輯器新增和刪除資料欄映射。還有一個按鈕可重設映射。

映射欄位

- 來源資料欄名稱 – 來源檔案中代表要匯入量值的資料欄名稱。當您使用匯入來源資料欄 時， 的時間串流 LiveAnalytics 會自動填入此值。
- 目標資料表資料欄名稱 – 選擇性輸入，指出目標資料表中量值的資料欄名稱。
- Timestream 屬性類型 – 指定來源欄中資料的屬性類型，例如 `DIMENSION`。
  - TIMESTAMP – 指定收集量值的時間。
  - MULTI – 表示多個量值。
  - DIMENSION – 時間序列中繼資料。
  - MEASURE_NAME – 對於單一量值記錄，這是量值名稱。
- 資料類型 – Timestream 資料欄的類型，例如 `BOOLEAN`。
  - BIGINT – 64 位元整數。
  - BOOLEAN – 邏輯的兩個真相值：true 和 false。
  - DOUBLE – 64 位元變數精確度數字。
  - TIMESTAMP – 在 中使用奈秒精度時間，UTC並追蹤自 Unix epoch 以來的時間的執行個體。

搭配 使用批次載入 AWS CLI

設定

若要開始使用批次載入，請執行下列步驟。

1. AWS CLI 使用 的指示安裝 LiveAnalytics 使用 存取 Amazon Timestream AWS CLI。
2. 執行下列命令，以確認 Timestream CLI命令已更新。驗證 create-batch-load-task 是否在 清單中。

   ```
   aws timestream-write help
   ```
3. 使用 的說明準備資料來源準備批次載入資料檔案。
4. 使用 的說明建立資料庫和資料表 LiveAnalytics 使用 存取 Amazon Timestream AWS CLI。
5. 建立報告輸出的 S3 儲存貯體。儲存貯體必須位於相同的 區域中。如需儲存貯體的詳細資訊，請參閱建立、設定和使用 Amazon S3 儲存貯體。
6. 建立批次載入任務。如需這些步驟，請參閱 建立批次載入任務。
7. 確認任務的狀態。如需這些步驟，請參閱 描述批次載入任務。

建立批次載入任務

您可以使用 `create-batch-load-task`命令建立批次載入任務。當您使用 建立批次載入任務時 CLI，您可以使用 JSON 參數 `cli-input-json`，這可讓您將參數彙總到單一JSON片段。您也可以使用包括 `data-model-configuration`、`data-source-configuration`、`report-configuration`、 `target-database-name`和 等其他參數來分開這些詳細資訊`target-table-name`。

如需範例，請參閱 建立批次載入任務範例

描述批次載入任務

您可以擷取批次載入任務描述，如下所示。

```
aws timestream-write describe-batch-load-task --task-id <value>
```

以下是回應範例：

```
{
    "BatchLoadTaskDescription": {
```

```
        "TaskId": "<TaskId>",
        "DataSourceConfiguration": {
            "DataSourceS3Configuration": {
                "BucketName": "test-batch-load-west-2",
                "ObjectKeyPrefix": "sample.csv"
            },
            "CsvConfiguration": {},
            "DataFormat": "CSV"
        },
        "ProgressReport": {
            "RecordsProcessed": 2,
            "RecordsIngested": 0,
            "FileParseFailures": 0,
            "RecordIngestionFailures": 2,
            "FileFailures": 0,
            "BytesIngested": 119
        },
        "ReportConfiguration": {
            "ReportS3Configuration": {
                "BucketName": "test-batch-load-west-2",
                "ObjectKeyPrefix": "<ObjectKeyPrefix>",
                "EncryptionOption": "SSE_S3"
            }
        },
        "DataModelConfiguration": {
            "DataModel": {
                "TimeColumn": "timestamp",
                "TimeUnit": "SECONDS",
                "DimensionMappings": [
                    {
                        "SourceColumn": "vehicle",
                        "DestinationColumn": "vehicle"
                    },
                    {
                        "SourceColumn": "registration",
                        "DestinationColumn": "license"
                    }
                ],
                "MultiMeasureMappings": {
                    "TargetMultiMeasureName": "test",
                    "MultiMeasureAttributeMappings": [
                        {
                            "SourceColumn": "wgt",
                            "TargetMultiMeasureAttributeName": "weight",
```

```
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "SourceColumn": "spd",
                            "TargetMultiMeasureAttributeName": "speed",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "SourceColumn": "fuel",
                            "TargetMultiMeasureAttributeName": "fuel",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "SourceColumn": "miles",
                            "TargetMultiMeasureAttributeName": "miles",
                            "MeasureValueType": "DOUBLE"
                        }
                    ]
                }
            }
        },
        "TargetDatabaseName": "BatchLoadExampleDatabase",
        "TargetTableName": "BatchLoadExampleTable",
        "TaskStatus": "FAILED",
        "RecordVersion": 1,
        "CreationTime": 1677167593.266,
        "LastUpdatedTime": 1677167602.38
    }
}
```

列出批次載入任務

您可以列出批次載入任務，如下所示。

```
aws timestream-write list-batch-load-tasks
```

輸出顯示如下。

```
{
    "BatchLoadTasks": [
        {
            "TaskId": "<TaskId>",
            "TaskStatus": "FAILED",
```

```
            "DatabaseName": "BatchLoadExampleDatabase",
            "TableName": "BatchLoadExampleTable",
            "CreationTime": 1677167593.266,
            "LastUpdatedTime": 1677167602.38
        }
    ]
}
```

恢復批次載入任務

您可以繼續批次載入任務，如下所示。

```
aws timestream-write resume-batch-load-task --task-id <value>
```

回應可以指示成功或包含錯誤資訊。

建立批次載入任務範例

Example

1. 為名為 的 LiveAnalytics 資料庫BatchLoad和名為 的資料表建立 TimestreamBatchLoadTest。驗證 ，並視需要調整 MemoryStoreRetentionPeriodInHours和 的值MagneticStoreRetentionPeriodInDays。

   ```
   aws timestream-write create-database --database-name BatchLoad \

   aws timestream-write create-table --database-name BatchLoad \
   --table-name BatchLoadTest \
   --retention-properties "{\"MemoryStoreRetentionPeriodInHours\": 12,
    \"MagneticStoreRetentionPeriodInDays\": 100}"
   ```

2. 使用主控台建立 S3 儲存貯體，並將sample.csv檔案複製到該位置。您可以在範例 下載CSV範例。 CSV
3. 使用主控台為 Timestream 建立 S3 儲存貯體 LiveAnalytics ，以便在批次載入任務完成時撰寫報告。
4. 建立批次載入任務。請務必取代 *$INPUT_BUCKET* 以及 *$REPORT_BUCKET* 您在上述步驟中建立的儲存貯體。

   ```
   aws timestream-write create-batch-load-task \
   --data-model-configuration "{\
               \"DataModel\": {\
   ```

```
              \"TimeColumn\": \"timestamp\",\
              \"TimeUnit\": \"SECONDS\",\
              \"DimensionMappings\": [\
                {\
                  \"SourceColumn\": \"vehicle\"\
                },\
                {\
                  \"SourceColumn\": \"registration\",\
                  \"DestinationColumn\": \"license\"\
                }\
              ],
              \"MultiMeasureMappings\": {\
                \"TargetMultiMeasureName\": \"mva_measure_name\",\
                \"MultiMeasureAttributeMappings\": [\
                  {\
                    \"SourceColumn\": \"wgt\",\
                    \"TargetMultiMeasureAttributeName\": \"weight\",\
                    \"MeasureValueType\": \"DOUBLE\"\
                  },\
                  {\
                    \"SourceColumn\": \"spd\",\
                    \"TargetMultiMeasureAttributeName\": \"speed\",\
                    \"MeasureValueType\": \"DOUBLE\"\
                  },\
                  {\
                    \"SourceColumn\": \"fuel_consumption\",\
                    \"TargetMultiMeasureAttributeName\": \"fuel\",\
                    \"MeasureValueType\": \"DOUBLE\"\
                  },\
                  {\
                    \"SourceColumn\": \"miles\",\
                    \"MeasureValueType\": \"BIGINT\"\
                  }\
                ]\
              }\
            }\
          }" \
--data-source-configuration "{
            \"DataSourceS3Configuration\": {\
              \"BucketName\": \"$INPUT_BUCKET\",\
              \"ObjectKeyPrefix\": \"$INPUT_OBJECT_KEY_PREFIX\"
            },\
            \"DataFormat\": \"CSV\"\
          }" \
```

```
--report-configuration "{\
            \"ReportS3Configuration\": {\
              \"BucketName\": \"$REPORT_BUCKET\",\
              \"EncryptionOption\": \"SSE_S3\"\
            }\
          }" \
--target-database-name BatchLoad \
--target-table-name BatchLoadTest
```

上述命令會傳回下列輸出。

```
{
    "TaskId": "TaskId "
}
```

5. 檢查任務的進度。確定您取代 *$TASK_ID* 具有上一個步驟中傳回的任務 ID。

```
aws timestream-write describe-batch-load-task --task-id $TASK_ID
```

範例輸出

```
{
    "BatchLoadTaskDescription": {
        "ProgressReport": {
            "BytesIngested": 1024,
            "RecordsIngested": 2,
            "FileFailures": 0,
            "RecordIngestionFailures": 0,
            "RecordsProcessed": 2,
            "FileParseFailures": 0
        },
        "DataModelConfiguration": {
            "DataModel": {
                "DimensionMappings": [
                    {
                        "SourceColumn": "vehicle",
                        "DestinationColumn": "vehicle"
                    },
                    {
                        "SourceColumn": "registration",
                        "DestinationColumn": "license"
```

```
                }
            ],
            "TimeUnit": "SECONDS",
            "TimeColumn": "timestamp",
            "MultiMeasureMappings": {
                "MultiMeasureAttributeMappings": [
                    {
                        "TargetMultiMeasureAttributeName": "weight",
                        "SourceColumn": "wgt",
                        "MeasureValueType": "DOUBLE"
                    },
                    {
                        "TargetMultiMeasureAttributeName": "speed",
                        "SourceColumn": "spd",
                        "MeasureValueType": "DOUBLE"
                    },
                    {
                        "TargetMultiMeasureAttributeName": "fuel",
                        "SourceColumn": "fuel_consumption",
                        "MeasureValueType": "DOUBLE"
                    },
                    {
                        "TargetMultiMeasureAttributeName": "miles",
                        "SourceColumn": "miles",
                        "MeasureValueType": "DOUBLE"
                    }
                ],
                "TargetMultiMeasureName": "mva_measure_name"
            }
        }
    },
    "TargetDatabaseName": "BatchLoad",
    "CreationTime": 1672960381.735,
    "TaskStatus": "SUCCEEDED",
    "RecordVersion": 1,
    "TaskId": "TaskId ",
    "TargetTableName": "BatchLoadTest",
    "ReportConfiguration": {
        "ReportS3Configuration": {
            "EncryptionOption": "SSE_S3",
            "ObjectKeyPrefix": "ObjectKeyPrefix ",
            "BucketName": "test-report-bucket"
        }
    },
```

```
        "DataSourceConfiguration": {
            "DataSourceS3Configuration": {
                "ObjectKeyPrefix": "sample.csv",
                "BucketName": "test-input-bucket"
            },
            "DataFormat": "CSV",
            "CsvConfiguration": {}
        },
        "LastUpdatedTime": 1672960387.334
    }
}
```

搭配 使用批次載入 AWS SDKs

如需如何使用 建立、描述和列出批次載入任務 AWS 的範例SDKs，請參閱 建立批次載入任務、列出批次載入任務、 描述批次載入任務和 恢復批次載入任務。

使用批次載入錯誤報告

批次載入任務具有下列其中一個狀態值：

- CREATED （已建立 ） – 任務已建立。
- IN_PROGRESS （進行中） – 任務正在進行中。
- FAILED （失敗 ） – 任務已完成。但偵測到一個或多個錯誤。
- SUCCEEDED （已完成 ） – 任務已完成，沒有錯誤。
- PROGRESS_STOPPED （進度已停止） – 任務已停止但未完成。您可以嘗試繼續任務。
- PENDING_RESUME （擱置繼續） – 任務正在等待繼續。

發生錯誤時，會在為此定義的 S3 儲存貯體中建立錯誤日誌報告。錯誤分類為 taskErrors 或 fileErrors 的個別陣列。以下是錯誤報告範例。

```
{
    "taskId": "9367BE28418C5EF902676482220B631C",
    "taskErrors": [],
    "fileErrors": [
        {
            "fileName": "example.csv",
            "errors": [
                {
```

```
                    "reason": "The record timestamp is outside the time range of the
 data ingestion window.",
                    "lineRanges": [
                        [
                            2,
                            3
                        ]
                    ]
                }
            ]
        }
    ]
}
```

在 Timestream 中使用 的排程查詢 LiveAnalytics

的 Amazon Timestream 中的排程查詢功能 LiveAnalytics 是完全受管、無伺服器和可擴展的解決方案，用於計算和儲存彙總、彙總和其他形式的預先處理資料，通常用於操作儀表板、業務報告、臨時分析和其他應用程式。排程查詢可讓即時分析更具效能且符合成本效益，因此您可以從資料中衍生其他洞見，並可以繼續做出更好的業務決策。

透過排程查詢，您可以定義即時分析查詢，以計算資料彙總、彙總和其他操作，以及 Amazon Timestream LiveAnalytics 定期執行這些查詢，並可靠地將查詢結果寫入單獨的資料表。通常會在幾分鐘內計算資料並將其更新為這些資料表。

然後，您可以指向儀表板和報告來查詢包含彙總資料的資料表，而不是查詢較大的來源資料表。這會導致效能和成本增加，可能會超過數量級的訂單。這是因為彙總資料的資料表比來源資料表包含的資料少得多，因此它們提供更快的查詢和更便宜的資料儲存。

此外，具有排程查詢的資料表提供 Timestream for LiveAnalytics Table 的所有現有功能。例如，您可以使用 查詢資料表SQL。您可以使用 Grafana 視覺化儲存在資料表中的資料。您也可以使用 Amazon Kinesis 、Amazon MSK、 AWS IoT Core 和 Telegraf 將資料擷取至資料表。您可以在這些資料表上設定資料保留政策，以進行自動資料生命週期管理。

由於包含彙總資料之資料表的資料保留與來源資料表的資料完全解耦，因此您也可以選擇減少來源資料表的資料保留，並將彙總資料保留更長的時間，而且只需資料儲存成本的一小部分。排程查詢可讓更多客戶更快、更便宜地存取即時分析，因此他們可以監控其應用程式並推動更好的資料驅動型商業決策。

主題

- 排程查詢優點

排程查詢優點

以下是排程查詢的優點：

- 操作簡易 – 排程查詢是無伺服器且完全受管的。
- 效能和成本 – 由於排程查詢會預先計算資料的彙總、彙總或其他即時分析操作，並將結果儲存在資料表中，因此存取由排程查詢填入之資料表的查詢所含資料少於來源資料表。因此，在這些資料表上執行的查詢更快速且更便宜。由排程運算填入的資料表包含的資料少於來源資料表，因此有助於降低儲存成本。您也可以在記憶體存放區中保留此資料較長的時間，而成本僅為在記憶體存放區中保留來源資料成本的一小部分。
- 互通性 – 由排程查詢填入的資料表提供 Timestream 對 LiveAnalytics 資料表的所有現有功能，並且可以與與 Timestream for 搭配使用的所有服務和工具搭配使用 LiveAnalytics。如需詳細資訊，請參閱使用其他服務。

排程查詢使用案例

您可以針對業務報告使用排程查詢，以摘要來自應用程式的終端使用者活動，以便訓練機器學習模型以進行個人化。您也可以針對偵測異常、網路入侵或詐騙活動的警示使用排程查詢，以便立即採取補救措施。

此外，您可以使用排程查詢來實現更有效的資料管理。您可以只授予排程查詢的來源資料表存取權，並僅提供開發人員由排程查詢填入的資料表存取權。這可將意外、長時間執行查詢的影響降至最低。

範例：使用即時分析來偵測詐騙付款並做出更好的業務決策

請考慮使用付款系統，處理從美國主要都會城市分佈的多個 point-of-sale終端機傳送的交易。您想要使用 的 Amazon Timestream LiveAnalytics 來存放和分析交易資料，以便您可以偵測詐騙交易並執行即

時分析查詢。這些查詢可協助您回答商業問題，例如識別每小時最繁忙和最不使用 point-of-sale的終端機、每個城市一天中最繁忙的時段，以及每小時大多數交易的城市。

系統每分鐘處理約 10 萬筆交易。存放在 Amazon Timestream LiveAnalytics中的每個交易都是 100 個位元組。您已設定每分鐘執行 10 個查詢，以偵測各種類型的詐騙付款。您也建立了 25 個查詢，這些查詢會彙總資料並沿著各種維度進行切片/切割，以協助回答您的業務問題。每個查詢都會處理最後一個小時的資料。

您已建立儀表板，以顯示這些查詢所產生的資料。儀表板包含 25 個小工具，每小時重新整理一次，且通常隨時有 10 個使用者存取。最後，您的記憶體存放區設定為 2 小時資料保留期，而磁性存放區設定為具有 6 個月資料保留期。

在此情況下，您可以使用即時分析查詢，在每次存取和重新整理儀表板時重新計算資料，或使用儀表板的衍生資料表。根據即時分析查詢的儀表板查詢成本為每月 120.70 美元。相反地，衍生資料表支援的儀表板查詢成本為每月 12.27 美元（請參閱 Amazon Timestream 以取得 LiveAnalytics定價 ）。在此情況下，使用衍生資料表可減少約 10 倍的查詢成本。

排程查詢概念

查詢字串 - 這是您要預先計算和儲存在另一個 Timestream 中 LiveAnalytics 資料表結果的查詢。您可以使用 Timestream for 的完整SQL表面積來定義排程查詢 LiveAnalytics，這可讓您靈活地使用常用的資料表表達式、巢狀查詢、視窗函數或 Timestream 支援的查詢語言 的任何彙總和純量函數來撰寫 LiveAnalytics 查詢。

排程表達式 - 可讓您指定何時執行排程查詢執行個體。您可以使用 cron 表達式（例如UTC每天上午 8 點執行）或速率表達式（例如每 10 分鐘執行一次）來指定表達式。

目標組態 - 可讓您指定如何將排程查詢的結果映射到儲存此排程查詢結果的目的地資料表。

通知組態 - Timestream 會根據您的排程表達 LiveAnalytics 式自動執行排程查詢的執行個體。當您建立排程查詢時，您設定SNS的主題上執行的每個此類查詢都會收到通知。此通知指定執行個體是否已成功執行或遇到任何錯誤。此外，它還提供位元組計量、寫入目標資料表的資料、下次調用時間等資訊。

以下是此類通知訊息的範例。

```
{
    "type":"AUTO_TRIGGER_SUCCESS",
    "arn":"arn:aws:timestream:us-east-1:123456789012:scheduled-query/
 PT1mPerMinutePerRegionMeasureCount-9376096f7309",
    "nextInvocationEpochSecond":1637302500,
```

```
    "scheduledQueryRunSummary":
    {
        "invocationEpochSecond":1637302440,
        "triggerTimeMillis":1637302445697,
        "runStatus":"AUTO_TRIGGER_SUCCESS",
        "executionStats":
        {
            "executionTimeInMillis":21669,
            "dataWrites":36864,
            "bytesMetered":13547036820,
            "recordsIngested":1200,
            "queryResultRows":1200
        }
    }
}
```

在此通知訊息中， bytesMetered是在來源資料表上掃描查詢的位元組， dataWrites 以及寫入目標資料表的位元組。

> **Note**
>
> 如果您以程式設計方式使用這些通知，請注意，未來可以將新欄位新增至通知訊息。

錯誤報告位置 - 已排程查詢非同步地執行和存放目標資料表中的資料。如果執行個體遇到任何錯誤（例如，無法儲存的無效資料），則遇到錯誤的記錄會寫入您在建立排程查詢時指定的錯誤報告位置中的錯誤報告。您可以指定位置的 S3 儲存貯體和字首。的時間串流會將排程查詢名稱和調用時間 LiveAnalytics 附加到此字首，以協助您識別與排程查詢的特定執行個體相關聯的錯誤。

標記 - 您可以選擇性地指定可與排程查詢建立關聯的標籤。如需詳細資訊，請參閱標記資源的 LiveAnalytics Timestream。

範例

在下列範例中，您可以使用排程查詢來計算簡單的彙總：

```
SELECT region, bin(time, 1m) as minute,
    SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as numDataPoints
FROM raw_data.devops
WHERE time BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m
GROUP BY bin(time, 1m), region
```

`@scheduled_runtime parameter` - 在此範例中，您會注意到查詢接受特殊的命名參數 `@scheduled_runtime`。這是特殊參數（時間戳記類型），服務會在叫用排程查詢的特定執行個體時設定，以便您可以決定控制排程查詢的特定執行個體分析來源資料表中資料的時間範圍。您可以在預期時間戳記類型的任何位置的查詢`@scheduled_runtime`中使用 。

請考慮您設定排程表達式的範例：cron（0/5 * * * ？ *），其中排程查詢會在每小時的分鐘 0、5、10、15、20、25、30、35、40、45、50、55 執行。對於在 2021-12-01 00：05：00 觸發的執行個體，@scheduled_runtime 參數會初始化為此值，因此目前執行個體會操作 2021-11-30 23：55：00 至 2021-12-01 00：06：00 範圍內的資料。

具有重疊時間範圍的執行個體 - 如本範例所示，排程查詢的兩個後續執行個體可能會在其時間範圍中重疊。這是您可以根據需求、您指定的時間述詞和排程表達式來控制的內容。在此情況下，此重疊允許這些運算根據其到達稍有延遲的任何資料更新彙總，在此範例中最多 10 分鐘。2021-12-01 00：00：00 觸發的查詢執行將涵蓋時間範圍 2021-11-30 23：50：00 至 2021-12-30 00：01：00，而 2021-12-01 00：05：00 觸發的查詢執行將涵蓋範圍 2021-11-30 23：55：00 至 2021-12-01 00：06：00。

為了確保正確性，並確保儲存在目標資料表中的彙總符合來源資料表中計算的彙總， 的 Timestream LiveAnalytics 可確保 2021-12-01 00：05：00 的運算只會在 2021-12-01 00：00：00 的運算完成之後執行。如果產生較新的值，後者運算的結果可以更新任何先前具體化的彙總。在內部， 的 Timestream LiveAnalytics 會使用記錄版本，其中排程查詢後者執行個體產生的記錄會獲指派較高的版本號碼。因此，假設來源資料表上有較新的資料可用，則 2021-12-01 00：05：00 的調用所計算的彙總可以更新 2021-12-01 00：00：00 的調用所計算的彙總。

自動觸發程序與手動觸發程序 - 建立排程查詢後， 的 Timestream LiveAnalytics 會根據指定的排程自動執行執行個體。此類自動觸發程序完全由 服務管理。

不過，在某些情況下，您可能想要手動啟動排程查詢的某些執行個體。例如，如果特定執行個體在查詢執行中失敗，如果在自動排程執行後來源資料表中有延遲到達的資料或更新，或者如果您想要更新目標資料表，以取得自動查詢執行未涵蓋的時間範圍（例如，建立排程查詢之前的時間範圍）。

您可以使用 ExecuteScheduledQuery API傳遞 參數，手動啟動排程查詢的特定執行個體，該 InvocationTime 參數是 @scheduled_runtime 參數所使用的值。以下是使用 時的一些重要考量 ExecuteScheduledQuery API：

- 如果您觸發多個這些調用，則需要確保這些調用不會產生重疊時間範圍的結果。如果您無法確保非重疊的時間範圍，請確定這些查詢執行依序啟動。如果您同時啟動在其時間範圍內重疊的多個查詢執行，則可以看到觸發失敗，您可能會在這些查詢執行的錯誤報告中看到版本衝突。
- 您可以使用 @scheduled_runtime 的任何時間戳記值來啟動調用。因此，您有責任適當地設定值，以便在目標資料表中更新適當的時間範圍，對應於來源資料表中更新資料的範圍。

排程查詢的排程表達式

您可以使用 Amazon Timestream 來建立自動排程的排程查詢，方法是使用 cron 或速率表達式的 LiveAnalytics 排程查詢。所有排程的查詢都會使用UTC時區，且排程的最低可能精確度為 1 分鐘。

指定排程表達式的兩種方法是 cron 和 rate 。Cron 表達式提供更精細的排程控制，而速率表達式更易於表達，但缺乏精細控制。

例如，使用 cron 表達式，您可以定義排程查詢，該查詢會在每週或每月特定日期的指定時間觸發，或只在週一至週五每小時指定分鐘，以此類推。相反地，速率表達式會以一般速率啟動排程查詢，例如每分鐘、每小時或每天一次，從建立排程查詢的確切時間開始。

Cron 表達式

- 語法

```
cron(fields)
```

Cron 表達式有六個必要欄位，以空格隔開。

欄位	值	萬用字元
分鐘	0-59	, - * /
小時	0-23	, - * /
Day-of-month	1-31	, - * ? / L W
月	1-12 或 JAN-DEC	, - * /
Day-of-week	1-7 或 SUN-SAT	, - * ? L #
年	1970-2199	, - * /

萬用字元

- * , *（逗號） 萬用字元包含其他值。在月份欄位中，JANFEB， MAR將包含 1 月、2 月和 3 月。
- *-*（破折號） 萬用字元指定範圍。在 Day (日) 欄位，1-15 包含指定月份的 1 至 15 號。

- *** （星號） 萬用字元包含 欄位中的所有值。在時數欄位中，*** 將包含每小時。您無法在 和 Day-of-week 欄位中使用 Day-of-month ***。如果您將其用於一個 ，則必須在另一個 中使用 * ? *。
- */* （正斜線） 萬用字元會指定增量。在分鐘欄位中，您可以輸入 1/10 以指定每 10 分鐘一次，從小時的第一分鐘開始 （例如，第 11、第 21 和第 31 分鐘等）。
- * ? * （問號） 萬用字元指定一個或另一個。在 Day-of-month欄位中，您可以輸入 *7*，如果您不在乎第 7 個星期的哪一天，您可以在欄位中輸入 Day-of-week * ? *。
- 或 Day-of-week 欄位中的 Day-of-month *L* 萬用字元指定當月或當週的最後一天。
- 欄位中的 Day-of-month W 萬用字元指定工作日。在 Day-of-month 欄位中，3W 會指定最接近當月第三天的工作日。
- 欄位中的 Day-of-week *#* 萬用字元會指定一個月內一週中指定日期的特定執行個體。例如，3#2 代表則該月的第二個星期二：3 是指星期二，因為它是每週的第三天，2 指的是一個月內該類型的第二天。

> **Note**
>
> 如果您使用 '#' 字元，您只能在 day-of-week 欄位中定義一個表達式。例如："3#1,6#3" 是無效的，因為它被轉譯為兩個表達式。

限制

- 您無法在相同的 cron 表達式中指定 Day-of-month 和 Day-of-week 欄位。如果您在其中一個欄位中指定值 （或 * ），則必須在另一個欄位中使用 * ? * （問號）。
- 不支援頻率多於 1 分鐘的 Cron 表達式。

範例

分鐘	小時	月中的日	月	週中的日	年	意義
0	10	*	*	?	*	每日上午 10：00 （UTC） 執行。

分鐘	小時	月中的日	月	週中的日	年	意義
15	12	*	*	?	*	每日中午 12：15（UTC）執行。
0	18	?	*	MON-FRI	*	週一至週五下午 6：00（UTC）執行。
0	8	1	*	?	*	每個月第一天的上午 8：00（UTC）執行。
0/15	*	*	*	?	*	每 15 分鐘執行一次。
0/10	*	*	*	MON-FRI	*	週一至週五每 10 分鐘執行一次。
0/5	8-17	?	*	MON-FRI	*	週一至週五，上午 8：00 至下午 5：55（），每 5 分鐘執行一次UTC。

Rate 運算式

- Rate 表達式在您建立排程事件規則時開始，然後在其定義的排程上執行。Rate 表達式有兩個必要欄位。欄位是以空格隔開。

  語法

  ```
  rate(value unit)
  ```

  - `value`：正數。
  - `unit`：時間單位。值為 1 （例如分鐘） 和值超過 1 （例如分鐘） 時，需要不同的單位。有效值：minute | minutes | hour | hours | day | days (分鐘、數分鐘、小時、數小時、天、數天)

排程查詢的資料模型映射

的 Timestream LiveAnalytics 支援其資料表中資料的彈性建模，且此相同的彈性適用於已具體化至另一個 Timestream for LiveAnalytics Table 的排程查詢結果。透過排程查詢，您可以查詢任何資料表，無論其在多測量記錄或單一測量記錄中具有資料，並使用多測量或單一測量記錄寫入查詢結果。

您可以在排程查詢的規格 TargetConfiguration 中使用 ，將查詢結果映射到目的地衍生資料表中的適當資料欄。下列各節說明指定此項目的不同方式， TargetConfiguration 以達成衍生資料表中的不同資料模型。具體而言，您會看到：

- 當查詢結果沒有量值名稱，且您在 中指定目標量值名稱時，如何寫入多量值記錄 TargetConfiguration。
- 如何在查詢結果中使用量值名稱來寫入多量值記錄。
- 如何定義模型，以寫入具有不同多測量屬性的多個記錄。
- 如何定義模型，以寫入衍生資料表中的單一測量記錄。
- 如何查詢排程查詢中的單一測量記錄和/或多測量記錄，並將結果具體化為單一測量記錄或多測量記錄，讓您選擇資料模型的彈性。

範例：多量值記錄的目標量值名稱

在此範例中，您會看到查詢正在從具有多測量資料的資料表中讀取資料，並使用多測量記錄將結果寫入另一個資料表。排程查詢結果沒有自然量值名稱資料欄。在此，您可以使用 中的 TargetMultiMeasureName 屬性，在衍生資料表中指定量值名稱 TargetConfigurationTimestreamConfiguration。

```
{
    "Name" : "CustomMultiMeasureName",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, AVG(memory_cached)
 as avg_mem_cached_1h, MIN(memory_free) as min_mem_free_1h, MAX(memory_used) as
 max_mem_used_1h, SUM(disk_io_writes) as sum_1h, AVG(disk_used) as avg_disk_used_1h,
 AVG(disk_free) as avg_disk_free_1h, MAX(cpu_user) as max_cpu_user_1h, MIN(cpu_idle) as
 min_cpu_idle_1h, MAX(cpu_system) as max_cpu_system_1h FROM raw_data.devops_multi WHERE
 time BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h
 AND measure_name = 'metrics' GROUP BY region, bin(time, 1h)",
    "ScheduleConfiguration" : {
        "ScheduleExpression" : "cron(0 0/1 * * ? *)"
    },
    "NotificationConfiguration" : {
        "SnsConfiguration" : {
            "TopicArn" : "******"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******",
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_1",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
                }
            ],
            "MultiMeasureMappings" : {
                "TargetMultiMeasureName": "dashboard-metrics",
                "MultiMeasureAttributeMappings" : [
                    {
                        "SourceColumn" : "avg_mem_cached_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName" : "avgMemCached"
                    },
                    {
                        "SourceColumn" : "min_mem_free_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_mem_used_1h",
```

```
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "sum_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName" : "totalDiskWrites"
                    },
                    {
                        "SourceColumn" : "avg_disk_used_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "avg_disk_free_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_cpu_user_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName" : "CpuUserP100"
                    },
                    {
                        "SourceColumn" : "min_cpu_idle_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_cpu_system_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName" : "CpuSystemP100"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

此範例中的映射會建立一個多測量記錄，其中包含測量名稱儀表板指標
和屬性名稱 avgMemCached、min_mem_free_1h、max_mem_used_1h
totalDiskWrites、avg_disk_used_1h、avg_disk_free_1h、 CpuUserP100、min_cpu_idle_1h、
CpuSystemP100。請注意，選用 TargetMultiMeasureAttributeName 將查詢輸出資料欄重新命名為用
於結果具體化的不同屬性名稱。

以下是完成此排程查詢後，目的地資料表的結構描述。如您在下列結果中從 Timestream 中看到
LiveAnalytics 的屬性類型，結果會具體化為具有單一測量名稱 的多測量記錄`dashboard-metrics`，
如測量結構描述所示。

資料行	Type	LiveAnalytics 屬性類型的時間串流
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
time	timestamp	TIMESTAMP
CpuSystemP100	double	MULTI
avgMemCached	double	MULTI
min_cpu_idle_1h	double	MULTI
avg_disk_free_1h	double	MULTI
avg_disk_used_1h	double	MULTI
totalDiskWrites	double	MULTI
max_mem_used_1h	double	MULTI
min_mem_free_1h	double	MULTI
CpuUserP100	double	MULTI

以下是透過SHOWMEASURES查詢取得的對應量值。

measure_name	data_type	維度
儀表板指標	多重	【{'dimension_name' : 'region' ， 'data_type' : 'varchar'}】

範例：在多量值記錄中使用來自排程查詢的量值名稱

在此範例中，您會看到從具有單一測量記錄的資料表讀取查詢，並將結果具體化為多測量記錄。在此情況下，排程查詢結果有一個資料欄，其值可以用作目標資料表中的量值名稱，其中排程查詢的結果會具體化。然後，您可以使用 中的 MeasureNameColumn 屬性，在衍生資料表中指定多量值記錄的量值名稱 TargetConfigurationTimestreamConfiguration。

```
{
    "Name" : "UsingMeasureNameFromQueryResult",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, measure_name, AVG(CASE WHEN
 measure_name IN ('memory_cached', 'disk_used', 'disk_free') THEN measure_value::double
 ELSE NULL END) as avg_1h, MIN(CASE WHEN measure_name IN ('memory_free', 'cpu_idle')
 THEN measure_value::double ELSE NULL END) as min_1h, SUM(CASE WHEN measure_name
 IN ('disk_io_writes') THEN measure_value::double ELSE NULL END) as sum_1h,
 MAX(CASE WHEN measure_name IN ('memory_used', 'cpu_user', 'cpu_system') THEN
 measure_value::double ELSE NULL END) as max_1h FROM raw_data.devops WHERE time
 BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h AND
 measure_name IN ('memory_free', 'memory_used', 'memory_cached', 'disk_io_writes',
 'disk_used', 'disk_free', 'cpu_user', 'cpu_system', 'cpu_idle') GROUP BY region,
 measure_name, bin(time, 1h)",
    "ScheduleConfiguration" : {
        "ScheduleExpression" : "cron(0 0/1 * * ? *)"
    },
    "NotificationConfiguration" : {
        "SnsConfiguration" : {
            "TopicArn" : "******"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******",
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_2",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
```

```
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
                }
            ],
            "MeasureNameColumn" : "measure_name",
            "MultiMeasureMappings" : {
                "MultiMeasureAttributeMappings" : [
                    {
                        "SourceColumn" : "avg_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "min_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName": "p0_1h"
                    },
                    {
                        "SourceColumn" : "sum_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName": "p100_1h"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

此範例中的映射會建立屬性為 avg_1h、p0_1h、 sum_1h、p100_1h 的多量值記錄，並將使用查詢結果中 measure_name 資料欄的值作為目的地資料表中多量值記錄的量值名稱。此外請注意，先前的範例可選擇使用 TargetMultiMeasureAttributeName 搭配映射的子集來重新命名屬性。例如，min_1h 已重新命名為 p0_1h，max_1h 已重新命名為 p100_1h。

以下是完成此排程查詢後，目的地資料表的結構描述。如您在下列結果中從 Timestream 中看到 LiveAnalytics 的屬性類型，結果會具體化為多測量記錄。如果您查看量值結構描述，則擷取了九個不同的量值名稱，對應於查詢結果中看到的值。

資料行	Type	LiveAnalytics 屬性類型的時間串流
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
time	timestamp	TIMESTAMP
sum_1h	double	MULTI
p100_1h	double	MULTI
p0_1h	double	MULTI
avg_1h	double	MULTI

以下是透過SHOWMEASURES查詢取得的對應量值。

measure_name	data_type	維度
cpu_idle	多重	【{'dimension_name' : 'region' ， 'data_type' : 'varchar'}】
cpu_system	多重	【{'dimension_name' : 'region' ， 'data_type' : 'varchar'}】
cpu_user	多重	【{'dimension_name' : 'region' ， 'data_type' : 'varchar'}】

measure_name	data_type	維度
disk_free	多重	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
disk_io_writes	多重	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
disk_used	多重	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
memory_cached	多重	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
memory_free	多重	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
memory_free	多重	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】

範例：將結果映射到具有不同屬性的不同多測量記錄

下列範例示範如何將查詢結果中的不同資料欄映射到具有不同量值名稱的不同多量值記錄。如果您看到下列排程查詢定義，查詢的結果具有下列資料欄：區域、小時、avg_mem_cached_1h、min_mem_free_1h、max_mem_used_1h、total_disk_io_writes_1h、avg_disk_used_1h、avg_disk_free_1h、max_cpu_user_1h、max_cpu_system_1h、`region` 會映射至維度，並hour映射至時間資料欄。

中的 MixedMeasureMappings 屬性 TargetConfiguration。TimestreamConfiguration 指定如何將量值對應至衍生資料表中的多量值記錄。

在此特定範例中，avg_mem_cached_1h、min_mem_free_1h、max_mem_used_1h 用於一個具有 mem_aggregates、total_disk_io_writes_1h、avg_disk_used_1h、avg_disk_free_1h 等測量名稱的多測量記錄中，而最後 max_cpu_user_1h、max_cpu_system_1h、min_cpu_system_1h 用於另一個具有測量名稱 cpu_aggregates 的多測量記錄中。

在這些映射中，您也可以選擇性地使用 TargetMultiMeasureAttributeName 重新命名查詢結果欄，以在目的地資料表中具有不同的屬性名稱。例如，結果欄 avg_mem_cached_1h 重新命名為 avgMemCached，Total_disk_io_writes_1h 重新命名為 totalIOWrites等。

當您定義多測量記錄的映射時， 的 Timestream 會 LiveAnalytics 檢查查詢結果中的每一列，並自動忽略具有NULL值的資料欄值。因此，在具有多個量值名稱的映射中，如果映射中該群組的所有資料欄值都NULL用於指定資料列，則不會擷取該量值名稱的值。

例如，在以下映射中，avg_mem_cached_1h、min_mem_free_1h 和 max_mem_used_1h 會映射以測量名稱 mem_aggregates。如果查詢結果的指定資料列，所有這些資料欄值都是 NULL，則 的 Timestream LiveAnalytics 將不會擷取該資料列的 Mem_aggregates 量值。如果指定資料列的所有九個資料欄都是 NULL，則您會在錯誤報告中看到使用者錯誤。

```
{
    "Name" : "AggsInDifferentMultiMeasureRecords",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, AVG(CASE WHEN measure_name
 = 'memory_cached' THEN measure_value::double ELSE NULL END) as avg_mem_cached_1h,
 MIN(CASE WHEN measure_name = 'memory_free' THEN measure_value::double ELSE
 NULL END) as min_mem_free_1h, MAX(CASE WHEN measure_name = 'memory_used' THEN
 measure_value::double ELSE NULL END) as max_mem_used_1h, SUM(CASE WHEN measure_name =
 'disk_io_writes' THEN measure_value::double ELSE NULL END) as total_disk_io_writes_1h,
 AVG(CASE WHEN measure_name = 'disk_used' THEN measure_value::double ELSE NULL END) as
 avg_disk_used_1h, AVG(CASE WHEN measure_name = 'disk_free' THEN measure_value::double
 ELSE NULL END) as avg_disk_free_1h, MAX(CASE WHEN measure_name = 'cpu_user' THEN
 measure_value::double ELSE NULL END) as max_cpu_user_1h, MAX(CASE WHEN measure_name
 = 'cpu_system' THEN measure_value::double ELSE NULL END) as max_cpu_system_1h,
 MIN(CASE WHEN measure_name = 'cpu_idle' THEN measure_value::double ELSE NULL END)
 as min_cpu_system_1h FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime,
 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h AND measure_name IN ('memory_cached',
 'memory_free', 'memory_used', 'disk_io_writes', 'disk_used', 'disk_free', 'cpu_user',
 'cpu_system', 'cpu_idle') GROUP BY region, bin(time, 1h)",
    "ScheduleConfiguration" : {
        "ScheduleExpression" : "cron(0 0/1 * * ? *)"
    },
    "NotificationConfiguration" : {
        "SnsConfiguration" : {
            "TopicArn" : "******"
```

```
        }
    },
    "ScheduledQueryExecutionRoleArn": "******",
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_3",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
                }
            ],
            "MixedMeasureMappings" : [
                {
                    "MeasureValueType" : "MULTI",
                    "TargetMeasureName" : "mem_aggregates",
                    "MultiMeasureAttributeMappings" : [
                        {
                            "SourceColumn" : "avg_mem_cached_1h",
                            "MeasureValueType" : "DOUBLE",
                            "TargetMultiMeasureAttributeName": "avgMemCached"
                        },
                        {
                            "SourceColumn" : "min_mem_free_1h",
                            "MeasureValueType" : "DOUBLE"
                        },
                        {
                            "SourceColumn" : "max_mem_used_1h",
                            "MeasureValueType" : "DOUBLE",
                            "TargetMultiMeasureAttributeName": "maxMemUsed"
                        }
                    ]
                },
                {
                    "MeasureValueType" : "MULTI",
                    "TargetMeasureName" : "disk_aggregates",
                    "MultiMeasureAttributeMappings" : [
                        {
                            "SourceColumn" : "total_disk_io_writes_1h",
                            "MeasureValueType" : "DOUBLE",
                            "TargetMultiMeasureAttributeName": "totalIOWrites"
                        },
```

```
                        {
                            "SourceColumn" : "avg_disk_used_1h",
                            "MeasureValueType" : "DOUBLE"
                        },
                        {
                            "SourceColumn" : "avg_disk_free_1h",
                            "MeasureValueType" : "DOUBLE"
                        }
                    ]
                },
                {
                    "MeasureValueType" : "MULTI",
                    "TargetMeasureName" : "cpu_aggregates",
                    "MultiMeasureAttributeMappings" : [
                        {
                            "SourceColumn" : "max_cpu_user_1h",
                            "MeasureValueType" : "DOUBLE"
                        },
                        {
                            "SourceColumn" : "max_cpu_system_1h",
                            "MeasureValueType" : "DOUBLE"
                        },
                        {
                            "SourceColumn" : "min_cpu_idle_1h",
                            "MeasureValueType" : "DOUBLE",
                            "TargetMultiMeasureAttributeName": "minCpuIdle"
                        }
                    ]
                }
            ]
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

以下是完成此排程查詢後，目的地資料表的結構描述。

資料行	Type	LiveAnalytics 屬性類型的時間串流
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
time	timestamp	TIMESTAMP
minCpuIdle	double	MULTI
max_cpu_system_1h	double	MULTI
max_cpu_user_1h	double	MULTI
avgMemCached	double	MULTI
maxMemUsed	double	MULTI
min_mem_free_1h	double	MULTI
avg_disk_free_1h	double	MULTI
avg_disk_used_1h	double	MULTI
totalIOWrites	double	MULTI

以下是透過SHOWMEASURES查詢取得的對應量值。

measure_name	data_type	維度
cpu_aggregates	多重	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
disk_aggregates	多重	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】

measure_name	data_type	維度
mem_aggregates	多重	【{'dimension_name' : 'region' ， 'data_type' : 'varchar'}】

範例：使用查詢結果中的量值名稱，將結果映射至單一量值記錄

以下是排程查詢的範例，其結果會具體化為單一測量記錄。在此範例中，查詢結果具有 measure_name 資料欄，其值將用作目標資料表中的量值名稱。您可以使用 TargetConfiguration. 中的 MixedMeasureMappings 屬性TimestreamConfiguration ，指定查詢結果欄與目標資料表中純量量量值的對應。

在下列範例定義中，查詢結果預期為九個不同的 measure_name 值。您可以在映射中列出所有這些量值名稱，並指定要用於該量值名稱之單一量值的欄。例如，在此映射中，如果在給定結果列中看到 memory_cached 的量值名稱，則 avg_1h 資料欄中的值會用作將資料寫入目標資料表時的量值。您可以選擇性地使用 TargetMeasureName 為此值提供新的量值名稱。

```
{
    "Name" : "UsingMeasureNameColumnForSingleMeasureMapping",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, measure_name, AVG(CASE WHEN
 measure_name IN ('memory_cached', 'disk_used', 'disk_free') THEN measure_value::double
 ELSE NULL END) as avg_1h, MIN(CASE WHEN measure_name IN ('memory_free', 'cpu_idle')
 THEN measure_value::double ELSE NULL END) as min_1h, SUM(CASE WHEN measure_name
 IN ('disk_io_writes') THEN measure_value::double ELSE NULL END) as sum_1h,
 MAX(CASE WHEN measure_name IN ('memory_used', 'cpu_user', 'cpu_system') THEN
 measure_value::double ELSE NULL END) as max_1h FROM raw_data.devops WHERE time
 BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h AND
 measure_name IN ('memory_free', 'memory_used', 'memory_cached', 'disk_io_writes',
 'disk_used', 'disk_free', 'cpu_user', 'cpu_system', 'cpu_idle') GROUP BY region,
 bin(time, 1h), measure_name",
    "ScheduleConfiguration" : {
        "ScheduleExpression" : "cron(0 0/1 * * ? *)"
    },
    "NotificationConfiguration" : {
        "SnsConfiguration" : {
            "TopicArn" : "******"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******",
```

```
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_4",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
                }
            ],
            "MeasureNameColumn" : "measure_name",
            "MixedMeasureMappings" : [
                {
                    "MeasureName" : "memory_cached",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_1h",
                    "TargetMeasureName" : "AvgMemCached"
                },
                {
                    "MeasureName" : "disk_used",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_1h"
                },
                {
                    "MeasureName" : "disk_free",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_1h"
                },
                {
                    "MeasureName" : "memory_free",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "min_1h",
                    "TargetMeasureName" : "MinMemFree"
                },
                {
                    "MeasureName" : "cpu_idle",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "min_1h"
                },
                {
                    "MeasureName" : "disk_io_writes",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "sum_1h",
```

```
                    "TargetMeasureName" : "total-disk-io-writes"
                },
                {
                    "MeasureName" : "memory_used",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_1h",
                    "TargetMeasureName" : "maxMemUsed"
                },
                {
                    "MeasureName" : "cpu_user",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_1h"
                },
                {
                    "MeasureName" : "cpu_system",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_1h"
                }
            ]
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

以下是完成此排程查詢後，目的地資料表的結構描述。如您從結構描述中看到的，資料表使用單一測量記錄。如果您列出資料表的量值結構描述，您會看到根據規格中提供的映射寫入 的九個量值。

資料行	Type	LiveAnalytics 屬性類型的時間串流
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
time	timestamp	TIMESTAMP

資料行	Type	LiveAnalytics 屬性類型的時間串流
measure_value::double	double	MEASURE_VALUE

以下是透過SHOWMEASURES查詢取得的對應量值。

measure_name	data_type	維度
AvgMemCached	double	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
MinMemFree	double	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
cpu_idle	double	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
cpu_system	double	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
cpu_user	double	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
disk_free	double	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
disk_used	double	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】

measure_name	data_type	維度
maxMemUsed	double	【{'dimension_name' : 'region' ， 'data_type' : 'varchar'}】
total-disk-io-writes	double	【{'dimension_name' : 'region' ， 'data_type' : 'varchar'}】

範例：將結果映射到具有查詢結果資料欄的單一測量記錄作為測量名稱

在此範例中，您有一個查詢，其結果沒有量值名稱資料欄。相反地，您希望查詢結果資料欄名稱作為將輸出映射至單一測量記錄時的測量名稱。先前有一個範例，其中類似的結果寫入多測量記錄。在此範例中，您將了解如何在符合您應用程式案例的情況下將其映射至單一測量記錄。

同樣地，您可以使用 中的 MixedMeasureMappings 屬性指定此映射 TargetConfigurationTimestreamConfiguration。在下列範例中，您會看到查詢結果有九個資料欄。您可以使用結果資料欄作為測量名稱，並將值作為單一測量值。

例如，對於查詢結果中的指定資料列，資料欄名稱 avg_mem_cached_1h 用作與資料欄相關聯的資料欄名稱和值，而 avg_mem_cached_1h 用作單一測量記錄的測量值。您也可以使用 TargetMeasureName 來使用目標資料表中的不同量值名稱。例如，對於資料欄 sum_1h 中的值，映射指定使用 total_disk_io_writes_1h 作為目標資料表中的量值名稱。如果任何資料欄的值為 NULL，則會忽略對應的量值。

```
{
    "Name" : "SingleMeasureMappingWithoutMeasureNameColumnInQueryResult",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, AVG(CASE WHEN measure_name
 = 'memory_cached' THEN measure_value::double ELSE NULL END) as avg_mem_cached_1h,
 AVG(CASE WHEN measure_name = 'disk_used' THEN measure_value::double ELSE NULL END) as
 avg_disk_used_1h, AVG(CASE WHEN measure_name = 'disk_free' THEN measure_value::double
 ELSE NULL END) as avg_disk_free_1h, MIN(CASE WHEN measure_name = 'memory_free' THEN
 measure_value::double ELSE NULL END) as min_mem_free_1h, MIN(CASE WHEN measure_name =
 'cpu_idle' THEN measure_value::double ELSE NULL END) as min_cpu_idle_1h, SUM(CASE WHEN
 measure_name = 'disk_io_writes' THEN measure_value::double ELSE NULL END) as sum_1h,
 MAX(CASE WHEN measure_name = 'memory_used' THEN measure_value::double ELSE NULL END)
 as max_mem_used_1h, MAX(CASE WHEN measure_name = 'cpu_user' THEN measure_value::double
 ELSE NULL END) as max_cpu_user_1h, MAX(CASE WHEN measure_name = 'cpu_system' THEN
 measure_value::double ELSE NULL END) as max_cpu_system_1h FROM raw_data.devops WHERE
```

```
time BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h
AND measure_name IN ('memory_free', 'memory_used', 'memory_cached', 'disk_io_writes',
'disk_used', 'disk_free', 'cpu_user', 'cpu_system', 'cpu_idle') GROUP BY region,
bin(time, 1h)",
   "ScheduleConfiguration" : {
       "ScheduleExpression" : "cron(0 0/1 * * ? *)"
   },
   "NotificationConfiguration" : {
       "SnsConfiguration" : {
           "TopicArn" : "******"
       }
   },
   "ScheduledQueryExecutionRoleArn": "******",
   "TargetConfiguration": {
       "TimestreamConfiguration": {
           "DatabaseName" : "derived",
           "TableName" : "dashboard_metrics_1h_agg_5",
           "TimeColumn" : "hour",
           "DimensionMappings" : [
               {
                   "Name": "region",
                   "DimensionValueType" : "VARCHAR"
               }
           ],
           "MixedMeasureMappings" : [
               {
                   "MeasureValueType" : "DOUBLE",
                   "SourceColumn" : "avg_mem_cached_1h"
               },
               {
                   "MeasureValueType" : "DOUBLE",
                   "SourceColumn" : "avg_disk_used_1h"
               },
               {
                   "MeasureValueType" : "DOUBLE",
                   "SourceColumn" : "avg_disk_free_1h"
               },
               {
                   "MeasureValueType" : "DOUBLE",
                   "SourceColumn" : "min_mem_free_1h"
               },
               {
                   "MeasureValueType" : "DOUBLE",
                   "SourceColumn" : "min_cpu_idle_1h"
```

```
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "sum_1h",
                    "TargetMeasureName" : "total_disk_io_writes_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_mem_used_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_cpu_user_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_cpu_system_1h"
                }
            ]
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

以下是完成此排程查詢後，目的地資料表的結構描述。如您所見，目標資料表正在儲存類型為雙的單一測量值記錄。同樣地，資料表的量值結構描述會顯示九個量值名稱。另請注意，由於映射已將 sum_1h 重新命名為 total_disk_io_writes_1h，因此存在測量名稱 total_disk_io_writes_1h。

資料行	Type	LiveAnalytics 屬性類型的時間串流
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
time	timestamp	TIMESTAMP

資料行	Type	LiveAnalytics 屬性類型的時間串流
measure_value::double	double	MEASURE_VALUE

以下是透過SHOWMEASURES查詢取得的對應量值。

measure_name	data_type	維度
avg_disk_free_1h	double	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
avg_disk_used_1h	double	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
avg_mem_cached_1h	double	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
max_cpu_system_1h	double	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
max_cpu_user_1h	double	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
max_mem_used_1h	double	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】
min_cpu_idle_1h	double	【{'dimension_name' : 'region' , 'data_type' : 'varchar'}】

measure_name	data_type	維度
min_mem_free_1h	double	【{'dimension_name'：'region'，'data_type'：'varchar'}】
total-disk-io-writes	double	【{'dimension_name'：'region'，'data_type'：'varchar'}】

排程查詢通知訊息

本節說明 Timestream 在建立、刪除、執行或更新排程查詢狀態 LiveAnalytics 時傳送的訊息。

通知訊息名稱	結構	描述
CreatingNotificationMessage	`CreatingNotificationMessage { String arn; NotificationType type; }`	此通知訊息會在傳送 的回應之前傳送`CreateScheduledQuery` 。傳送此通知後，就會啟用排程查詢。 arn - 正在建立的排程查詢ARN的 。 類型 - SCHEDULED_QUERY_CREATING
UpdateNotificationMessage	`UpdateNotificationMessage { String arn; NotificationType type; QueryState state; }`	更新排程查詢時，會傳送此通知訊息。如果遇到無法復原的錯誤， 的 Timestream LiveAnalytics 可以自動停用排程查詢，例如： • AssumeRole 失敗 • 指定客戶受管KMS金鑰KMS時，與 通訊時遇到的任何 4xx 錯誤。

通知訊息名稱	結構	描述
		• 在執行排程查詢期間遇到的任何 4xx 個錯誤。 • 擷取查詢結果期間遇到的任何 4xx 個錯誤 arn - 正在更新的排程查詢ARN的 。 類型 - SCHEDULED_QUERY_UPDATE 狀態 - ENABLED或DISABLED
DeleteNotificationMessage	`DeletionNotificationMessage { String arn; NotificationType type; }`	刪除排程查詢時，會傳送此通知訊息。 arn - 正在建立的排程查詢ARN的 。 類型 - SCHEDULED_QUERY_DELETED

通知訊息名稱	結構	描述
SuccessNotificationMessage	`SuccessNotificationMessage { NotificationType type; String arn; Date nextInvocationEpochSecond; ScheduledQueryRunSummary runSummary; }` `ScheduledQueryRunSummary { Date invocationTime; Date triggerTime; String runStatus; ExecutionStats executionstats; ErrorReportLocation errorReportLocation; String failureReason; }` `ExecutionStats { Long bytesMetered; Long dataWrites; Long queryResultRows; Long recordsIngested; Long executionTimeInMillis; }` `ErrorReportLocation {`	此通知訊息會在執行排程查詢且成功擷取結果後傳送。 ARN - 正在刪除的排程查詢ARN的 。 NotificationType - AUTO_TRIGGER_SUCCESS 或 MANUAL_TRIGGER_SUCCESS。 nextInvocationEpoch第二 - 下次 的 LiveAnalytics Timestream 將執行排程查詢。 runSummary - 排程查詢執行的相關資訊。

通知訊息名稱	結構	描述
	S3ReportLocation s3ReportLocation; } S3ReportLocation { String bucketName; String objectKey; }	

通知訊息名稱	結構	描述
FailureNotificationMessage	FailureNotificationMessage { NotificationType type; String arn; ScheduledQueryRunSummary runSummary; } ScheduledQueryRunSummary { Date invocationTime; Date triggerTime; String runStatus; ExecutionStats executionstats; ErrorReportLocation errorReportLocation; String failureReason; } ExecutionStats { Long bytesMetered; Long dataWrites; Long queryResultRows; Long recordsIngested; Long executionTimeInMillis; } ErrorReportLocation { S3ReportLocation s3ReportLocation; }	在排程查詢執行期間遇到失敗，或擷取查詢結果時，會傳送此通知訊息。 arn - 正在執行的排程查詢ARN的 。 類型 - AUTO_TRIGGER_FAILURE 或 MANUAL_TRIGGER_FAILURE。 runSummary - 排程查詢執行的相關資訊。

通知訊息名稱	結構	描述
	`S3ReportLocation { String bucketName; String objectKey; }`	

排程查詢錯誤報告

本節說明執行排程查詢遇到錯誤 LiveAnalytics 時，Timestream 產生錯誤報告的位置、格式和原因。

主題

- 排程查詢錯誤報告原因
- 排程查詢錯誤報告位置
- 排程查詢錯誤報告格式
- 排程查詢錯誤類型
- 排程查詢錯誤報告範例

排程查詢錯誤報告原因

針對可復原的錯誤產生錯誤報告。無法復原的錯誤不會產生錯誤報告。的 Timestream LiveAnalytics 可以在遇到無法復原的錯誤時自動停用排程查詢。其中包含：

- `AssumeRole` 失敗
- 指定客戶受管KMS金鑰與 通訊KMS時遇到的任何 4xx 錯誤
- 排程查詢執行時遇到的任何 4xx 個錯誤
- 擷取查詢結果期間遇到的任何 4xx 個錯誤

對於無法復原的錯誤， 的 Timestream 會 LiveAnalytics 傳送失敗通知，其中包含無法復原的錯誤訊息。系統也會傳送更新通知，指出排程查詢已停用。

排程查詢錯誤報告位置

排程查詢錯誤報告位置具有下列命名慣例：

```
s3://customer-bucket/customer-prefix/
```

以下是排程查詢 的範例ARN：

```
arn:aws:timestream:us-east-1:000000000000:scheduled-query/test-query-hd734tegrgfd
```

```
s3://customer-bucket/customer-prefix/test-query-hd734tegrgfd/<InvocationTime>/<Auto or
 Manual>/<Actual Trigger Time>
```

Auto 指出 Timestream 為 LiveAnalytics 和 自動排程的排程查詢 *Manual* 表示使用者透過 Amazon Timestream for Query 中的ExecuteScheduledQueryAPI動作手動觸發的排程 LiveAnalytics 查詢。如需 的詳細資訊ExecuteScheduledQuery，請參閱 ExecuteScheduledQuery。

排程查詢錯誤報告格式

錯誤報告具有下列JSON格式：

```
{
    "reportId": <String>,            // A unique string ID for all error reports
 belonging to a particular scheduled query run
    "errors": [ <Error>, ... ],      // One or more errors
}
```

排程查詢錯誤類型

Error 物件可以是三種類型之一：

- 記錄擷取錯誤

```
{
    "reason": <String>,              // The error message String
    "records": [ <Record>, ... ],    // One or more rejected records )
}
```

- 資料列剖析和驗證錯誤

```
{
    "reason": <String>,        // The error message String
    "rawLine": <String>,       // [Optional] The raw line String that is being parsed
 into record(s) to be ingested. This line has encountered the above-mentioned parse
 error.
```

```
}
```

- 一般錯誤

```
{
    "reason": <String>,        // The error message
}
```

排程查詢錯誤報告範例

以下是因擷取錯誤而產生的錯誤報告範例。

```
{
    "reportId": "C9494AABE012D1FBC162A67EA2C18255",
    "errors": [
        {
            "reason": "The record timestamp is outside the time range
 [2021-11-12T14:18:13.354Z, 2021-11-12T16:58:13.354Z) of the memory store.",
            "records": [
                {
                    "dimensions": [
                        {
                            "name": "dim0",
                            "value": "d0_1",
                            "dimensionValueType": null
                        },
                        {
                            "name": "dim1",
                            "value": "d1_1",
                            "dimensionValueType": null
                        }
                    ],
                    "measureName": "random_measure_value",
                    "measureValue": "3.141592653589793",
                    "measureValues": null,
                    "measureValueType": "DOUBLE",
                    "time": "1637166175635000000",
                    "timeUnit": "NANOSECONDS",
                    "version": null
                },
                {
                    "dimensions": [
```

```
                    {
                        "name": "dim0",
                        "value": "d0_2",
                        "dimensionValueType": null
                    },
                    {
                        "name": "dim1",
                        "value": "d1_2",
                        "dimensionValueType": null
                    }
                ],
                "measureName": "random_measure_value",
                "measureValue": "6.283185307179586",
                "measureValues": null,
                "measureValueType": "DOUBLE",
                "time": "1637166175636000000",
                "timeUnit": "NANOSECONDS",
                "version": null
            },
            {
                "dimensions": [
                    {
                        "name": "dim0",
                        "value": "d0_3",
                        "dimensionValueType": null
                    },
                    {
                        "name": "dim1",
                        "value": "d1_3",
                        "dimensionValueType": null
                    }
                ],
                "measureName": "random_measure_value",
                "measureValue": "9.42477796076938",
                "measureValues": null,
                "measureValueType": "DOUBLE",
                "time": "1637166175637000000",
                "timeUnit": "NANOSECONDS",
                "version": null
            },
            {
                "dimensions": [
                    {
                        "name": "dim0",
```

```
                            "value": "d0_4",
                            "dimensionValueType": null
                        },
                        {
                            "name": "dim1",
                            "value": "d1_4",
                            "dimensionValueType": null
                        }
                    ],
                    "measureName": "random_measure_value",
                    "measureValue": "12.566370614359172",
                    "measureValues": null,
                    "measureValueType": "DOUBLE",
                    "time": "1637166175638000000",
                    "timeUnit": "NANOSECONDS",
                    "version": null
                }
            ]
        }
    ]
}
```

排程查詢模式和範例

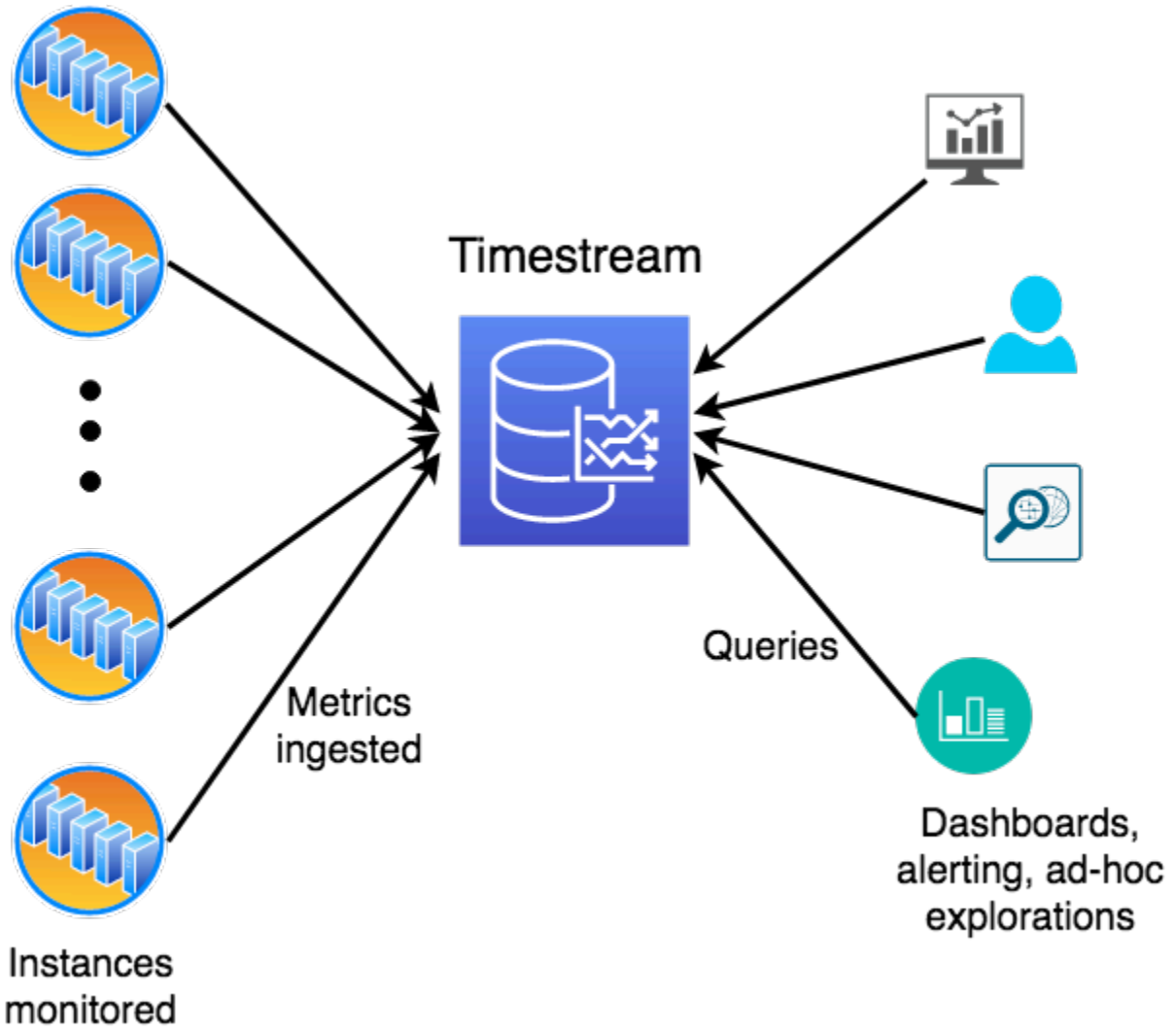
本節說明排程查詢的使用模式和 end-to-end範例。

主題

- 排程查詢範例結構描述
- 排程查詢模式
- 排程查詢範例

排程查詢範例結構描述

在此範例中，我們將使用範例應用程式模擬來自大型伺服器機群的 DevOps 案例監控指標。使用者想要提醒異常的資源用量、建立彙總機群行為和使用率的儀表板，以及對最近和歷史資料執行複雜的分析，以尋找關聯性。下圖提供一組受監控執行個體向 Timestream 發出 指標的設定圖解 LiveAnalytics。另一組並行使用者會發出警示、儀表板或臨時分析的查詢，其中查詢和擷取會平行執行。



受監控的應用程式會建模為高度橫向擴展的服務，部署於全球數個區域。每個區域都會進一步細分為數個稱為儲存格的擴展單位，這些儲存格在區域內的基礎設施方面具有隔離層級。每個儲存格都會進一步細分為孤島，代表軟體隔離的程度。每個孤島都有五個微服務，其中包含一個獨立的服務執行個體。每個微服務都有數個伺服器，具有不同的執行個體類型和作業系統版本，這些伺服器部署在三個可用區域。這些識別發出指標的伺服器屬性會在 Timestream 中建模為 的維度 LiveAnalytics。在此架構中，我們有一個維度階層（例如區域、儲存格、孤島和 microservice_name），以及跨階層的其他維度（例如 instance_type 和 availability_zone）。

應用程式會發出各種指標（例如 cpu_user 和 memory_free）和事件（例如 task_completed 和 gc_reclaimed）。每個指標或事件都與八個維度（例如區域或儲存格）相關聯，這些維度可唯一識別發出它的伺服器。資料會寫入 20 個指標一起存放在具有測量名稱指標的多測量記錄中，而所有 5 個事件都會一起存放在另一個具有測量名稱事件的多測量記錄中。資料模型、結構描述和資料產生可以在開放原始碼的發電機 中找到。除了結構描述和資料分佈之外，資料產生器還提供使用多個寫入器平行擷

取資料的範例，使用 Timestream 的擷取擴展 LiveAnalytics 每秒擷取數百萬個測量。下方顯示 結構描述 （資料表和測量結構描述） 和資料集的一些範例資料。

主題

多測量記錄

資料表結構描述

以下是使用多測量記錄擷取資料後的資料表結構描述。這是DESCRIBE查詢的輸出。假設資料擷取到資料庫 raw_data 和資料表開發，以下是查詢。

```
DESCRIBE "raw_data"."devops"
```

資料行	Type	LiveAnalytics 屬性類型的時間串流
availability_zone	varchar	DIMENSION
microservice_name	varchar	DIMENSION
instance_name	varchar	DIMENSION
process_name	varchar	DIMENSION
os_version	varchar	DIMENSION
jdk_version	varchar	DIMENSION
儲存格	varchar	DIMENSION
region	varchar	DIMENSION
孤島	varchar	DIMENSION
instance_type	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME

資料行	Type	LiveAnalytics 屬性類型的時間串流
time	timestamp	TIMESTAMP
memory_free	double	MULTI
cpu_steal	double	MULTI
cpu_iowait	double	MULTI
cpu_user	double	MULTI
memory_cached	double	MULTI
disk_io_reads	double	MULTI
cpu_hi	double	MULTI
latency_per_read	double	MULTI
network_bytes_out	double	MULTI
cpu_idle	double	MULTI
disk_free	double	MULTI
memory_used	double	MULTI
cpu_system	double	MULTI
file_descriptors_in_use	double	MULTI
disk_used	double	MULTI
cpu_nice	double	MULTI
disk_io_writes	double	MULTI
cpu_si	double	MULTI
latency_per_write	double	MULTI

資料行	Type	LiveAnalytics 屬性類型的時間串流
network_bytes_in	double	MULTI
task_end_state	varchar	MULTI
gc_pause	double	MULTI
task_completed	bigint	MULTI
gc_reclaimed	double	MULTI

測量結構描述

以下是SHOWMEASURES查詢傳回的量值結構描述。

```
SHOW MEASURES FROM "raw_data"."devops"
```

measure_name	data_type	維度
事件	多重	【{"data_type"："varchar"，"dimension_name"："availability_zone"}，{"data_type"："varchar"，"dimension_name"："microservice_name"}，{"data_type"："varchar"，"dimension_name"："instance_name"}，{"data_type"："varchar"，"dimension_name"："process_name"}，{"data_type"："varchar"，"dimension_name"："jdk_version"}，{"data_type"："varchar"，"dimension_name"name"："" ，{"

measure_name	data_type	維度
指標	多重	【{"data_type" : "varchar" , "dimension_name" : "availability_zone"} , {"data_type" : "varchar" , "dimension_name" : "microservice_name"} , {"data_type" : "varchar" , "dimension_name" : "instance_name"} , {"data_type" : "varchar" , "dimension_name" : "os_version"} , {"data_type" : "varchar" , "dimension_name" : "cell"} , {"data_type" : "varchar" , "dimension_name" : "region"} , {"data_type" : " , "dimension"

範例資料

re	信有格	錢洛	ai ity	mi ic	in n	in ty	o n	p a	jo o	m a	時間	c	c m	c	c	c t	c	c	c	c	m e	m cl	d e	la e	d ri	la e	d	d	n y	n y	fil rin	m e	ta st	g	ta le	gs m	sc la ed
us e	us e- c	us e- cs il	us e- -1	a	i- za k- a us e- cs il 0	m e	A				指標	1 1 1 4	6	0	3	0	0	0	0	0	5	8	5	9	3	2	6	2	8	4	3	5					

re	信右格	鎂洛	ar ity	mic	in n	in ty	op on	pa	jo	ma	時間	cp	cp m	cp	cp	cp t	cp	cp	cp	me	mc cle	de	la e	di ri	la e	de	de	ny	ny yt	fil rin	me e	ta st	ge	ta le	ge ml	se ed	la
					zm																																
us e	us e- co	us e- co sil	us e- -1	ar	i- za k- au us e- co sil 0 zm	me e	AV			排模	1 1 1 4	5	0	3	0	0	0	0	0	9	3	5	3	2	9	5	3	7	5	6	2						
us e	us e- co	us e- co sil	us e- -1	ar	i- za k- au us e- co sil 0 zm	me e	AV			排模	1 1 1 3	4	0	4	0	0	0	0	0	9	4	5	3	8	3	7	6	8	5	4	8						

re	信右格	鎂洛	ar ity	m ic	in n	in ty	o n	p a	jo o	m a	時間	c	c m	c	c	c t	c	c	c	m e	m cl	d e	la e	d ri	la e	d	d	n yt	n yt	fil rin	m e	ta st	g	ta le	gscemla med
us e	us e- cc	us e- cs ilc	us e- -1	ar	i- zk- au e- cs ilc 0 zc m	m e:	A			指標	1 1 1: 3	3	0	5	0	0	0	0	0	4	7	2	4	4	3	8	6	2	1	2	1					
us e	us e- cc	us e- cs ilc	us e- -1	ar	i- zk- au e- cs ilc 0 zc m	m e:	A			指標	1 1 1: 3	5	0	3	0	0	0	0	0	5	3	8	5	7	1	8	6	7	6	5	3					

re	信有格	錢洛	ar ity	mi ic	in n	in ty	o n	pa	jo o	ma a	時間	c	c m	c	c	c t	c	c	c	m e	m cl	d e	la e	d ri	la e	d	d	n yt	n yt	fil rin	m e	ta st	g	ta le	gs eo ml	seconda
us e	us e- c	us e- c sil	us e -1	ar	i- za k- au us e- c sil 0 zo m			ho g	JI	事件	1 1 1 4																				7	SI TS	8	3	63	3.8 RE	W
us e	us e- c	us e- c sil	us e -1	ar	i- za k- au us e- c sil 0 zo m			ho g	JI	事件	1 1 1 4																				7	SI TT	2	4	73	3.5 JL	W

re	信存格	錄況	a ity	m ic	in n	ir ty	o n	p a	jo o	m a	時間	c	c m	c	c	c t	c	c	c	m e	m cl	d e	la e	d ri	la e	d	d	n yt	n yt	fil ri n	m e	ta st	g	ta le	g m	scen comp led
u e	u e - c	u e - c s il	u e -1	a	i- z k a u e - c s il 0 z m			h g	J	事件	1 1 1. 3																				6	S I T	6	2	7	2_3W JL
u e	u e - c	u e - c s il	u e -1	a	i- z k a u e - c s il 0 z m			h g	J	事件	1 1 1. 3																				2	S I T	5	1	9	9_W JL

re	信存格	錄浴	ai ity	mi ic	in n	in ty	o n	pa a	jo o	m a	時間	c	c m	c	c	c t	c	c	c	c	m e	m cl	d e	la e	d ri	la e	d	d	n yt	n yt	fil ri n	m e	ta st	g	ta le	gs ml	sc la
us e	us e- c	us - cs il	us e- -1	a	i- za k- a u e- c s il 0 z m			ho g	JI	事件	1 1 1 3																					3	S IT S	7	2	82	2.9 RE

單一測量記錄

的 Timestream LiveAnalytics 也可讓您擷取資料，每個時間序列記錄使用一個量值。以下是使用單一量值記錄擷取時的結構描述詳細資訊。

資料表結構描述

以下是使用多測量記錄擷取資料後的資料表結構描述。這是DESCRIBE查詢的輸出。假設資料擷取到資料庫 raw_data 和資料表開發，以下是查詢。

```
DESCRIBE "raw_data"."devops_single"
```

資料行	Type	LiveAnalytics 屬性類型的時間串流
availability_zone	varchar	DIMENSION
microservice_name	varchar	DIMENSION

資料行	Type	LiveAnalytics 屬性類型的時間串流
instance_name	varchar	DIMENSION
process_name	varchar	DIMENSION
os_version	varchar	DIMENSION
jdk_version	varchar	DIMENSION
儲存格	varchar	DIMENSION
region	varchar	DIMENSION
孤島	varchar	DIMENSION
instance_type	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
time	timestamp	TIMESTAMP
measure_value::double	double	MEASURE_VALUE
measure_value::bigint	bigint	MEASURE_VALUE
measure_value::varchar	varchar	MEASURE_VALUE

測量結構描述

以下是SHOWMEASURES查詢傳回的量值結構描述。

```
SHOW MEASURES FROM "raw_data"."devops_single"
```

measure_name	data_type	維度
cpu_hi	double	【{'dimension_name'： 'availability_zone'， 'data_type'： 'varchar'}， {'dimensi

measure_name	data_type	維度
		on_name' : 'microser vice_name' , 'data_typ e' : 'varchar'} , {'dimensi on_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_versi on' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's
cpu_idle	double	【{'dimension_name' : 'availability_zone' , 'data_typ e' : 'varchar'} , {'dimensi on_name' : 'microser vice_name' , 'data_typ e' : 'varchar'} , {'dimensi on_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_versi on' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's

measure_name	data_type	維度
cpu_iowait	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's
cpu_nice	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's

measure_name	data_type	維度
cpu_si	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's
cpu_steal	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's

measure_name	data_type	維度
cpu_system	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's
cpu_user	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's

measure_name	data_type	維度
disk_free	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's
disk_io_reads	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's

measure_name	data_type	維度
disk_io_writes	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's
disk_used	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's

measure_name	data_type	維度
file_descriptors_in_use	double	【{'dimension_name' : 'availability_zone' , 'data_typ e' : 'varchar'} , {'dimensi on_name' : 'microser vice_name' , 'data_typ e' : 'varchar'} , {'dimensi on_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_versi on' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's
gc_pause	double	【{'dimension_name' : 'availability_zone' , 'data_typ e' : 'varchar'} , {'dimensi on_name' : 'microser vice_name' , 'data_typ e' : 'varchar'} , {'dimensi on_name' : 'instance_name' , 'data_type' : 'varchar' } , {'dimension_name' : 'process_name' , 'data_typ e' : 'varchar' , {'dimensi on_name' : 'jdk_version' , 'data_type' : 'varchar'} , {dimension_name' : 'cell' , 'type' : 'divarchar'

measure_name	data_type	維度
gc_reclaimed	double	【{'dimension_name' : 'availability_zone' , 'data_typ e' : 'varchar'} , {'dimensi on_name' : 'microser vice_name' , 'data_typ e' : 'varchar'} , {'dimensi on_name' : 'instance_name' , 'data_type' : 'varchar' } , {'dimension_name' : 'process_name' , 'data_typ e' : 'varchar' , {'dimensi on_name' : 'jdk_version' , 'data_type' : 'varchar'} , {dimension_name' : 'cell' , 'type' : 'divarchar'
latency_per_read	double	【{'dimension_name' : 'availability_zone' , 'data_typ e' : 'varchar'} , {'dimensi on_name' : 'microser vice_name' , 'data_typ e' : 'varchar'} , {'dimensi on_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_versi on' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's

measure_name	data_type	維度
latency_per_write	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's
memory_cached	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's

measure_name	data_type	維度
memory_free	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar' } , {'dimension_name' : 'process_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar'} , {'dimension_name' : 'jdk_version' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name' : '區域' , 'data_type' : 'varchar' } , {'dimension_name' : 「孤島」 , 'data_type' : 'varchar' } , {'dimension_name' : 'instance_type' , 'data_type' : 'varchar'}】

measure_name	data_type	維度
memory_used	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar'} , {'dimension_name' : 'cell' , 'data_type' : 'varchar'} , {'dimension_name'' : 'region' , 's
network_bytes_in	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar'} , {'dimension_name' : 'cell' , 'data_type' : 'varchar'} , {'dimension_name'' : 'region' , 's

measure_name	data_type	維度
network_bytes_out	double	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'os_version' , 'data_type' : 'varchar' } , {'dimension_name' : 'cell' , 'data_type' : 'varchar' } , {'dimension_name'' : 'region' , 's
task_completed	bigint	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar' } , {'dimension_name' : 'process_name' , 'data_type' : 'varchar' , {'dimension_name' : 'jdk_version' , 'data_type' : 'varchar'} , {dimension_name' : 'cell' , 'type' : 'divarchar'

measure_name	data_type	維度
task_end_state	varchar	【{'dimension_name' : 'availability_zone' , 'data_type' : 'varchar'} , {'dimension_name' : 'microservice_name' , 'data_type' : 'varchar'} , {'dimension_name' : 'instance_name' , 'data_type' : 'varchar' } , {'dimension_name' : 'process_name' , 'data_type' : 'varchar' , {'dimension_name' : 'jdk_version' , 'data_type' : 'varchar'} , {dimension_name' : 'cell' , 'type' : 'divarchar'

範例資料

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	儲存格	regio	錫洛	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	遺傳者	i-zaZsvk-hercus-eu-west-1-cell-9-silo-20000(		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	cpu_l	34 : 57.2	0.871		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	儲存格	regio	錫洛	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
		mazo com												
eu- west- -1	遺傳者	i- zaZs\ k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_i	34： 57.2	3.462		
eu- west- -1	遺傳者	i- zaZs\ k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_i t	34： 57.2	0.102		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	儲存格	regio	錫洛	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	遺傳者	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	cpu_r	34：57.2	0.630		
eu-west--1	遺傳者	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	cpu_s	34：57.2	0.164		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	儲存格	regio	錫洛	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	遺傳者	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_s	34： 57.2	0.107		
eu- west- -1	遺傳者	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_s m	34： 57.2	0.457		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	儲存格	regio	錫洛	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	遺傳者	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	cpu_u	34：57.2	20448年94月		
eu-west--1	遺傳者	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	disk_	34：57.2	72.51		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	儲存格	regio	錫洛	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	遺傳者	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	disk_ieads	34：57.2	81.73		
eu-west--1	遺傳者	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	disk_irites	34：57.2	77.11		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	儲存格	regio	錫洛	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	遺傳者	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200006mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	disk_	34：57.2	89.42		
eu-west--1	遺傳者	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200006mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	file_driptorsn_use	34：57.2	30.08		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	儲存格	regio	錫洛	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	遺傳者	i-zaZsv k-hercu s-eu-west -1-cell-9 - silo-2 0000(mazo com	伺服器		JDK_	eu-west- - cell-9	eu-west-	eu-west- - cell-9 s ilo-2		gc_pa	34：57.2	60.28		
eu-west--1	遺傳者	i-zaZsv k-hercu s-eu-west -1-cell-9 - silo-2 0000(mazo com	伺服器		JDK_	eu-west- - cell-9	eu-west-	eu-west- - cell-9 s ilo-2		gc_re med	34：57.2	75.28		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	儲存格	regio	錫洛	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	遺傳者	i-zaZsv k-hercu s-eu-west -1-cell-9 - silo-2 0000(mazo com		AL20		eu-west- -cell-9	eu-west-	eu-west- -cell-9 s ilo-2	r5.4xl e	laten(er_re	34 : 57.2	8.076		
eu-west--1	遺傳者	i-zaZsv k-hercu s-eu-west -1-cell-9 - silo-2 0000(mazo com		AL20		eu-west- -cell-9	eu-west-	eu-west- -cell-9 s ilo-2	r5.4xl e	laten(er_wr	34 : 57.2	58.11		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	儲存格	regio	錫洛	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	遺傳者	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	memched	34：57.2	87.56		
eu-west--1	遺傳者	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom	伺服器		JDK_	eu-west--cell-9	eu-west-	eu-west--cell-9silo-2		memee	34：57.2	18.95		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	儲存格	regio	錫洛	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	遺傳者	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	mem ee	34： 57.2	2052 年 7 月 97 日		
eu- west- -1	遺傳者	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	mem ed	34： 57.2	12.37		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	儲存格	regio	錫洛	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	遺傳者	i-zaZsv k-hercu s-eu-west -1-cell-9 - silo-2 0000(mazo com		AL20		eu-west- -cell-9	eu-west-	eu-west- -cell-9 s ilo-2	r5.4xl e	netwo ytes_	34：57.2	2065年2月31日		
eu-west--1	遺傳者	i-zaZsv k-hercu s-eu-west -1-cell-9 - silo-2 0000(mazo com		AL20		eu-west- -cell-9	eu-west-	eu-west- -cell-9 s ilo-2	r5.4xl e	netwo ytes_	34：57.2	0.514		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	儲存格	regio	錫洛	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue:: varchar
eu-west--1	遺傳者	i-zaZsv k-hercu s-eu-west -1-cell-9 - silo-2 0000(mazo com	伺服器		JDK_	eu-west- - cell-9	eu-west-	eu-west- - cell-9 s ilo-2		task_ leted	34：57.2		69	
eu-west--1	遺傳者	i-zaZsv k-hercu s-eu-west -1-cell-9 - silo-2 0000(mazo com	伺服器		JDK_	eu-west- - cell-9	eu-west-	eu-west- - cell-9 s ilo-2		task_ state	34：57.2			SUCCESS_ ITH_RESUL T

排程查詢模式

在本節中，您會找到一些常見模式，說明如何使用 Amazon Timestream for LiveAnalytics Scheduled Queries 來最佳化儀表板，以更快且降低成本。下列範例使用 DevOps 應用程式案例來說明適用於一般排程查詢的關鍵概念，無論應用程式案例為何。

Timestream 中的排程查詢 LiveAnalytics 可讓您使用 Timestream for 的完整SQL表面積來表達查詢 LiveAnalytics。您的查詢可以包含一或多個來源資料表、針對 LiveAnalyticsSQL的語言執行 Timestream 允許的彙總或任何其他查詢，然後在 Timestream 中的另一個目的地資料表中具體化查詢的結果 LiveAnalytics。為了便於表達，本節將排程查詢的此目標資料表稱為衍生資料表 。

以下是本節涵蓋的要點。

- 使用簡單的機群層級彙總來說明如何定義排程查詢並了解一些基本概念。
- 如何結合排程查詢 （衍生資料表） 目標的結果與來源資料表的結果，以取得排程查詢的成本和效能優勢。
- 設定排程查詢的重新整理期間時，您的取捨是什麼。
- 針對某些常見案例使用排程查詢。
  - 追蹤特定日期之前每個執行個體的最後一個資料點。
  - 要用於填入儀表板中變數的維度差異值。
- 您在排程查詢內容中處理延遲到達資料的方式。
- 如何使用一次性手動執行來處理排程查詢自動觸發程序未直接涵蓋的各種案例。

主題

案例

下列範例使用 中概述的 DevOps 監控案例排程查詢範例結構描述。

這些範例提供排程查詢定義，您可以在其中插入適當的組態，以便接收排程查詢的執行狀態通知、接收執行排程查詢期間所遇到錯誤的報告，以及排程查詢用來執行其操作IAM的角色。

您可以在填寫上述選項、建立目標 （或衍生） 資料表，以及透過 執行 AWS 之後，建立這些排程查詢 CLI。例如，假設排程查詢定義儲存在 檔案 中scheduled_query_example.json。您可以使用 CLI 命令建立查詢。

```
aws timestream-query create-scheduled-query --cli-input-json  file://
scheduled_query_example.json --profile aws_profile --region us-east-1
```

在上述命令中，使用 --profile 選項傳遞的設定檔必須具有建立排程查詢的適當許可。如需政策和許可的詳細說明，請參閱排程查詢的身分型政策。

簡單機群層級彙總

第一個範例會逐步解說使用簡單範例運算機群層級彙總處理排程查詢時的一些基本概念。使用此範例，您將學到以下內容。

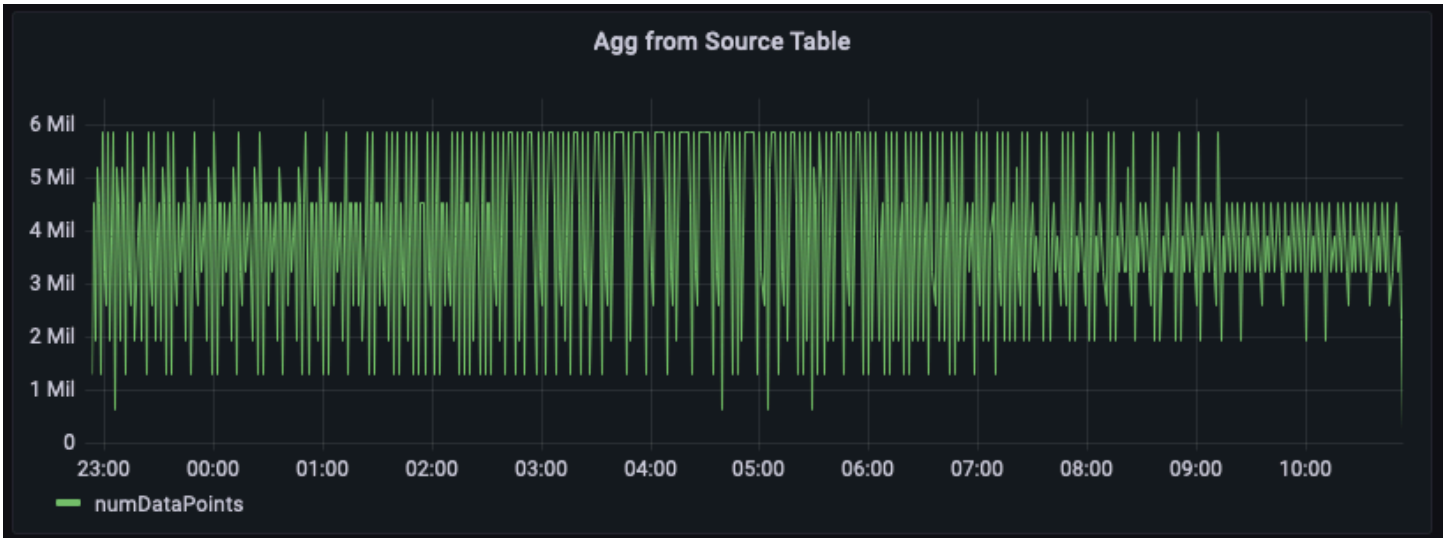
- 如何取得用於取得彙總統計資料的儀表板查詢，並將其對應至排程查詢。
- 的 Timestream 如何 LiveAnalytics 管理排程查詢不同執行個體的執行。
- 如何讓排程查詢的不同執行個體在時間範圍中重疊，以及如何在目標資料表上維護資料的正確性，以確保使用排程查詢結果的儀表板為您提供的結果與原始資料上計算的相同彙總相符。
- 如何設定排程查詢的時間範圍和重新整理節奏。
- 如何自助追蹤排程查詢的結果以進行調校，讓查詢執行個體的執行延遲落在重新整理儀表板的可接受延遲範圍內。

主題

- 從來源資料表彙總
- 預先運算彙總的排程查詢
- 從衍生資料表彙總
- 彙總合併來源和衍生資料表
- 從經常重新整理的排程運算彙總

從來源資料表彙總

在此範例中，您要每分鐘追蹤指定區域內伺服器發出的指標數量。下圖是繪製 us-east-1 區域此時間序列的範例。



以下是從原始資料計算此彙總的範例查詢。它會篩選 us-east-1 區域的列，然後計算每分鐘總和，並考慮 20 個指標（如果 measure_name 是指標）或 5 個事件（如果 measure_name 是事件）。在此範例中，圖表圖解顯示發出的指標數量在每分鐘 150 萬到 600 萬之間。繪製此時間序列數小時（此圖中過去 12 小時）時，原始資料上的此查詢會分析數億列。

```
WITH grouped_data AS (
    SELECT region, bin(time, 1m) as minute, SUM(CASE WHEN measure_name = 'metrics' THEN
 20 ELSE 5 END) as numDataPoints
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636699996445) AND
 from_milliseconds(1636743196445)
        AND region = 'us-east-1'
    GROUP BY region, measure_name, bin(time, 1m)
)
SELECT minute, SUM(numDataPoints) AS numDataPoints
FROM grouped_data
GROUP BY minute
ORDER BY 1 desc, 2 desc
```

預先運算彙總的排程查詢

如果您想要最佳化儀表板以更快載入，並透過掃描較少的資料來降低成本，您可以使用排程查詢來預先計算這些彙總。Timestream for 中的排程查詢 LiveAnalytics 可讓您在另一個 Timestream 中實現這些預先運算的 LiveAnalytics 資料表，之後可用於儀表板。

建立排程查詢的第一個步驟是識別您要預先運算的查詢。請注意，上述儀表板已繪製為 us-east-1 區域。不過，不同的使用者可能會想要不同區域的相同彙總，例如 us-west-2 或 eu-west-1。為了避免

為每個此類查詢建立排程查詢，您可以預先計算每個區域的彙總，並在另一個 Timestream 中具體化 LiveAnalytics 每個區域的彙總。

下面的查詢提供對應的預先運算範例。如您所見，它類似於原始資料查詢中使用的常見資料表表達式 grouped_data，除了兩個差異：1）它不使用區域述詞，因此我們可以針對所有區域使用一個查詢來預先計算；以及 2）它使用具有特殊參數 @scheduled_runtime 的參數化時間述詞，詳細解釋如下。

```
SELECT region, bin(time, 1m) as minute,
    SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as numDataPoints
FROM raw_data.devops
WHERE time BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m
GROUP BY bin(time, 1m), region
```

上述查詢可以使用下列規格轉換為排程查詢。排程查詢會指派一個名稱，這是一種易於使用的語詞。然後包含 QueryString，即 ScheduleConfiguration，這是 cron 表達式 。它指定將查詢結果 TargetConfiguration 映射到 Timestream 中目的地資料表的 LiveAnalytics。最後，它會指定許多其他組態，例如 NotificationConfiguration，其中會傳送通知以進行查詢的個別執行， ErrorReportConfiguration 其中會撰寫報告，以防查詢遇到任何錯誤，以及 ScheduledQueryExecutionRoleArn，這是用來執行排程查詢操作的角色。

```
{
    "Name": "MultiPT5mPerMinutePerRegionMeasureCount",
    "QueryString": "SELECT region, bin(time, 1m) as minute, SUM(CASE WHEN measure_name
 = 'metrics' THEN 20 ELSE 5 END) as numDataPoints FROM raw_data.devops WHERE time
 BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m GROUP BY bin(time, 1m),
 region",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/5 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "per_minute_aggs_pt5m",
            "TimeColumn": "minute",
            "DimensionMappings": [
                {
                    "Name": "region",
```

```
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

在此範例中， ScheduleExpression cron （ 0/5 * * ？ * ）表示查詢會在每天每小時的 5、10、15、.. 分鐘執行一次。觸發此查詢的特定執行個體時，這些時間戳記會轉換為查詢中使用的 @scheduled_runtime 參數。例如，請考慮在 2021-12-01 00：00：00 執行此排程查詢的執行個體。在此執行個體中，@scheduled_runtime 參數會在叫用查詢時初始化為時間戳記 2021-12-01 00：00：00。因此，此特定執行個體將在時間戳記 2021-12-01 00：00：00 執行，並計算從時間範圍 2021-11-30 23：50：00 到 2021-12-01 00：01：00 的每分鐘彙總。同樣地，此查詢的下一個執行個體會在時間戳記 2021-12-01 00：05：00 觸發，在這種情況下，查詢將計算從時間範圍 2021-11-30 23：55：00 到 2021-12-01 00：06：00 的每分鐘彙總。因此，@scheduled_runtime 參數提供排程查詢，以使用查詢的調用時間預先計算已設定時間範圍的彙總。

請注意，查詢的兩個後續執行個體在其時間範圍中重疊。這是您可以根據需求控制的項目。在這種情況下，此重疊允許這些查詢根據其到達稍有延遲的任何資料更新彙總，在此範例中最多 5 分鐘。為了確保具體化查詢的正確性， 的 Timestream LiveAnalytics 可確保在 2021-12-01 00：05：00 的查詢只有在 2021-12-01 00：00：00 完成查詢後才會執行，而後者查詢的結果可以在產生較新的值時，使用 更新任何先前具體化的彙總。例如，如果在時間戳記 2021-11-30 23：59：00 的某些資料在執行 2021-12-01 00：00：00 的查詢之後，但在 2021-12-01 00：05：00 的查詢之前到達，則 2021-12-01 00：05：00 的執行將重新計算 2021-11-30 23：59：00 分鐘的彙總，這將導致先前的彙總更新為新計

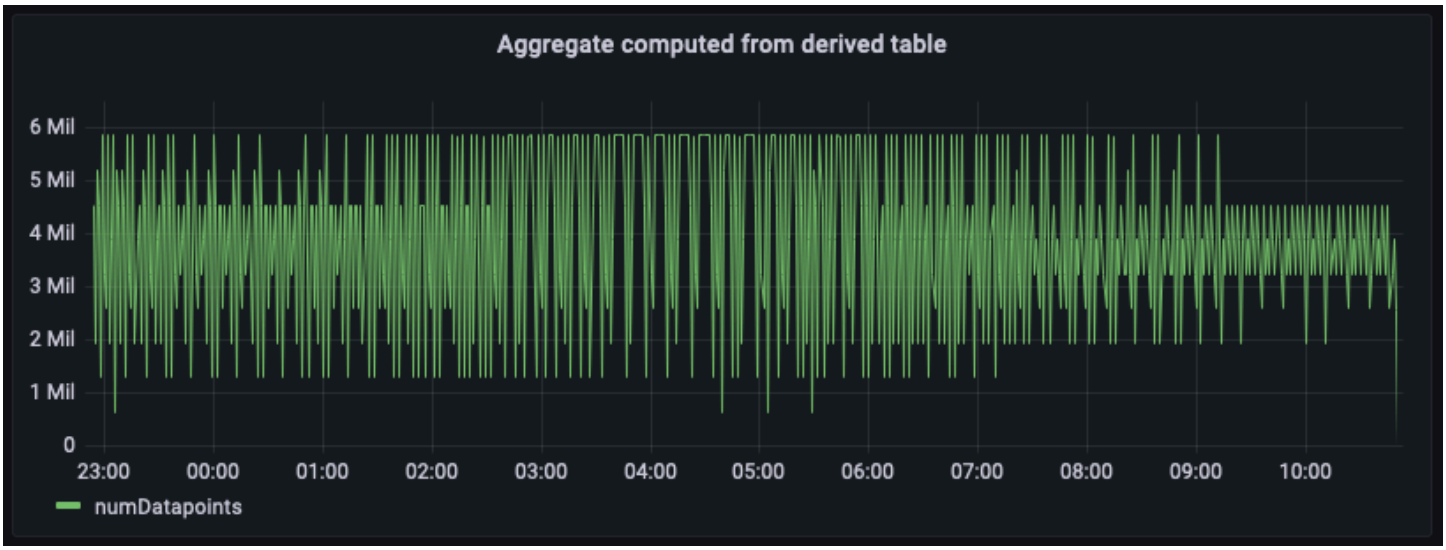
算的值。您可以依賴排程查詢的這些語義，在更新預先計算的速度與如何優雅地處理某些延遲到達的資料之間取得權衡。以下討論其他考量事項，包括如何以資料的新鮮度權衡此重新整理節奏，以及如何解決更新抵達資料延遲時間更久的彙總，或排程運算的來源是否有更新值，這需要重新計算彙總。

每個排程運算都有通知組態，其中 的 Timestream 會 LiveAnalytics 傳送每次執行排程組態的通知。您可以設定 SNS的主題，以接收每次調用的通知。除了特定執行個體的成功或失敗狀態之外，它還有數個統計資料，例如此運算執行所花費的時間、掃描的運算位元組數，以及運算寫入其目的地資料表的位元組數。您可以使用這些統計資料來進一步調整查詢、排程組態，或追蹤排程查詢的支出。值得注意的一個方面是執行個體的執行時間。在此範例中，排程運算設定為每 5 分鐘執行一次。執行時間將決定可用的預先運算延遲，當您在儀表板中使用預先計算的資料時，也會定義儀表板中的延遲。此外，如果此延遲持續高於重新整理間隔，例如，如果設定為每 5 分鐘重新整理一次的運算的執行時間超過 5 分鐘，請務必調整運算以更快地執行，以避免儀表板中進一步延遲。

從衍生資料表彙總

現在您已設定排程查詢，且彙總已預先計算並具體化至排程運算目標組態中指定 LiveAnalytics 資料表的另一個 Timestream，您可以使用該資料表中的資料撰寫SQL查詢來為儀表板提供動力。以下是使用具體化預先彙總來產生 us-east-1 每分鐘資料點計數彙總的查詢。

```
SELECT bin(time, 1m) as minute, SUM(numDataPoints) as numDatapoints
FROM "derived"."per_minute_aggs_pt5m"
WHERE time BETWEEN from_milliseconds(1636699996445) AND
 from_milliseconds(1636743196445)
    AND region = 'us-east-1'
GROUP BY bin(time, 1m)
ORDER BY 1 desc
```



上一個圖形會繪製從彙總資料表計算的彙總。將此面板與從原始來源資料計算的面板進行比較，您會注意到它們完全相符，儘管這些彙總延遲幾分鐘，但受到您為排程運算設定的重新整理間隔以及執行時間的控制。

相較於透過原始來源資料計算的彙總，此預先計算資料上的查詢會掃描數個數量級更少的資料。根據彙總的精細程度，這種減少可以輕鬆降低 100X的成本和查詢延遲。執行此排程運算需要成本。不過，根據這些儀表板的重新整理頻率以及並行使用者載入這些儀表板的頻率，您最終會使用這些預先運算大幅降低整體成本。而且儀表板的載入時間快上 10-100X。

彙總合併來源和衍生資料表

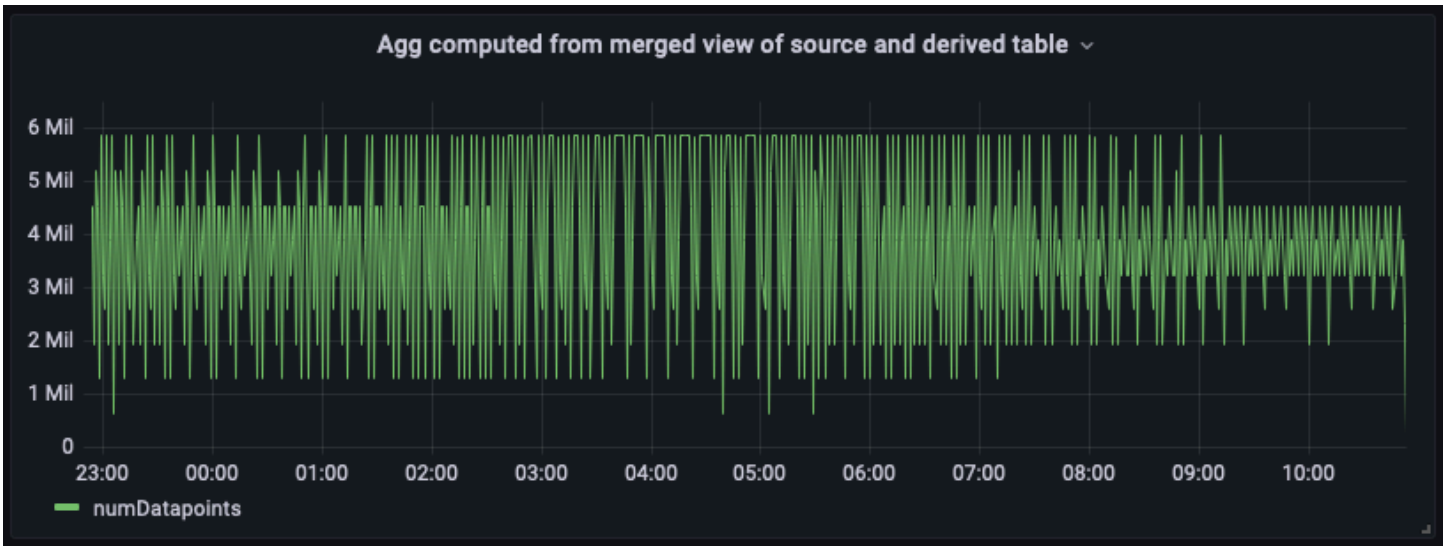
使用衍生資料表建立的儀表板可能會有延遲。如果您的應用程式案例需要儀表板擁有最新的資料，則您可以使用 Timestream 的強大功能和彈性，支援 LiveAnalyticsSQL將來源資料表的最新資料與衍生資料表的歷史彙總結合，以形成合併檢視。此合併檢視使用來源SQL和衍生資料表的 和 非重疊時間範圍的聯合語義。在下面的範例中，我們使用「derived」."per_minute_aggs_pt5m」衍生的資料表。由於衍生資料表的排程運算每 5 分鐘重新整理一次（根據排程表達式規格），下面的查詢使用來源資料表中最近 15 分鐘的資料， 以及從衍生資料表中超過 15 分鐘的任何資料，然後結合結果以建立具有兩個世界最佳效能的合併檢視： 從衍生的資料表讀取較舊的預先計算彙總，以及來源資料表的彙總新鮮度，以為您的即時分析使用案例提供動力，進而經濟實惠且低延遲。

請注意，與只查詢衍生的資料表相比，此聯合方法的查詢延遲略高，而且掃描的資料略高，因為其正在即時彙總原始資料以填入最新的時間間隔。不過，與從來源資料表快速彙總相比，此合併檢視仍然會明顯更快且更便宜，特別是對於呈現數天或數週資料的儀表板。您可以調整此範例的時間範圍，以符合應用程式的重新整理需求和延遲容差。

```
WITH aggregated_source_data AS (
    SELECT bin(time, 1m) as minute, SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE
 5 END) as numDatapoints
    FROM "raw_data"."devops"
    WHERE time BETWEEN bin(from_milliseconds(1636743196439), 1m) - 15m AND
 from_milliseconds(1636743196439)
        AND region = 'us-east-1'
    GROUP BY bin(time, 1m)
), aggregated_derived_data AS (
    SELECT bin(time, 1m) as minute, SUM(numDataPoints) as numDatapoints
    FROM "derived"."per_minute_aggs_pt5m"
    WHERE time BETWEEN from_milliseconds(1636699996439) AND
 bin(from_milliseconds(1636743196439), 1m) - 15m
        AND region = 'us-east-1'
    GROUP BY bin(time, 1m)
)
```

```
SELECT minute, numDatapoints
FROM (
    (
    SELECT *
    FROM aggregated_derived_data
    )
    UNION
    (
    SELECT *
    FROM aggregated_source_data
    )
)
ORDER BY 1 desc
```

以下是具有此統一合併檢視的儀表板面板。如您所見，儀表板看起來幾乎與從衍生資料表計算的檢視相同，但最右邊的 將具有最多 up-to-date彙總。



從經常重新整理的排程運算彙總

根據儀表板的載入頻率和儀表板所需的延遲，還有另一種方法可在儀表板中取得更新鮮的結果：讓排程運算更頻繁地重新整理彙總。例如，以下是相同排程運算的組態，除了每分鐘重新整理一次（請注意排程 express cron （ 0/1 * * * ？ * ））。使用此設定時，衍生的 table per_minute_aggs_pt1m 將比運算指定重新整理排程為每 5 分鐘一次的情況有更近期的彙總。

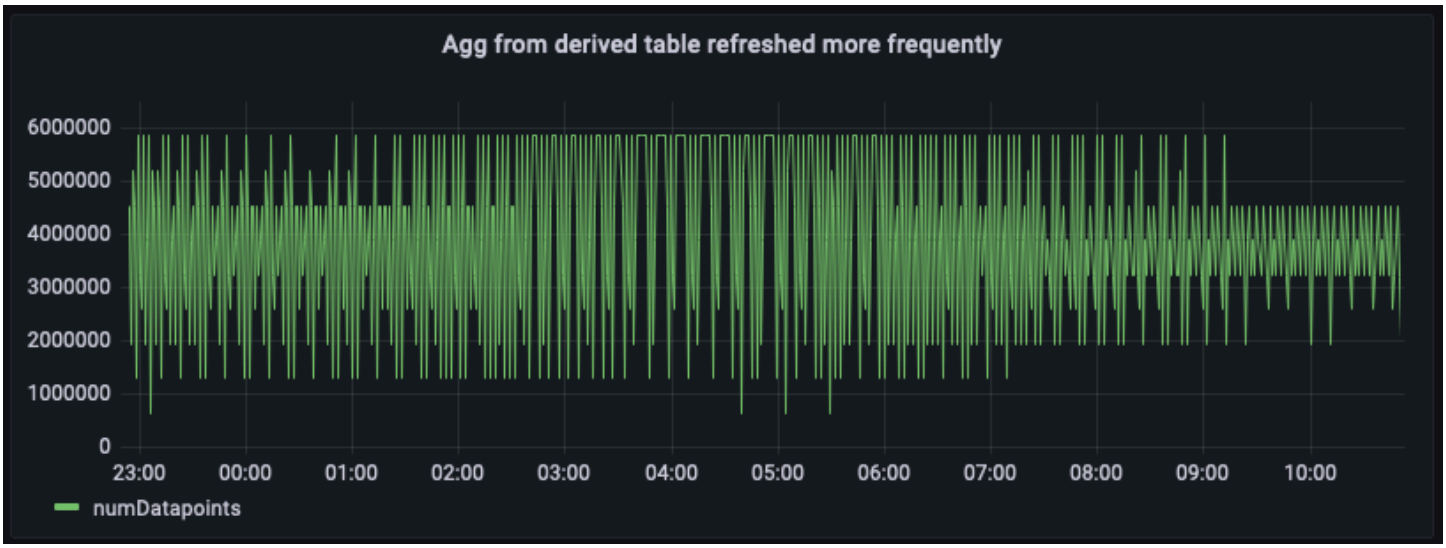
```
{
    "Name": "MultiPT1mPerMinutePerRegionMeasureCount",
    "QueryString": "SELECT region, bin(time, 1m) as minute, SUM(CASE WHEN measure_name
 = 'metrics' THEN 20 ELSE 5 END) as numDataPoints FROM raw_data.devops WHERE time
```

```
 BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m GROUP BY bin(time, 1m),
 region",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/1 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "per_minute_aggs_pt1m",
            "TimeColumn": "minute",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

```
SELECT bin(time, 1m) as minute, SUM(numDataPoints) as numDatapoints
```

```
FROM "derived"."per_minute_aggs_pt1m"
WHERE time BETWEEN from_milliseconds(1636699996446) AND
 from_milliseconds(1636743196446)
    AND region = 'us-east-1'
GROUP BY bin(time, 1m), region
ORDER BY 1 desc
```

由於衍生的資料表具有較新的彙總，因此您現在可以直接查詢衍生的資料表 per_minute_aggs_pt1m，以取得較新的彙總，如先前的查詢和下方的儀表板快照所示。



請注意，以更快的排程重新整理排程的運算 （例如 1 分鐘相較於 5 分鐘） 將增加排程運算的維護成本。每個運算執行的通知訊息都會提供掃描多少資料，以及寫入衍生資料表多少資料的統計資料。同樣地，如果您使用合併檢視來結合衍生的資料表，您會查詢合併檢視的成本，而且儀表板載入延遲會比只查詢衍生的資料表更高。因此，您選擇的方法取決於儀表板重新整理的頻率，以及排程查詢的維護成本。如果您有數十個使用者每分鐘重新整理一次儀表板，則更頻繁地重新整理衍生資料表可能會導致整體成本降低。

每個裝置的最後一個點

您的應用程式可能需要您讀取裝置發出的上次測量。在指定的 之前，可能會有更一般的使用案例來取得裝置的最後一次測量date/time or the first measurement for a device after a given date/time。當您擁有數百萬個裝置和多年的資料時，此搜尋可能需要掃描大量資料。

您會看到以下範例，說明如何使用排程查詢來最佳化搜尋裝置發出的最後一個點。如果您的應用程式需要，您也可以使用相同的模式來最佳化第一個點查詢。

主題

- 從來源資料表運算

- 衍生資料表，以每日精細程度預先運算
- 從衍生資料表運算
- 從來源和衍生資料表合併

從來源資料表運算

以下是尋找特定部署中（例如，指定區域、儲存格、孤島和 availability_zone 內特定微服務之伺服器）服務所發出之上次測量的範例查詢。在範例應用程式中，此查詢將傳回數百個伺服器的上次測量。另請注意，此查詢具有未繫結的時間述詞，並尋找任何早於指定時間戳記的資料。

> **Note**
>
> 如需 max和 max_by函數的相關資訊，請參閱 彙總函數。

```
SELECT instance_name, MAX(time) AS time, MAX_BY(gc_pause, time) AS last_measure
FROM "raw_data"."devops"
WHERE time < from_milliseconds(1636685271872)
    AND measure_name = 'events'
    AND region = 'us-east-1'
    AND cell = 'us-east-1-cell-10'
    AND silo = 'us-east-1-cell-10-silo-3'
    AND availability_zone = 'us-east-1-1'
    AND microservice_name = 'hercules'
GROUP BY region, cell, silo, availability_zone, microservice_name,
    instance_name, process_name, jdk_version
ORDER BY instance_name, time DESC
```

衍生資料表，以每日精細程度預先運算

您可以將上述使用案例轉換為排定的運算。如果您的應用程式需求需要跨多個區域、儲存格、孤島、可用區域和微服務取得整個機群的這些值，您可以使用一個排程運算來預先計算整個機群的值。這就是 Timestream 對 無 LiveAnalytics伺服器排程查詢的強大功能，允許這些查詢擴展到您應用程式的擴展需求。

以下是在特定日期預先計算所有伺服器最後一個點的查詢。請注意，查詢只有時間述詞，而不是維度的述詞。時間述詞會將查詢限制為根據指定排程表達式觸發運算後的過去一天。

```
SELECT region, cell, silo, availability_zone, microservice_name,
    instance_name, process_name, jdk_version,
```

```
    MAX(time) AS time, MAX_BY(gc_pause, time) AS last_measure
FROM raw_data.devops
WHERE time BETWEEN bin(@scheduled_runtime, 1d) - 1d AND bin(@scheduled_runtime, 1d)
    AND measure_name = 'events'
GROUP BY region, cell, silo, availability_zone, microservice_name,
    instance_name, process_name, jdk_version
```

以下是使用上述查詢的排程運算組態，該查詢UTC會在每天 01：00 執行該查詢，以計算過去一天的彙總。排程表達式 cron（0 1 * ？ *） 控制此行為，並在一天結束後執行一小時，以考慮到達延遲一天之前的任何資料。

```
{
    "Name": "PT1DPerInstanceLastpoint",
    "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
 instance_name, process_name, jdk_version, MAX(time) AS time, MAX_BY(gc_pause, time)
 AS last_measure FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime, 1d) -
 1d AND bin(@scheduled_runtime, 1d) AND measure_name = 'events' GROUP BY region, cell,
 silo, availability_zone, microservice_name, instance_name, process_name, jdk_version",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0 1 * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "per_timeseries_lastpoint_pt1d",
            "TimeColumn": "time",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
```

```
                },
                {
                    "Name": "availability_zone",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "process_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "jdk_version",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "last_measure",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "last_measure",
                        "MeasureValueType": "DOUBLE"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

從衍生資料表運算

使用上述組態定義衍生的資料表，且排程查詢的至少一個執行個體已將資料具體化到衍生的資料表後，您現在可以查詢衍生的資料表以取得最新的測量。以下是衍生資料表的範例查詢。

```
SELECT instance_name, MAX(time) AS time, MAX_BY(last_measure, time) AS last_measure
FROM "derived"."per_timeseries_lastpoint_pt1d"
WHERE time < from_milliseconds(1636746715649)
    AND measure_name = 'last_measure'
    AND region = 'us-east-1'
    AND cell = 'us-east-1-cell-10'
    AND silo = 'us-east-1-cell-10-silo-3'
    AND availability_zone = 'us-east-1-1'
    AND microservice_name = 'hercules'
GROUP BY region, cell, silo, availability_zone, microservice_name,
    instance_name, process_name, jdk_version
ORDER BY instance_name, time DESC
```

從來源和衍生資料表合併

與先前的範例類似，衍生資料表中的任何資料都不會有最新的寫入。因此，您可以再次使用與先前類似的模式，以合併舊資料衍生資料表中的資料，並針對剩餘的提示使用來源資料。以下是使用類似UNION方法的此類查詢範例。由於應用程式需求是在一段時間之前尋找最新的測量，而且此開始時間可以過去，因此撰寫此查詢的方式是使用提供的時間，從指定時間起使用最多一天的來源資料，然後在舊資料上使用衍生的資料表。如下方查詢範例所示，來源資料上的時間述詞會繫結。這可確保在具有明顯較高資料量的來源資料表上進行有效處理，然後未繫結的時間述詞位於衍生的資料表上。

```
WITH last_point_derived AS (
    SELECT instance_name, MAX(time) AS time, MAX_BY(last_measure, time) AS last_measure
    FROM "derived"."per_timeseries_lastpoint_pt1d"
    WHERE time < from_milliseconds(1636746715649)
        AND measure_name = 'last_measure'
        AND region = 'us-east-1'
        AND cell = 'us-east-1-cell-10'
        AND silo = 'us-east-1-cell-10-silo-3'
        AND availability_zone = 'us-east-1-1'
        AND microservice_name = 'hercules'
    GROUP BY region, cell, silo, availability_zone, microservice_name,
        instance_name, process_name, jdk_version
), last_point_source AS (
    SELECT instance_name, MAX(time) AS time, MAX_BY(gc_pause, time) AS last_measure
    FROM "raw_data"."devops"
```

```
    WHERE time < from_milliseconds(1636746715649) AND time >
 from_milliseconds(1636746715649) - 26h
        AND measure_name = 'events'
        AND region = 'us-east-1'
        AND cell = 'us-east-1-cell-10'
        AND silo = 'us-east-1-cell-10-silo-3'
        AND availability_zone = 'us-east-1-1'
        AND microservice_name = 'hercules'
    GROUP BY region, cell, silo, availability_zone, microservice_name,
        instance_name, process_name, jdk_version
)
SELECT instance_name, MAX(time) AS time, MAX_BY(last_measure, time) AS last_measure
FROM (
    SELECT * FROM last_point_derived
    UNION
    SELECT * FROM last_point_source
)
GROUP BY instance_name
ORDER BY instance_name, time DESC
```

上一個只是如何建構衍生資料表的一個說明。如果您有多年的資料，您可以使用更多層級的彙總。例如，除了每日彙總之外，您還可以擁有每月彙總，而且您可以在每日彙總之前每小時彙總。因此，您可以合併最近的 來填寫上一個小時、每小時填寫上一天的 、上個月填寫的 ，以及每月填寫較舊的 。您設定與重新整理排程的層級數量，取決於您對這些問題的頻率，以及同時發出這些查詢的使用者數量的需求。

唯一維度值

您可能有一個使用案例，其中有儀表板，您想要使用維度的唯一值作為變數，以深入探索對應於特定資料層的指標。下面的快照是範例，其中儀表板會預先填入多個維度的唯一值，例如區域、儲存格、孤島、微服務和 availability_zone。此處我們示範如何使用排程查詢，從您追蹤的指標大幅加速運算這些變數的不同值。

主題

- 在原始資料上
- 運算前唯一維度值
- 從衍生資料表計算變數

在原始資料上

您可以使用 SELECT DISTINCT 來計算從資料中看到的不同值。例如，如果您想要取得區域的不同值，您可以使用此表單的查詢。

```
SELECT DISTINCT region
FROM "raw_data"."devops"
WHERE time > ago(1h)
ORDER BY 1
```

您可能正在追蹤數百萬個裝置和數十億個時間序列。但是，在大多數情況下，這些有趣的變數適用於較低的基數維度，其中您有幾到十個值。DISTINCT 從原始資料運算可能需要掃描大量資料。

運算前唯一維度值

您希望這些變數快速載入，以便儀表板具有互動性。此外，這些變數通常在每個儀表板負載上計算，因此您希望它們也具有成本效益。您可以使用排程查詢來最佳化尋找這些變數，並在衍生的資料表中具體化它們。

首先，您需要識別計算值時，在述詞中使用的DISTINCT值或資料欄所需的維度DISTINCT。

在此範例中，您可以看到儀表板正在填入維度區域、儲存格、孤島、 availability_zone 和微服務的不同值。因此，您可以使用下面的查詢來預先計算這些唯一值。

```
SELECT region, cell, silo, availability_zone, microservice_name,
    min(@scheduled_runtime) AS time, COUNT(*) as numDataPoints
FROM raw_data.devops
WHERE time BETWEEN @scheduled_runtime - 15m AND @scheduled_runtime
GROUP BY region, cell, silo, availability_zone, microservice_name
```

這裡需要注意幾個重要事項。

- 您可以使用一個排程運算來預先計算許多不同查詢的值。例如，您正在使用上述查詢來預先計算五個不同變數的值。因此，您不需要每個變數各一個。您可以使用相同的模式來識別跨多個面板的共用運算，以最佳化您需要維護的排程查詢數目。
- 維度的唯一值不是固有的時間序列資料。因此，您可以使用 @scheduled_runtime 將此 轉換為時間序列。透過將此資料與 @scheduled_runtime 參數建立關聯，您也可以追蹤在指定時間點出現的唯一值，進而從中建立時間序列資料。
- 在上一個範例中，您會看到正在追蹤的指標值。此範例使用 COUNT (*) 。如果您想要追蹤儀表板的其他有意義的彙總，您可以計算這些彙總。

以下是使用上一個查詢的排程運算組態。在此範例中，其設定為使用排程運算式 cron (0/15 * * ? *) 。

```
{
    "Name": "PT15mHighCardPerUniqueDimensions",
    "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
 min(@scheduled_runtime) AS time, COUNT(*) as numDataPoints FROM raw_data.devops WHERE
 time BETWEEN @scheduled_runtime - 15m AND @scheduled_runtime GROUP BY region, cell,
 silo, availability_zone, microservice_name",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/15 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "hc_unique_dimensions_pt15m",
            "TimeColumn": "time",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "availability_zone",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                }
```

```
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "count_multi",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

從衍生資料表計算變數

一旦排程運算預先材料化衍生資料表 hc_unique_dimensions_pt15m 中的唯一值，您可以使用衍生資料表有效率地計算維度的唯一值。以下是如何計算唯一值，以及如何在這些唯一值查詢中使用其他變數作為述詞的範例查詢。

區域

```
SELECT DISTINCT region
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
ORDER BY 1
```

儲存格

```
SELECT DISTINCT cell
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
  AND region = '${region}'
ORDER BY 1
```

錫洛

```
SELECT DISTINCT silo
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
   AND region = '${region}' AND cell = '${cell}'
ORDER BY 1
```

Microservice

```
SELECT DISTINCT microservice_name
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
   AND region = '${region}' AND cell = '${cell}'
ORDER BY 1
```

可用區域

```
SELECT DISTINCT availability_zone
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
   AND region = '${region}' AND cell = '${cell}' AND silo = '${silo}'
ORDER BY 1
```

處理延遲抵達的資料

您可能會遇到資料可能明顯延遲到達的情況，例如，與擷取資料列相關聯的時間戳記相比，資料擷取到 Timestream 的時間 LiveAnalytics 會大幅延遲。在上述範例中，您已了解如何使用 @scheduled_runtime 參數定義的時間範圍來考慮某些延遲到達的資料。不過，如果您有使用案例，其中資料可能會延遲數小時或數天，您可能需要不同的模式，以確保衍生資料表中的預先運算已適當更新，以反映此類延遲到達的資料。如需延遲抵達資料的一般資訊，請參閱 寫入資料（插入 和 upserts）。

在以下，您將看到兩種不同的方法來解決此延遲到達資料。

- 如果您的資料到達有可預測的延遲，則可以使用另一個「擷取」排程運算來更新延遲到達資料的彙總。
- 如果您有無法預測的延遲或偶爾的延遲抵達資料，您可以使用手動執行來更新衍生的資料表。

此討論涵蓋延遲資料抵達的情況。不過，相同的原則適用於資料更正，其中您已修改來源資料表中的資料，並想要更新衍生資料表中的彙總。

主題

- 排程的追蹤查詢
- 無法預測的延遲到達資料的手動執行

排程的追蹤查詢

查詢彙總及時到達的資料

以下模式將說明如何在資料到達時有可預測的延遲時，使用自動方式更新彙總。請考慮下列排程運算的其中一個先前範例。此排程運算會每 30 分鐘重新整理衍生資料表一次，並已考量資料延遲長達一小時。

```
{
    "Name": "MultiPT30mPerHrPerTimeseriesDPCount",
    "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
 instance_type, os_version, instance_name, process_name, jdk_version, bin(time,
 1h) as hour, SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as
 numDataPoints FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime, 1h)
 - 1h AND @scheduled_runtime + 1h GROUP BY region, cell, silo, availability_zone,
 microservice_name, instance_type, os_version, instance_name, process_name,
 jdk_version, bin(time, 1h)",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/30 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "dp_per_timeseries_per_hr",
            "TimeColumn": "hour",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
```

```
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "availability_zone",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_type",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "os_version",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "process_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "jdk_version",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
```

```
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

更新延遲到達資料的彙總的擷取查詢

現在，如果您考慮資料延遲約 12 小時的情況。以下是相同查詢的變體。不過，差別在於，它計算與觸發排程運算時相比延遲長達 12 小時的資料彙總。例如，您會在以下範例中看到查詢，此查詢的目標時間範圍是在觸發查詢之前 2 小時到 14 小時之間。此外，如果您注意到排程表達式 cron （0 0，12 * ？ * ），則會在UTC每天 00：00 UTC和 12：00 觸發運算。因此，在 2021-12-01 00：00：00 觸發查詢時，查詢會在 2021-11-30 10：00：00 至 2021-11-30 22：00：00 期間更新彙總。排程查詢使用類似於 Timestream LiveAnalytics的 upsert 語義進行 的寫入，如果視窗中有延遲到達的資料，或如果找到較新的彙總 （例如，新的群組在此彙總中顯示，當原始排程運算觸發時不存在），則新的彙總將插入衍生的資料表中，則新的彙總值會以較新的值更新。同樣地，當下一個執行個體在 2021-12-01 12：00：00 觸發時，該執行個體會將 2021-11-30 22：00：00 至 2021-12-01 10：00：00 範圍內的彙總更新。

```
      {
   "Name": "MultiPT12HPerHrPerTimeseriesDPCountCatchUp",
   "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
instance_type, os_version, instance_name, process_name, jdk_version, bin(time, 1h)
as hour, SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as numDataPoints
FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 14h AND
bin(@scheduled_runtime, 1h) - 2h GROUP BY region, cell, silo, availability_zone,
microservice_name, instance_type, os_version, instance_name, process_name,
jdk_version, bin(time, 1h)",
   "ScheduleConfiguration": {
       "ScheduleExpression": "cron(0 0,12 * * ? *)"
   },
   "NotificationConfiguration": {
       "SnsConfiguration": {
```

```
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "dp_per_timeseries_per_hr",
            "TimeColumn": "hour",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "availability_zone",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_type",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "os_version",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "process_name",
                    "DimensionValueType": "VARCHAR"
```

```
                },
                {
                    "Name": "jdk_version",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

上述範例是假設延遲到達限制為 12 小時的圖示，且每 12 小時更新一次衍生資料表，即可用於晚於即時時段的資料。您可以調整此模式每小時更新一次衍生資料表，以便衍生資料表能更快反映延遲到達的資料。同樣地，您可以將時間範圍調整為超過 12 小時，例如一天或甚至一週或更長時間，以處理可預測的延遲抵達資料。

無法預測的延遲到達資料的手動執行

在某些情況下，您可能會出現無法預測的延遲到達資料，或者您對來源資料進行了變更，並在事實發生後更新了某些值。在所有這類情況下，您可以手動觸發排程查詢以更新衍生資料表。以下是如何達成此目標的範例。

假設您擁有使用案例，其中將運算寫入衍生的資料表 dp_per_timeseries_per_hr。您在資料表開發集中的基本資料已更新至時間範圍 2021-11-30 23：00：00 - 2021-12-01 00：00：00。有兩種不同的排程查詢可用於更新此衍生資料表：Multi PT30mPerHrPerTimeseriesDPCount 和 Multi PT12HPerHrPerTimeseriesDPCountCatchUp。您在 Timestream 中為 建立的每個排程運算

LiveAnalytics 都有唯一的 ARN，您可以在建立運算或執行清單操作時取得該 。您可以使用 ARN進行運算，以及查詢為執行此操作而使用參數 @scheduled_runtime 的值。

假設 Multi PT30mPerHrPerTimeseriesDPCount 的運算具有 ARN arn_1，且您想要使用此運算來更新衍生的資料表。由於上述排程運算會在 @scheduled_runtime 值之前 1 小時和之後 1 小時更新彙總，因此您可以使用 @scheduled_runtime 參數的值 2021-12-01 00：00：00 涵蓋更新 (2021-11-30 23：00：00 - 2021-12-01 00：00：00 的時間範圍。您可以使用 ExecuteScheduledQuery API傳遞此運算ARN的 ，以及以 epoch 秒為單位的時間參數值 （ 以 為單位UTC ），以達成此目標。以下是使用 的範例AWSCLI，您可以使用 Timestream SDKs支援的任何 遵循相同的模式 LiveAnalytics。

```
aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638316800 --profile profile --region us-east-1
```

在先前的範例中，設定檔是具有適當權限的AWS設定檔，可進行此API呼叫，而 1638316800 對應至 2021-12-01 00：00：00 的 epoch 秒。此手動觸發程序的行為幾乎與自動觸發程序相似，假設系統在所需時段觸發此調用。

如果您在更長的時間內更新 ，則表示基礎資料已於 2021-11-30 23：00：00 - 2021-12-01 11：00：00 更新，則您可以多次觸發上述查詢，以涵蓋整個時間範圍。例如，您可以執行六個不同的執行，如下所示。

```
aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638316800 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638324000 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638331200 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638338400 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638345600 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638352800 --profile profile --region us-east-1
```

先前的六個命令對應於在 2021-12-01 00：00：00、2021-12-01 02：00：00、2021-12-01 04：0：00、2021-12-01 06：00：00、2021-12-01 08：00：00 和 2021-12-01 10：00 叫用的排程運算：

或者，您可以在 2021-12-01 13：00：00 對一次執行使用觸發的運算多PT12HPerHrPerTimeseriesDPCountCatchUp工，以更新整個 12 小時時間範圍的彙總。例如，如果 arn_2 是該運算ARN的 ，您可以從 執行下列命令CLI。

```
aws timestream-query execute-scheduled-query --scheduled-query-arn arn_2 --invocation-
time 1638363600 --profile profile --region us-east-1
```

值得注意的是，對於手動觸發程序，您可以使用時間戳記來設定不需要與該自動觸發程序時間戳記相符的調用時間參數。例如，在上一個範例中，即使自動排程只在時間戳記 2021-12-01 10：00：00、2021-12-01 12021-12-012：00：00 和 2021-12-02 00：00：00 觸發，您仍會在 13：00：00 時觸發運算。的 Timestream LiveAnalytics 可讓您靈活地使用手動操作所需的適當值來觸發它。

以下是使用 時的一些重要考量 ExecuteScheduledQuery API。

- 如果您觸發多個這些調用，則需要確保這些調用不會產生重疊時間範圍的結果。例如，在先前的範例中，有六個叫用。每個調用都涵蓋 2 小時的時間範圍，因此調用時間戳記會分散兩小時，以避免更新中的任何重疊。這可確保衍生資料表中的資料最終處於相符的狀態，該狀態是來源資料表的彙總。如果您無法確保非重疊的時間範圍，則請確保依序觸發這些執行。如果您同時觸發多個執行，而這些執行在其時間範圍內重疊，則可以看到觸發失敗，您可能會在這些執行的錯誤報告中看到版本衝突。排程查詢調用所產生的結果會根據調用觸發時間指派版本。因此，較新的調用產生的資料列具有較高的版本。較高的版本記錄可以覆寫較低的版本記錄。對於自動觸發的排程查詢， 的 LiveAnalytics Timestream 會自動管理排程，因此即使後續調用具有重疊的時間範圍，也不會看到這些問題。
- 如前所述，您可以為 @scheduled_runtime 觸發具有任何時間戳記值的叫用。因此，您有責任適當地設定值，以便在衍生資料表中更新適當的時間範圍，對應來源資料表中更新資料的範圍。
- 您也可以將這些手動觸發程序用於處於 DISABLED 狀態的排程查詢。這可讓您定義未在自動排程中執行的特殊查詢，因為它們處於 DISABLED 狀態。相反地，您可以使用手動觸發程序來管理資料更正或延遲到達使用案例。

回填歷史預先運算

當您建立排程運算時， 的 Timestream 會 LiveAnalytics 管理後續查詢的執行，其中重新整理受您提供的排程表達式所管理。視來源資料表的歷史資料數量而定，您可能想要使用與歷史資料對應的彙總來更新衍生資料表。您可以使用上述邏輯進行手動觸發，以回填歷史彙總。

例如，如果我們考慮衍生的資料表 per_timeseries_lastpoint_pt1d，則排程運算會在過去一天更新一次。如果您的來源資料表具有一年的資料，您可以使用 ARN 進行此排程運算，並每天手動觸發，直到一年，以便衍生的資料表已填入所有歷史查詢。請注意，此處適用手動觸發程序的所有注意事項。此

外，如果衍生資料表的設定方式是歷史擷取將寫入衍生資料表上的磁性存放區，請注意寫入磁性存放區時的最佳實務和限制。

排程查詢範例

本節包含範例，說明如何使用 Timestream for LiveAnalytics的排程查詢，在視覺化機群整體統計資料時最佳化成本和儀表板載入時間，以有效地監控您的裝置機群。Timestream 中的排程查詢 LiveAnalytics 可讓您使用 Timestream for 的完整SQL表面積來表達查詢 LiveAnalytics。您的查詢可以包含一或多個來源資料表、針對 LiveAnalyticsSQL的語言執行彙總或 Timestream 允許的任何其他查詢，然後將查詢結果存放在 Timestream 中的另一個目的地資料表中 LiveAnalytics。

本節將排程查詢的目標資料表稱為衍生資料表 。

例如，我們將使用一個 DevOps 應用程式，用於監控部署在多個部署 （ 例如區域、儲存格和孤島 ）、多個微服務的大型伺服器機群，而您正在使用 Timestream for 追蹤機群範圍的統計資料 LiveAnalytics。我們將使用的範例結構描述描述於排程查詢範例結構描述 。

將說明下列案例。

- 如何轉換儀表板、將擷取到 Timestream 的原始資料的彙總統計資料繪製 LiveAnalytics 為排程查詢，然後如何使用預先計算的彙總來建立新的儀表板，顯示彙總統計資料。
- 如何合併排程查詢以取得彙總檢視和原始精細資料，以深入研究詳細資訊。這可讓您儲存和分析原始資料，同時使用排程查詢來最佳化常見的機群範圍操作。
- 如何使用排程查詢，找出在多個儀表板中使用的彙總，並讓相同的排程查詢填入相同或多個儀表板中的多個面板，以最佳化成本。

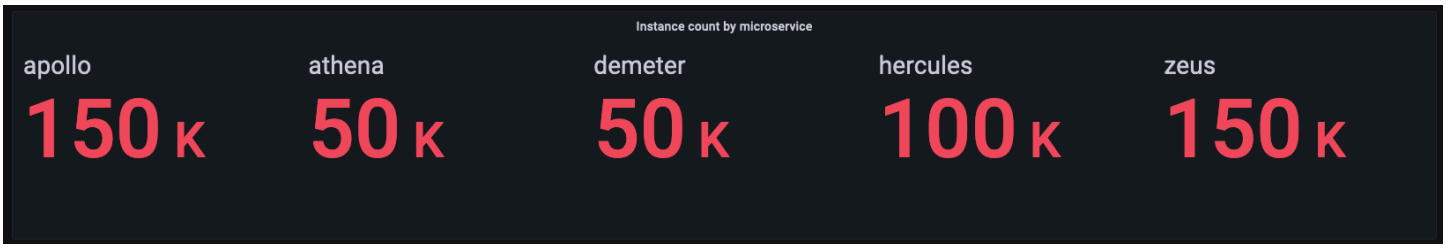
主題

- 將彙總儀表板轉換為排程查詢
- 使用排程查詢和原始資料進行深入分析
- 透過跨儀表板共用排程查詢來最佳化成本
- 比較基礎資料表上的查詢與排程查詢結果的查詢

將彙總儀表板轉換為排程查詢

假設您正在計算機群範圍的統計資料，例如透過五個微服務，以及部署服務的六個區域，在機群中計算主機計數。從下面的快照中，您可以看到有 500K部伺服器發出指標，而一些較大的區域 （ 例如 us-east-1 ） 具有 >20 萬部伺服器。

計算這些彙總，在其中，您正在計算數百 GB 資料的不同執行個體名稱，除了掃描資料的成本之外，還可能導致查詢延遲數十秒。



原始儀表板查詢

dasboard 面板中顯示的彙總會使用下列查詢，從原始資料計算。查詢使用多個SQL建構，例如不同的計數和多個彙總函數。

```
SELECT CASE WHEN microservice_name = 'apollo' THEN num_instances ELSE NULL END AS
 apollo,
    CASE WHEN microservice_name = 'athena' THEN num_instances ELSE NULL END AS athena,
    CASE WHEN microservice_name = 'demeter' THEN num_instances ELSE NULL END AS
 demeter,
    CASE WHEN microservice_name = 'hercules' THEN num_instances ELSE NULL END AS
 hercules,
    CASE WHEN microservice_name = 'zeus' THEN num_instances ELSE NULL END AS zeus
FROM (
    SELECT microservice_name, SUM(num_instances) AS num_instances
    FROM (
        SELECT microservice_name, COUNT(DISTINCT instance_name) as num_instances
        FROM "raw_data"."devops"
        WHERE time BETWEEN from_milliseconds(1636526171043) AND
 from_milliseconds(1636612571043)
            AND measure_name = 'metrics'
        GROUP BY region, cell, silo, availability_zone, microservice_name
    )
    GROUP BY microservice_name
)
```

轉換為排程查詢

先前的查詢可以轉換為排程查詢，如下所示。您會先在區域、儲存格、孤島、可用區域和微服務的特定部署中計算不同的主機名稱。然後，您可以新增主機來計算每個微服務主機計數每小時的 。透過使用排程查詢支援的`@scheduled_runtime`參數，您可以在叫用查詢時重新計算過去一小時的參數。內部查詢子`WHERE`句`bin(@scheduled_runtime, 1h)`中的 可確保即使查詢排程在一小時中間的時間，您仍然可以取得整小時的資料。

即使查詢會每小時計算彙總，就像您在排程的運算組態中看到的一樣，它設定為每半小時重新整理一次，以便您更快在衍生資料表中取得更新。您可以根據您的新鮮度需求進行調整，例如每 15 分鐘重新計算彙總，或在小時界限重新計算彙總。

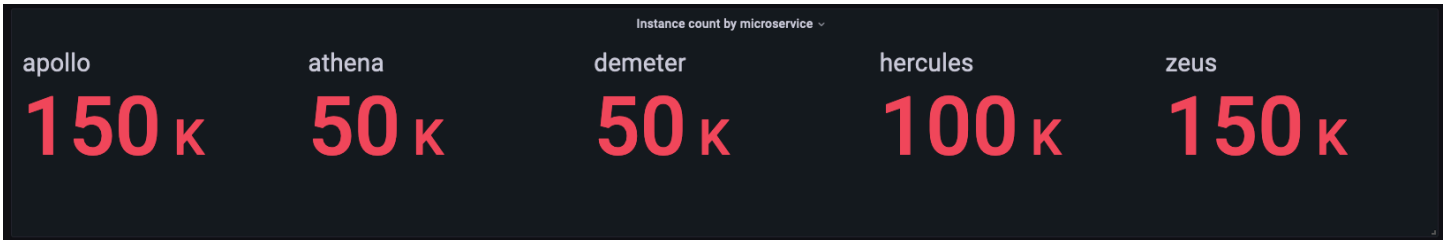
```
SELECT microservice_name, hour, SUM(num_instances) AS num_instances
FROM (
        SELECT microservice_name, bin(time, 1h) AS hour,
            COUNT(DISTINCT instance_name) as num_instances
       FROM raw_data.devops
       WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND @scheduled_runtime

           AND measure_name = 'metrics'
       GROUP BY region, cell, silo, availability_zone, microservice_name, bin(time, 1h)

     )
GROUP BY microservice_name, hour
```

```
{
    "Name": "MultiPT30mHostCountMicroservicePerHr",
    "QueryString": "SELECT microservice_name, hour, SUM(num_instances) AS num_instances
    FROM (        SELECT microservice_name, bin(time, 1h) AS hour, COUNT(DISTINCT
 instance_name) as num_instances        FROM raw_data.devops        WHERE time BETWEEN
 bin(@scheduled_runtime, 1h) - 1h AND @scheduled_runtime            AND measure_name
 = 'metrics'        GROUP BY region, cell, silo, availability_zone, microservice_name,
 bin(time, 1h)    )    GROUP BY microservice_name, hour",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/30 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "host_count_pt1h",
            "TimeColumn": "hour",
            "DimensionMappings": [
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                }
```

```
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "num_instances",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "num_instances",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

使用預先計算的結果產生新的儀表板

您現在將了解如何使用您建立之排程查詢的衍生資料表來建立彙總檢視儀表板。從儀表板快照中，您也可以驗證從衍生資料表和基礎資料表計算的彙總是否也相符。使用衍生資料表建立儀表板後，您會注意到相較於從原始資料計算這些彙總，使用衍生資料表的載入時間會明顯更快，而且成本更低。以下是使用預先計算資料的儀表板快照，以及使用儲存在「derived」資料表中的預先計算資料來轉譯此面板的查詢。」host_count_pt1h」。請注意，查詢的結構與原始資料儀表板中使用的查詢非常相似，但它使用已計算此查詢正在彙總之不同計數的衍生資料表。




```
SELECT CASE WHEN microservice_name = 'apollo' THEN num_instances ELSE NULL END AS
 apollo,
    CASE WHEN microservice_name = 'athena' THEN num_instances ELSE NULL END AS athena,
    CASE WHEN microservice_name = 'demeter' THEN num_instances ELSE NULL END AS
 demeter,
```

```
    CASE WHEN microservice_name = 'hercules' THEN num_instances ELSE NULL END AS
 hercules,
    CASE WHEN microservice_name = 'zeus' THEN num_instances ELSE NULL END AS zeus
FROM (
    SELECT microservice_name, AVG(num_instances) AS num_instances
    FROM (
        SELECT microservice_name, bin(time, 1h), SUM(num_instances) as num_instances
        FROM "derived"."host_count_pt1h"
        WHERE time BETWEEN from_milliseconds(1636567785421) AND
 from_milliseconds(1636654185421)
            AND measure_name = 'num_instances'
        GROUP BY microservice_name, bin(time, 1h)
    )
    GROUP BY microservice_name
)
```

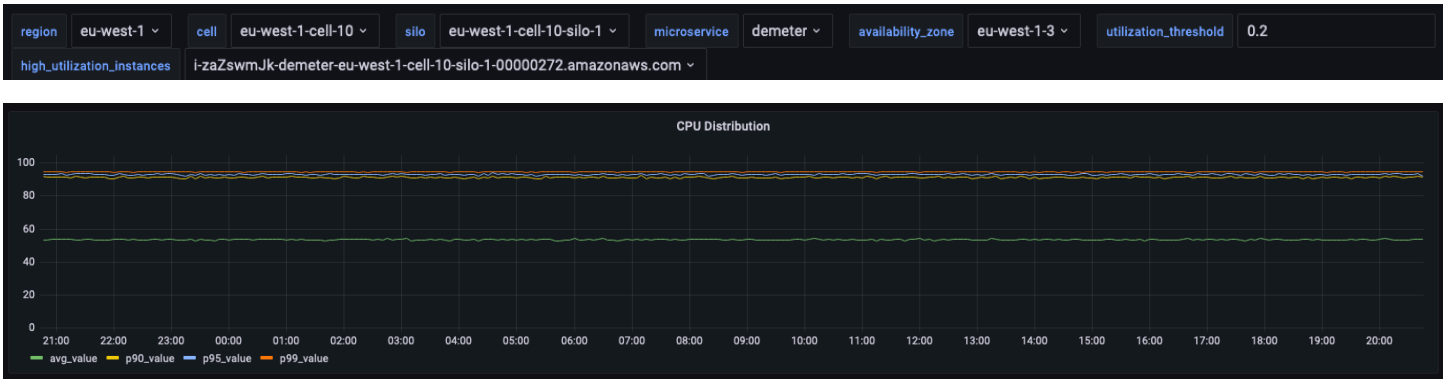
使用排程查詢和原始資料進行深入分析

您可以使用整個機群的彙總統計資料來識別需要深入分析的區域，然後使用原始資料深入分析精細資料，以取得更深入的洞見。

在此範例中，您將了解如何使用彙總儀表板來識別任何似乎比其他部署具有更高CPU使用率的部署（部署適用於指定區域、儲存格、孤島和可用區域中的特定微服務）。然後，您可以向下切入，以更了解使用原始資料。由於這些向下切入可能很少發生，並且只存取與部署相關的資料，因此您可以使用原始資料進行此分析，而且不需要使用排程查詢。

每次部署向下切入

下面的儀表板提供特定部署中更精細和伺服器層級的統計資料。為了協助您深入了解機群的不同部分，此儀表板會使用區域、儲存格、孤島、微服務和 availability_zone 等變數。然後，它會顯示該部署的一些彙總統計資料。



在下面的查詢中，您可以看到在變數下拉式清單中選擇的值用作查詢WHERE子句中的述詞，這可讓您僅專注於部署的資料。然後，面板會繪製該部署中執行個體的彙總CPU指標。您可以使用原始資料，以互動式查詢延遲執行此深入分析，以衍生更深入的洞見。

```
SELECT bin(time, 5m) as minute,
    ROUND(AVG(cpu_user), 2) AS avg_value,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.9), 2) AS p90_value,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.95), 2) AS p95_value,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.99), 2) AS p99_value
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099476) AND
 from_milliseconds(1636613499476)
    AND region = 'eu-west-1'
    AND cell = 'eu-west-1-cell-10'
    AND silo = 'eu-west-1-cell-10-silo-1'
    AND microservice_name = 'demeter'
    AND availability_zone = 'eu-west-1-3'
    AND measure_name = 'metrics'
GROUP BY bin(time, 5m)
ORDER BY 1
```

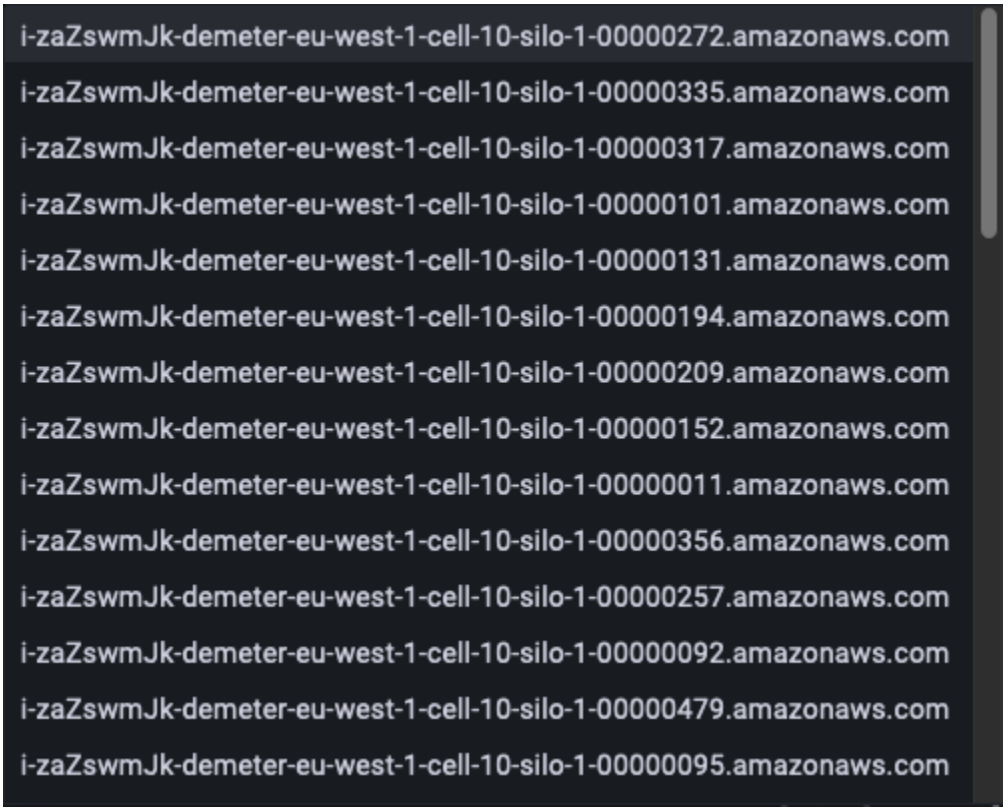
執行個體層級統計資料

此儀表板會進一步計算另一個變數，該變數也會列出高CPU使用率的伺服器/執行個體，並依使用率的遞減順序排序。用於計算此變數的查詢顯示如下。

```
WITH microservice_cell_avg AS (
    SELECT AVG(cpu_user) AS microservice_avg_metric
    FROM "raw_data"."devops"
    WHERE $__timeFilter
        AND measure_name = 'metrics'
        AND region = '${region}'
        AND cell = '${cell}'
        AND silo = '${silo}'
        AND availability_zone = '${availability_zone}'
        AND microservice_name = '${microservice}'
), instance_avg AS (
    SELECT instance_name,
        AVG(cpu_user) AS instance_avg_metric
    FROM "raw_data"."devops"
    WHERE $__timeFilter
        AND measure_name = 'metrics'
```

```
        AND region = '${region}'
        AND cell = '${cell}'
        AND silo = '${silo}'
        AND microservice_name = '${microservice}'
        AND availability_zone = '${availability_zone}'
    GROUP BY availability_zone, instance_name
)
SELECT i.instance_name
FROM instance_avg i CROSS JOIN microservice_cell_avg m
WHERE i.instance_avg_metric > (1 + ${utilization_threshold}) *
 m.microservice_avg_metric
ORDER BY i.instance_avg_metric DESC
```

在上述查詢中，會根據為其他變數選擇的值動態重新計算變數。為部署填入變數後，您可以從清單中選擇個別執行個體，以進一步視覺化該執行個體的指標。您可以從執行個體名稱的下拉式清單中選擇不同的執行個體，如下列快照所示。



先前面板會顯示所選執行個體的統計資料，下方是用來擷取這些統計資料的查詢。

```
SELECT BIN(time, 30m) AS time_bin,
    AVG(cpu_user) AS avg_cpu,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.99), 2) as p99_cpu
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099477) AND
 from_milliseconds(1636613499477)
    AND measure_name = 'metrics'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
GROUP BY BIN(time, 30m)
ORDER BY time_bin desc
```

```
SELECT BIN(time, 30m) AS time_bin,
    AVG(memory_used) AS avg_memory,
    ROUND(APPROX_PERCENTILE(memory_used, 0.99), 2) as p99_memory
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099477) AND
 from_milliseconds(1636613499477)
    AND measure_name = 'metrics'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
```

```
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
GROUP BY BIN(time, 30m)
ORDER BY time_bin desc
```

```
SELECT COUNT(gc_pause)
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099477) AND
 from_milliseconds(1636613499478)
    AND measure_name = 'events'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
```

```
SELECT avg(gc_pause) as avg, round(approx_percentile(gc_pause, 0.99), 2) as p99
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099478) AND
 from_milliseconds(1636613499478)
    AND measure_name = 'events'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
```

```
SELECT BIN(time, 30m) AS time_bin,
    AVG(disk_io_reads) AS avg,
    ROUND(APPROX_PERCENTILE(disk_io_reads, 0.99), 2) as p99
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099478) AND
 from_milliseconds(1636613499478)
    AND measure_name = 'metrics'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
GROUP BY BIN(time, 30m)
ORDER BY time_bin desc
```

透過跨儀表板共用排程查詢來最佳化成本

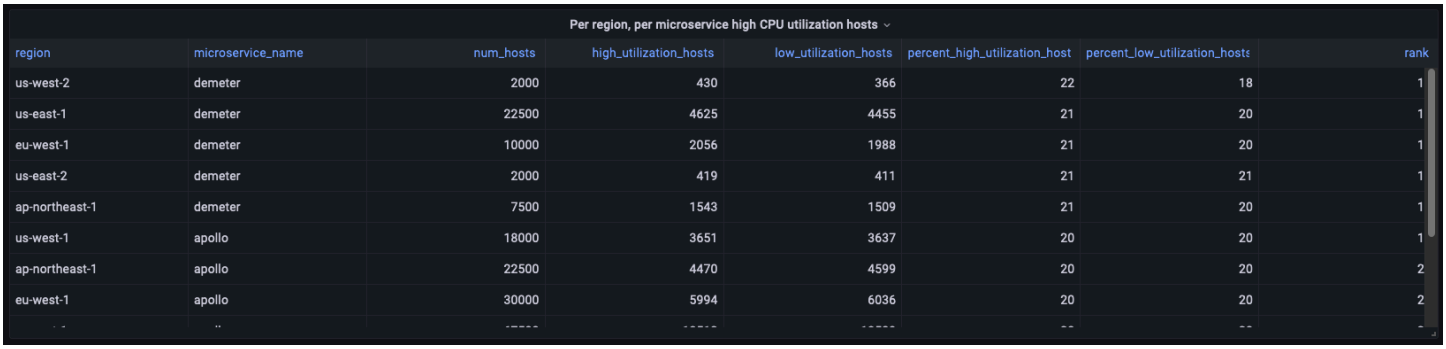
在此範例中，我們將看到一個案例，其中多個儀表板面板顯示類似資訊的變化（尋找高CPU使用率的CPU大量主機和機群的部分），以及如何使用相同的排程查詢來預先計算結果，然後用於填入多個面板。重複使用會進一步最佳化您的成本，其中 不使用不同的排程查詢，每個面板一個查詢只會使用擁有者。

具有原始資料的儀表板面板

CPU 每個微服務每個區域的使用率

第一個面板會運算平均CPU使用率低於或高於上述CPU使用率的執行個體，用於區域、儲存格、孤島、可用區域和微服務內的指定部署。然後，它會排序具有高使用率主機最高百分比的區域和微服務。它有助於識別特定部署的伺服器正在執行的熱度，然後向下切入以更好地了解問題。

面板的查詢示範了 Timestream 的靈活性， LiveAnalyticsSQL支援 使用常見的資料表表達式、視窗函數、聯結等來執行複雜的分析任務。



region	microservice_name	num_hosts	high_utilization_hosts	low_utilization_hosts	percent_high_utilization_host	percent_low_utilization_hosts	rank
us-west-2	demeter	2000	430	366	22	18	1
us-east-1	demeter	22500	4625	4455	21	20	1
eu-west-1	demeter	10000	2056	1988	21	20	1
us-east-2	demeter	2000	419	411	21	21	1
ap-northeast-1	demeter	7500	1543	1509	21	20	1
us-west-1	apollo	18000	3651	3637	20	20	1
ap-northeast-1	apollo	22500	4470	4599	20	20	2
eu-west-1	apollo	30000	5994	6036	20	20	2

查詢：

```
WITH microservice_cell_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, AVG(cpu_user) AS
 microservice_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526593876) AND
 from_milliseconds(1636612993876)
        AND measure_name = 'metrics'
    GROUP BY region, cell, silo, availability_zone, microservice_name
), instance_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, instance_name,
        AVG(cpu_user) AS instance_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526593876) AND
 from_milliseconds(1636612993876)
```

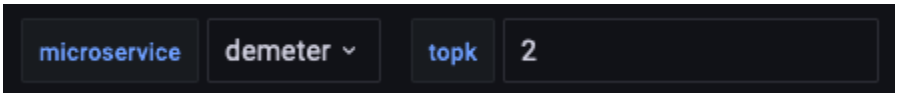
```
        AND measure_name = 'metrics'
    GROUP BY region, cell, silo, availability_zone, microservice_name, instance_name
), instances_above_threshold AS (
  SELECT i.*,
    CASE WHEN i.instance_avg_metric > (1 + 0.2) * m.microservice_avg_metric THEN 1 ELSE
 0 END AS high_utilization,
    CASE WHEN i.instance_avg_metric < (1 - 0.2) * m.microservice_avg_metric THEN 1 ELSE
 0 END AS low_utilization
  FROM instance_avg i INNER JOIN microservice_cell_avg m
    ON i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND
 i.availability_zone = m.availability_zone
      AND m.microservice_name = i.microservice_name
), per_deployment_high AS (
SELECT region, microservice_name, COUNT(*) AS num_hosts, SUM(high_utilization) AS
 high_utilization_hosts, SUM(low_utilization) AS low_utilization_hosts,
    ROUND(SUM(high_utilization) * 100.0 / COUNT(*), 0) AS
 percent_high_utilization_hosts,
    ROUND(SUM(low_utilization) * 100.0 / COUNT(*), 0) AS percent_low_utilization_hosts
FROM instances_above_threshold
GROUP BY region, microservice_name
), per_region_ranked AS (
    SELECT *,
        DENSE_RANK() OVER (PARTITION BY region ORDER BY percent_high_utilization_hosts
 DESC, high_utilization_hosts DESC) AS rank
    FROM per_deployment_high
)
SELECT *
FROM per_region_ranked
WHERE rank <= 2
ORDER BY percent_high_utilization_hosts desc, rank asc
```

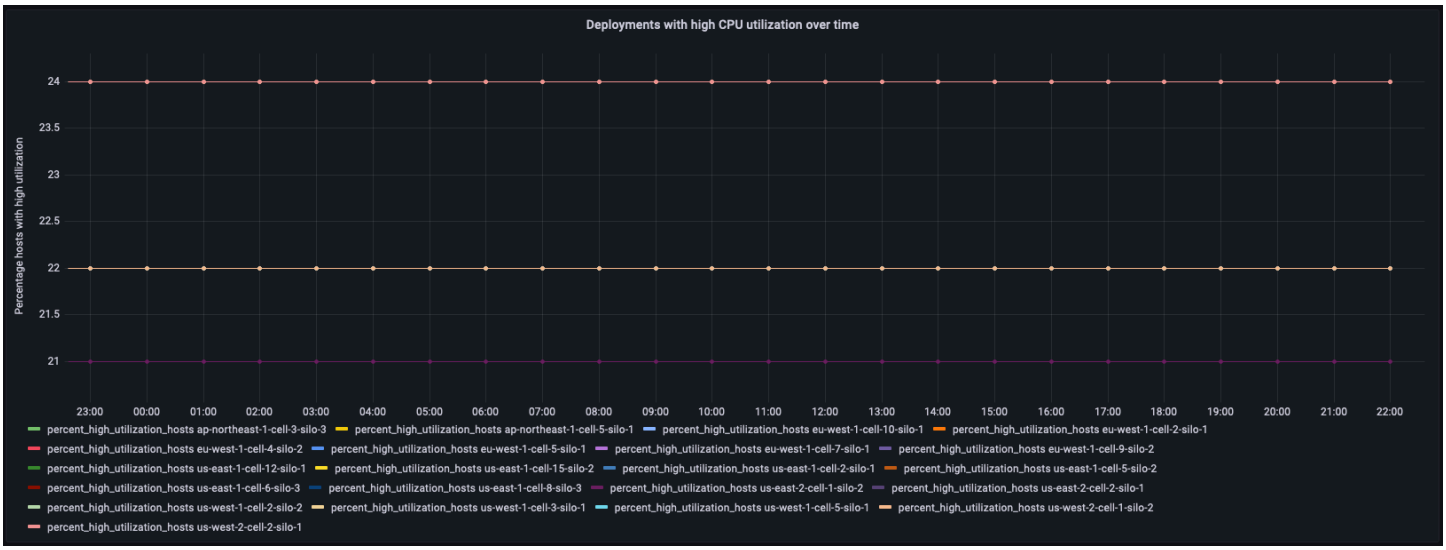
深入探索微服務以尋找熱點

下一個儀表板可讓您深入探索其中一個微服務，以找出該微服務的特定區域、儲存格和孤島，正在以更高的CPU使用率執行其機群的部分。例如，在機群範圍儀表板中，您看到微服務指標顯示在前幾個排名位置，因此在此儀表板中，您想要深入探索該微服務。

此儀表板使用變數來挑選微服務來深入探索，並使用維度的唯一值填入變數的值。選取微服務後，儀表板的其餘部分會重新整理。

如下方所示，第一個面板會繪製一段時間內部署中的主機百分比（微服務的區域、儲存格和孤島），以及用於繪製儀表板的對應查詢。此圖表本身會識別具有較高 之主機百分比的特定部署CPU。





查詢：

```
WITH microservice_cell_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, bin(time, 1h) as
 hour, AVG(cpu_user) AS microservice_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526898831) AND
 from_milliseconds(1636613298831)
        AND measure_name = 'metrics'
        AND microservice_name = 'demeter'
    GROUP BY region, cell, silo, availability_zone, microservice_name, bin(time, 1h)
), instance_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h) as hour,
        AVG(cpu_user) AS instance_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526898831) AND
 from_milliseconds(1636613298831)
        AND measure_name = 'metrics'
        AND microservice_name = 'demeter'
    GROUP BY region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h)
), instances_above_threshold AS (
  SELECT i.*,
    CASE WHEN i.instance_avg_metric > (1 + 0.2) * m.microservice_avg_metric THEN 1 ELSE
 0 END AS high_utilization
```

```
  FROM instance_avg i INNER JOIN microservice_cell_avg m
    ON i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND
 i.availability_zone = m.availability_zone
      AND m.microservice_name = i.microservice_name AND m.hour = i.hour
), high_utilization_percent AS (
    SELECT region, cell, silo, microservice_name, hour, COUNT(*) AS num_hosts,
 SUM(high_utilization) AS high_utilization_hosts,
        ROUND(SUM(high_utilization) * 100.0 / COUNT(*), 0) AS
 percent_high_utilization_hosts
    FROM instances_above_threshold
    GROUP BY region, cell, silo, microservice_name, hour
), high_utilization_ranked AS (
    SELECT region, cell, silo, microservice_name,
        DENSE_RANK() OVER (PARTITION BY region ORDER BY
 AVG(percent_high_utilization_hosts) desc, AVG(high_utilization_hosts) desc) AS rank
    FROM high_utilization_percent
    GROUP BY region, cell, silo, microservice_name
)
SELECT hup.silo, CREATE_TIME_SERIES(hour, hup.percent_high_utilization_hosts) AS
 percent_high_utilization_hosts
FROM high_utilization_percent hup INNER JOIN high_utilization_ranked hur
    ON hup.region = hur.region AND hup.cell = hur.cell AND hup.silo = hur.silo AND
 hup.microservice_name = hur.microservice_name
WHERE rank <= 2
GROUP BY hup.region, hup.cell, hup.silo
ORDER BY hup.silo
```

轉換為單一排程查詢以重複使用

請務必注意，在兩個儀表板的不同面板中，也會進行類似的運算。您可以為每個面板定義個別的排程查詢。在這裡，您將了解如何定義一個排程查詢，以進一步最佳化成本，這些查詢可用來呈現所有三個面板的結果。

下列查詢會擷取計算並用於所有不同面板的彙總。您會在此排程查詢的定義中觀察幾個重要層面。

- 排程查詢支援的SQL表面積的彈性和功能，您可以在其中使用常見的資料表表達式、聯結、案例陳述式等。
- 您可以使用一個排程查詢，以比特定儀表板更精細的精細程度計算統計資料，以及儀表板可能用於不同變數的所有值。例如，您會看到彙總會跨區域、儲存格、孤島和微服務進行計算。因此，您可以結合這些項目來建立區域層級或區域，以及微服務層級彙總。同樣地，相同的查詢會運算所有區域、儲存格、孤島和微服務的彙總。它可讓您在這些資料欄上套用篩選條件，以取得值子集的彙總。例如，

您可以計算任何一個區域的彙總，例如 us-east-1，或任何一個微服務，例如 demeter 或深入鑽研區域、儲存格、孤島和微服務中的特定部署。此方法會進一步最佳化維護預先計算彙總的成本。

```
WITH microservice_cell_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, bin(time, 1h) as
 hour, AVG(cpu_user) AS microservice_avg_metric
    FROM raw_data.devops
    WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime, 1h)
 + 1h
        AND measure_name = 'metrics'
    GROUP BY region, cell, silo, availability_zone, microservice_name, bin(time, 1h)

), instance_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h) as hour,
        AVG(cpu_user) AS instance_avg_metric
   FROM raw_data.devops
   WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime, 1h)
 + 1h
       AND measure_name = 'metrics'
   GROUP BY region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h)
), instances_above_threshold AS (
    SELECT i.*,
        CASE WHEN i.instance_avg_metric > (1 + 0.2) * m.microservice_avg_metric THEN 1
 ELSE 0 END AS high_utilization,
        CASE WHEN i.instance_avg_metric < (1 - 0.2) * m.microservice_avg_metric THEN 1
 ELSE 0 END AS low_utilization
   FROM instance_avg i INNER JOIN microservice_cell_avg m
       ON i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND
 i.availability_zone = m.availability_zone
           AND m.microservice_name = i.microservice_name AND m.hour = i.hour
)
SELECT region, cell, silo, microservice_name, hour,
     COUNT(*) AS num_hosts, SUM(high_utilization) AS high_utilization_hosts,
 SUM(low_utilization) AS low_utilization_hosts
FROM instances_above_threshold GROUP BY region, cell, silo, microservice_name, hour
```

以下是先前查詢的排程查詢定義。排程表達式設定為每 30 分鐘重新整理一次，並使用 bin (@scheduled_runtime， 1h) 建構再次重新整理資料最長一小時，以取得完整的小時事件。視應用程式的新鮮度需求而定，您可以將其設定為更頻繁或更頻繁地重新整理。透過使用WHERE時間

BETWEEN bin (@scheduled_runtime， 1h ） - 1h AND bin (@scheduled_runtime， 1h ） + 1h，我們可以確保即使您每 15 分鐘重新整理一次，仍會取得目前小時和前一小時的完整小時資料。

稍後，您將看到三個面板如何使用寫入 table deployment_cpu_stats_per_hr 的這些彙總來視覺化與該面板相關的指標。

```
{
    "Name": "MultiPT30mHighCpuDeploymentsPerHr",
    "QueryString": "WITH microservice_cell_avg AS (    SELECT region, cell,
 silo, availability_zone, microservice_name, bin(time, 1h) as hour, AVG(cpu_user)
 AS microservice_avg_metric    FROM raw_data.devops    WHERE time BETWEEN
 bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime, 1h) + 1h    AND
 measure_name = 'metrics'    GROUP BY region, cell, silo, availability_zone,
 microservice_name, bin(time, 1h)    ), instance_avg AS (    SELECT region,
 cell, silo, availability_zone, microservice_name, instance_name, bin(time, 1h)
 as hour,    AVG(cpu_user) AS instance_avg_metric    FROM raw_data.devops
 WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime,
 1h) + 1h    AND measure_name = 'metrics'    GROUP BY region, cell, silo,
 availability_zone, microservice_name, instance_name, bin(time, 1h)    ),
 instances_above_threshold AS (    SELECT i.*,    CASE WHEN i.instance_avg_metric >
 (1 + 0.2) * m.microservice_avg_metric THEN 1 ELSE 0 END AS high_utilization,    CASE
 WHEN i.instance_avg_metric < (1 - 0.2) * m.microservice_avg_metric THEN 1 ELSE 0 END
 AS low_utilization    FROM instance_avg i INNER JOIN microservice_cell_avg m    ON
 i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND i.availability_zone
 = m.availability_zone    AND m.microservice_name = i.microservice_name AND m.hour =
 i.hour    )    SELECT region, cell, silo, microservice_name, hour,         COUNT(*)
 AS num_hosts, SUM(high_utilization) AS high_utilization_hosts, SUM(low_utilization) AS
 low_utilization_hosts    FROM instances_above_threshold GROUP BY region, cell, silo,
 microservice_name, hour",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/30 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "deployment_cpu_stats_per_hr",
            "TimeColumn": "hour",
            "DimensionMappings": [
```

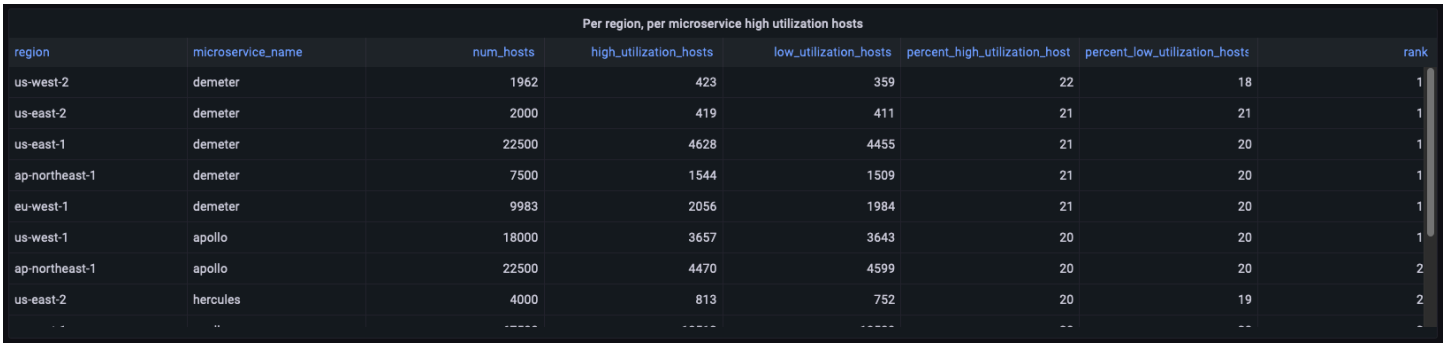
```
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "cpu_user",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "num_hosts",
                        "MeasureValueType": "BIGINT"
                    },
                    {
                        "SourceColumn": "high_utilization_hosts",
                        "MeasureValueType": "BIGINT"
                    },
                    {
                        "SourceColumn": "low_utilization_hosts",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
```

```
}
```

預先計算結果的儀表板

高CPU使用率主機

對於高使用率主機，您會看到不同的面板如何使用 deployment_cpu_stats_per_hr 中的資料來計算面板所需的不同彙總。例如，此面板提供區域層級資訊，因此會報告依區域和微服務分組的彙總，而不會篩選任何區域或微服務。



Per region, per microservice high utilization hosts

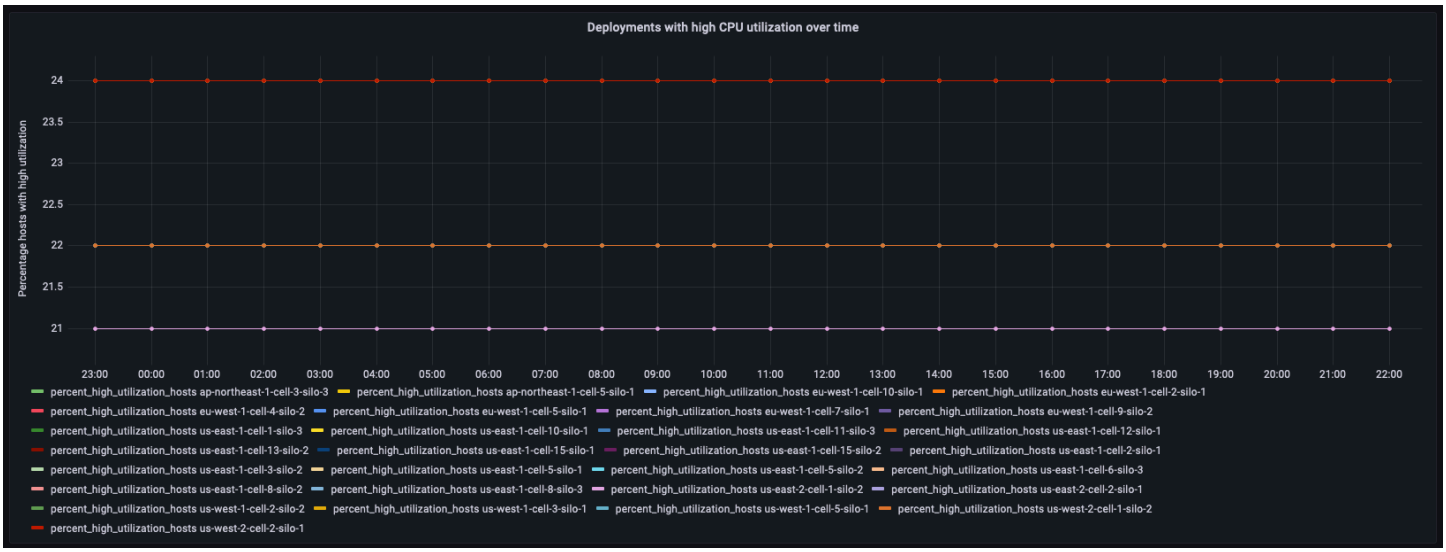
region	microservice_name	num_hosts	high_utilization_hosts	low_utilization_hosts	percent_high_utilization_host	percent_low_utilization_hosts	rank
us-west-2	demeter	1962	423	359	22	18	1
us-east-2	demeter	2000	419	411	21	21	1
us-east-1	demeter	22500	4628	4455	21	20	1
ap-northeast-1	demeter	7500	1544	1509	21	20	1
eu-west-1	demeter	9983	2056	1984	21	20	1
us-west-1	apollo	18000	3657	3643	20	20	1
ap-northeast-1	apollo	22500	4470	4599	20	20	2
us-east-2	hercules	4000	813	752	20	19	2

```
WITH per_deployment_hosts AS (
    SELECT region, cell, silo, microservice_name,
        AVG(num_hosts) AS num_hosts,
        AVG(high_utilization_hosts) AS high_utilization_hosts,
        AVG(low_utilization_hosts) AS low_utilization_hosts
    FROM "derived"."deployment_cpu_stats_per_hr"
    WHERE time BETWEEN from_milliseconds(1636567785437) AND
 from_milliseconds(1636654185437)
        AND measure_name = 'cpu_user'
    GROUP BY region, cell, silo, microservice_name
), per_deployment_high AS (
    SELECT region, microservice_name,
        SUM(num_hosts) AS num_hosts,
        ROUND(SUM(high_utilization_hosts), 0) AS high_utilization_hosts,
        ROUND(SUM(low_utilization_hosts),0) AS low_utilization_hosts,
        ROUND(SUM(high_utilization_hosts) * 100.0 / SUM(num_hosts)) AS
 percent_high_utilization_hosts,
        ROUND(SUM(low_utilization_hosts) * 100.0 / SUM(num_hosts)) AS
 percent_low_utilization_hosts
    FROM per_deployment_hosts
    GROUP BY region, microservice_name
),
per_region_ranked AS (
    SELECT *,
```

```
        DENSE_RANK() OVER (PARTITION BY region ORDER BY percent_high_utilization_hosts
 DESC, high_utilization_hosts DESC) AS rank
    FROM per_deployment_high
)
SELECT *
FROM per_region_ranked
WHERE rank <= 2
ORDER BY percent_high_utilization_hosts desc, rank asc
```

深入探索微服務以尋找高CPU用量部署

下一個範例會再次使用 deployment_cpu_stats_per_hr 衍生的資料表，但現在會套用特定微服務的篩選條件（此範例中的demeter，因為其在彙總儀表板中報告了高使用率主機）。此面板會追蹤一段時間內高CPU使用率主機的百分比。




```
WITH high_utilization_percent AS (
    SELECT region, cell, silo, microservice_name, bin(time, 1h) AS hour, MAX(num_hosts)
 AS num_hosts,
        MAX(high_utilization_hosts) AS high_utilization_hosts,
        ROUND(MAX(high_utilization_hosts) * 100.0 / MAX(num_hosts)) AS
 percent_high_utilization_hosts
    FROM "derived"."deployment_cpu_stats_per_hr"
    WHERE time BETWEEN from_milliseconds(1636525800000) AND
 from_milliseconds(1636612200000)
        AND measure_name = 'cpu_user'
        AND microservice_name = 'demeter'
    GROUP BY region, cell, silo, microservice_name, bin(time, 1h)
), high_utilization_ranked AS (
    SELECT region, cell, silo, microservice_name,
```

```
        DENSE_RANK() OVER (PARTITION BY region ORDER BY
 AVG(percent_high_utilization_hosts) desc, AVG(high_utilization_hosts) desc) AS rank
    FROM high_utilization_percent
    GROUP BY region, cell, silo, microservice_name
)
SELECT hup.silo, CREATE_TIME_SERIES(hour, hup.percent_high_utilization_hosts) AS
 percent_high_utilization_hosts
FROM high_utilization_percent hup INNER JOIN high_utilization_ranked hur
    ON hup.region = hur.region AND hup.cell = hur.cell AND hup.silo = hur.silo AND
 hup.microservice_name = hur.microservice_name
WHERE rank <= 2
GROUP BY hup.region, hup.cell, hup.silo
ORDER BY hup.silo
```

比較基礎資料表上的查詢與排程查詢結果的查詢

在此 Timestream 查詢範例中，我們使用下列結構描述、範例查詢和輸出，將基礎資料表上的查詢與排程查詢結果衍生資料表上的查詢進行比較。透過規劃良好的排程查詢，您可以取得具有較少資料列和其他特性的衍生資料表，這些特性可能會導致查詢比原始基礎資料表快。

如需描述此案例的影片，請參閱在 Amazon Timestream 中使用排程查詢來改善查詢效能並降低成本 LiveAnalytics。

在此範例中，我們使用下列案例：

- 區域 – us-east-1
- 基礎資料表 – `"clickstream"."shopping"`
- 衍生資料表 – `"clickstream"."aggregate"`

基本資料表

以下說明基礎資料表的結構描述。

資料行	Type	LiveAnalytics 屬性類型的時間串流
通道	varchar	MULTI
description	varchar	MULTI

資料行	Type	LiveAnalytics 屬性類型的時間串流
事件	varchar	DIMENSION
ip_address	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
產品	varchar	MULTI
product_id	varchar	MULTI
數量	double	MULTI
query	varchar	MULTI
session_id	varchar	DIMENSION
user_group	varchar	DIMENSION
user_id	varchar	DIMENSION

以下說明基本資料表的量值。基礎資料表是指在 Timestream 中執行排程查詢的資料表。

- measure_name – `metrics`
- 資料 – 多
- 維度：

```
[ ( user_group, varchar ),( user_id, varchar ),( session_id, varchar ),( ip_address,
 varchar ),( event, varchar ) ]
```

基礎資料表上的查詢

以下是臨時查詢，可在指定時間範圍內透過 5 分鐘彙總收集計數。

```
SELECT BIN(time, 5m) as time,
channel,
product_id,
```

```
SUM(quantity) as product_quantity
FROM "clickstream"."shopping"
WHERE BIN(time, 5m) BETWEEN '2023-05-11 10:10:00.000000000' AND '2023-05-11
 10:30:00.000000000'
AND channel = 'Social media'
and product_id = '431412'
GROUP BY BIN(time, 5m),channel,product_id
```

輸出：

```
duration:1.745 sec
Bytes scanned: 29.89 MB
Query Id: AEBQEANMHG7MHHBHCKJ3BSOE3QUGIDBGWCCP5I6J6YUW5CVJZ2M3JCJ27QRMM7A
Row count:5
```

排程查詢

以下是每 5 分鐘執行一次的排程查詢。

```
SELECT BIN(time, 5m) as time, channel as measure_name, product_id, product,
SUM(quantity) as product_quantity
FROM "clickstream"."shopping"
WHERE time BETWEEN BIN(@scheduled_runtime, 5m) - 10m AND BIN(@scheduled_runtime, 5m) -
 5m
AND channel = 'Social media'
GROUP BY BIN(time, 5m), channel, product_id, product
```

衍生資料表上的查詢

以下是衍生資料表上的臨時查詢。衍生的資料表是指包含排程查詢結果的時間串流資料表。

```
SELECT time, measure_name, product_id,product_quantity
FROM "clickstream"."aggregate"
WHERE time BETWEEN '2023-05-11 10:10:00.000000000' AND '2023-05-11 10:30:00.000000000'
AND measure_name = 'Social media'
and product_id = '431412'
```

輸出：

```
duration: 0.2960 sec
Bytes scanned: 235.00 B
QueryID: AEBQEANMHHAAQU4FFTT6CFM6UYXTL4SMLZV22MFP4KV2Z7IRVOPLOMLDD6BR33Q
```

```
Row count: 5
```

Comparison (比較)

以下是基礎資料表查詢結果與衍生資料表查詢的比較。在衍生資料表上，透過排程查詢完成彙總結果的相同查詢會以較少的掃描位元組更快完成。

這些結果顯示使用排程查詢來彙總資料以加快查詢速度的值。

	基礎資料表的查詢	衍生資料表的查詢
持續時間	1.745 秒	0.2960 秒
掃描的位元組	29.89 MB	235 個位元組
列計數	5	5

使用 UNLOAD 將查詢結果從 Timestream 匯出至 S3 LiveAnalytics

適用於 LiveAnalytics 的 Amazon Timestream 現在可讓您使用 UNLOAD陳述式，以具成本效益且安全的方式將查詢結果匯出至 Amazon S3。您現在可以使用UNLOAD陳述式，以 Apache Parquet 或逗號分隔值 （CSV） 格式將時間序列資料匯出至選取的 S3 儲存貯體，這可讓您靈活地將時間序列資料與其他 服務存放、合併和分析。UNLOAD 陳述式可讓您以壓縮方式匯出資料，從而減少傳輸的資料和所需的儲存空間。 在匯出資料時， UNLOAD也支援根據選取的屬性進行分割，從而改善效能並減少下游服務存取資料的處理時間。此外，您可以使用 Amazon S3 受管金鑰 （SSE-S3） 或 AWS Key Management Service （AWS KMS） 受管金鑰 （SSE-KMS） 來加密匯出的資料。

UNLOAD Timestream for 的優點 LiveAnalytics

使用 UNLOAD陳述式的主要優點如下。

- 操作簡易 – 使用 UNLOAD陳述式，您可以在單一查詢請求中以 Apache Parquet 或 CSV 格式匯出 GB 的資料，提供彈性，為您的下游處理需求選擇最適合的格式，並更輕鬆地建置資料湖。
- 安全且符合成本效益 – UNLOAD陳述式可讓您以壓縮方式將資料匯出至 S3 儲存貯體，並使用客戶受管金鑰加密 （SSE-KMS 或 SSE_S3） 資料，降低資料儲存成本並防止未經授權的存取。
- 效能 – 使用 UNLOAD 陳述式，您可以在匯出至 S3 儲存貯體時分割資料。分割資料可讓下游服務平行處理資料，縮短其處理時間。此外，下游服務只能處理他們所需的資料，減少所需的處理資源，進而降低相關的成本。

UNLOAD 從 Timestream 的 使用案例 LiveAnalytics

您可以使用 UNLOAD陳述式將資料寫入 S3 儲存貯體，並執行下列動作。

- 建置 Data Warehouse – 您可以將 GB 的查詢結果匯出至 S3 儲存貯體，並更輕鬆地將時間序列資料新增至資料湖。您可以使用 Amazon Athena 和 Amazon Redshift 等服務，將時間序列資料與其他相關資料結合，以衍生複雜的業務洞察。
- 建置 AI 和 ML 資料管道 – 此UNLOAD陳述式可讓您輕鬆地為機器學習模型建置資料管道，以存取時間序列資料，進而更輕鬆地將時間序列資料與 Amazon SageMaker 和 Amazon 等服務搭配使用 EMR。
- 簡化ETL處理：將資料匯出至 S3 儲存貯體可以簡化在資料上執行提取、轉換、載入（ETL）操作的程序，讓您順暢地使用第三方工具 AWS 或服務，例如 AWS Glue 來處理和轉換資料。

UNLOAD 概念

語法

```
UNLOAD (SELECT statement)
 TO 's3://bucket-name/folder'
 WITH ( option = expression [, ...] )
```

其中 option 是

```
{ partitioned_by = ARRAY[ col_name[,…] ]
 | format = [ '{ CSV | PARQUET }' ]
 | compression = [ '{ GZIP | NONE }' ]
 | encryption = [ '{ SSE_KMS | SSE_S3 }' ]
 | kms_key = '<string>'
 | field_delimiter ='<character>'
 | escaped_by = '<character>'
 | include_header = ['{true, false}']
 | max_file_size = '<value>'
 | }
```

參數

SELECT 陳述式

用於從 LiveAnalytics 資料表的一或多個 Timestream 選取和擷取資料的查詢陳述式。

```
(SELECT column 1, column 2, column 3 from database.table
      where measure_name = "ABC" and timestamp between ago (1d) and now() )
```

TO 子句

```
TO 's3://bucket-name/folder'
```

或

```
TO 's3://access-point-alias/folder'
```

UNLOAD 陳述式中的TO子句指定查詢結果輸出的目的地。您需要提供完整路徑，包括 Amazon S3 儲存貯體名稱或 Amazon S3 access-point-alias，其中 的資料夾位置位於 Amazon S3，其中 Timestream for LiveAnalytics 會寫入輸出檔案物件。S3 儲存貯體應該由相同帳戶擁有，且位於相同區域。除了查詢結果集之外， 的 Timestream 也會將資訊清單和中繼資料檔案 LiveAnalytics 寫入指定的目的地資料夾。

PARTITIONED_BY 子句

```
partitioned_by = ARRAY [col_name[,…] , (default: none)
```

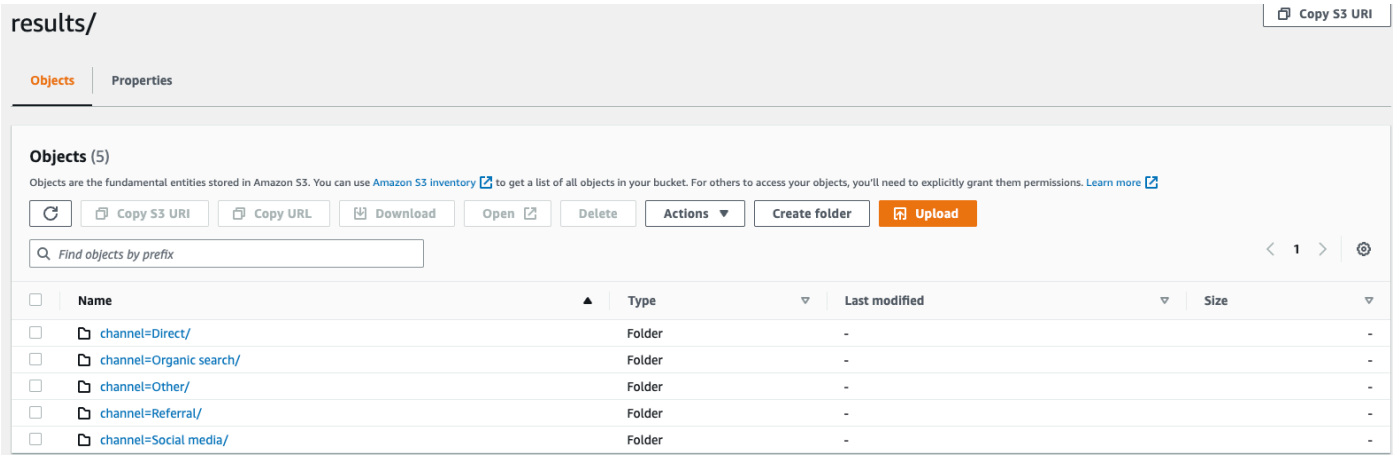
partitioned_by 子句會用於查詢，以精細層級分組和分析資料。當您將查詢結果匯出至 S3 儲存貯體時，您可以選擇根據選取查詢中的一或多個資料欄來分割資料。分割資料時，匯出的資料會根據分割區資料欄分為子集，且每個子集都存放在個別的資料夾中。在包含匯出資料的結果資料夾中，folder/results/partition column = partition value/會自動建立子資料夾。不過請注意，分割的資料欄不包含在輸出檔案中。

partitioned_by 不是語法中的強制性子句。如果您選擇在沒有分割的情況下匯出資料，則可以在語法中排除 子句。

Example

假設您正在監控網站的點擊串流資料，並擁有 5 個流量通道，即 direct、Social Media、Organic Search、 Other和 Referral。匯出資料時，您可以選擇使用資料欄 分割資料Channel。在您的 資料資料夾中s3://bucketname/results，您會有五個資料夾，每個資料夾都有各自的頻道名稱，例如，s3://bucketname/results/channel=Social Media/.在此資料夾中，您會找到透過Social Media頻道登陸網站的所有客戶的資料。同樣地，您將擁有其他資料夾用於其餘頻道。

依頻道資料欄分割的匯出資料



FORMAT

```
format = [ '{ CSV | PARQUET }' , default: CSV
```

用來指定寫入 S3 儲存貯體之查詢結果格式的關鍵字。您可以使用逗號（ ，CSV ）作為預設分隔符號，以逗號分隔值（ ） 匯出資料，或以 Apache Parquet 格式匯出資料，這是用於分析的有效開放資料欄儲存格式。

COMPRESSION

```
compression = [ '{ GZIP | NONE }' ], default: GZIP
```

您可以使用壓縮演算法壓縮匯出的資料，GZIP或指定 NONE選項使其取消壓縮。

ENCRYPTION

```
encryption = [ '{ SSE_KMS | SSE_S3 }' ], default: SSE_S3
```

Amazon S3 上的輸出檔案會使用您選取的加密選項進行加密。除了您的資料之外，資訊清單和中繼資料檔案也會根據您選取的加密選項進行加密。我們目前支援 SSE_S3 和 SSE_KMS 加密。SSE_S3 是一種伺服器端加密，Amazon S3 使用 256 位元進階加密標準（ AES ） 加密資料。SSE_KMS 是一種伺服器端加密，可使用客戶管理的金鑰加密資料。

KMS_KEY

```
kms_key = '<string>'
```

KMS 金鑰是客戶定義的金鑰，用於加密匯出的查詢結果。KMS 金鑰由 AWS Key Management Service （ AWS KMS ） 安全管理，用於加密 Amazon S3 上的資料檔案。

FIELD_DELIMITER

```
field_delimiter ='<character>' , default: (,)
```

以 CSV 格式匯出資料時，此欄位會指定用於分隔輸出檔案中欄位的單一ASCII字元，例如管道字元（|）、逗號（，）或索引標籤（/t）。CSV 檔案的預設分隔符號是逗號字元。如果資料中的值包含選取的分隔符號，則會以引號字元引述分隔符號。例如，如果您資料中的值包含 `Time,stream`，則會將此值引述為匯出資料`"Time,stream"`中的 。Timestream 使用的引號字元 LiveAnalytics 是雙引號（"）。

`FIELD_DELIMITER` 如果您想要在 中包含標頭，請避免指定歸位字元（ASCII 13， 十六進位 0D， 文字 '\r'）或換行字元（ASCII 10， 十六進位 0A ， 文字 '\n'）CSV，因為這會阻止許多剖析器在產生的CSV輸出中正確剖析標頭。

ESCAPED_BY

```
escaped_by = '<character>', default: (\)
```

以 CSV 格式匯出資料時，此欄位會指定在寫入 S3 儲存貯體的資料檔案中應視為逸出字元的字元。逸出會在下列情況下發生：

1. 如果值本身包含引號字元（"），則會使用逸出字元逸出。例如，如果值為 `Time"stream`，其中（\）是設定的逸出字元，則會以 逸出`Time\"stream`。
2. 如果值包含設定的逸出字元，則會逸出。例如，如果值為 `Time\stream`，則會以 的形式逸出`Time\\stream`。

> **Note**
>
> 如果匯出的輸出包含類似 Arrays、Rows 或 Timeseries 的複雜資料類型，則會將其序列化為JSON字串。以下是範例。

資料類型	實際值	如何逸出CSV格式為 【序列化JSON字串】 的值
陣列	`[ 23,24,25 ]`	`"[23,24,25]"`
Row	`( x=23.0, y=hello )`	`"{\"x\":23.0,\"y\":\"hello\"}"`

資料類型	實際值	如何逸出CSV格式為 【序列化JSON字串】 的值
時間序列	`[ ( time=1970-01-01 00:00:00.000000010 , value=100.0 ), ( time=1970-01-01 00:00:00.000000012, value=120.0 ) ]`	`"[{\"time\":\"1970-01-01 00:00:00.000000010Z\",\"value\":100.0},{\"time\":\"1970-01-01 00:00:00.000000012Z\",\"value\":120.0}]"`

INCLUDE_HEADER

```
include_header = 'true' , default: 'false'
```

以 CSV 格式匯出資料時，此欄位可讓您將資料欄名稱包含為匯出CSV資料檔案的第一列。

接受的值為 'true' 和 'false'，預設值為 'false'。文字轉換選項，例如 `escaped_by`和也`field_delimiter`適用於標頭。

> **Note**
>
> 包含標頭時，請務必不要選取歸位字元（ASCII 13，十六進位 0D文字 '\r'）或換行字元（ASCII 10，十六進位 0A，文字 '\n'）作為 `FIELD_DELIMITER`，因為這會阻止許多剖析器在產生的CSV輸出中正確剖析標頭。

MAX_FILE_SIZE

```
max_file_size = 'X[MB|GB]' , default: '78GB'
```

此欄位會指定UNLOAD陳述式在 Amazon S3 中建立的檔案大小上限。UNLOAD 陳述式可以建立多個檔案，但寫入 Amazon S3 的每個檔案的大小上限大約是此欄位中指定的大小。

欄位的值必須介於 16 MB 和 78 GB 之間，包括在內。您可以指定整數，例如 `12GB`，或小數，例如 `0.5GB`或 `24.7MB`。預設值為 78 GB。

實際檔案大小會在寫入檔案時近似，因此實際大小上限可能不等於您指定的數字。

寫入 S3 儲存貯體的內容為何？

對於每個成功執行的UNLOAD查詢， 的 Timestream 會將您的查詢結果、中繼資料檔案和資訊清單檔案 LiveAnalytics 寫入 S3 儲存貯體。如果您已分割資料，則結果資料夾中會包含所有分割區資料夾。清單檔案包含由 UNLOAD命令寫入的檔案清單。中繼資料檔案包含描述書面資料特性、屬性和屬性的資訊。

匯出的檔案名稱是什麼？

匯出的檔案名稱包含兩個元件，第一個元件是 queryID，第二個元件是唯一的識別符。

CSV 檔案

```
S3://bucket_name/results/<queryid>_<UUID>.csv
S3://bucket_name/results/<partitioncolumn>=<partitionvalue>/<queryid>_<UUID>.csv
```

壓縮CSV檔案

```
S3://bucket_name/results/<partitioncolumn>=<partitionvalue>/<queryid>_<UUID>.gz
```

Parquet 檔案

```
S3://bucket_name/results/<partitioncolumn>=<partitionvalue>/<queryid>_<UUID>.parquet
```

中繼資料和清單檔案

```
S3://bucket_name/<queryid>_<UUID>_manifest.json
S3://bucket_name/<queryid>_<UUID>_metadata.json
```

由於 CSV 格式的資料儲存在檔案層級，當您在匯出至 S3 時壓縮資料時，檔案將具有「.gz」副檔名。不過，Parquet 中的資料會在資料欄層級壓縮，因此即使您在匯出時壓縮資料， 檔案仍會有 .parquet 副檔名。

每個檔案包含哪些資訊？

清單檔案

資訊清單檔案提供與UNLOAD執行一起匯出之檔案清單的相關資訊。清單檔案可在提供的 S3 儲存貯體中使用，檔案名稱為：s3://<bucket_name>/<queryid>_<UUID>_manifest.json。資訊清單檔案將包含結果資料夾中檔案的 URL、個別檔案的記錄數目和大小，以及查詢中繼資料（這是匯出至 S3 的查詢總位元組數和總列數）。

```
{
  "result_files": [
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 32295,
                "row_count": 10
            }
    },
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 62295,
                "row_count": 20
            }
    },
  ],
  "query_metadata":
    {
      "content_length_in_bytes": 94590,
      "total_row_count": 30,
      "result_format": "CSV",
      "result_version": "Amazon Timestream version 1.0.0"
    },
  "author": {
        "name": "Amazon Timestream",
        "manifest_file_version": "1.0"
  }
}
```

中繼資料

中繼資料檔案提供有關資料集的其他資訊，例如資料欄名稱、資料欄類型和結構描述。中繼資料檔案可在提供的 S3 儲存貯體中使用，檔案名稱為：S3：//bucket_name/<queryid>_<UUID>_metadata.json

以下是中繼資料檔案的範例。

```
{
    "ColumnInfo": [
        {
            "Name": "hostname",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "region",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "measure_name",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "cpu_utilization",
            "Type": {
                "TimeSeriesMeasureValueColumnInfo": {
                    "Type": {
                        "ScalarType": "DOUBLE"
                    }
                }
            }
        }
  ],
  "Author": {
        "Name": "Amazon Timestream",
        "MetadataFileVersion": "1.0"
  }
}
```

在中繼資料檔案中共用的資料欄資訊與查詢API回應中ColumnInfo傳送的結構相同SELECT。

結果

結果資料夾包含 Apache Parquet 或 CSV 格式的匯出資料。

範例

當您透過UNLOAD查詢 提交如下所示的查詢時API，

```
UNLOAD(SELECT user_id, ip_address, event, session_id, measure_name, time, query,
 quantity, product_id, channel
                    FROM sample_clickstream.sample_shopping WHERE time BETWEEN ago(2d)
 AND now())
                TO 's3://my_timestream_unloads/withoutpartition/' WITH ( format='CSV',
 compression='GZIP')
```

UNLOAD 查詢回應將有 1 列 * 3 欄。這 3 欄為：

- 類型的資料列 BIGINT - 指示匯出的資料列數目
- metadataFile 類型 VARCHAR - 即匯出URI的中繼資料檔案的 S3
- manifestFile 類型 VARCHAR - 即匯出清單檔案URI的 S3

您會從查詢 收到下列回應API：

```
{
    "Rows": [
        {
            "Data": [
                {
                    "ScalarValue": "20" # No of rows in output across all files
                },
                {
                    "ScalarValue": "s3://my_timestream_unloads/withoutpartition/
AEDAAANGH3D7FYHOBQGQQMEAISCJ45B42OWWJMOT4N6RRJICZUA7R25VYVOHJIY_<UUID>_metadata.json"
 #Metadata file
                },
                {
                    "ScalarValue": "s3://my_timestream_unloads/withoutpartition/
AEDAAANGH3D7FYHOBQGQQMEAISCJ45B42OWWJMOT4N6RRJICZUA7R25VYVOHJIY_<UUID>_manifest.json"
 #Manifest file
```

```
                }
            ]
        }
    ],
    "ColumnInfo": [
        {
            "Name": "rows",
            "Type": {
                "ScalarType": "BIGINT"
            }
        },
        {
            "Name": "metadataFile",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "manifestFile",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        }
    ],
    "QueryId": "AEDAAANGH3D7FYHOBQGQQMEAISCJ45B42OWWJMOT4N6RRJICZUA7R25VYVOHJIY",
    "QueryStatus": {
        "ProgressPercentage": 100.0,
        "CumulativeBytesScanned": 1000,
        "CumulativeBytesMetered": 10000000
    }
}
```

資料類型

UNLOAD 陳述式支援支援的資料類型除了 time和 所述的 LiveAnalytics查詢語言的所有 Timestream 資料類型unknown。

UNLOAD 從 Timestream for 的 先決條件 LiveAnalytics

以下是使用 UNLOAD Timestream for 將資料寫入 S3 的先決條件 LiveAnalytics。

- 您必須具有從 Timestream 讀取資料的權限，才能在UNLOAD命令中使用 LiveAnalytics 資料表（s）。

- 您必須在與 Timestream 相同的 AWS 區域擁有 Amazon S3 儲存貯體，才能 LiveAnalytics取得 資源。
- 對於選取的 S3 儲存貯體，請確定 S3 儲存貯體政策也具有允許 Timestream LiveAnalytics 匯出資料的許可。
- 用於執行UNLOAD查詢的憑證必須具有必要的 AWS Identity and Access Management （ IAM ） 許可，允許 Timestream LiveAnalytics 將資料寫入 S3。範例政策如下：

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "timestream:Select",
                "timestream:ListMeasures",
                "timestream:WriteRecords",
                "timestream:Unload"
            ],
            "Resource": "arn:aws:timestream:<region>:<account_id>:database/
<database_name>/table/<table_name>"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketAcl",
                "s3:PutObject",
                "s3:GetObjectMetadata",
                "s3:AbortMultipartUpload"
            ],
            "Resource": [
                "arn:aws:s3:::<S3_Bucket_Created>",
                "arn:aws:s3:::<S3_Bucket_Created>/*"
            ]
        }
    ]
}
```

如需這些 S3 寫入許可的其他內容，請參閱 Amazon Simple Storage Service 指南 。如果您使用 KMS 金鑰來加密匯出的資料，請參閱下列其他必要IAM政策。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
            "kms:DescribeKey",
            "kms:Decrypt",
            "kms:GenerateDataKey*"
        ],
        "Resource": "<account_id>-arn:aws:kms:<region>:<account_id>:key/*",
        "Condition": {
            "ForAnyValue:StringLike": {
                "kms:ResourceAliases": "alias/<Alias_For_Generated_Key>"
            }
        }
    }, {
        "Effect": "Allow",
        "Action": [
            "kms:CreateGrant"
        ],
        "Resource": "<account_id>-arn:aws:kms:<region>:<account_id>:key/*",
        "Condition": {
            "ForAnyValue:StringEquals": {
                "kms:EncryptionContextKeys": "aws:timestream:<database_name>"
            },
            "Bool": {
                "kms:GrantIsForAWSResource": true
            },
            "StringLike": {
                "kms:ViaService": "timestream.<region>.amazonaws.com"
            },
            "ForAnyValue:StringLike": {
                "kms:ResourceAliases": "alias/<Alias_For_Generated_Key>"
            }
        }
    }
]
}
```

UNLOAD 從 Timestream for 的 最佳實務 LiveAnalytics

以下是與UNLOAD命令相關的最佳實務。

- 使用 UNLOAD命令可匯出至 S3 儲存貯體的資料量不受限。不過，查詢會在 60 分鐘內逾時，建議您在單一查詢中匯出不超過 60GB 的資料。如果您需要匯出超過 60GB 的資料，請將任務分割為多個查詢。
- 雖然您可以將數千個請求傳送至 S3 以上傳資料，但建議您將寫入操作平行到多個 S3 字首。請參閱此處的文件。當多個讀取器/寫入器存取相同的資料夾時，可能會調節 S3 API呼叫率。
- 鑑於定義字首的 S3 金鑰長度限制，我們建議您使用 10-15 個字元內的儲存貯體和資料夾名稱，特別是在使用`partitioned_by`子句時。
- 當您收到包含 UNLOAD陳述式的查詢的 4XX 或 5XX 時，可能會將部分結果寫入 S3 儲存貯體。的 Timestream LiveAnalytics 不會刪除儲存貯體中的任何資料。在使用相同的 S3 目的地執行另一個UNLOAD查詢之前，建議您手動刪除失敗查詢建立的檔案。您可以使用對應的 來識別由失敗查詢寫入的檔案`QueryExecutionId`。對於失敗的查詢， 的 Timestream LiveAnalytics 不會將資訊清單檔案匯出至 S3 儲存貯體。
- 的 Timestream LiveAnalytics 使用分段上傳將查詢結果匯出至 S3。當您從 Timestream 收到 的 4XX 或 5XX LiveAnalytics 來查詢包含 UNLOAD 陳述式時，Timestream for LiveAnalytics 會盡最大努力中止分段上傳，但可能會留下一些不完整的部分。因此，我們建議您遵循此處的準則，在 S3 儲存貯體中設定未完成分段上傳的自動清除。

使用CSV剖析器以 CSV 格式存取資料的建議

- CSV 剖析器不允許您在分隔符號、逸出和引號字元中具有相同的字元。
- CSV 有些剖析器無法解譯複雜的資料類型，例如 Arrays，建議您透過JSON還原序列化器解譯這些類型。

以 Parquet 格式存取資料的建議

1. 如果您的使用案例需要在結構描述 aka 資料欄名稱中支援 UTF-8 個字元，我們建議您使用 Parquet-mr 程式庫 。
2. 結果中的時間戳記會以 12 位元組整數表示 （INT96）
3. Timeseries 將以 表示`array<row<time, value>>`，其他巢狀結構將使用 Parquet 格式支援的對應資料類型

使用 partition_by 子句

- 欄位中使用的資料欄partitioned_by應該是選取查詢中的最後一欄。如果partitioned_by欄位中使用多個資料欄，則資料欄應該是選取查詢中的最後一個資料欄，並且與partition_by欄位中使用的順序相同。
- 用於分割資料（partitioned_by 欄位）的資料欄值只能包含 ASCII 個字元。雖然 的 Timestream 在值中 LiveAnalytics 允許 UTF-8 個字元，但 S3 僅支援ASCII字元作為物件金鑰。

UNLOAD 從 Timestream 的 使用案例範例 LiveAnalytics

假設您正在監控電子商務網站的使用者工作階段指標、流量來源和產品購買。您正在使用 Timestream for LiveAnalytics 來即時深入分析使用者行為、產品銷售，並對將客戶導向網站的流量管道（有機搜尋、社交媒體、直接流量、付費行銷活動等）執行行銷分析。

主題

- 匯出不含任何分割區的資料
- 依通道分割資料
- 依事件分割資料
- 依頻道和事件分割資料
- 清單和中繼資料檔案
- 使用 Glue 爬蟲程式建置 Glue Data Catalog

匯出不含任何分割區的資料

您想要以 CSV 格式匯出資料的最後兩天。

```
UNLOAD(SELECT user_id, ip_address, event, session_id, measure_name, time,
query, quantity, product_id, channel
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/withoutpartition'
WITH (  format='CSV',
compression='GZIP')
```

依通道分割資料

您想要以 CSV 格式匯出最後兩天的資料，但想要將來自每個流量頻道的資料匯出到個別資料夾中。若要這麼做，您需要使用資料channel欄分割資料，如下所示。

```
UNLOAD(SELECT user_id, ip_address, event, session_id, measure_name, time,
query, quantity, product_id, channel
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/partitionbychannel/'
WITH (
partitioned_by = ARRAY ['channel'],
format='CSV',
compression='GZIP')
```

依事件分割資料

您想要以 CSV 格式匯出最後兩天的資料，但想要將每個事件的資料存放在個別的資料夾中。若要這麼做，您需要使用資料event欄分割資料，如下所示。

```
UNLOAD(SELECT user_id, ip_address, channel, session_id, measure_name, time,
query, quantity, product_id, event
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/partitionbyevent/'
WITH (
partitioned_by = ARRAY ['event'],
format='CSV',
compression='GZIP')
```

依頻道和事件分割資料

您想要以 CSV 格式匯出最後兩天的資料，但想要將每個頻道的資料，以及在頻道內將每個事件儲存在個別資料夾中。若要這麼做，您需要使用 channel和 event資料欄來分割資料，如下所示。

```
UNLOAD(SELECT user_id, ip_address, session_id, measure_name, time,
query, quantity, product_id, channel,event
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/partitionbychannelevent/'
WITH (
```

```
partitioned_by = ARRAY ['channel','event'],
format='CSV',
compression='GZIP')
```

清單和中繼資料檔案

清單檔案

資訊清單檔案提供與UNLOAD執行一起匯出的檔案清單資訊。清單檔案可在提供的 S3 儲存貯體中使用，檔案名稱為：S3://bucket_name/<queryid>_<UUID>_manifest.json。資訊清單檔案將包含結果資料夾中檔案的 URL、個別檔案的記錄數目和大小，以及查詢中繼資料（這是匯出至 S3 以進行查詢的總位元組數和總列數）。

```
{
  "result_files": [
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 32295,
                "row_count": 10
            }
    },
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 62295,
                "row_count": 20
            }
    },
  ],
  "query_metadata":
    {
      "content_length_in_bytes": 94590,
      "total_row_count": 30,
      "result_format": "CSV",
      "result_version": "Amazon Timestream version 1.0.0"
    },
  "author": {
        "name": "Amazon Timestream",
```

```
        "manifest_file_version": "1.0"
  }
}
```

中繼資料

中繼資料檔案提供有關資料集的其他資訊，例如資料欄名稱、資料欄類型和結構描述。中繼資料檔案可在提供的 S3 儲存貯體中使用，檔案名稱為：S3：//bucket_name/<queryid>_<UUID>_metadata.json

以下是中繼資料檔案的範例。

```
{
    "ColumnInfo": [
        {
            "Name": "hostname",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "region",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "measure_name",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "cpu_utilization",
            "Type": {
                "TimeSeriesMeasureValueColumnInfo": {
                    "Type": {
                        "ScalarType": "DOUBLE"
                    }
                }
            }
        }
  ],
  "Author": {
```

```
        "Name": "Amazon Timestream",
        "MetadataFileVersion": "1.0"
    }
}
```

在中繼資料檔案中共用的資料欄資訊與查詢API回應中ColumnInfo傳送的結構相同SELECT。

使用 Glue 爬蟲程式建置 Glue Data Catalog

1. 使用 Admin 憑證登入您的帳戶，以進行下列驗證。
2. 使用此處提供的指導方針建立 Glue 資料庫的 Crawler。請注意，資料來源中提供的 S3 資料夾應該是UNLOAD結果資料夾，例如 s3://my_timestream_unloads/results。
3. 依照此處的指導方針執行爬蟲程式。
4. 檢視 Glue 資料表。
   - 前往 AWS Glue → 資料表 。
   - 您將看到在建立爬蟲程式時，使用提供的資料表字首建立新的資料表。
   - 您可以按一下資料表詳細資訊檢視來查看結構描述和分割區資訊。

以下是使用 AWS Glue Data Catalog 的其他 AWS 服務和開放原始碼專案。

- Amazon Athena – 如需詳細資訊，請參閱 Amazon Athena 使用者指南中的了解資料表、資料庫和資料目錄。
- Amazon Redshift Spectrum – 如需詳細資訊，請參閱 Amazon Redshift 資料庫開發人員指南中的使用 Amazon Redshift Spectrum 查詢外部資料。
- Amazon EMR – 如需詳細資訊，請參閱 Amazon 管理指南中的使用資源型政策來EMR存取 AWS Glue Data Catalog。 EMR
- AWS 適用於 Apache Hive 中繼存放區的 Glue Data Catalog 用戶端 – 如需此 GitHub專案的詳細資訊，請參閱AWS 適用於 Apache Hive 中繼存放區的 Glue Data Catalog Client。

UNLOAD 從 Timestream 的 限制 LiveAnalytics

以下是與UNLOAD命令相關的限制。

- 使用 UNLOAD陳述式查詢的並行為每秒 1 個查詢 （ QPS ）。超過查詢速率可能會導致限流。
- 包含UNLOAD陳述式的查詢每個查詢最多可以匯出 100 個分割區。建議您先檢查所選資料欄的不同計數，再使用它來分割匯出的資料。

- 包含UNLOAD陳述式逾時的查詢會在 60 分鐘後逾時。
- UNLOAD 陳述式在 Amazon S3 中建立的檔案大小上限為 78 GB。

如需 的 Timestream 其他限制 LiveAnalytics，請參閱 配額

使用查詢洞察功能來最佳化 Amazon Timestream 中的查詢

查詢洞察是一種效能調校功能，可協助您最佳化查詢、改善其效能並降低成本。透過查詢洞察，您可以評估查詢的時間、時間和空間分割區金鑰型剪除效率。您還可以使用查詢洞察，識別需要改進的區域，以增強查詢效能。此外，透過查詢洞察，您可以評估查詢使用時間型和分割區金鑰型索引來最佳化資料擷取的有效性。若要最佳化查詢效能，請務必微調管理查詢執行的時序和空間參數。

主題

- 查詢洞察的優點
- 最佳化 Amazon Timestream 中的資料存取
- 在 Amazon Timestream 中啟用查詢洞察
- 使用查詢洞察回應最佳化查詢

查詢洞察的優點

以下是使用查詢洞察的主要優點：

- 識別低效率查詢 – 查詢洞察提供查詢存取資料表的時間型和屬性型剪除資訊。此資訊可協助您識別次佳存取的資料表。
- 最佳化資料模型和分割 – 您可以使用查詢洞察資訊來存取和微調資料模型和分割策略。
- 調整查詢 – 查詢洞見強調更有效地使用索引的機會。

最佳化 Amazon Timestream 中的資料存取

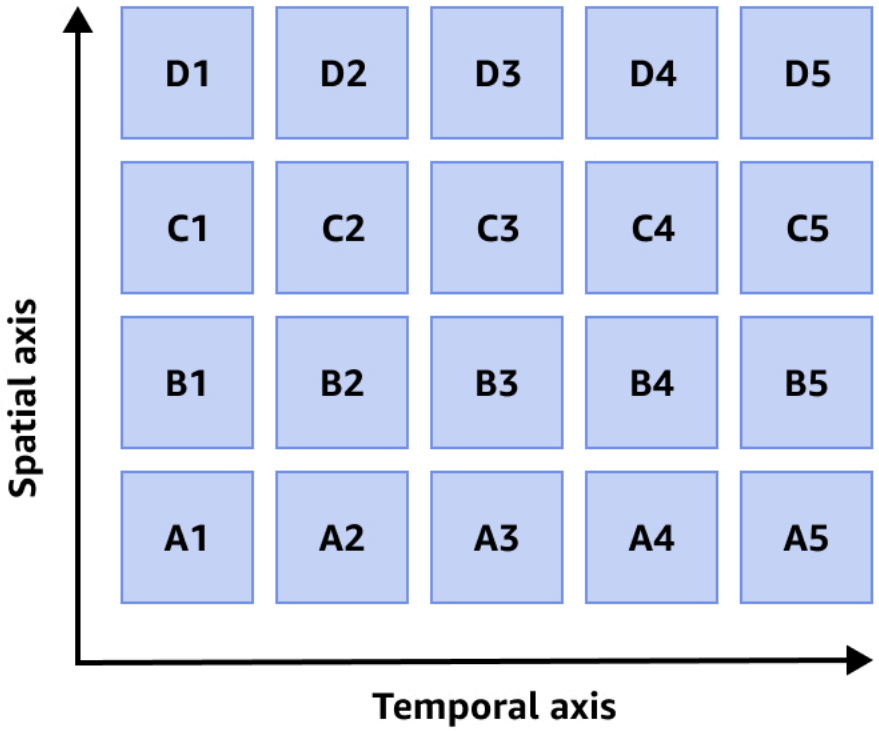
您可以使用 Timestream 分割方案或資料組織技術，最佳化 Amazon Timestream 中的資料存取模式。

主題

- 時間串流分割方案
- 資料組織

時間串流分割方案

Amazon Timestream 使用高度可擴展的分割方案，其中每個 Timestream 資料表可以有數百個、數千個或甚至數百萬個獨立分割區。高可用性的分割區追蹤和索引服務可管理分割區，將故障的影響降至最低，並讓系統更具彈性。



資料組織

Timestream 會將擷取的每個資料點存放在單一分割區中。當您將資料擷取到 Timestream 資料表時，Timestream 會根據時間戳記、分割區金鑰和資料中的其他內容屬性自動建立分割區。除了按時間分割資料（時間分割）之外，Timestream 也會根據選取的分割金鑰和其他維度（空間分割）分割資料。此方法旨在分發寫入流量，並允許有效截斷查詢的資料。

查詢洞察功能提供查詢縮減效率的寶貴洞察，包括查詢空間涵蓋範圍和查詢時間涵蓋範圍。

主題

- QuerySpatialCoverage
- QueryTemporalCoverage

QuerySpatialCoverage

此QuerySpatialCoverage指標提供已執行查詢的空間涵蓋範圍，以及空間修剪效率最差資料表的深入見解。此資訊可協助您識別分割策略中需要改進的區域，以增強空間刪除。QuerySpatialCoverage指標的值範圍介於 0 和 1 之間。指標的值越低，在空間軸上剪除的查詢越最佳。例如，值 0.1 表示查詢掃描 10% 的空間軸。值 1 表示查詢掃描 100% 的空間軸。

Example 使用查詢洞察分析查詢的空間涵蓋範圍

假設您有一個存放天氣資料的 Timestream 資料庫。假設每小時從位於美國不同州的氣象站記錄溫度。假設您選擇 State作為客戶定義的分割金鑰 （CDPK），依狀態分割資料。

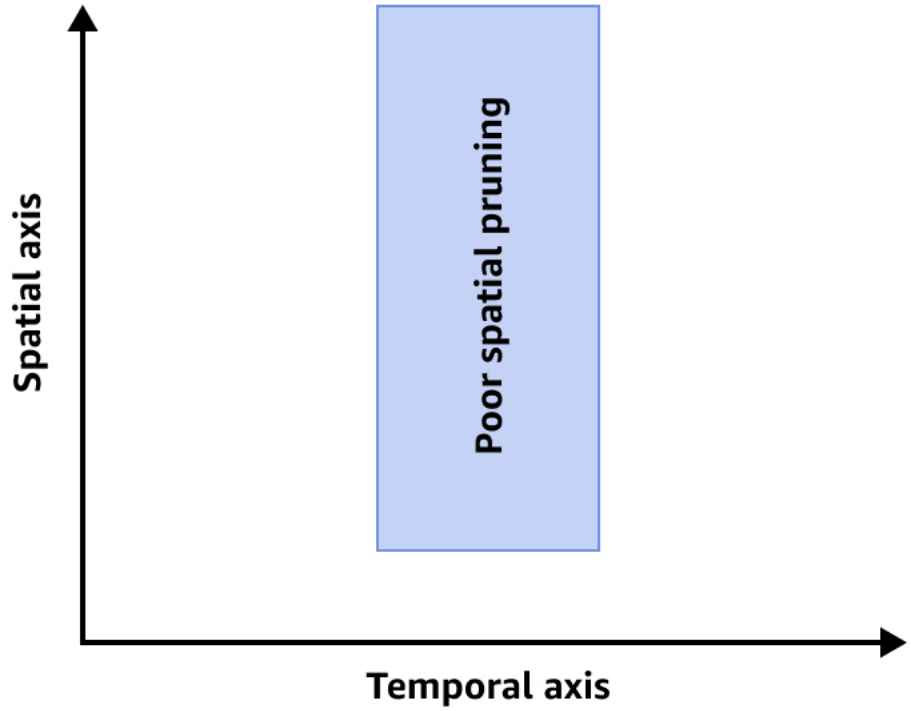
假設您執行查詢，以擷取加州所有氣象站在特定日期的下午 2 點到下午 4 點之間的平均溫度。下列範例顯示此案例的查詢。

```
SELECT AVG(temperature)
FROM "weather_data"."hourly_weather"
WHERE time >= '2024-10-01 14:00:00' AND time < '2024-10-01 16:00:00'
  AND state = 'CA';
```

使用查詢洞察功能，您可以分析查詢的空間涵蓋範圍。假設QuerySpatialCoverage指標傳回 0.02 的值。這表示查詢只會掃描 2% 的空間軸，這很有效。在此情況下，查詢能夠有效地剪除空間範圍，僅從 California 擷取資料，並忽略其他狀態的資料。

相反地，如果QuerySpatialCoverage指標傳回 0.8 的值，則表示查詢掃描了 80% 的空間軸，效率較低。這可能表示需要改進分割策略，才能改善空間修剪。例如，您可以選取分割區金鑰作為城市或區域，而非狀態。透過分析QuerySpatialCoverage指標，您可以識別最佳化分割策略並改善查詢效能的機會。

下圖顯示空間修剪不良。



若要提高空間修剪效率，您可以執行下列其中一項或兩項操作：

- 新增 measure_name、預設剖析金鑰，或在查詢中使用述CDPK詞。
- 如果您已新增上一個點中提到的屬性，請移除這些屬性或子句周圍的函數，例如 LIKE。

QueryTemporalCoverage

此QueryTemporalCoverage指標提供對已執行查詢掃描的時間範圍的洞察，包括掃描最大時間範圍的資料表。QueryTemporalCoverage 指標的值是以奈秒為單位的時間範圍。此指標的值越低，查詢在時間範圍內剪除越最佳。例如，查詢掃描最後幾分鐘的資料效能高於掃描資料表整個時間範圍的查詢。

Example

假設您有一個存放 IoT 感應器資料的 Timestream 資料庫，每分鐘從位於製造工廠的裝置進行測量。假設您已透過 分割資料device_ID。

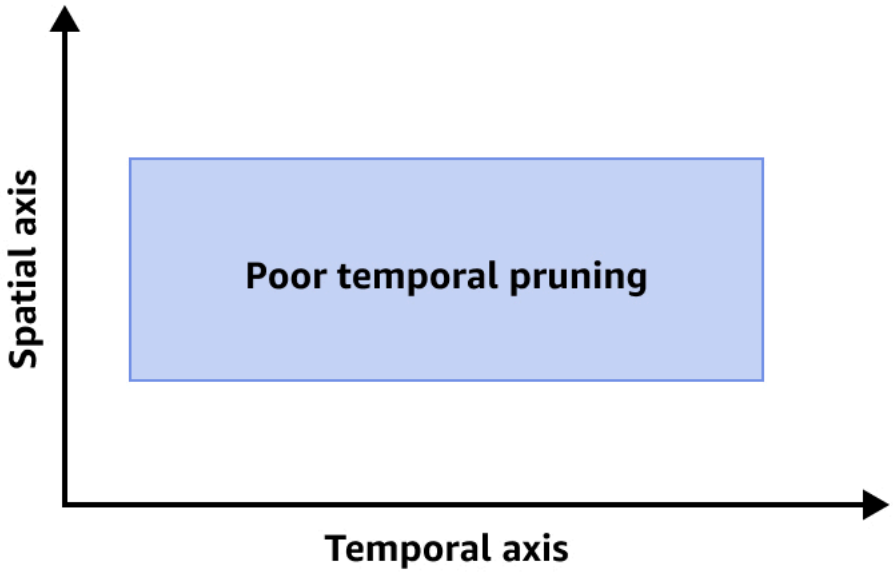
假設您執行查詢，以擷取過去 30 分鐘內特定裝置的平均感應器讀數。下列範例顯示此案例的查詢。

```
SELECT AVG(sensor_reading)
FROM "sensor_data"."factory_1"
WHERE device_id = 'DEV_123'
  AND time >= NOW() - INTERVAL 30 MINUTE and time < NOW();
```

使用查詢洞察功能，您可以分析查詢掃描的時間範圍。假設QueryTemporalCoverage指標傳回 1800000000000 奈秒 （ 30 分鐘 ） 的值。這表示查詢只會掃描過去 30 分鐘的資料，這是相對狹窄的時間範圍。這是個好跡象，因為它表示查詢能夠有效地剪除時間分割，並且只擷取請求的資料。

相反地，如果QueryTemporalCoverage指標傳回值 1 年，以奈秒為單位，則表示查詢掃描了資料表中一年的時間範圍，效率較低。這可能表示查詢未針對時間剪除進行最佳化，而您可以透過新增時間篩選條件來改善查詢。

下圖顯示時間修剪不良。



為了改善時間修剪，建議您執行下列其中一項或所有動作：

- 在查詢中新增缺少的時間述詞，並確認時間述詞正在剪除所需的時段。
- 在時間述詞周圍移除 函數MAX()，例如 。
- 將時間述詞新增至所有子查詢。如果您的子查詢正在加入大型資料表或執行複雜的操作，這很重要。

在 Amazon Timestream 中啟用查詢洞察

您可以使用透過查詢回應直接傳遞的洞察，為查詢啟用查詢洞察。啟用查詢洞察不需要額外的基礎設施或產生任何額外的成本。當您啟用查詢洞察時，除了查詢結果之外，還會傳回查詢效能相關的中繼資料欄位，作為查詢回應的一部分。您可以使用此資訊來調整查詢，以改善查詢效能並降低查詢成本。

如需啟用查詢洞察的資訊，請參閱 執行查詢。

若要檢視啟用查詢洞察傳回的回應範例，請參閱排程查詢的範例 。

> **Note**
>
> - 當您啟用查詢洞察時，其速率會將查詢限制為每秒 1 個查詢 （QPS）。為了避免效能影響，強烈建議您在查詢的評估階段啟用查詢洞察，然後再將其部署到生產環境。
> - 查詢洞察中提供的洞察最終一致，這表示隨著新資料持續擷取到資料表中，它們可能會發生變化。

使用查詢洞察回應最佳化查詢

假設您正在使用 的 Amazon Timestream LiveAnalytics 來監控不同位置的能源消耗。假設資料庫中有兩個名為 `raw-metrics`和 的資料表`aggregate-metrics`。

`raw-metrics` 資料表會將詳細的能源資料存放在裝置層級，並包含下列資料欄：

- 時間戳記
- 狀態，例如 Washington
- 裝置 ID
- 能源消耗

此資料表的資料會精細地收集和儲存 minute-by-minute。資料表使用 `State`作為 CDPK。

`aggregate-metrics` 資料表會儲存排程查詢的結果，以每小時彙總所有裝置的能耗資料。此資料表包含下列資料欄：

- 時間戳記
- 狀態，例如 Washington
- 總能耗

aggregate-metrics 資料表以每小時的精細程度存放此資料。資料表使用 State作為 CDPK。

主題

查詢過去 24 小時的能源消耗

假設您想要擷取過去 24 小時內在華盛頓消耗的總能源。若要尋找此資料，您可以利用資料表： raw-metrics和 的強度aggregate-metrics。aggregate-metrics 資料表提供過去 23 小時的每小時能耗資料，而raw-metrics資料表則提供過去 1 小時的分鐘精細資料。透過跨兩個資料表查詢，您可以取得過去 24 小時內在華盛頓的能耗完整且準確的影像。

```
SELECT am.time, am.state, am.total_energy_consumption,
rm.time, rm.state, rm.device_id, rm.energy_consumption
FROM
 "metrics"."aggregate-metrics" am
 LEFT JOIN "metrics"."raw-metrics" rm ON am.state = rm.state
WHERE rm.time >= ago(1h) and rm.time < now()
```

此範例查詢僅供說明用途，可能無法正常運作。它旨在示範概念，但您可能需要修改它以符合您的特定使用案例或環境。

執行此查詢後，您可能會注意到查詢回應時間比預期慢。若要識別此效能問題的根本原因，您可以使用查詢洞察功能來分析查詢的效能並最佳化其執行。

下列範例顯示查詢洞察回應。

```
queryInsightsResponse={
                QuerySpatialCoverage: {
                    Max: {
                        Value: 1.0,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/raw-metrics,
                        PartitionKey: [State]
                    }
                },
                QueryTemporalRange: {
```

```
                    Max: {
                        Value:31540000000000000 //365 days,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics
                    }
                },
                QueryTableCount: 2,
                OutputRows: 83,
                OutputBytes: 590
```

查詢洞察回應提供下列資訊：

- 時間範圍 ：查詢掃描aggregate-metrics了資料表過多的 365 天時間範圍。這表示時間篩選的使用效率低下。
- 空間涵蓋範圍 ：查詢掃描了raw-metrics資料表的整個空間範圍 （ 100% ）。這表示空間篩選未有效使用。

如果您的查詢存取多個資料表，查詢洞察會提供具有最多次最佳存取模式之資料表的指標。

最佳化時間範圍的查詢

根據查詢洞察回應，您可以最佳化時間範圍的查詢，如下列範例所示。

```
SELECT am.time, am.state, am.total_energy_consumption,
rm.time, rm.state, rm.device_id, rm.energy_consumption
FROM
  "metrics"."aggregate-metrics" am
  LEFT JOIN "metrics"."raw-metrics" rm ON am.state = rm.state
WHERE
  am.time >=  ago(23h) and am.time < now()
  AND rm.time >=  ago(1h) and rm.time < now()
  AND rm.state = 'Washington'
```

如果您再次執行QueryInsights命令，則會傳回下列回應。

```
queryInsightsResponse={
                QuerySpatialCoverage: {
                    Max: {
                        Value: 1.0,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics,
```

```
                        PartitionKey: [State]
                    }
                },
                QueryTemporalRange: {
                    Max: {
                        Value: 82800000000000 //23 hours,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics
                    }
                },
                QueryTableCount: 2,
                OutputRows: 83,
                OutputBytes: 590
```

此回應顯示aggregate-metrics資料表的空間覆蓋率仍然是 100%，這是無效的。下一節說明如何最佳化空間涵蓋範圍的查詢。

最佳化空間涵蓋範圍的查詢

根據查詢洞察回應，您可以最佳化空間涵蓋範圍的查詢，如下列範例所示。

```
SELECT am.time, am.state, am.total_energy_consumption,
rm.time, rm.state, rm.device_id, rm.energy_consumption
FROM
  "metrics"."aggregate-metrics" am
  LEFT JOIN "metrics"."raw-metrics" rm ON am.state = rm.state
WHERE
  am.time >=  ago(23h) and am.time < now()
  AND am.state ='Washington'
  AND rm.time >=  ago(1h) and rm.time < now()
  AND rm.state = 'Washington'
```

如果您再次執行QueryInsights命令，則會傳回下列回應。

```
queryInsightsResponse={
                QuerySpatialCoverage: {
                    Max: {
                        Value: 0.02,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics,
                        PartitionKey: [State]
                    }
                },
```

```
                QueryTemporalRange: {
                    Max: {
                        Value: 82800000000000 //23 hours,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics
                    }
                },
                QueryTableCount: 2,
                OutputRows: 83,
                OutputBytes: 590
```

改善查詢效能

最佳化查詢後，查詢洞察會提供下列資訊：

- `aggregate-metrics` 資料表的暫時修剪為 23 小時。這表示只會掃描 23 小時的時間範圍。
- `aggregate-metrics` 資料表的空間修剪為 0.02。這表示只會掃描資料表空間範圍資料的 2%。查詢會掃描資料表的一小部分，導致效能快速且資源使用率降低。改善的剪除效率表示查詢現在已針對效能進行最佳化。

使用 AWS Backup

適用於 的 Amazon Timestream 中的資料保護功能 LiveAnalytics 是完全受管的解決方案，可協助您符合法規合規和業務連續性要求。此功能是透過原生整合 來啟用 AWS Backup，這是一種統一備份服務，旨在簡化備份的建立、遷移、還原和刪除，同時提供改進的報告和稽核。透過與 的整合 AWS Backup，您可以使用完全受管、政策驅動的集中式資料保護解決方案來建立不可變備份，並集中管理跨越 Timestream 和其他 支援之 AWS服務的應用程式資料保護 AWS Backup。

若要使用 功能，您必須選擇加入以允許 AWS Backup 來保護您的 Timestream 資源。選擇加入選項適用於特定帳戶和 AWS 區域，因此您可能必須使用相同的帳戶選擇加入多個區域。如需備份的詳細資訊 AWS ，請參閱 AWS Backup 開發人員指南 。

透過 提供的資料保護功能 AWS Backup 包括以下內容。

排程備份：您可以使用備份計畫為 LiveAnalytics 資料表設定 Timestream 的定期排程備份。

跨帳戶和跨區域複製 — 您可以將備份自動複製到不同 AWS 區域或帳戶中的另一個備份保存庫，這可讓您支援資料保護需求。

冷儲存體分層：您可以設定備份以實作生命週期規則，以刪除或轉換備份至較冷的儲存體。這可協助您最佳化備份成本。

標籤：您可以自動標記備份，用於計費和成本分配。

加密：您的備份資料會儲存在保存庫中 AWS Backup 。這可讓您使用獨立於 AWS KMS Timestream 的金鑰來加密和保護備份，以進行 LiveAnalytics 資料表加密金鑰。

使用WORM模型保護備份：您可以使用 AWS Backup 保存庫鎖定為備份啟用 write-once-read-many（WORM） 設定。透過保存 AWS Backup 庫鎖定，您可以新增額外的防禦層，保護備份免受意外或惡意刪除操作、備份保留期的變更，以及生命週期設定的更新。如需進一步了解，請參閱AWS Backup 保存庫鎖。

資料保護功能適用於所有區域 若要進一步了解此功能，請參閱 AWS Backup 開發人員指南 。

備份和還原 Timestream 資料表：運作方式

您可以建立 Amazon Timestream 資料表的備份。本節概述備份與還原程序期間所發生的情況。

主題

備份

您可以使用隨需備份功能來建立 LiveAnalytics 資料表的 Amazon Timestream 完整備份。本節概述備份與還原程序期間所發生的情況。

您可以精細地建立 Timestream 資料的備份。您可以使用 Timestream 主控台或 AWS Backup 主控台或 SDK啟動所選資料表的備份CLI。備份會以非同步方式建立，且資料表中的所有資料都會包含在備份中，直到備份啟動時間為止。不過，備份進行中時，某些擷取到資料表的資料也可能包含在備份中。若要保護您的資料，您可以建立一次性隨需備份或排程資料表的重複備份。

備份進行中時，您無法執行下列動作。

- 暫停或取消備份操作。
- 刪除備份的來源資料表。
- 在資料表的備份進行時，停用該資料表的備份。

設定完成後， AWS Backup 會提供自動化備份排程、保留管理和生命週期管理，消除對自訂指令碼和手動程序的需求。如需詳細資訊，請參閱 AWS Backup 開發人員指南

備份的所有 Timestream LiveAnalytics 都是增量性質，表示資料表的第一個備份是完整備份，而相同資料表的每個後續備份都是增量備份，僅複製自上次備份以來對資料所做的變更。由於 的 Timestream 中的資料 LiveAnalytics 儲存在分割區集合中，因此在後續備份期間會複製因擷取新資料或更新現有資料而變更的所有分割區。

如果您使用 Timestream for LiveAnalytics 主控台，則為帳戶中所有資源建立的備份會列在備份索引標籤中。此外，備份也會列在資料表詳細資訊中。

還原

您可以從 LiveAnalytics 主控台的 Timestream 或 AWS Backup 主控台 SDK或 還原資料表 AWS CLI。您可以從備份還原整個資料，也可以設定資料表保留設定以還原選取的資料。啟動還原時，您可以設定下列資料表設定。

- Database Name (資料庫名稱)
- 資料表名稱
- 記憶體存放區保留
- 磁性存放區保留
- 啟用磁性儲存寫入
- S3 錯誤日誌位置 （選用）
- IAM 還原備份時 AWS Backup 將擔任的角色

上述組態獨立於來源資料表。若要還原備份中的所有資料，建議您設定新的資料表設定，讓記憶體存放區保留期和磁性存放區保留期的總和大於最舊時間戳記與現在之間的差異。當您選取要還原的增量備份時，所有資料 （增量 + 基礎完整資料） 都會還原。成功還原後，資料表處於作用中狀態，您可以在還原的資料表上執行擷取和/或查詢操作。不過，您無法在還原進行時執行這些操作。還原後，資料表與您帳戶中的任何其他資料表類似。

Example 從備份還原所有資料

此範例具有下列假設。

最早的時間戳記 —`August 1, 2021 0:00:00`

- 現在 —`November 9, 2022 0:00:00`

若要從備份還原所有資料，請輸入並比較值，如下所示。

1. 輸入記憶體存放區保留和 Magnetic 存放區保留 。例如，假設這些值。
   - 記憶體存放區保留 - 12 小時
   - 磁性存放區保留 — 500 天
2. 尋找記憶體存放區保留和 Magnetic 存放區保留 的總和。

```
12 hours + (500 * 24 hours) =
12 hours + 12,000 hours =
12,012 hours
```

3. 尋找最早時間戳記與現在之間的差異。

```
November 9, 2022 0:00:00 - August 1, 2021 0:00:00 =
465 days =
465 * 24 hours =
11,160 hours
```

4. 確保第二個步驟中的保留值總和大於第三個步驟中的時間差。視需要調整保留時間。

```
12,012 > 11,160
true
```

Example 從備份還原選取的資料

此範例具有下列假設。

- 現在 —November 9, 2022 0:00:00

若要僅還原從備份中選取的資料，請輸入並比較值，如下所示。

1. 判斷所需的最早時間戳記。例如，假設 December 4, 2021 0:00:00。
2. 尋找所需最早時間戳記與現在之間的差異。

```
November 9, 2022 0:00:00 - December 4, 2021 0:00:00 =
340 days =
340 * 24 hours =
8,160 hours
```

3. 輸入記憶體存放區保留所需的值 。例如，輸入 12 小時。
4. 從第二個步驟的差異中減去 值。

```
8,160 hours - 12 hours =
8148 hours
```

5. 輸入 Magnetic Store 保留 的值。

您可以將 Timestream 的 LiveAnalytics 資料表資料備份複製到不同 AWS 區域，然後在該新區域中還原。您可以在 AWS 商業區域和 AWS GovCloud （美國） 區域之間複製備份，然後還原備份。您只需為從來源區域中傳輸出來的資料，以及還原為目標區域中的新資料表付費。

資料表還原後，您必須在還原的資料表上手動設定下列項目。

- AWS Identity and Access Management （IAM） 政策
- 標籤
- 排程查詢

還原時間與資料表的組態直接相關。這包括資料表的大小、基礎分割區的數量、還原至記憶體存放區的資料量，以及其他變數。規劃災難復原時的最佳實務是定期記錄平均還原完成時間，並建立這些時間如何影響您的整體復原時間目標 （RTO）。

所有備份和還原主控台和API動作都會擷取並記錄在 中 AWS CloudTrail，以供記錄、持續監控和稽核。

建立 Amazon Timestream 資料表的備份

本節說明如何啟用 AWS Backup 和建立 Amazon Timestream 的隨需和排程備份。

主題

- 啟用 AWS Backup 來保護 LiveAnalytics 資料的時間串流
- 建立隨需備份
- 排程備份

啟用 AWS Backup 來保護 LiveAnalytics 資料的時間串流

您必須啟用 AWS Backup ，才能與 的 Timestream 搭配使用 LiveAnalytics。

若要在適用於 LiveAnalytics 主控台的 Timestream AWS Backup 中啟用 ，請執行下列步驟。

1. 登入 AWS 管理主控台 。
2. 快顯橫幅會顯示在 LiveAnalytics 儀表板 Timestream 頁面的頂端，以啟用 AWS Backup 以支援 LiveAnalytics 資料 Timestream。否則，從導覽窗格中，選擇 Backups 。
3. 在備份視窗中，您會看到要啟用 的橫幅 AWS Backup。選擇 啟用 。

   LiveAnalytics 資料表的 Timestream AWS Backup 現在可使用 到 的資料保護。

若要透過 啟用 AWS Backup，請參閱 AWS Backup 文件，透過主控台以程式設計方式啟用 。

如果您選擇在啟用 Timestream 後 AWS Backup 停用保護 LiveAnalytics 資料，請透過 AWS Backup 主控台登入並將切換移至左側。

如果您無法啟用或停用這些 AWS Backup 功能，您的 AWS 管理員可能需要執行這些動作。

建立隨需備份

若要為 LiveAnalytics 資料表建立 Timestream 的隨需備份，請遵循下列步驟。

1. 登入 AWS Management Console 。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 選擇 Create on-demand backup (建立隨需備份)。
4. 繼續選取備份視窗中的設定。
5. 您可以立即建立備份、立即啟動備份，或選取備份時段以開始備份。
6. 選取備份的生命週期管理政策。您可以將備份資料轉換為冷儲存，您必須在其中將備份保留至少 90 天。您可以設定備份所需的保留期 您可以選取現有的保存庫，或選取建立新的備份保存庫，以導覽至 AWS Backup 主控台並建立新的備份保存庫 <在此處建立新備份保存庫的文件連結 >
7. 選取適當的IAM角色。
8. 如果您要將一或多個標籤指派至您的隨需備份，請輸入 key (索引鍵) 和選用的 value (值)，然後選擇 Add tag (新增標籤)。
9. 選擇建立隨需備份。這將帶您前往備份頁面，您將在其中看到任務清單。
10. 選擇您選擇要備份之資源的 Backup job ID (備份任務 ID)，以查看該任務的詳細資料。

排程備份

若要排程備份，請參閱建立排程備份。

還原 Amazon Timestream 資料表的備份

本節說明如何還原 Amazon Timestream 資料表的備份。

主題

從 還原 LiveAnalytics 資料表的時間串流 AWS Backup

若要還原 LiveAnalytics 資料表的 Timestream，使其不再 AWS Backup 使用適用於 LiveAnalytics 主控台的 Timestream，請遵循下列步驟。

1. 登入 AWS 管理主控台 。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 若要還原資源，請選擇資源復原點 ID 旁的選項按鈕。在窗格右上角，選擇 Restore (還原)。
4. 輸入資料表組態設定，即資料庫名稱和資料表名稱 。請注意，還原的資料表名稱應與原始來源資料表名稱不同。
5. 設定記憶體和磁性存放區保留設定。
6. 針對還原角色 ，選擇此還原 AWS Backup 將擔任IAM的角色。
7. 選擇 Restore backup (還原備份)。頁面頂端的訊息提供還原任務的相關資訊。

> **Note**
>
> 無論設定的記憶體和磁性存放區保留期為何，您都必須支付還原整個備份的費用。不過，還原完成後，還原的資料表只會包含設定保留期間內的資料。

將 LiveAnalytics 資料表的時間串流還原至其他區域或帳戶

若要將 LiveAnalytics 資料表的時間串流還原至其他區域或帳戶，您必須先將備份複製到該新區域或帳戶。為了複製到另一個帳戶，該帳戶必須先授予您許可。複製要 LiveAnalytics 備份到新區域或帳戶的 Timestream 後，可以使用上一節中的程序還原。

複製 Amazon Timestream 資料表的備份

您可以製作現有備份的複製。您可以視需要將備份複製到多個 AWS 帳戶或 AWS 區域，或自動作為排程備份計畫的一部分。如果您有業務持續性或合規性要求，需要將備份儲存在與生產資料最短距離的位置，則跨區域複寫特別有用。

跨帳戶備份對於將備份安全地複製到組織內一個或多個 AWS 帳戶的運作或安全性考量相當有助益。如果不小心刪除您的原始備份，您可以將備份從其目的地帳戶複製到其來源帳戶，然後啟動還原。您必須先在 Organizations 服務中擁有屬於相同組織的兩個帳戶，以及帳戶所需的許可，才能執行此操作。當您將增量備份複製到另一個帳戶或區域時，也會複製相關聯的完整備份。

除非您另有指定，否則副本會繼承來源備份的組態。有一種例外狀況。如果您將新複本指定為「永不」過期。使用此設定，新複本仍會繼承其來源到期日。如果您希望新備份副本是永久性的，請將來源備份設定為永不過期，或指定新副本在建立後 100 年過期。

若要從 Timestream 主控台複製備份，請遵循下列步驟。

1. 登入 AWS 管理主控台 。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 選擇資源復原點 ID 旁的選項按鈕。在窗格的右上角，選取動作，然後選擇複製 。
4. 選取繼續 AWS 備份，然後遵循跨帳戶備份 的步驟。

Timestream 目前不支援將隨需和排程備份複製到 LiveAnalytics 主控台的帳戶和區域，您必須導覽至 AWS Backup 才能執行操作。

刪除備份

本節說明如何刪除 LiveAnalytics 資料表的時間串流備份。

若要從 Timestream 主控台刪除備份，請遵循下列步驟。

1. 登入 AWS 管理主控台 。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 選擇資源復原點 ID 旁的選項按鈕。在窗格的右上角，選取動作，然後選擇刪除 。
4. 選取繼續 AWS 備份，然後依照刪除備份 的步驟刪除備份。

> **Note**
>
> 當您刪除增量備份時，只會刪除增量備份，也不會刪除基礎完整備份。

配額和限制

AWS Backup 會將備份限制為每個資源一個並行備份。因此，資源的其他排程或隨需備份請求會排入佇列，且只會在現有備份任務完成後啟動。如果備份任務未在備份時段內啟動或完成，請求會失敗。如需 AWS Backup 限制的詳細資訊，請參閱 AWS 備份開發人員指南中的備份限制。 AWS

建立備份時，每個帳戶最多可以執行四個並行備份。同樣地，您可以為每個帳戶執行一次並行還原。當您同時啟動四個以上的備份任務時，只會啟動四個備份任務，剩餘的任務也會定期重試。啟動後，如果備份任務未在設定的備份時段持續時間內完成，則備份任務會失敗。如果失敗的備份任務是隨需備份，您可以重試備份，而對於排程備份，則會在下列排程中嘗試任務。

客戶定義的分割金鑰

適用於 LiveAnalytics 客戶定義分割金鑰的 Amazon Timestream 是 Timestream 中的功能 LiveAnalytics ，可讓客戶為其資料表定義自己的分割金鑰。分割是一種技術，用於在多個實體儲存單元之間分發資料，從而實現更快、更有效率的資料擷取。透過客戶定義的分割區金鑰，客戶可以建立分割區結構描述，以更符合其查詢模式和使用案例。

使用適用於 LiveAnalytics 客戶定義分割金鑰的 Timestream，客戶可以選擇一個維度名稱作為資料表的分割金鑰。這可讓您更靈活地定義其資料的分割結構描述。透過選取正確的分割區金鑰，客戶可以最佳化其資料模型、改善查詢效能，並減少查詢延遲。

主題

- 使用自訂複合分割區金鑰進行 LiveAnalytics 結構描述驗證的時間串流

使用客戶定義的分割金鑰

如果您有定義良好的查詢模式，具有高基數維度，且需要低查詢延遲， LiveAnalytics 則客戶定義分割區金鑰的 Timestream 可能是增強資料模型的有用工具。例如，如果您是追蹤網站上的客戶互動的零售公司，則主要存取模式可能會是依客戶 ID 和時間戳記。透過將客戶 ID 定義為分割區金鑰，您的資料可以平均分佈，從而減少延遲，最終改善使用者體驗。

另一個範例是醫療保健產業，其中可穿戴裝置會收集感應器資料，以追蹤患者的生命徵象。主要存取模式會依裝置 ID 和時間戳記，在兩個維度上都有高基數。透過將裝置 ID 定義為分割區金鑰， 可以最佳化查詢執行，並確保持續的長期查詢效能。

總而言之，當您具有清晰的查詢模式、高基數維度，且查詢需要低延遲時， LiveAnalytics 客戶定義的分割區金鑰的 Timestream 最有用。透過定義與您的查詢模式一致的分割區金鑰，您可以最佳化查詢執行，並確保持續的長期效能查詢效能。

開始使用客戶定義的分割金鑰

從主控台中，選擇資料表並建立新資料表。您也可以使用 SDK存取`CreateTable`動作，以建立新的資料表，其中包含客戶定義的分割區金鑰。

使用維度類型分割區索引鍵建立資料表

您可以使用下列程式碼片段建立具有維度類型分割區金鑰的資料表。

Java

```
public void createTableWithDimensionTypePartitionKeyExample() {
        System.out.println("Creating table");
        CreateTableRequest createTableRequest = new CreateTableRequest();
        createTableRequest.setDatabaseName(DATABASE_NAME);
        createTableRequest.setTableName(TABLE_NAME);
        final RetentionProperties retentionProperties = new RetentionProperties()
                .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);
        createTableRequest.setRetentionProperties(retentionProperties);

        // Can specify enforcement level with OPTIONAL or REQUIRED
```

```
        final List<PartitionKey> partitionKeyWithDimensionAndOptionalEnforcement =
 Collections.singletonList(new PartitionKey()
            .withName(COMPOSITE_PARTITION_KEY_DIM_NAME)
            .withType(PartitionKeyType.DIMENSION)
            .withEnforcementInRecord(PartitionKeyEnforcementLevel.OPTIONAL));
        Schema schema = new Schema();

 schema.setCompositePartitionKey(partitionKeyWithDimensionAndOptionalEnforcement);
        createTableRequest.setSchema(schema);

        try {
            writeClient.createTable(createTableRequest);
            System.out.println("Table [" + TABLE_NAME + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
 DATABASE_NAME + "] . Skipping database creation");
        }
    }
```

Java v2

```
public void createTableWithDimensionTypePartitionKeyExample() {
        System.out.println("Creating table");
        final RetentionProperties retentionProperties =
 RetentionProperties.builder()
                .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS)
                .build();
        // Can specify enforcement level with OPTIONAL or REQUIRED
        final List<PartitionKey> partitionKeyWithDimensionAndOptionalEnforcement =
 Collections.singletonList(PartitionKey
                .builder()
                .name(COMPOSITE_PARTITION_KEY_DIM_NAME)
                .type(PartitionKeyType.DIMENSION)
                .enforcementInRecord(PartitionKeyEnforcementLevel.OPTIONAL)
                .build());
        final Schema schema = Schema.builder()

 .compositePartitionKey(partitionKeyWithDimensionAndOptionalEnforcement).build();
        final CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .databaseName(DATABASE_NAME)
                .tableName(TABLE_NAME)
                .retentionProperties(retentionProperties)
```

```
                .schema(schema)
                .build();

        try {
            writeClient.createTable(createTableRequest);
            System.out.println("Table [" + TABLE_NAME + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
 DATABASE_NAME + "] . Skipping database creation");
        }
    }
```

Go v1

```
func createTableWithDimensionTypePartitionKeyExample(){
        // Can specify enforcement level with OPTIONAL or REQUIRED
        partitionKeyWithDimensionAndOptionalEnforcement :=
 []*timestreamwrite.PartitionKey{
                {
                    Name:                aws.String(CompositePartitionKeyDimName),
                    EnforcementInRecord: aws.String("OPTIONAL"),
                    Type:                aws.String("DIMENSION"),
                },
        }
        createTableInput := &timestreamwrite.CreateTableInput{
             DatabaseName: aws.String(*databaseName),
             TableName:    aws.String(*tableName),
             // Enable MagneticStoreWrite for Table
             MagneticStoreWriteProperties:
 &timestreamwrite.MagneticStoreWriteProperties{
                 EnableMagneticStoreWrites: aws.Bool(true),
                 // Persist MagneticStoreWrite rejected records in S3
                 MagneticStoreRejectedDataLocation:
 &timestreamwrite.MagneticStoreRejectedDataLocation{
                     S3Configuration: &timestreamwrite.S3Configuration{
                         BucketName:       aws.String("timestream-sample-bucket"),
                         ObjectKeyPrefix:  aws.String("TimeStreamCustomerSampleGo"),
                         EncryptionOption: aws.String("SSE_S3"),
                     },
                 },
             },
             Schema: &timestreamwrite.Schema{
```

```
                CompositePartitionKey:
partitionKeyWithDimensionAndOptionalEnforcement,
            }
        }
        _, err := writeSvc.CreateTable(createTableInput)
    }
```

Go v2

```
 func (timestreamBuilder TimestreamBuilder)
CreateTableWithDimensionTypePartitionKeyExample() error {
        partitionKeyWithDimensionAndOptionalEnforcement := []types.PartitionKey{
            {
                Name:                aws.String(CompositePartitionKeyDimName),
                EnforcementInRecord: types.PartitionKeyEnforcementLevelOptional,
                Type:                types.PartitionKeyTypeDimension,
            },
        }
        _, err := timestreamBuilder.WriteSvc.CreateTable(context.TODO(),
&timestreamwrite.CreateTableInput{
            DatabaseName: aws.String(databaseName),
            TableName:    aws.String(tableName),
            MagneticStoreWriteProperties: &types.MagneticStoreWriteProperties{
                EnableMagneticStoreWrites: aws.Bool(true),
                // Persist MagneticStoreWrite rejected records in S3
                MagneticStoreRejectedDataLocation:
&types.MagneticStoreRejectedDataLocation{
                    S3Configuration: &types.S3Configuration{
                        BucketName:       aws.String(s3BucketName),
                        EncryptionOption: "SSE_S3",
                    },
                },
            },
            Schema: &types.Schema{
                CompositePartitionKey:
partitionKeyWithDimensionAndOptionalEnforcement,
            },
        })

        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
```

```
             fmt.Println("Create table is successful")
         }
         return err
     }
```

Python

```
def create_table_with_measure_name_type_partition_key(self):
        print("Creating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': CT_TTL_DAYS
        }
        partitionKey_with_measure_name = [
            {'Type': 'MEASURE'}
        ]
        schema = {
            'CompositePartitionKey': partitionKey_with_measure_name
        }
        try:
            self.client.create_table(DatabaseName=DATABASE_NAME,
 TableName=TABLE_NAME,
                                     RetentionProperties=retention_properties,
 Schema=schema)
            print("Table [%s] successfully created." % TABLE_NAME)
        except self.client.exceptions.ConflictException:
            print("Table [%s] exists on database [%s]. Skipping table creation" % (
                TABLE_NAME, DATABASE_NAME))
        except Exception as err:
            print("Create table failed:", err)
```

檢查分割結構描述組態

您可以用幾種方式檢查分割結構描述的資料表組態。從主控台中，選擇資料庫，然後選擇要檢查的資料表。您也可以使用 SDK來存取DescribeTable動作。

描述具有分割區金鑰的資料表

您可以使用下列程式碼片段來描述具有分割區金鑰的資料表。

Java

```
   public void describeTable() {
       System.out.println("Describing table");
       final DescribeTableRequest describeTableRequest = new
DescribeTableRequest();
       describeTableRequest.setDatabaseName(DATABASE_NAME);
       describeTableRequest.setTableName(TABLE_NAME);
       try {
           DescribeTableResult result =
amazonTimestreamWrite.describeTable(describeTableRequest);
           String tableId = result.getTable().getArn();
           System.out.println("Table " + TABLE_NAME + " has id " + tableId);
           // If table is created with composite partition key, it can be described
with
           //
System.out.println(result.getTable().getSchema().getCompositePartitionKey());
       } catch (final Exception e) {
           System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
           throw e;
       }
   }
```

以下為範例輸出。

1. 資料表具有維度類型分割區索引鍵

```
[{Type: DIMENSION,Name: hostId,EnforcementInRecord: OPTIONAL}]
```

2. 資料表具有量值名稱類型分割區索引鍵

```
[{Type: MEASURE,}]
```

3. 從建立的資料表取得複合分割區金鑰，而不指定複合分割區金鑰

```
[{Type: MEASURE,}]
```

Java v2

```
 public void describeTable() {
       System.out.println("Describing table");
```

```
        final DescribeTableRequest describeTableRequest =
DescribeTableRequest.builder()
                .databaseName(DATABASE_NAME).tableName(TABLE_NAME).build();
        try {
            DescribeTableResponse response =
writeClient.describeTable(describeTableRequest);
            String tableId = response.table().arn();
            System.out.println("Table " + TABLE_NAME + " has id " + tableId);
            // If table is created with composite partition key, it can be described
with
            //
System.out.println(response.table().schema().compositePartitionKey());
        } catch (final Exception e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        }
    }
```

以下為範例輸出。

1. 資料表具有維度類型分割區索引鍵

```
[PartitionKey(Type=DIMENSION, Name=hostId, EnforcementInRecord=OPTIONAL)]
```

2. 資料表具有量值名稱類型分割區索引鍵

```
[PartitionKey(Type=MEASURE)]
```

3. 從建立的資料表取得複合分割區金鑰，而不指定複合分割區金鑰將會傳回

```
[PartitionKey(Type=MEASURE)]
```

Go v1

```
   <tablistentry>
    <tabname> Go </tabname>
    <tabcontent>
     <programlisting language="go"></programlisting>
    </tabcontent>
   </tablistentry>
```

以下為範例輸出。

```
{
  Table: {
    Arn: "arn:aws:timestream:us-west-2:533139590831:database/devops/table/
host_metrics_dim_pk_1",
    CreationTime: 2023-05-31 01:52:00.511 +0000 UTC,
    DatabaseName: "devops",
    LastUpdatedTime: 2023-05-31 01:52:00.511 +0000 UTC,
    MagneticStoreWriteProperties: {
      EnableMagneticStoreWrites: true,
      MagneticStoreRejectedDataLocation: {
        S3Configuration: {
          BucketName: "timestream-sample-bucket-west",
          EncryptionOption: "SSE_S3",
          ObjectKeyPrefix: "TimeStreamCustomerSampleGo"
        }
      }
    },
    RetentionProperties: {
      MagneticStoreRetentionPeriodInDays: 73000,
      MemoryStoreRetentionPeriodInHours: 6
    },
    Schema: {
      CompositePartitionKey: [{
          EnforcementInRecord: "OPTIONAL",
          Name: "hostId",
          Type: "DIMENSION"
        }]
    },
    TableName: "host_metrics_dim_pk_1",
    TableStatus: "ACTIVE"
  }
}
```

Go v2

```
 func (timestreamBuilder TimestreamBuilder) DescribeTable()
 (*timestreamwrite.DescribeTableOutput, error) {
         describeTableInput := &timestreamwrite.DescribeTableInput{
             DatabaseName: aws.String(databaseName),
             TableName:    aws.String(tableName),
         }
```

```
        describeTableOutput, err :=
 timestreamBuilder.WriteSvc.DescribeTable(context.TODO(), describeTableInput)

        if err != nil {
            fmt.Printf("Failed to describe table with Error: %s", err.Error())
        } else {
            fmt.Printf("Describe table is successful : %s\n",
 JsonMarshalIgnoreError(*describeTableOutput))
            // If table is created with composite partition key, it will be included
 in the output
        }

        return describeTableOutput, err
    }
```

以下為範例輸出。

```
{
  "Table": {
    "Arn":"arn:aws:timestream:us-east-1:351861611069:database/cdpk-wr-db/table/
host_metrics_dim_pk",
    "CreationTime":"2023-05-31T22:36:10.66Z",
    "DatabaseName":"cdpk-wr-db",
    "LastUpdatedTime":"2023-05-31T22:36:10.66Z",
    "MagneticStoreWriteProperties":{
      "EnableMagneticStoreWrites":true,
      "MagneticStoreRejectedDataLocation":{
        "S3Configuration":{
          "BucketName":"error-configuration-sample-s3-bucket-cq8my",
          "EncryptionOption":"SSE_S3",
          "KmsKeyId":null,"ObjectKeyPrefix":null
        }
      }
    },
    "RetentionProperties":{
      "MagneticStoreRetentionPeriodInDays":73000,
      "MemoryStoreRetentionPeriodInHours":6
    },
    "Schema":{
      "CompositePartitionKey":[{
        "Type":"DIMENSION",
        "EnforcementInRecord":"OPTIONAL",
        "Name":"hostId"
```

```
      }]
    },
    "TableName":"host_metrics_dim_pk",
    "TableStatus":"ACTIVE"
  },
  "ResultMetadata":{}
}
```

Python

```
 def describe_table(self):
       print('Describing table')
       try:
           result = self.client.describe_table(DatabaseName=DATABASE_NAME,
TableName=TABLE_NAME)
           print("Table [%s] has id [%s]" % (TABLE_NAME, result['Table']['Arn']))
           # If table is created with composite partition key, it can be described
with
           # print(result['Table']['Schema'])
       except self.client.exceptions.ResourceNotFoundException:
           print("Table doesn't exist")
       except Exception as err:
           print("Describe table failed:", err)
```

以下為範例輸出。

1. 資料表具有維度類型分割區索引鍵

```
[{'CompositePartitionKey': [{'Type': 'DIMENSION', 'Name': 'hostId',
 'EnforcementInRecord': 'OPTIONAL'}]}]
```

2. 資料表具有量值名稱類型分割區索引鍵

```
[{'CompositePartitionKey': [{'Type': 'MEASURE'}]}]
```

3. 從建立的資料表取得複合分割區金鑰，而不指定複合分割區金鑰

```
[{'CompositePartitionKey': [{'Type': 'MEASURE'}]}]
```

更新分割結構描述組態

您可以使用SDK可存取UpdateTable動作的 更新分割結構描述的資料表組態。

使用分割區金鑰更新資料表

您可以使用下列程式碼片段來更新具有分割區金鑰的資料表。

Java

```
   public void updateTableCompositePartitionKeyEnforcement() {
       System.out.println("Updating table");

       UpdateTableRequest updateTableRequest = new UpdateTableRequest();
       updateTableRequest.setDatabaseName(DATABASE_NAME);
       updateTableRequest.setTableName(TABLE_NAME);

       // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
       final List<PartitionKey> partitionKeyWithDimensionAndRequiredEnforcement =
Collections.singletonList(new PartitionKey()
           .withName(COMPOSITE_PARTITION_KEY_DIM_NAME)
           .withType(PartitionKeyType.DIMENSION)
           .withEnforcementInRecord(PartitionKeyEnforcementLevel.REQUIRED));
       Schema schema = new Schema();

schema.setCompositePartitionKey(partitionKeyWithDimensionAndRequiredEnforcement);
       updateTableRequest.withSchema(schema);

       writeClient.updateTable(updateTableRequest);
       System.out.println("Table updated");
```

Java v2

```
   public void updateTableCompositePartitionKeyEnforcement() {
       System.out.println("Updating table");
       // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
       final List<PartitionKey> partitionKeyWithDimensionAndRequiredEnforcement =
Collections.singletonList(PartitionKey
           .builder()
           .name(COMPOSITE_PARTITION_KEY_DIM_NAME)
           .type(PartitionKeyType.DIMENSION)
```

```
            .enforcementInRecord(PartitionKeyEnforcementLevel.REQUIRED)
            .build());
        final Schema schema = Schema.builder()

.compositePartitionKey(partitionKeyWithDimensionAndRequiredEnforcement).build();
        final UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()

.databaseName(DATABASE_NAME).tableName(TABLE_NAME).schema(schema).build();

        writeClient.updateTable(updateTableRequest);
        System.out.println("Table updated");
```

Go v1

```
// Update table partition key enforcement attribute
   updateTableInput := &timestreamwrite.UpdateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
        // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
        Schema: &timestreamwrite.Schema{
            CompositePartitionKey: []*timestreamwrite.PartitionKey{
                {
                        Name:
aws.String(CompositePartitionKeyDimName),
                        EnforcementInRecord: aws.String("REQUIRED"),
                        Type:                aws.String("DIMENSION"),
                },
            }},
    }
    updateTableOutput, err := writeSvc.UpdateTable(updateTableInput)
        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
            fmt.Println("Update table is successful, below is the output:")
            fmt.Println(updateTableOutput)
        }
```

Go v2

```
// Update table partition key enforcement attribute
        updateTableInput := &timestreamwrite.UpdateTableInput{
```

```
            DatabaseName: aws.String(*databaseName),
            TableName:    aws.String(*tableName),
            // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
            Schema: &types.Schema{
                CompositePartitionKey: []types.PartitionKey{
                    {
                        Name:
aws.String(CompositePartitionKeyDimName),
                        EnforcementInRecord:
types.PartitionKeyEnforcementLevelRequired,
                        Type:               types.PartitionKeyTypeDimension,
                    },
                }},
        }
        updateTableOutput, err :=
timestreamBuilder.WriteSvc.UpdateTable(context.TODO(), updateTableInput)
        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
            fmt.Println("Update table is successful, below is the output:")
            fmt.Println(updateTableOutput)
        }
```

Python

```
   def update_table(self):
       print('Updating table')
       try:
           # Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
           partition_key_with_dimension_and_required_enforcement = [
               {
                   'Type': 'DIMENSION',
                   'Name': COMPOSITE_PARTITION_KEY_DIM_NAME,
                   'EnforcementInRecord': 'REQUIRED'
               }
           ]
           schema = {
               'CompositePartitionKey':
partition_key_with_dimension_and_required_enforcement
           }
```

```
            self.client.update_table(DatabaseName=DATABASE_NAME,
TableName=TABLE_NAME,
                                     Schema=schema)
            print('Table updated.')
        except Exception as err:
            print('Update table failed:', err)
```

客戶定義分割金鑰的優點

增強的查詢效能：客戶定義的分割金鑰可讓您最佳化查詢執行，並改善整體查詢效能。透過定義與您的查詢模式相符的分割區金鑰，您可以將資料掃描降至最低並最佳化資料刪除，進而降低查詢延遲。

更好的長期效能可預測性：客戶定義的分割區金鑰可讓客戶平均分散跨分割區的資料，進而提升資料管理的效率。這將確保查詢效能在資料儲存規模隨時間擴展時保持穩定。

客戶定義分割金鑰的限制

身為 LiveAnalytics 使用者的 Timestream，請務必記住客戶分割區金鑰的相關限制。首先，它需要充分了解您的工作負載和查詢模式。這表示您應該清楚了解哪些維度最常用作查詢中的主要篩選條件，並具有高基數，以最有效地使用分割區金鑰。

其次，需要在建立資料表時定義分割區金鑰，且無法新增至現有資料表。這表示在建立資料表之前，您應該仔細考慮分割策略，以確保其符合您業務需求。

最後，請務必注意，一旦建立資料表，之後就無法變更分割區金鑰。這表示您應該在承諾分割策略之前，先徹底測試和評估分割策略。考慮到這些限制，Timestream 的客戶定義分割區金鑰可以大幅提升查詢效能和長期滿意度。

客戶定義的分割金鑰和低基數維度

如果您決定使用具有極低基數的分割區索引鍵，例如特定區域或狀態，請務必注意，其他實體的資料，例如 `customerID`、 `ProductCategory`和其他實體，有時可能會分散到太多分割區，但資料很少或不存在。這可能會導致查詢執行效率低下和效能降低。

為了避免這種情況，我們建議您選擇不僅屬於金鑰篩選條件的維度，而且具有更高的基數。這將有助於確保資料平均分佈在分割區中，並改善查詢效能。

為現有資料表建立分割區金鑰

如果您已有適用於 的 Timestream 資料表， LiveAnalytics 且想要使用客戶定義的分割金鑰，則需要將資料遷移至具有所需分割結構描述定義的新資料表。這可以使用匯出至 S3 和批次載入一起完成，這涉及將資料從現有資料表匯出至 S3、修改資料以包含分割區金鑰 （如有必要），以及將標頭新增至CSV檔案，然後將資料匯入已定義所需分割區結構描述的新資料表。請記住，這種方法可能耗時且昂貴，特別是對於大型資料表。

或者，您可以使用排程查詢，將資料遷移至具有所需分割結構描述的新資料表。此方法涉及建立從現有資料表讀取並寫入新資料表的排程查詢。排程查詢可以設定為定期執行，直到所有資料遷移完畢為止。請記住，在遷移過程中讀取和寫入資料需要支付費用。

使用自訂複合分割區金鑰進行 LiveAnalytics 結構描述驗證的時間串流

的 Timestream 中的結構描述驗證 LiveAnalytics 有助於確保擷取至資料庫的資料符合指定的結構描述、盡可能減少擷取錯誤並改善資料品質。特別是，在採用客戶定義的分割區金鑰時，結構描述驗證特別有用，目標是最佳化查詢效能。

什麼是使用客戶定義分割金鑰進行 LiveAnalytics 結構描述驗證的 Timestream？

LiveAnalytics 結構描述驗證的 Timestream 功能，會根據預先定義的結構描述驗證擷取至資料表之 LiveAnalytics Timestream 中的資料。此結構描述定義資料模型，包括要插入之記錄的分割區金鑰、資料類型和限制條件。

使用客戶定義的分割區金鑰時，結構描述驗證變得更加重要。分割區金鑰可讓您指定分割區金鑰，這可決定資料在 Timestream 中存放的方式 LiveAnalytics。透過使用自訂分割區金鑰針對結構描述驗證傳入資料，您可以強制執行資料一致性、及早偵測錯誤，以及改善 的 Timestream 中存放資料的整體品質 LiveAnalytics。

如何使用 Timestream 搭配自訂複合分割區金鑰進行 LiveAnalytics 結構描述驗證

若要使用 Timestream 搭配自訂複合分割區金鑰進行 LiveAnalytics 結構描述驗證，請遵循下列步驟：

思考您的查詢模式會是什麼樣子：若要正確選擇和定義 LiveAnalytics 資料表的時間串流結構描述，您應該從查詢需求開始。

指定自訂複合分割區金鑰：建立資料表時，請指定自訂分割區金鑰。此金鑰決定將用於分割資料表資料的屬性。您可以選擇維度鍵和量值鍵進行分割。維度索引鍵會根據維度名稱分割資料，而 會根據量值名稱測量索引鍵分割區資料。

設定強制執行層級：為了確保適當的資料分割及其附帶的優點，適用於 的 Amazon Timestream LiveAnalytics 可讓您為結構描述中的每個分割區金鑰設定強制執行層級。強制執行層級會決定擷取記錄時是否需要或選用分割區金鑰維度。您可以選擇兩個選項：REQUIRED，表示擷取的記錄中必須存在分割區金鑰，而 OPTIONAL則表示不需要存在分割區金鑰。建議您在使用客戶定義的分割區時，使用REQUIRED強制執行層級，以確保資料已正確分割，並取得此功能的完整優點。此外，您可以在建立結構描述之後隨時變更強制執行層級組態，以根據資料擷取需求進行調整。

擷取資料：將資料擷取至資料表的時間串流 LiveAnalytics時，結構描述驗證程序會使用自訂複合分割區金鑰，根據定義的結構描述檢查記錄。如果記錄未遵循結構描述， 的 Timestream LiveAnalytics 將傳回驗證錯誤。

處理驗證錯誤：如果發生驗證錯誤， 的 Timestream LiveAnalytics 會根據錯誤RejectedRecordsException類型傳回 ValidationException或 。請務必在應用程式中處理這些例外狀況，並採取適當動作，例如修正不正確的記錄並重試擷取。

更新強制執行層級：如有必要，您可以使用 UpdateTable動作在建立資料表後更新分割區金鑰的強制執行層級。不過，請務必注意，分割區金鑰組態的某些方面，例如名稱和類型，無法在建立資料表後變更。如果您將強制執行層級從 變更為 REQUIRED OPTIONAL，則無論選擇作為客戶定義分割金鑰的屬性是否存在，都會接受所有記錄。相反地，如果您將強制執行層級從 變更為 OPTIONAL REQUIRED，您可能會開始看到不符合此條件之記錄的 4xx 個寫入錯誤。因此，在建立資料表時，請務必根據資料的分割需求，為使用案例選擇適當的強制執行層級。

何時使用 Timestream 搭配自訂複合分割區金鑰進行 LiveAnalytics 結構描述驗證

在資料一致性、品質和最佳化分割至關重要的情況下，應使用具有自訂複合分割區金鑰的 LiveAnalytics 結構描述驗證時間串流。透過在資料擷取期間強制執行結構描述，您可以防止可能導致分析不正確或遺失寶貴洞見的錯誤和不一致。

與批次載入任務的互動

設定批次載入任務以使用客戶定義的分割區金鑰將資料匯入資料表時，有一些案例可能會影響程序：

1. 如果強制執行層級設定為 OPTIONAL，則如果分割區金鑰未在任務組態期間映射，則在建立流程中，主控台上會顯示警示。使用 API或 時，不會出現此提醒CLI。
2. 如果強制執行層級設定為 REQUIRED，則除非分割區金鑰對應至來源資料欄，否則任務建立會遭到拒絕。
3. 如果在建立任務REQUIRED之後將強制執行層級變更為 ，任務將繼續執行，但任何沒有正確對應分割區金鑰的記錄都會遭到拒絕，並出現 4xx 錯誤。

與排程查詢的互動

設定排程查詢任務以計算和儲存彙總、彙總和其他形式的預先處理資料至具有客戶定義分割金鑰的資料表時，有一些案例可能會影響程序：

1. 如果強制執行層級設定為 OPTIONAL，則在任務組態期間未對應分割區金鑰時，會顯示警示。使用API或 時，不會出現此提醒CLI。
2. 如果強制執行層級設定為 REQUIRED，則除非分割區金鑰對應至來源資料欄，否則任務建立會遭到拒絕。
3. 如果在建立任務REQUIRED之後將強制執行層級變更為 ，且排程的查詢結果不包含分割區金鑰維度，則任務的所有下一個迭代都會失敗。

將標籤新增至資源

您可以使用標籤 為 LiveAnalytics 資源標記 Amazon Timestream。標籤可讓您以不同方式分類資源，例如，依用途、擁有者、環境或其他條件。標籤可協助您執行以下作業：

- 根據您指派給資源的標籤來快速識別資源。
- 請參閱依標籤細分的 AWS 帳單。

Amazon Elastic Compute Cloud （Amazon EC2）、Amazon Simple Storage Service （Amazon S3）LiveAnalytics、適用於 的 Timestream 等 AWS 服務支援標記。有效標記可讓您跨帶有特定標籤的服務來建立報告，以提供成本深入資訊。

若要開始使用標記，請執行以下操作：

1. 了解標記限制。
2. 使用標記操作 建立標籤。

最後，最好遵循最佳標記策略。如需資訊，請參閱 AWS 標記策略。

標記限制

每個標記皆包含由您定義的索引鍵和值。將適用以下限制：

- LiveAnalytics 資料表的每個 Timestream 只能有一個具有相同金鑰的標籤。如果您嘗試新增現有的標籤，現有的標籤值會更新為新的值。

- 值充當標籤類別內的描述符。在 Timestream LiveAnalytics 中， 值不能為空白或 null。
- 標籤鍵與值皆區分大小寫。
- 鍵長度上限為 128 個 Unicode 字元。
- 值長度上限為 256 個 Unicode 字元。
- 允許的字元為字母、空格和數字，以及下列特殊字元：`+ - = . _ : /`
- 每一資源標籤數最多為 50。
- AWS- 指派的標籤名稱和值會自動指派前`aws:`綴，您無法指派。AWS- 指派的標籤名稱不會計入 50 的標籤限制。使用者指派的標籤名稱在成本分配報告中具有字首 `user:`。
- 標籤的套用不可回溯。

標記操作

您可以使用適用於 LiveAnalytics主控台、查詢語言或 AWS Command Line Interface （ ） 的 Amazon Timestream 新增、列出、編輯或刪除資料庫和資料表的標籤AWS CLI。

主題

- 使用主控台將標籤新增至新的或現有的資料庫和資料表

使用主控台將標籤新增至新的或現有的資料庫和資料表

您可以在建立新資料庫、資料表和排程查詢時，使用適用於 LiveAnalytics 主控台的 Timestream 將標籤新增至新資料庫、資料表和排程查詢。您也可以新增、編輯或刪除現有資料表的標籤。

在建立資料庫時標記資料庫 （主控台）

1. 在 https://console.aws.amazon.com/timestream 開啟 Timestream 主控台。
2. 在導覽窗格中，選擇資料庫 ，然後選擇建立資料庫 。
3. 在建立資料庫頁面上，提供資料庫的名稱。輸入標籤的索引鍵和值，然後選擇新增標籤 。
4. 選擇建立資料庫。

在建立資料表時標記資料表 （主控台）

1. 在 https://console.aws.amazon.com/timestream 開啟 Timestream 主控台。
2. 在導覽窗格中，選擇 Tables (資料表)，然後選擇 Create table (建立資料表)。

3. 在建立 LiveAnalytics 資料表的時間串流頁面上，提供資料表的名稱。輸入標籤的索引鍵和值，然後選擇新增標籤 。
4. 選擇 建立資料表 。

在建立排程查詢時加上標籤 （主控台）

1. 在 https://console.aws.amazon.com/timestream 開啟 Timestream 主控台。
2. 在導覽窗格中，選擇排程查詢 ，然後選擇建立排程查詢 。
3. 在步驟 3 上。設定查詢設定頁面，選擇新增標籤 。為標籤輸入金鑰和值。選擇新增標籤以新增其他標籤。
4. 選擇 Next (下一步)。

為現有的資源加上標籤 (主控台)

1. 在 https://console.aws.amazon.com/timestream 開啟 Timestream 主控台。
2. 在導覽窗格中，選擇資料庫 、資料表 或排程查詢 。
3. 選擇清單中的資料庫或資料表。然後選擇管理標籤以新增、編輯或刪除您的標籤。

如需標籤結構的相關資訊，請參閱 標記限制。

Timestream 中的安全性 LiveAnalytics

的雲端安全 AWS 是最高優先順序。身為 AWS 客戶，您受益於為滿足最安全敏感組織的需求而建置的資料中心和網路架構。

安全性是 AWS 和 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 也提供您可以安全使用的服務。第三方稽核人員定期檢測及驗證安全的效率也是我們 AWS 合規計劃的一部分。若要了解適用於 的 Timestream 的合規計劃 LiveAnalytics，請參閱AWS 依合規計劃範圍中的服務。
- 雲端安全性 – 您的責任取決於您使用 AWS 的服務。您也必須對其他因素負責，包括資料的敏感度、您組織的需求和適用的法律及法規。

本文件將協助您了解如何在使用 Timestream for 時套用共同責任模型 LiveAnalytics。下列主題說明如何設定 的 Timestream LiveAnalytics ，以符合您的安全和合規目標。您也將了解如何使用其他服務 AWS ，以協助您監控和保護 Timestream LiveAnalytics 的資源。

主題

Timestream 中的資料保護 LiveAnalytics

AWS 共同責任模型適用於 Amazon Timestream Live Analytics 中的資料保護。如本模型所述， AWS 負責保護執行所有 的全域基礎設施 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 AWS 服務 的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱資料隱私權。 FAQ如需歐洲資料保護的相關資訊，請參閱AWS 安全部落格 上的AWS 共同責任模型和GDPR部落格文章。

為了資料保護目的，我們建議您保護 AWS 帳戶 憑證，並使用 AWS IAM Identity Center 或 AWS Identity and Access Management （ ） 設定個別使用者IAM。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 對每個帳戶使用多重要素驗證 （ MFA ）。
- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 和建議 TLS 1.3。
- 使用 設定 API和使用者活動日誌 AWS CloudTrail。如需使用 CloudTrail 線索擷取 AWS 活動的資訊，請參閱 AWS CloudTrail 使用者指南 中的使用 CloudTrail 線索。
- 使用 AWS 加密解決方案，以及 中的所有預設安全控制項 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在 AWS 透過命令列介面或 FIPS 存取 時需要 140-3 個經過驗證的密碼編譯模組API，請使用FIPS端點。如需可用FIPS端點的詳細資訊，請參閱聯邦資訊處理標準 （ FIPS ） 140-3。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用 Timestream Live Analytics 或其他 AWS 服務 使用主控台API AWS CLI、 或 時 AWS SDKs。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您將 URL提供給外部伺服器，強烈建議您在 中不要包含憑證資訊，URL以驗證您對該伺服器的請求。

如需 Timestream 資料保護 LiveAnalytics 主題的詳細資訊，例如靜態加密和金鑰管理，請選取下列任何可用的主題。

主題

靜態加密

靜態 LiveAnalytics 加密的時間串流透過使用 AWS Key Management Service （ AWS KMS ） 中存放的加密金鑰加密您靜態所有資料，提供增強的安全性。此功能協助降低了保護敏感資料所涉及的操作負擔和複雜性。您可以透過靜態加密，建立符合嚴格加密合規和法規要求，而且對安全性要求甚高的應用程式。

- 在 LiveAnalytics 資料庫的 Timestream 上，預設會開啟加密，且無法關閉。業界標準 AES-256 加密演算法是使用的預設加密演算法。
- AWS KMS 在 的 Timestream 中，靜態加密需要 LiveAnalytics。
- 您無法僅加密資料表中的項目子集。
- 您不需要修改資料庫用戶端應用程式即可使用加密。

如果您沒有提供金鑰， 的 Timestream 會 LiveAnalytics 建立並使用`alias/aws/timestream`帳戶中名為 的 AWS KMS 金鑰。

您可以在 中使用自己的客戶受管金鑰KMS來加密 Timestream 中的 LiveAnalytics 資料。如需 Timestream for 中金鑰的詳細資訊 LiveAnalytics，請參閱 金鑰管理。

將資料 LiveAnalytics 儲存在兩個儲存層、記憶體存放區和磁性存放區的時間串流。使用 Timestream for LiveAnalytics Service 金鑰加密記憶體存放區資料。磁性存放區資料會使用金鑰 AWS KMS加密。

Timestream Query 服務需要憑證才能存取您的資料。這些憑證會使用KMS金鑰加密。

> **Note**
>
> 的 Timestream LiveAnalytics 不會 AWS KMS 呼叫每個解密操作。相反地，它會保留具有作用中流量的 金鑰本機快取 5 分鐘。任何許可變更都會在最多 5 分鐘內透過 Timestream 針對 LiveAnalytics 系統傳播，最終保持一致。

傳輸中加密

您的所有 Timestream Live Analytics 資料都會在傳輸中加密。根據預設，所有與 的 Timestream 往來的通訊 LiveAnalytics 都會使用 Transport Layer Security （ TLS ） 加密進行保護。

金鑰管理

您可以使用 Key AWS Management Service （ AWS KMS ） 管理 Amazon Timestream Live Analytics 的金鑰。Timestream Live Analytics 需要使用 KMS 來加密您的資料。您擁有下列金鑰管理選項，具體取決於您需要對金鑰進行多少控制：

資料庫和資料表資源

- Timestream Live Analytics 受管金鑰：如果您不提供金鑰，Timestream Live Analytics 將使用 建立`alias/aws/timestream`金鑰KMS。
- 客戶受管金鑰：支援KMS客戶受管金鑰。如果您需要對金鑰的許可和生命週期進行更多控制，包括每年自動輪換它們的功能，請選擇此選項。

排程查詢資源

- Timestream Live Analytics 擁有的金鑰：如果您不提供金鑰，Timestream Live Analytics 將使用自己的KMS金鑰來加密查詢資源，此金鑰會存在於時間串流帳戶中。如需詳細資訊，請參閱KMS開發人員指南中的AWS 擁有金鑰。
- 客戶受管金鑰：支援KMS客戶受管金鑰。如果您需要對金鑰的許可和生命週期進行更多控制，包括每年自動輪換它們的功能，請選擇此選項。

KMS 不支援外部金鑰存放區 （ XKS ） 中的金鑰。

適用於 的 Amazon Timestream 身分和存取管理 LiveAnalytics

AWS Identity and Access Management （ IAM ） 是一種 AWS 服務 ，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員會控制誰可以進行身分驗證 （ 登入 ） 和授權 （ 具有許可 ），以使用 Timestream 作為 LiveAnalytics 資源。IAM 是 AWS 服務 您可以免費使用的 。

主題

物件

使用 AWS Identity and Access Management （ IAM ） 的方式會有所不同，具體取決於您在 Timestream 中為 執行的工作 LiveAnalytics。

服務使用者 – 如果您使用 Timestream LiveAnalytics 進行服務，則管理員會為您提供所需的憑證和許可。當您將更多 Timestream 用於執行任務 LiveAnalytics 的功能時，您可能需要額外的許可。了解存取許可的管理方式可協助您向管理員請求正確的許可。如果您無法存取 Timestream 中的 功能 LiveAnalytics，請參閱 為 Amazon Timestream 進行 LiveAnalytics 身分和存取疑難排解。

服務管理員 – 如果您負責公司 LiveAnalytics 資源的 Timestream，您可能可以完整存取 的 Timestream LiveAnalytics。您的任務是判斷服務使用者應存取的功能 LiveAnalytics 和資源的 Timestream。然後，您必須向IAM管理員提交請求，以變更服務使用者的許可。請檢閱此頁面上的資訊，以了解 的基本概念IAM。若要進一步了解貴公司如何IAM搭配 Timestream for 使用 LiveAnalytics，請參閱 適用於 的 Amazon Timestream 如何搭配 LiveAnalytics 使用 IAM。

IAM 管理員：如果您是IAM管理員，您可能想要了解如何撰寫政策以管理 的 Timestream 存取權的詳細資訊 LiveAnalytics。若要檢視您可以在 中使用的 LiveAnalytics 身分型政策的 Timestream 範例IAM，請參閱 適用於 LiveAnalytics 身分型政策範例的 Amazon Timestream。

使用身分驗證

驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者身分、IAM使用者身分或擔任IAM角色來驗證 （登入 AWS）。

您可以使用透過身分來源提供的憑證，以聯合身分 AWS 身分登入 。 AWS IAM Identity Center （IAM Identity Center） 使用者、您公司的單一登入身分驗證，以及您的 Google 或 Facebook 憑證，都是聯合身分的範例。當您以聯合身分登入時，您的管理員先前會使用 IAM角色設定身分聯合。當您 AWS 使用聯合來存取 時，您會間接擔任 角色。

根據您身分的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需登入的詳細資訊 AWS，請參閱 使用者指南 中的如何登入 AWS 帳戶您的 。 AWS 登入

如果您以 AWS 程式設計方式存取 ， AWS 會提供軟體開發套件 （SDK） 和命令列介面 （CLI），以使用您的 憑證以密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，則必須自行簽署請求。如需使用建議方法自行簽署請求的詳細資訊，請參閱 IAM 使用者指南 中的AWS 簽章第 4 版以取得API請求。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如， AWS 建議您使用多重要素驗證 （MFA） 來提高帳戶的安全性。若要進一步了解，請參閱 AWS IAM Identity Center 使用者指南 中的多重要素驗證和 IAM 使用者指南 AWS 中的多重要素驗證IAM。

IAM 使用者和群組

IAM 使用者是 中具有單一個人或應用程式特定許可 AWS 帳戶 的身分。在可能的情況下，我們建議依賴臨時憑證，而不是建立具有密碼和存取金鑰等長期憑證IAM的使用者。不過，如果您有特定使用案例需要IAM使用者長期憑證，建議您輪換存取金鑰。如需詳細資訊，請參閱 IAM 使用者指南 中的定期輪換需要長期憑證的使用案例存取金鑰。

IAM 群組是指定IAM使用者集合的身分。您無法以群組身分簽署。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的程序變得更為容易。例如，您可以擁有名為 的群組IAMAdmins，並授予該群組管理IAM資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供暫時憑證。若要進一步了解，請參閱 IAM 使用者指南 中的IAM使用者使用案例。

IAM 角色

IAM 角色是 中具有特定許可 AWS 帳戶 的身分。它類似於IAM使用者，但與特定人員無關。若要暫時在 中擔任IAM角色 AWS Management Console，您可以從使用者切換至IAM角色 （主控台）。您可以

透過呼叫 或 AWS API AWS CLI 操作，或使用自訂 來擔任角色URL。如需使用角色方法的詳細資訊，請參閱 IAM 使用者指南 中的擔任角色的方法。

IAM 具有臨時憑證的角色在下列情況下很有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需聯合角色的相關資訊，請參閱 IAM 使用者指南 中的為第三方身分提供者建立角色。如果您使用 IAM Identity Center，您可以設定許可集。若要控制身分在身分驗證後可以存取的內容，IAMIdentity Center 會將許可集與 中的角色相關聯IAM。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的許可集。
- 臨時IAM使用者許可 – IAM使用者或角色可以擔任IAM角色，暫時接受特定任務的不同許可。
- 跨帳戶存取 – 您可以使用 IAM角色，允許不同帳戶中的某人（受信任的主體）存取您帳戶中的資源。角色是授予跨帳戶存取權的主要方式。不過，使用某些 AWS 服務，您可以將政策直接連接至資源（而不是使用角色作為代理）。若要了解跨帳戶存取的角色和資源型政策之間的差異，請參閱 IAM 使用者指南 中的跨帳戶資源存取IAM。
- 跨服務存取 – 有些 AWS 服務 使用其他 中的功能 AWS 服務。例如，當您在 服務中撥打電話時，該服務通常會在 Amazon 中執行應用程式EC2或在 Amazon S3 中儲存物件。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉送存取工作階段（FAS）– 當您使用IAM使用者或角色在 中執行動作時 AWS，您會被視為主體。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫 的委託人許可 AWS 服務，結合 AWS 服務 請求向下游服務提出請求。FAS 只有在服務收到需要與其他 AWS 服務 或 資源互動才能完成的請求時，才會發出請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出FAS請求的政策詳細資訊，請參閱轉送存取工作階段 。
  - 服務角色 – 服務角色是服務代表您執行動作所擔任IAM的角色。IAM 管理員可以從 內部建立、修改和刪除服務角色IAM。如需詳細資訊，請參閱 使用者指南 中的建立角色以將許可委派給 AWS 服務 。 IAM
  - 服務連結角色 – 服務連結角色是連結至 的服務角色類型 AWS 服務。服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的 中 AWS 帳戶 ，並由 服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon 上執行的應用程式 EC2 – 您可以使用 IAM角色來管理在EC2執行個體上執行之應用程式的臨時憑證，以及提出 AWS CLI 或 AWS API請求。最好將存取金鑰存放在EC2執行個體中。若要將 AWS 角色指派給EC2執行個體並將其提供給其所有應用程式，您可以建立連接至執行個體的執行個體設定檔。執行個體設定檔包含 角色，並啟用在EC2執行個體上執行的程式，以取得臨時憑證。如需詳細資訊，請參閱 IAM 使用者指南 中的使用 IAM角色將許可授予在 Amazon EC2執行個體上執行的應用程式。

使用政策管理存取權

您可以透過建立政策並將其連接至 AWS 身分或資源 AWS 來控制 中的存取。政策是 AWS 其中的物件，當與身分或資源相關聯時， 會定義其許可。當主體 （使用者、根使用者或角色工作階段） 發出請求時， 會 AWS 評估這些政策。政策中的許可決定是否允許或拒絕請求。大多數政策都以JSON文件 AWS 形式儲存在 中。如需JSON政策文件結構和內容的詳細資訊，請參閱 IAM 使用者指南 中的JSON政策概觀。

管理員可以使用 AWS JSON政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對所需資源執行動作的許可，IAM管理員可以建立IAM政策。然後，管理員可以將IAM政策新增至角色，使用者可以擔任角色。

IAM 無論您用來執行操作的方法為何，政策都會定義動作的許可。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該政策的使用者可以從 AWS Management Console、 AWS CLI或 AWS 取得角色資訊API。

身分型政策

身分型政策是JSON許可政策文件，您可以附加到身分，例如IAM使用者、使用者群組或角色。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分型政策，請參閱 IAM 使用者指南 中的使用客戶受管政策定義自訂IAM許可。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管政策是獨立的政策，您可以連接到 中的多個使用者、群組和角色 AWS 帳戶。受管政策包括 AWS 受管政策和客戶受管政策。若要了解如何在受管政策或內嵌政策之間進行選擇，請參閱 IAM 使用者指南 中的在受管政策與內嵌政策之間進行選擇。

資源型政策

資源型政策是您連接至資源JSON的政策文件。資源型政策的範例包括IAM角色信任政策和 Amazon S3 儲存貯體政策 。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。主體可以包括帳戶、使用者、角色、聯合使用者或 AWS 服務。

資源型政策是位於該服務中的內嵌政策。您無法在資源型政策IAM中使用來自 的 AWS 受管政策。

存取控制清單 （ACLs）

存取控制清單 （ACLs） 控制哪些主體 （帳戶成員、使用者或角色） 具有存取 資源的許可。ACLs 類似於資源型政策，雖然它們不使用JSON政策文件格式。

Amazon S3 AWS WAF和 Amazon VPC是支援 的服務範例ACLs。若要進一步了解 ACLs，請參閱 Amazon Simple Storage Service 開發人員指南 中的存取控制清單 （ACL） 概觀。

其他政策類型

AWS 支援其他較不常見的政策類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可界限是一項進階功能，您可以在其中設定身分型政策可授予IAM實體 （IAM使用者或角色） 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可界限的詳細資訊，請參閱 IAM 使用者指南 中的IAM實體許可界限。
- 服務控制政策 （SCPs） – SCPs是在 中指定組織或組織單位 （OU） 最大許可JSON的政策 AWS Organizations。 AWS Organizations 是一項用於分組和集中管理您企業擁有 AWS 帳戶 之多個的服務。如果您啟用組織中的所有功能，則可以將服務控制政策 （SCPs） 套用至任何或所有帳戶。SCP 限制成員帳戶中實體的許可，包括每個 AWS 帳戶根使用者。如需 Organizations 和 的詳細資訊SCPs，請參閱 AWS Organizations 使用者指南 中的服務控制政策。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合使用者的暫時工作階段時，作為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參閱 IAM 使用者指南 中的工作階段政策。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。若要了解如何 AWS 決定是否在涉及多種政策類型時允許請求，請參閱 IAM 使用者指南 中的政策評估邏輯。

適用於 的 Amazon Timestream 如何搭配 LiveAnalytics 使用 IAM

使用 IAM 管理 Timestream for 的存取權之前 LiveAnalytics，您應該了解哪些IAM功能可與 Timestream for 搭配使用 LiveAnalytics。若要取得 LiveAnalytics 和其他 AWS 服務如何與 搭配使用的高階檢視IAM，請參閱 IAM 使用者指南 中的 AWS 與 搭配使用的服務IAM。

主題

- 身分型政策的時間 LiveAnalytics串流
- 資源型政策的時間 LiveAnalytics串流
- 根據 LiveAnalytics 標籤的 Timestream 授權
- 角色的時間 LiveAnalytics IAM串流

身分型政策的時間 LiveAnalytics串流

透過身分IAM型政策，您可以指定允許或拒絕的動作和資源，以及允許或拒絕動作的條件。的 Timestream LiveAnalytics 支援特定動作和資源，以及條件索引鍵。若要了解您在JSON政策中使用的所有元素，請參閱 IAM 使用者指南 中的IAMJSON政策元素參考。

動作

管理員可以使用 AWS JSON政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 `Action`元素說明您可以用來允許或拒絕政策中存取的動作。政策動作通常具有與相關聯 AWS API操作相同的名稱。有一些例外狀況，例如沒有相符API操作的僅限許可動作。也有一些作業需要政策中的多個動作。這些額外的動作稱為相依動作。

政策會使用動作來授予執行相關聯動作的許可。

您可以在IAM政策陳述式的動作元素中指定下列動作。使用政策授予許可，以在 中執行 操作AWS。當您在政策中使用 動作時，通常會允許或拒絕存取具有相同名稱API的操作、CLI命令或SQL命令。

在某些情況下，單一動作會控制對 API 操作和 SQL命令的存取。或者，某些操作需要多種不同的動作。

如需 支援的 Timestream 清單 LiveAnalytics `Action`，請參閱下表：

> **Note**
>
> 對於所有資料庫特定的 `Actions`，您可以指定資料庫ARN，將動作限制在特定資料庫。

動作	描述	存取層級	資源類型 (*必填項目)
DescribeEndpoints	傳回後續請求必須向其提出的 Timestream 端點。	全部	*
Select	在從一或多個資料表中選取資料的 Timestream 上執行查	讀取	資料表*

動作	描述	存取層級	資源類型 (*必填項目)
	詢。如需詳細說明，請參閱此備註		
CancelQuery	取消查詢。	讀取	*
ListTables	取得資料表清單。	清單	資料庫*
ListDatabases	取得資料庫清單。	清單	*
ListMeasures	取得量值清單。	讀取	資料表*
DescribeTable	取得資料表描述。	讀取	資料表*
DescribeDatabase	取得資料庫描述。	讀取	資料庫*
SelectValues	執行不需要指定特定資源的查詢。如需詳細說明，請參閱此備註。	讀取	*
WriteRecords	將資料插入 Timestream。	寫入	資料表*
CreateTable	建立 資料表。	寫入	資料庫*
CreateDatabase	建立資料庫。	寫入	*
DeleteDatabase	刪除資料庫。	寫入	*
UpdateDatabase	更新資料庫。	寫入	*
DeleteTable	刪除資料表。	寫入	資料庫*
UpdateTable	更新資料表。	寫入	資料庫*

SelectValues 與 選取：

`SelectValues` 是 `Action`，用於不需要資源的查詢。不需要資源的查詢範例如下：

```
SELECT 1
```

請注意，此查詢不會參考特定 Timestream LiveAnalytics 的資源。考慮另一個範例：

```
SELECT now()
```

此查詢會傳回使用 now()函數的目前時間戳記，但不需要指定資源。 SelectValues 經常用於測試，因此 的 Timestream LiveAnalytics 可以在沒有資源的情況下執行查詢。現在，請考慮Select查詢：

```
SELECT * FROM database.table
```

這種類型的查詢需要資源，具體而言是 的時間 LiveAnalytics table串流，以便可以從資料表擷取指定的資料。

資源

管理員可以使用 AWS JSON政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Resource JSON 政策元素會指定動作套用的物件。陳述式必須包含 Resource 或 NotResource 元素。最佳實務是使用其 Amazon Resource Name （ ARN ） 指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出操作)，請使用萬用字元 (*) 來表示陳述式適用於所有資源。

```
"Resource": "*"
```

在 LiveAnalytics 資料庫和資料表的 Timestream 中，可以在IAM許可Resource元素中使用。

LiveAnalytics 資料庫資源的 Timestream 具有下列 ARN：

```
arn:${Partition}:timestream:${Region}:${Account}:database/${DatabaseName}
```

LiveAnalytics 資料表資源的 Timestream 具有下列 ARN：

```
arn:${Partition}:timestream:${Region}:${Account}:database/${DatabaseName}/table/
${TableName}
```

如需有關 格式的詳細資訊ARNs，請參閱 Amazon Resource Names （ ARNs ） AWS 和服務命名空間。

例如，若要在陳述式中指定database鍵空間，請使用下列 ARN：

```
"Resource": "arn:aws:timestream:us-east-1:123456789012:database/mydatabase"
```

若要指定屬於特定帳戶的所有資料庫，請使用萬用字元（*）：

```
"Resource": "arn:aws:timestream:us-east-1:123456789012:database/*"
```

某些 LiveAnalytics 動作的 Timestream，例如用於建立資源的動作，無法對特定資源執行。在這些情況下，您必須使用萬用字元 (*)。

```
"Resource": "*"
```

條件索引鍵

的 Timestream LiveAnalytics 不提供任何服務特定的條件金鑰，但確實支援使用某些全域條件金鑰。若要查看所有 AWS 全域條件索引鍵，請參閱 IAM 使用者指南 中的AWS 全域條件內容索引鍵。

範例

若要檢視 LiveAnalytics 身分型政策的 Timestream 範例，請參閱 適用於 LiveAnalytics 身分型政策範例的 Amazon Timestream。

資源型政策的時間 LiveAnalytics串流

的 Timestream LiveAnalytics 不支援資源型政策。若要檢視詳細資源類型政策頁面的範例，請參閱 https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html。

根據 LiveAnalytics 標籤的 Timestream 授權

您可以使用標籤來管理對 Timestream 進行 LiveAnalytics 資源的存取。若要根據標籤管理資源存取，您可以使用 `aws:RequestTag/key-name`、 `timestream:ResourceTag/key-name`或 條件索引鍵，在政策的條件元素中提供標籤資訊。 `aws:TagKeys`如需為 LiveAnalytics 資源標記 Timestream 的詳細資訊，請參閱 the section called “標記 資源”。

若要檢視身分型政策範例，用於根據該資源的標籤來限制資源的存取權，請參閱「 根據標籤進行 LiveAnalytics 資源存取的時間串流」。

角色的時間 LiveAnalytics IAM串流

IAM 角色是您 AWS 帳戶中具有特定許可的實體。

將臨時憑證與 的 Timestream 搭配使用 LiveAnalytics

您可以使用臨時憑證來登入聯合、擔任IAM角色或擔任跨帳戶角色。您可以透過呼叫 AWS STS API AssumeRole或 等操作來取得臨時安全憑證GetFederationToken。

服務連結角色

的 Timestream LiveAnalytics 不支援服務連結角色。

服務角色

的 Timestream LiveAnalytics 不支援服務角色。

AWS Amazon Timestream Live Analytics 的 受管政策

AWS 受管政策是由 AWS AWS .managed 政策建立和管理的獨立政策旨在為許多常見使用案例提供許可，以便您可以開始將許可指派給使用者、群組和角色。

請記住， AWS 受管政策可能不會授予特定使用案例的最低權限許可，因為這些許可可供所有 AWS 客戶使用。我們建議您定義使用案例專屬的客戶管理政策，以便進一步減少許可。

您無法變更 AWS 受管政策中定義的許可。如果 AWS 更新受管政策中 AWS 定義的許可，則更新會影響政策連接的所有主體身分（使用者、群組和角色）。 AWS 最有可能在 AWS 服務 啟動新的 或現有服務可用的新API操作時更新受 AWS 管政策。

如需詳細資訊，請參閱 IAM 使用者指南 中的 AWS 受管政策。

主題

- AWS 受管政策： AmazonTimestreamReadOnlyAccess
- AWS 受管政策： AmazonTimestreamConsoleFullAccess
- AWS 受管政策： AmazonTimestreamFullAccess
- 受 AWS 管政策的 Timestream Live Analytics 更新

AWS 受管政策： AmazonTimestreamReadOnlyAccess

您可以AmazonTimestreamReadOnlyAccess連接至使用者、群組和角色。此政策提供 Amazon Timestream 的唯讀存取權。

許可詳細資訊

此政策包含以下許可：

- Amazon Timestream – 提供 Amazon Timestream 的唯讀存取權。此政策也授予許可，以取消任何執行中的查詢。

若要以 JSON 格式檢閱此政策，請參閱 AmazonTimestreamReadOnlyAccess。

AWS 受管政策： AmazonTimestreamConsoleFullAccess

您可以AmazonTimestreamConsoleFullAccess連接至使用者、群組和角色。

此政策提供使用 管理 Amazon Timestream 的完整存取權 AWS Management Console。此政策也會授予特定 AWS KMS 操作和操作的許可，以管理您儲存的查詢。

許可詳細資訊

此政策包含以下許可：

- Amazon Timestream – 授予委託人對 Amazon Timestream 的完整存取權。
- AWS KMS – 允許主體列出別名並描述金鑰。
- Amazon S3 – 允許主體列出所有 Amazon S3 儲存貯體。
- Amazon SNS – 允許主體列出 Amazon SNS主題。
- IAM – 允許主體列出IAM角色。
- DBQMS – 允許主體存取、建立、刪除、說明和更新查詢。資料庫查詢中繼資料服務（dbqms）是僅限內部的服務。它為多個 上的查詢編輯器提供最近和儲存 AWS Management Console 的查詢 AWS 服務，包括 Amazon Timestream。

若要以 JSON 格式檢閱此政策，請參閱 AmazonTimestreamConsoleFullAccess。

AWS 受管政策： AmazonTimestreamFullAccess

您可以AmazonTimestreamFullAccess連接至使用者、群組和角色。

此政策提供 Amazon Timestream 的完整存取權。此政策也會授予特定 AWS KMS 操作的許可。

許可詳細資訊

此政策包含以下許可：

- Amazon Timestream – 授予委託人對 Amazon Timestream 的完整存取權。
- AWS KMS – 允許主體列出別名並描述金鑰。
- Amazon S3 – 允許主體列出所有 Amazon S3 儲存貯體。

若要以 JSON 格式檢閱此政策，請參閱 AmazonTimestreamFullAccess。

受 AWS 管政策的 Timestream Live Analytics 更新

檢視自此服務開始追蹤這些變更以來，Timestream Live Analytics 受 AWS 管政策更新的詳細資訊。如需此頁面變更的自動提醒，請在 Timestream Live Analytics 文件歷史記錄頁面上訂閱RSS摘要。

變更	描述	日期
AmazonTimestreamReadOnlyAccess – 更新現有政策	已將 timestream:DescribeAccountSettings 動作新增至現有的 AmazonTimestreamReadOnlyAccess 受管政策。此動作用於描述 AWS 帳戶 設定。 Timestream Live Analytics 也透過新增Sid欄位來更新此受管政策。 政策更新不會影響AmazonTimestreamReadOnlyAccess 受管政策的使用。	2024 年 6 月 3 日

變更	描述	日期
AmazonTimestreamReadOnlyAccess – 更新現有政策	已將 timestream:DescribeBatchLoadTask 和 timestream:ListBatchLoadTasks 動作新增至現有的 AmazonTimestreamReadOnlyAccess 受管政策。列出和描述批次載入任務時，會使用這些動作。 政策更新不會影響AmazonTimestreamReadOnlyAccess 受管政策的使用。	2023 年 2 月 24 日
AmazonTimestreamReadOnlyAccess – 更新現有政策	已將 timestream:DescribeScheduledQuery 和 timestream:ListScheduledQueries 動作新增至現有的 AmazonTimestreamReadOnlyAccess 受管政策。列出和描述現有的排程查詢時，會使用這些動作。 政策更新不會影響AmazonTimestreamReadOnlyAccess 受管政策的使用。	2021 年 11 月 29 日

變更	描述	日期
AmazonTimestreamConsoleFullAccess – 更新現有政策	已將 s3:ListAllMyBuckets 動作新增至現有的 AmazonTimestreamConsoleFullAccess 受管政策。當您為 Timestream 指定 Amazon S3 儲存貯體以記錄磁性存放區寫入錯誤時，就會使用此動作。 政策更新不會影響受 AmazonTimestreamConsoleFullAccess 管政策的使用。	2021 年 11 月 29 日
AmazonTimestreamFullAccess – 更新現有政策	已將 s3:ListAllMyBuckets 動作新增至現有的 AmazonTimestreamFullAccess 受管政策。當您為 Timestream 指定 Amazon S3 儲存貯體以記錄磁性存放區寫入錯誤時，就會使用此動作。 政策更新不會影響AmazonTimestreamFullAccess 受管政策的使用。	2021 年 11 月 29 日

變更	描述	日期
AmazonTimestreamConsoleFullAccess – 更新現有政策	從現有AmazonTimestreamConsoleFullAccess 受管政策中移除冗餘動作。此政策先前包含備援動作 dbqms:DescribeQueryHistory 。更新的政策會移除備援動作。 政策更新不會影響受 AmazonTimestreamConsoleFullAccess 管政策的使用。	2021 年 4 月 23 日
Timestream Live Analytics 開始追蹤變更	Timestream Live Analytics 開始追蹤其 AWS 受管政策的變更。	2021 年 4 月 21 日

適用於 LiveAnalytics 身分型政策範例的 Amazon Timestream

根據預設，IAM使用者和角色沒有為 LiveAnalytics 資源建立或修改 Timestream 的許可。他們也無法使用 AWS Management Console、CQLSH、 AWS CLI或 來執行任務 AWS API。IAM 管理員必須建立IAM政策，以授予使用者和角色對所需指定資源執行特定API操作的許可。然後，管理員必須將這些政策連接到需要這些許可IAM的使用者或群組。

若要了解如何使用這些範例政策文件建立以IAM身分為基礎的JSON政策，請參閱 IAM 使用者指南 中的在JSON索引標籤上建立政策。

主題

- 政策最佳實務
- 使用適用於 LiveAnalytics 主控台的 Timestream
- 允許使用者檢視他們自己的許可
- Timestream 中的常見操作 LiveAnalytics
- 根據標籤進行 LiveAnalytics 資源存取的時間串流
- 排程查詢

政策最佳實務

身分型政策會決定某人是否可以為帳戶中 LiveAnalytics 的資源建立、存取或刪除 Timestream。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用受AWS 管政策，將許可授予許多常見使用案例。它們可在您的 中使用 AWS 帳戶。建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需詳細資訊，請參閱 IAM 使用者指南 中的 AWS 受管政策或 AWS 任務功能的受管政策。
- 套用最低權限許可 – 當您使用IAM政策設定許可時， 只會授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的詳細資訊，請參閱 IAM 使用者指南 中的政策和許可IAM。
- 使用IAM政策中的條件來進一步限制存取：您可以將條件新增至政策，以限制對動作和資源的存取。例如，您可以撰寫政策條件來指定所有請求都必須使用 傳送SSL。如果透過特定 使用服務動作，例如 AWS 服務，您也可以使用 條件來授予其存取權 AWS CloudFormation。如需詳細資訊，請參閱 IAM 使用者指南 中的IAMJSON政策元素：條件。
- 使用 IAM Access Analyzer 驗證您的IAM政策，以確保安全且功能許可 – IAM Access Analyzer 會驗證新的和現有的政策，讓政策符合IAM政策語言 （JSON） 和IAM最佳實務。IAM Access Analyzer 提供超過 100 個政策檢查和可操作的建議，協助您撰寫安全且實用的政策。如需詳細資訊，請參閱 IAM 使用者指南 中的使用 IAM Access Analyzer 驗證政策。
- 需要多重要素身分驗證 （MFA） – 如果您有需要IAM使用者或 根使用者的案例 AWS 帳戶，請開啟 MFA 以獲得額外的安全性。若要在呼叫API操作MFA時要求 ，請將MFA條件新增至您的政策。如需詳細資訊，請參閱 IAM 使用者指南 中的使用 安全API存取MFA。

如需 中最佳實務的詳細資訊IAM，請參閱 IAM 使用者指南 中的安全最佳實務IAM。

使用適用於 LiveAnalytics 主控台的 Timestream

的 Timestream LiveAnalytics 不需要特定許可，即可存取適用於 LiveAnalytics 主控台的 Amazon Timestream。您需要至少 個唯讀許可，才能列出和檢視 AWS 帳戶中 LiveAnalytics 資源的 Timestream 詳細資訊。如果您建立的身分型政策比最低必要許可更嚴格，則主控台將無法與該政策針對實體 （IAM使用者或角色） 正常運作。

允許使用者檢視他們自己的許可

此範例示範如何建立政策，允許使用者檢視連接至其IAM使用者身分的內嵌和受管政策。此政策包含在主控台上完成此動作或使用 或 AWS CLI 以程式設計方式完成此動作的許可 AWS API。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Timestream 中的常見操作 LiveAnalytics

以下是允許 Timestream LiveAnalytics 中服務的一般操作IAM的政策範例。

主題

- 允許所有操作
- 允許SELECT操作
- 允許對多個資源進行SELECT操作

- 允許中繼資料操作
- 允許INSERT操作
- 允許CRUD操作
- 取消查詢並選取資料，而無需指定資源
- 建立、描述、刪除和描述資料庫
- 依標籤限制列出的資料庫{"Owner": "${username}"}
- 列出資料庫中的所有資料表
- 在資料表上建立、描述、刪除、更新和選取
- 依資料表限制查詢

允許所有操作

以下是允許 Timestream 中 的所有操作的範例政策 LiveAnalytics。

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

允許SELECT操作

下列範例政策允許 SELECT樣式查詢特定資源。

> **Note**
>
> <account_ID> 以您的 Amazon 帳戶 ID 取代 。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:Select",
                "timestream:DescribeTable",
                "timestream:ListMeasures"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/DevOps"
        },
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints",
                "timestream:SelectValues",
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

允許對多個資源進行SELECT操作

下列範例政策允許對多個資源進行 SELECT樣式查詢。

> **Note**
>
> <account_ID> 以您的 Amazon 帳戶 ID 取代 。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:Select",
                "timestream:DescribeTable",
                "timestream:ListMeasures"
            ],
            "Resource": [
```

```
                "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/table/
DevOps",
                "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/table/
DevOps1",
                "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/table/
DevOps2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints",
                "timestream:SelectValues",
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

允許中繼資料操作

下列範例政策允許使用者執行中繼資料查詢，但不允許使用者執行在 Timestream 中讀取或寫入 實際資料的操作 LiveAnalytics。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints",
                "timestream:DescribeTable",
                "timestream:ListMeasures",
                "timestream:SelectValues",
                "timestream:ListTables",
                "timestream:ListDatabases",
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

允許INSERT操作

下列範例政策允許使用者database/sampleDB/table/DevOps在 帳戶中對 執行 INSERT操作<account_id>。

> **Note**
>
> <account_ID> 以您的 Amazon 帳戶 ID 取代 。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "timestream:WriteRecords"
            ],
            "Resource": [
                "arn:aws:timestream:us-east-1:<account_id>:database/sampleDB/table/
DevOps"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

允許CRUD操作

下列範例政策允許使用者在 Timestream 中為 執行CRUD操作 LiveAnalytics。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "timestream:DescribeEndpoints",
                "timestream:CreateTable",
                "timestream:DescribeTable",
                "timestream:CreateDatabase",
                "timestream:DescribeDatabase",
                "timestream:ListTables",
                "timestream:ListDatabases",
                "timestream:DeleteTable",
                "timestream:DeleteDatabase",
                "timestream:UpdateTable",
                "timestream:UpdateDatabase"
            ],
            "Resource": "*"
        }
    ]
}
```

取消查詢並選取資料，而無需指定資源

下列範例政策允許使用者取消查詢，並對不需要資源規格的資料執行Select查詢：

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

建立、描述、刪除和描述資料庫

下列範例政策允許使用者建立、描述、刪除和描述資料庫 sampleDB：

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "timestream:CreateDatabase",
                "timestream:DescribeDatabase",
                "timestream:DeleteDatabase",
                "timestream:UpdateDatabase"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB"
        }
    ]
}
```

依標籤限制列出的資料庫**{"Owner": "${username}"}**

下列範例政策允許使用者列出所有以索引鍵值對 標記的資料庫{"Owner": "${username}"}：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:ListDatabases"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

列出資料庫中的所有資料表

下列範例政策可列出資料庫 中的所有資料表sampleDB：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "timestream:ListTables"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/"
        }
    ]
}
```

在資料表上建立、描述、刪除、更新和選取

下列範例政策允許使用者建立資料表、描述資料表、刪除資料表、更新資料表，以及對資料庫 DevOps中的資料表執行Select查詢sampleDB：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:CreateTable",
                "timestream:DescribeTable",
                "timestream:DeleteTable",
                "timestream:UpdateTable",
                "timestream:Select"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/DevOps"
        }
    ]
}
```

依資料表限制查詢

下列範例政策可讓使用者查詢資料庫 DevOps以外的所有資料表sampleDB：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:Select"
```

```
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "timestream:Select"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/DevOps"
        }
    ]
}
```

根據標籤進行 LiveAnalytics 資源存取的時間串流

您可以使用身分型政策中的條件，根據標籤控制 LiveAnalytics 對 Timestream 資源的存取。本節將提供一些範例。

下列範例示範如何建立政策，在資料表的 `Owner`包含該使用者使用者名稱的值時，將檢視資料表的許可授予使用者。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadOnlyAccessTaggedTables",
            "Effect": "Allow",
            "Action": "timestream:Select",
            "Resource": "arn:aws:timestream:us-east-2:111122223333:database/mydatabase/
table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

您可以將此政策連接至您帳戶中IAM的使用者。如果名為 的使用者`richard-roe`嘗試檢視 LiveAnalytics 資料表的時間串流，則必須標記資料表`Owner=richard-roe`或 `owner=richard-`

roe。否則，他便會被拒絕存取。條件標籤鍵 Owner 符合 Owner 和 owner，因為條件索引鍵名稱不區分大小寫。如需詳細資訊，請參閱 IAM 使用者指南 中的IAMJSON政策元素：條件。

如果傳入請求的標籤具有金鑰Owner和值 ，則下列政策會授予使用者建立具有標籤的資料表的許可username：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTagTableUser",
            "Effect": "Allow",
            "Action": [
                "timestream:Create",
                "timestream:TagResource"
            ],
            "Resource": "arn:aws:timestream:us-east-2:111122223333:database/mydatabase/
table/*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:RequestTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

下列政策允許在任何env標籤設定為 dev或 的資料庫DescribeDatabaseAPI上使用 test：

```
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDescribeEndpoints",
      "Effect": "Allow",
      "Action": [
        "timestream:DescribeEndpoints"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowDevTestAccess",
      "Effect": "Allow",
      "Action": [
```

```
        "timestream:DescribeDatabase"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "timestream:tag/env": [
              "dev",
              "test"
          ]
        }
      }
    }
  ]
}
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowTagAccessForDevResources",
      "Effect": "Allow",
      "Action": [
        "timestream:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": [
            "test",
            "dev"
          ]
        }
      }
    }
  ]
}
```

此政策使用 Condition金鑰，允許將具有 金鑰env和 test、 qa或 值的標籤dev新增至 資源。

排程查詢

列出、刪除、更新、執行 ScheduledQuery

下列範例政策允許使用者列出、刪除、更新和執行排程查詢。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DeleteScheduledQuery",
                "timestream:ExecuteScheduledQuery",
                "timestream:UpdateScheduledQuery",
                "timestream:ListScheduledQueries",
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*"
        }
    ]
}
```

CreateScheduledQuery 使用客戶受管KMS金鑰

下列範例政策允許使用者建立使用客戶受管KMS金鑰加密的排程查詢；*<keyid for ScheduledQuery>*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/ScheduledQueryExecutionRole"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "timestream:CreateScheduledQuery",
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
```

```
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-west-2:123456789012:key/<keyid for
 ScheduledQuery>",
            "Effect": "Allow"
        }
    ]
}
```

DescribeScheduledQuery 使用客戶受管KMS金鑰

下列範例政策允許使用者描述使用客戶受管KMS金鑰建立的排程查詢；*<keyid for ScheduledQuery>*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "timestream:DescribeScheduledQuery",
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-west-2:123456789012:key/<keyid for
 ScheduledQuery>",
            "Effect": "Allow"
        }
    ]
}
```

執行角色許可（使用客戶受管KMS金鑰進行排程查詢，KMS以及SSE使用錯誤報告）

將下列範例政策連接至 ScheduledQueryExecutionRoleArn 參數中指定的IAM角色，CreateScheduledQueryAPI該角色使用客戶受管KMS金鑰進行排程查詢加密，以及將錯誤報告SSE-KMS加密。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "kms:GenerateDataKey",
            ],
            "Resource": "arn:aws:kms:us-west-2:123456789012:key/<keyid for
 ScheduledQuery>",
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-west-2:123456789012:key/<keyid for database-1>",
                "arn:aws:kms:us-west-2:123456789012:key/<keyid for database-n>",
                "arn:aws:kms:us-west-2:123456789012:key/<keyid for ScheduledQuery>"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-west-2:123456789012:scheduled-query-notification-topic-
*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "timestream:Select",
                "timestream:SelectValues",
                "timestream:WriteRecords"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
```

```
                "s3:PutObject",
                "s3:GetBucketAcl"
            ],
            "Resource": [
                "arn:aws:s3:::scheduled-query-error-bucket",
                "arn:aws:s3:::scheduled-query-error-bucket/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

執行角色信任關係

以下是 的 ScheduledQueryExecutionRoleArn 參數中指定IAM角色的信任關係CreateScheduledQueryAPI。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "timestream.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

允許存取在 帳戶內建立的所有排程查詢

將下列範例政策連接至 CreateScheduledQuery 參數中指定的IAM角色ScheduledQueryExecutionRoleArn，API以允許存取在 帳戶內建立的所有排程查詢 *Account_ID*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Principal": {
                "Service": "timestream.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "Account_ID"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:timestream:us-
west-2:Account_ID:scheduled-query/*"
                }
            }
        }
    ]
}
```

允許存取具有特定名稱的所有排程查詢

將下列範例政策連接至 ScheduledQueryExecutionRoleArn 參數中指定的IAM角色，API以允許存取名稱為 CreateScheduledQuery 的所有排程查詢 *Scheduled_Query_Name*，在帳戶中 *Account_ID*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "timestream.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "Account_ID"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:timestream:us-
west-2:Account_ID:scheduled-query/Scheduled_Query_Name*"
                }
            }
        }
```

```
    ]
}
```

為 Amazon Timestream 進行 LiveAnalytics 身分和存取疑難排解

使用下列資訊來協助您診斷和修正使用 Timestream for LiveAnalytics 和 時可能遇到的常見問題IAM。

主題

- 我無權在 Timestream 中執行 的動作 LiveAnalytics
- 我無權執行 iam：PassRole
- 我想要允許 AWS 帳戶外的人員存取我的 Timestream 以取得 LiveAnalytics 資源

我無權在 Timestream 中執行 的動作 LiveAnalytics

如果 AWS Management Console 告訴您未獲授權執行動作，則必須聯絡管理員尋求協助。您的管理員是為您提供簽署憑證的人員。

當mateojacksonIAM使用者嘗試使用主控台檢視有關 的詳細資訊時，會發生下列錯誤範例 *table* 但沒有資料表的timestream:*Select*許可。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 timestream:Select on resource: mytable
```

在此情況下，Mateo 會請求管理員更新他的政策，允許他使用 *mytable* 動作存取 timestream:*Select* 資源。

我無權執行 iam：PassRole

如果您收到錯誤，表示您無權執行iam:PassRole動作，則必須更新政策，才能將角色傳遞給 的 Timestream LiveAnalytics。

有些 AWS 服務 允許您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

當名為 marymajor IAM的使用者嘗試使用主控台在 Timestream 中為 執行動作時，會發生下列錯誤範例 LiveAnalytics。但是，動作請求服務具備服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 `iam:PassRole` 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我想要允許 AWS 帳戶外的人員存取我的 Timestream 以取得 LiveAnalytics 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。對於支援資源型政策或存取控制清單 （ACLs） 的服務，您可以使用這些政策來授予人員對資源的存取權。

如需進一步了解，請參閱以下內容：

- 若要了解 的 Timestream 是否 LiveAnalytics 支援這些功能，請參閱 適用於 的 Amazon Timestream 如何搭配 LiveAnalytics 使用 IAM。
- 若要了解如何 AWS 帳戶 在您擁有的 資源之間提供存取權，請參閱 IAM 使用者指南 中的在您擁有 AWS 帳戶 的另一個資源中為IAM使用者提供存取權。
- 若要了解如何提供資源存取權給第三方 AWS 帳戶，請參閱 使用者指南 中的提供存取權給第三方 AWS 帳戶 擁有。 IAM
- 若要了解如何透過身分聯合提供存取權，請參閱 IAM 使用者指南 中的為外部驗證的使用者提供存取權 （身分聯合）。
- 若要了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱 IAM 使用者指南 中的跨帳戶資源存取IAM。

在 Timestream 中記錄和監控 LiveAnalytics

監控是維護 Timestream 和 AWS 解決方案可靠性、可用性 LiveAnalytics 和效能的重要部分。您應該從 AWS 解決方案的所有部分收集監控資料，以便在發生多點故障時更輕鬆地進行偵錯。不過，在開始監控 的 Timestream 之前 LiveAnalytics，您應該建立監控計畫，其中包含下列問題的答案：

- 監控目標是什麼？
- 要監控哪些資源？
- 監控這些資源的頻率為何？
- 要使用哪些監控工具？
- 誰將執行監控任務？
- 發生問題時應該通知誰？

下一個步驟是建立正常 Timestream 的基準，以測量您環境中 LiveAnalytics 效能的不同時間和不同的負載條件。當您監控 的 Timestream 時 LiveAnalytics，請儲存歷史監控資料，以便與目前的效能資料進行比較、識別正常效能模式和效能異常，以及設計解決問題的方法。

若要建立基準，您至少必須監控下列項目：

- 系統錯誤，以便您可以判斷是否有任何請求導致錯誤。

主題

監控工具

AWS 提供各種工具，您可以用來監控 的 Timestream LiveAnalytics。您可以設定其中一些工具來進行監控，但有些工具需要手動介入。建議您盡可能自動化監控任務。

主題

自動化監控工具

您可以使用下列自動監控工具來監看 Timestream 的 LiveAnalytics ，並在發生錯誤時報告：

- Amazon CloudWatch Alarms – 在您指定的時段內觀察單一指標，並根據指標在數個時段內相對於指定閾值的值執行一或多個動作。動作是傳送至 Amazon Simple Notification Service （Amazon SNS） 主題或 Amazon EC2 Auto Scaling Policy. CloudWatch alarms 的通知，不會單純因為動作處於特定狀態而叫用動作；狀態必須已變更並維持在指定的期間數。如需詳細資訊，請參閱使用 Amazon 監控 CloudWatch。

手動監控工具

監控 的另一個重要部分 LiveAnalytics 是手動監控 CloudWatch 警示未涵蓋的項目。 LiveAnalytics CloudWatch、 Trusted Advisor和其他 AWS Management Console 儀表板的 Timestream at-a-glance 提供 AWS 環境狀態的檢視。

- CloudWatch 首頁顯示以下內容：
  - 目前警示與狀態
  - 警示與資源的圖表
  - 服務運作狀態

  此外，您可以使用 CloudWatch 執行下列動作：
- 建立自定儀表板來監控您注重的服務
- 用於疑難排解問題以及探索驅勢的圖形指標資料。
- 搜尋和瀏覽您的所有 AWS 資源指標
- 建立與編輯要通知發生問題的警示

使用 記錄通話的時間串流 LiveAnalytics API AWS CloudTrail

的 Timestream 與 LiveAnalytics 整合 AWS CloudTrail，此服務提供 Timestream AWS 中使用者、角色或服務所採取動作的記錄，以 LiveAnalytics. CloudTrail captures Data Definition Language （DDL） API呼叫 Timestream LiveAnalytics 作為事件。擷取的呼叫包括從 Timestream 進行 LiveAnalytics 主控台的呼叫，以及從 Timestream 進行操作的 LiveAnalytics API程式碼呼叫。如果您建立追蹤，則可以啟用事件持續交付 CloudTrail至 Amazon Simple Storage Service （Amazon S3） 儲存貯體，包括 的 Timestream 事件 LiveAnalytics。如果您未設定追蹤，仍然可以在 事件歷史記錄 中檢視 CloudTrail 主控台上的最新事件。 使用 所收集的資訊 CloudTrail，您可以判斷對 Timestream 提出的請求 LiveAnalytics、提出請求的 IP 地址、提出請求的人員、提出的時間，以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱 AWS CloudTrail 使用者指南 。

中 LiveAnalytics 資訊的時間串流 CloudTrail

CloudTrail 當您建立 AWS 帳戶時， 會在您的帳戶上啟用。當 的活動在 Timestream 中發生時 LiveAnalytics，該活動會與 CloudTrail 事件歷史記錄 中的其他 AWS 服務事件一起記錄在事件中。您可以檢視、搜尋和下載 AWS 帳戶的最新事件。如需詳細資訊，請參閱檢視具有事件歷史記錄 CloudTrail 的事件。

> **Warning**
>
> 目前， 的 Timestream LiveAnalytics 會產生所有管理和`Query`API操作 CloudTrail 的事件，但不會產生 `WriteRecords`和 `DescribeEndpoints` 的事件APIs。

如需 AWS 帳戶中事件的持續記錄，包括 的 Timestream 事件 LiveAnalytics，請建立追蹤。追蹤可讓 CloudTrail 將日誌檔案傳遞至 Amazon S3 儲存貯體。根據預設，當您在主控台中建立追蹤時，追蹤會套用至所有AWS區域。追蹤會記錄 AWS 分割區中所有 區域的事件，並將日誌檔案傳送至您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析 CloudTrail 日誌中收集的事件資料並對其採取行動。

如需詳細資訊，請參閱 AWS CloudTrail 使用者指南中的以下主題：

- 建立追蹤的概觀
- CloudTrail 支援的服務和整合
- 設定 的 Amazon SNS Notifications CloudTrail
- 從多個區域接收 CloudTrail 日誌檔案
- 從多個帳戶接收 CloudTrail 日誌檔案
- 記錄資料事件

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：

- 請求是使用根還是 AWS Identity and Access Management （ IAM ） 使用者憑證提出
- 提出該請求時，是否使用了特定角色或聯合身分使用者的臨時安全憑證
- 該請求是否由其他 AWS 服務提出

如需詳細資訊，請參閱CloudTrail userIdentity元素 。

對於QueryAPI事件：

- 建立接收所有事件的追蹤，或使用 Timestream 選取 LiveAnalytics 資源類型 `AWS::Timestream::Database`或 的事件`AWS::Timestream::Table`。
- Query API 未存取任何資料庫或資料表，或由於查詢字串格式不正確而導致驗證例外狀況的 請求，會在 CloudTrail 中記錄資源類型`AWS::Timestream::Database`和 ARN值：

  ```
  arn:aws:timestream:(region):(accountId):database/NO_RESOURCE_ACCESSED
  ```

  這些事件只會傳送至接收資源類型 之事件的追蹤`AWS::Timestream::Database`。

Amazon Timestream Live Analytics 中的復原能力

AWS 全域基礎設施是以 AWS 區域和可用區域為基礎建置。 AWS 區域提供多個實體隔離和隔離的可用區域，這些區域與低延遲、高輸送量和高度備援的網路連接。透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 AWS 區域和可用區域的詳細資訊，請參閱AWS 全域基礎設施。

如需透過 取得之 Timestream 資料保護功能的相關資訊 AWS Backup，請參閱 使用 AWS Backup。

Amazon Timestream Live Analytics 中的基礎設施安全性

作為受管服務，Amazon Timestream Live Analytics 受到 Amazon Web Services：安全程序概觀白皮書中所述 AWS 的全球網路安全程序保護。

您可以使用 AWS 已發佈的API呼叫，透過網路存取 Timestream Live Analytics。用戶端必須支援 Transport Layer Security （ TLS ） 1.0 或更新版本。我們建議使用 TLS 1.2 或更新版本。用戶端還必須支援具有完美正向保密性 （ PFS ） 的密碼套件，例如 Ephemeral Diffie-Hellman （ DHE ） 或 Elliptic Curve Ephemeral Diffie-Hellman （ ECDHE ）。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

此外，必須使用與IAM委託人相關聯的存取金鑰 ID 和秘密存取金鑰來簽署請求。或者，您可以透過 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

Timestream Live Analytics 的架構可讓流量隔離到 Timestream Live Analytics 執行個體所在的特定 AWS 區域。

Timestream 中的組態和漏洞分析

組態和 IT 控制是 AWS 和客戶之間的共同責任。如需詳細資訊，請參閱 AWS 共同責任模型 。除了共同責任模型之外， LiveAnalytics 使用者的 Timestream 應注意下列事項：

- 客戶有責任修補他們的用戶端應用程式與相關的用戶端相依性。
- 客戶應考慮在適當情況下進行滲透測試 （ 請參閱https://aws.amazon.com/security/滲透測試/。 ）

Timestream 中的事件回應 LiveAnalytics

LiveAnalytics 服務事件的 Amazon Timestream 會在 Personal Health Dashboard 中報告。您可以在這裡進一步了解儀表板和 AWS Health https://docs.aws.amazon.com/health/latest/ug/what-is-aws-health.html。

的 Timestream LiveAnalytics 支援使用 報告 AWS CloudTrail。如需詳細資訊，請參閱使用 記錄通話的時間串流 LiveAnalytics API AWS CloudTrail。

VPC 端點 （ AWS PrivateLink ）

您可以建立介面VPC端點 ， LiveAnalytics 在 VPC與 Amazon Timestream 之間建立私有連線。介面端點採用 AWS PrivateLink，這項技術可讓您在沒有網際網路閘道、NAT裝置、VPN連線或 AWS Direct Connect 連線的情況下，以私有方式存取 的 LiveAnalytics APIs Timestream。您 中的執行個體VPC不需要公有 IP 地址，即可與 Timestream 通訊 LiveAnalytics APIs。的 VPC與 Timestream LiveAnalytics 之間的流量不會離開 Amazon 網路。

每個介面端點都是由您子網路中的一或多個彈性網路介面表示。如需介面VPC端點的詳細資訊，請參閱 Amazon VPC使用者指南 中的介面VPC端點 （ AWS PrivateLink ）。

為了開始使用適用於 LiveAnalytics 和 VPC端點的 Timestream，我們提供了有關 LiveAnalytics 適用於 的 Timestream 與VPC端點的特定考量、為 建立 Timestream 的介面VPC端點 LiveAnalytics、為 建立 Timestream 的VPC端點政策 LiveAnalytics，以及將 Timestream 用戶端 （ 適用於 Write 或 Query SDK ）與VPC端點搭配使用的資訊。

主題

- VPC 端點如何使用 Timestream
- 為 建立 Timestream 的介面VPC端點 LiveAnalytics
- 為 建立 Timestream 的VPC端點政策 LiveAnalytics

VPC 端點如何使用 Timestream

當您建立VPC端點以存取 Timestream Write 或 Timestream Query 時SDK，所有請求都會路由至 Amazon 網路內的端點，且不會存取公有網際網路。更具體地說，您的請求會路由至帳戶已針對指定區域映射至之儲存格的寫入和查詢端點。若要進一步了解 Timestream 的蜂巢式架構和特定蜂巢式端點，請參閱 行動架構。例如，假設您的帳戶已在 `cell1` 中映射至 `us-west-2`，且您已設定用於寫入（`ingest-cell1.timestream.us-west-2.amazonaws.com`）和查詢 （ ） 的VPC介面端

點query-cell1.timestream.us-west-2.amazonaws.com。在此情況下，使用這些端點傳送的任何寫入請求將完全保留在 Amazon 網路中，且不會存取公有網際網路。

Timestream VPC端點的考量事項

為 Timestream 建立VPC端點時，請考慮下列事項：

- 在您為 設定 Timestream 的介面VPC端點之前 LiveAnalytics，請務必檢閱 Amazon VPC使用者指南中的介面端點屬性和限制。
- 的 Timestream LiveAnalytics 支援從您的 呼叫其所有API動作VPC。
- VPC Timestream for 支援端點政策 LiveAnalytics。依預設，透過端點 LiveAnalytics 允許 的完整 Timestream 存取權。如需詳細資訊，請參閱 Amazon VPC使用者指南 中的使用VPC端點控制對 服務的存取。
- 由於 Timestream 的架構，對寫入和查詢動作的存取需要建立兩個VPC介面端點，每個 各一個 SDK。此外，您必須指定儲存格端點 （您只能為要映射的 Timestream 儲存格建立端點）。如需詳細資訊，請參閱本指南中為 Timestream 建立介面VPC端點 LiveAnalytics一節。

現在您已了解 的 Timestream 如何使用 LiveAnalytics VPC端點，請為 的 Timestream 建立介面VPC端點 LiveAnalytics。

為 建立 Timestream 的介面VPC端點 LiveAnalytics

您可以使用 Amazon VPC主控台或 AWS Command Line Interface （ ） 為 Timestream for LiveAnalytics Service 建立介面VPC端點AWS CLI。若要建立 Timestream 的VPC端點，請完成以下所述的 Timestream 特定步驟。

> **Note**
>
> 在完成下列步驟之前，請確定您了解 Timestream VPC端點的特定考量事項。

使用 Timestream 儲存格建置VPC端點服務名稱

由於 Timestream 的獨特架構，必須為每個 SDK（寫入和查詢） 建立單獨的VPC介面端點。此外，您必須指定 Timestream 儲存格端點 （您只能為要映射的 Timestream 儲存格建立端點）。若要使用介面VPC端點從 中直接連線至 TimestreamVPC，請完成下列步驟：

1. 首先，尋找可用的 Timestream 儲存格端點。若要尋找可用的儲存格端點，請使用 DescribeEndpoints動作（可透過寫入和查詢 取得APIs）來列出 Timestream 帳戶中可用的儲存格端點。如需更多詳細資訊，請參閱範例。
2. 選取要使用的儲存格端點後，請為 Timestream Write 或 Query 建立VPC介面端點字串API：
   - 對於寫入 API：

     ```
     com.amazonaws.<region>.timestream.ingest-<cell>
     ```

   - 對於查詢 API：

     ```
     com.amazonaws.<region>.timestream.query-<cell>
     ```

   where *<region>* 是有效的 AWS 區域碼，且 *<cell>* 是動作 傳回端點物件中的其中一個儲存格端點地址（例如 cell1或 DescribeEndpoints cell2）。如需更多詳細資訊，請參閱範例。
3. 現在您已建置VPC端點服務名稱，請建立介面端點 。當系統要求您提供VPC端點服務名稱時，請使用您在步驟 2 中建構的VPC端點服務名稱。

範例：建構您的VPC端點服務名稱

在下列範例中，DescribeEndpoints動作會在 AWS CLI中使用us-west-2區域中API的寫入來執行：

```
aws timestream-write describe-endpoints --region us-west-2
```

此命令將傳回下列輸出：

```
{
    "Endpoints": [
        {
            "Address": "ingest-cell1.timestream.us-west-2.amazonaws.com",
            "CachePeriodInMinutes": 1440
        }
    ]
}
```

在這種情況下，*cell1* 是 *<cell>* 和 *us-west-2* 是 *<region>*。 因此，產生的VPC端點服務名稱會如下所示：

```
com.amazonaws.us-west-2.timestream.ingest-cell1
```

現在您已為 的 Timestream 建立介面VPC端點 LiveAnalytics，請為 的 Timestream 建立VPC端點政策 LiveAnalytics。

為 建立 Timestream 的VPC端點政策 LiveAnalytics

您可以將端點政策連接至控制對 Timestream for 的存取的VPC端點 LiveAnalytics。此政策會指定下列資訊：

- 可執行動作的主體。
- 可執行的動作。
- 可供執行動作的資源。

如需詳細資訊，請參閱 Amazon VPC使用者指南 中的使用VPC端點控制對 服務的存取。

範例： LiveAnalytics 動作的 Timestream VPC端點政策

以下是 的 Timestream 端點政策範例 LiveAnalytics。連接至端點時，此政策會授予所有資源上所有主體對所列 Timestream LiveAnalytics 動作的存取權 （ 在此情況下為 `ListDatabases` ）。

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "timestream:ListDatabases"
         ],
         "Resource":"*"
      }
   ]
}
```

適用於 的 Amazon Timestream 安全最佳實務 LiveAnalytics

的 Amazon Timestream LiveAnalytics 提供許多安全功能，供您在開發和實作自己的安全政策時考慮。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。

主題

LiveAnalytics 預防性安全最佳實務的時間串流

下列最佳實務可協助您預測和預防 Timestream 中的 安全事件 LiveAnalytics。

靜態加密

使用儲存在 AWS Key Management Service （ AWS KMS ） 中的加密金鑰，靜態 LiveAnalytics 加密存放在資料表中的所有使用者資料的時間串流。如此可透過保護您的資料免於發生未經授權的基礎儲存體存取，為資料提供另一層保護。

的 Timestream LiveAnalytics 使用單一服務預設金鑰 （ AWS 擁有的 CMK ） 來加密您的所有資料表。如果此金鑰不存在，則會為您建立。無法停用服務預設金鑰。如需詳細資訊，請參閱 Timestream for LiveAnalytics Encryption at Rest 。

使用 IAM 角色來驗證 的 Timestream 存取權 LiveAnalytics

對於存取 Timestream for 的使用者、應用程式和其他 AWS 服務 LiveAnalytics，他們必須在其 AWS API請求中包含有效的 AWS 憑證。您不應將 AWS 憑證直接儲存在應用程式或EC2執行個體中。這些是不會自動輪換的長期登入資料，因此如果遭到盜用，可能會對業務造成嚴重的影響。IAM 角色可讓您取得可用來存取 AWS 服務和資源的臨時存取金鑰。

如需詳細資訊，請參閱 IAM 角色。

使用 Timestream IAM的政策進行 LiveAnalytics 基本授權

授予許可時，您可以決定誰取得許可、APIs他們取得許可的 LiveAnalytics Timestream，以及您想要針對這些資源允許的特定動作。對降低錯誤或惡意意圖所引起的安全風險和影響而言，實作最低權限是其中關鍵。

將許可政策連接至身分 IAM （即使用者、群組和角色），藉此授予許可，以在 Timestream LiveAnalytics 上執行資源操作。

您可以使用下列內容執行這項作業：

- AWS 受管 （預先定義） 政策
- 客戶受管政策
- 標籤型授權

考慮用戶端加密

如果您在 Timestream 中存放敏感或機密資料 LiveAnalytics，建議您盡可能將資料加密至接近其原始伺服器的位置，以便在整個生命週期保護您的資料。加密傳輸中和靜態的敏感資料有助於確保您的純文字資料不會提供給任何第三方。

使用其他 服務

的 Amazon Timestream 與各種 AWS 服務和熱門第三方工具 LiveAnalytics 整合。目前， 的 Timestream LiveAnalytics 支援與下列項目的整合：

主題

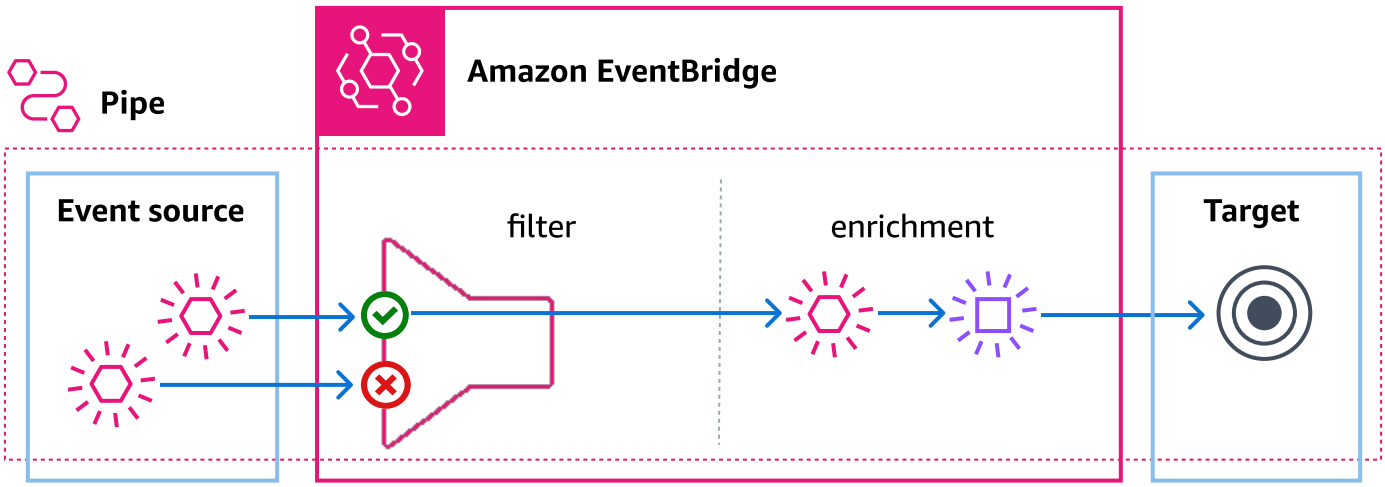
- Amazon DynamoDB
- AWS Lambda
- AWS IoT Core
- Amazon Managed Service for Apache Flink
- Amazon Kinesis
- Amazon MQ
- Amazon MSK
- Amazon QuickSight
- Amazon SageMaker
- Amazon SQS
- 使用 DBeaver 來使用 Amazon Timestream
- 格拉法納
- 使用 SquaredUp 搭配 Amazon Timestream
- 開放原始碼 Telegraf
- JDBC
- ODBC
- VPC 端點 （ AWS PrivateLink ）

Amazon DynamoDB

使用 EventBridge 管道將 DynamoDB 資料傳送至 Timestream

您可以使用 EventBridge 管道將資料從 DynamoDB 串流傳送至 Amazon Timestream for LiveAnalytics Table。

管道適用於 point-to-point支援來源和目標之間的整合，並支援進階轉換和擴充。管道可減少開發事件驅動架構時對專業知識和整合程式碼的需求。若要設定管道，您可以選擇來源、新增可選篩選、定義可選的擴充，以及選擇事件資料的目標。



如需 EventBridge 管道的詳細資訊，請參閱 EventBridge 使用者指南 中的EventBridge 管道。如需設定管道以將事件交付至 LiveAnalytics Amazon Timestream for Table 的相關資訊，請參閱EventBridge 管道目標詳細資訊 。

AWS Lambda

您可以建立與 Timestream for 互動的 Lambda 函數 LiveAnalytics。例如，您可以建立定期執行的 Lambda 函數，以在 Timestream 上執行查詢，並根據滿足一或多個條件的查詢結果傳送SNS通知。若要進一步了解 Lambda，請參閱 AWS Lambda 文件 。

主題

- 使用 Amazon Timestream for LiveAnalytics with Python 建置 AWS Lambda 函數
- 使用 Amazon Timestream for LiveAnalytics 搭配 建置 AWS Lambda 函數 JavaScript
- 使用 Amazon Timestream for LiveAnalytics with Go 建置 AWS Lambda 函數
- 使用 Amazon Timestream for LiveAnalytics with C# 建置 AWS Lambda 函數

使用 Amazon Timestream for LiveAnalytics with Python 建置 AWS Lambda 函數

若要使用 Amazon Timestream for LiveAnalytics with Python 建置 AWS Lambda 函數，請遵循下列步驟。

1. 為 Lambda 建立IAM角色以擔任 將授予存取 Timestream Service 所需的許可，如 中所述提供 Timestream 以供 LiveAnalytics 存取。
2. 編輯IAM角色的信任關係以新增 Lambda 服務。您可以使用下列命令來更新現有角色，以便 AWS Lambda 可以擔任該角色：

   a. 建立信任政策文件：

```
cat > Lambda-Role-Trust-Policy.json << EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

   b. 使用信任文件從上一個步驟更新角色

```
aws iam update-assume-role-policy --role-name <name_of_the_role_from_step_1> --
policy-document file://Lambda-Role-Trust-Policy.json
```

相關參考位於 TimestreamWrite和 TimestreamQuery。

使用 Amazon Timestream for LiveAnalytics 搭配 建置 AWS Lambda 函數 JavaScript

若要使用 的 Amazon Timestream LiveAnalytics 搭配 建置 AWS Lambda 函數 JavaScript，請遵循此處概述的指示。

相關參考位於 Timestream Write Client - AWS SDK for JavaScript v3 和 Timestream Query Client - AWS SDK for JavaScript v3。

使用 Amazon Timestream for LiveAnalytics with Go 建置 AWS Lambda 函數

若要使用 Amazon Timestream for LiveAnalytics with Go 建置 AWS Lambda 函數，請遵循此處概述的指示。

相關參考位於 timestreamwrite 和 timestreamquery 。

使用 Amazon Timestream for LiveAnalytics with C# 建置 AWS Lambda 函數

若要使用具有 C# LiveAnalytics 的 Amazon Timestream 建置 AWS Lambda 函數，請遵循此處概述的指示。

相關參考位於 Amazon.TimestreamWrite 和 Amazon.TimestreamQuery。

AWS IoT Core

您可以使用 IoT AWS Core 從 IoT 裝置收集資料，並透過 IoT Core 規則動作將資料路由至 Amazon Timestream。 AWS IoT 規則動作會指定觸發規則時應採取的動作。您可以定義將資料傳送至 Amazon Timestream 資料表、Amazon DynamoDB 資料庫，以及叫用 AWS Lambda 函數的動作。

IoT 規則中的 Timestream 動作用於將來自傳入訊息的資料直接插入 Timestream。此動作會剖析 IoT Core SQL 陳述式的結果，並將資料儲存在 Timestream 中。傳回SQL結果集中的欄位名稱會用作量值：：name，而欄位的值是量值：：value。

例如，請考慮陳述SQL式和範例訊息承載：

```
SELECT temperature, humidity from 'iot/topic'
```

```
{
  "dataFormat": 5,
  "rssi": -88,
  "temperature": 24.04,
  "humidity": 43.605,
  "pressure": 101082,
  "accelerationX": 40,
  "accelerationY": -20,
  "accelerationZ": 1016,
  "battery": 3007,
  "txPower": 4,
```

```
  "movementCounter": 219,
  "device_id": 46216,
  "device_firmware_sku": 46216
}
```

如果使用上述SQL陳述式建立 Timestream 的 IoT Core 規則動作，則兩個記錄會新增至 Timestream，其測量名稱分別為溫度和濕度，以及測量值分別為 24.04 和 43.605。

您可以使用 SELECT 陳述式中的 AS 運算子，修改要新增至 Timestream 的記錄量值名稱。下列SQL陳述式將建立具有訊息名稱溫度而非溫度的記錄。

測量的資料類型是從訊息承載值的資料類型推斷而來。JSON 整數、雙數、布林值和字串等資料類型會分別映射至 BIGINT、DOUBLE、 BOOLEAN和 的 Timestream 資料類型VARCHAR。資料也可以使用 cast ( ) 函數強制使用特定資料類型。您可以指定量值的時間戳記。如果時間戳記留空，則會使用處理項目的時間。

如需其他詳細資訊，請參閱 Timestream 規則動作文件

若要建立 IoT Core 規則動作以在 Timestream 中儲存資料，請依照下列步驟操作：

主題

- 必要條件
- 使用主控台
- 使用 CLI
- 範例應用程式
- 影片教學課程

必要條件

1. 使用 中所述的說明在 Amazon Timestream 中建立資料庫建立 資料庫。
2. 使用 中所述的指示在 Amazon Timestream 中建立資料表建立資料表。

使用主控台

1. 使用 AWS IoT Core 的 AWS 管理主控台，按一下管理 > 訊息路由 > 規則，後面接著建立規則 來建立規則。
2. 將規則名稱設定為您選擇的名稱，並將 SQL設定為如下所示的文字

```
SELECT temperature as temp, humidity from 'iot/topic'
```

3. 從動作清單中選取時間串流
4. 指定 Timestream 資料庫、資料表和維度名稱，以及將資料寫入 Timestream 的角色。如果角色不存在，您可以按一下建立角色來建立角色
5. 若要測試規則，請遵循此處所示的指示 。

使用 CLI

如果您尚未安裝 AWS 命令列介面 （ AWS CLI ），請從此處執行此操作。

1. 將下列規則承載儲存在名為 timestream_rule.json 的JSON檔案中。Replace (取代) *arn:aws:iam::123456789012:role/TimestreamRole* 您的角色，授予 AWS IoT 存取權，以在 Amazon Timestream 中儲存資料

```
{
    "actions": [
            {
                "timestream": {
                    "roleArn": "arn:aws:iam::123456789012:role/TimestreamRole",
                    "tableName": "devices_metrics",
                    "dimensions": [
                        {
                            "name": "device_id",
                            "value": "${clientId()}"
                        },
                        {
                            "name": "device_firmware_sku",
                            "value": "My Static Metadata"
                        }
                    ],
                    "databaseName": "record_devices"
                }
            }
    ],
    "sql": "select * from 'iot/topic'",
    "awsIotSqlVersion": "2016-03-23",
    "ruleDisabled": false
}
```

2. 使用下列命令建立主題規則

```
aws iot create-topic-rule --rule-name timestream_test --topic-rule-payload file://
<path/to/timestream_rule.json> --region us-east-1
```

3. 使用下列命令擷取主題規則的詳細資訊

```
aws iot get-topic-rule --rule-name timestream_test
```

4. 將下列訊息承載儲存在名為 timestream_msg.json 的檔案中

```
{
  "dataFormat": 5,
  "rssi": -88,
  "temperature": 24.04,
  "humidity": 43.605,
  "pressure": 101082,
  "accelerationX": 40,
  "accelerationY": -20,
  "accelerationZ": 1016,
  "battery": 3007,
  "txPower": 4,
  "movementCounter": 219,
  "device_id": 46216,
  "device_firmware_sku": 46216
}
```

5. 使用下列命令測試規則

```
aws iot-data publish --topic 'iot/topic' --payload file://<path/to/
timestream_msg.json>
```

範例應用程式

為了協助您開始使用 Timestream 搭配 AWS IoT Core，我們建立了全功能範例應用程式，可在 AWS IoT Core 和 Timestream 中建立必要的成品，以建立主題規則和範例應用程式來將資料發佈至主題。

1. 依照 的指示複製 AWS IoT Core 整合的範例應用程式的 GitHub 儲存庫 GitHub
2. 請遵循 中的指示README，使用 AWS CloudFormation 範本在 Amazon Timestream 和 AWS IoT Core 中建立必要的成品，並將範例訊息發佈至主題。

影片教學課程

本影片說明 IoT Core 如何與 Timestream 搭配使用。

Amazon Managed Service for Apache Flink

您可以使用 Apache Flink，將時間序列資料從 Amazon Managed Service for Apache Flink、Amazon MSK、Apache Kafka 和其他串流技術直接傳輸至 Amazon Timestream for LiveAnalytics。我們已為 Timestream 建立 Apache Flink 範例資料連接器。我們也建立了將資料傳送至 Amazon Kinesis 的範例應用程式，以便資料可以從 Kinesis 流向 Managed Service for Apache Flink，最後流向 Amazon Timestream。您可以在 中取得所有這些成品 GitHub。本影片教學課程說明 設定。

> **Note**
>
> Java 11 是使用 Managed Service for Apache Flink 應用程式的建議版本。如果您有多個 Java 版本，請務必將 Java 11 匯出到 JAVA_HOME 環境變數。

主題

- 範例應用程式
- 影片教學課程

範例應用程式

若要開始使用，請遵循下列程序：

1. 在 Timestream 中建立資料庫，名稱為 ，kdaflink請依照 中所述的指示進行 建立 資料庫
2. 在 Timestream 中建立名為 的資料表，kinesisdata1請遵循 中所述的指示 建立資料表
3. TimestreamTestStream 按照建立串流中所述的指示，使用名稱建立 Amazon Kinesis Data Stream https://docs.aws.amazon.com/streams/latest/dev/amazon-kinesis-streams.html#how-do-i-create-a-stream
4. 按照 的指示複製適用於 Timestream 的 Apache Flink 資料連接器的 GitHub 儲存庫 GitHub
5. 若要編譯、執行和使用範例應用程式，請遵循 Apache Flink 範例資料連接器中的指示 README
6. 遵循編譯應用程式程式碼的指示，編譯 Managed Service for Apache Flink 應用程式
7. 按照上傳 Apache Flink 串流程式碼的指示，上傳 Managed Service for Apache Flink 應用程式二進位檔

a. 按一下建立應用程式後，按一下應用程式IAM角色的連結

b. 連接 AmazonKinesisReadOnlyAccess和 IAM的政策AmazonTimestreamFullAccess。

> **Note**
>
> 上述IAM政策不限於特定資源，且不適用於生產用途。對於生產系統，請考慮使用限制存取特定資源的政策。

8. 依照 的指示，複製範例應用程式的 GitHub 儲存庫，將資料寫入 Kinesis https://github.com/awslabs/amazon-timestream-tools/blob/master/tools/kinesis_ingestor GitHub
9. 請依照 中的指示README執行範例應用程式，以將資料寫入 Kinesis
10. 在 Timestream 中執行一或多個查詢，以確保資料從 Kinesis 傳送至 Managed Service for Apache Flink to Timestream，並遵循 的指示 建立資料表

影片教學課程

本影片說明如何將 Timestream 與 Managed Service for Apache Flink 搭配使用。

Amazon Kinesis

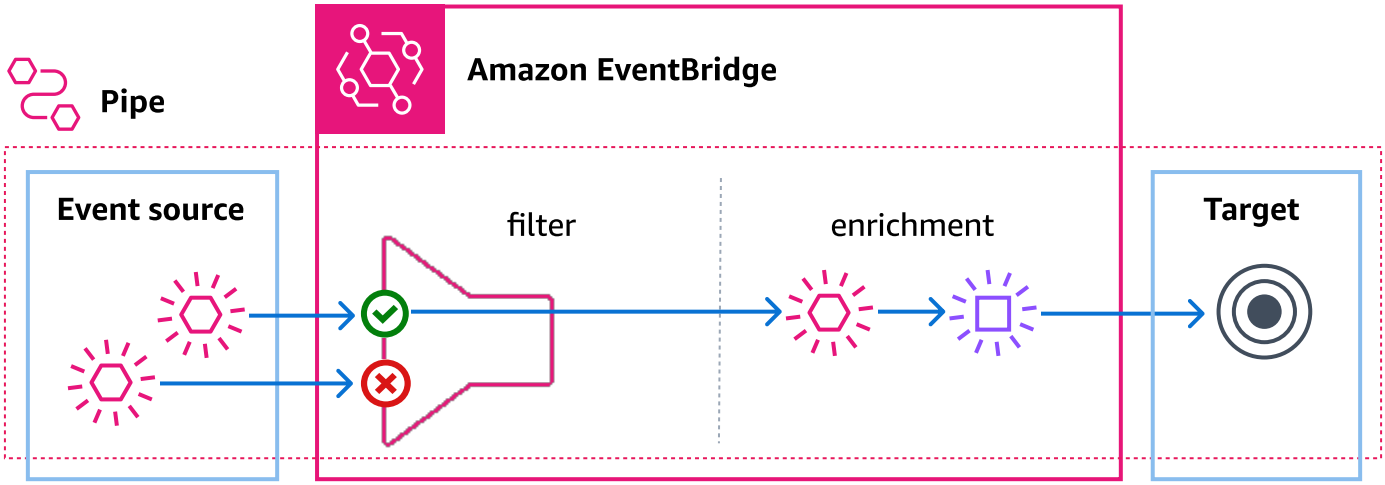
使用 Amazon Managed Service for Apache Flink

您可以將資料從 Kinesis Data Streams 傳送至 Timestream，以便 LiveAnalytics 使用 Managed Service for Apache Flink 的範例 Timestream 資料連接器。如需詳細資訊，請參閱 Amazon Managed Service for Apache Flink for Apache Flink。

使用 EventBridge 管道將 Kinesis 資料傳送至 Timestream

您可以使用 EventBridge 管道將資料從 Kinesis 串流傳送至 LiveAnalytics Amazon Timestream for Table。

管道適用於 point-to-point支援來源和目標之間的整合，並支援進階轉換和擴充。管道可減少開發事件驅動架構時對專業知識和整合程式碼的需求。若要設定管道，您可以選擇來源、新增可選篩選、定義可選的擴充，以及選擇事件資料的目標。



此整合可讓您利用 Timestream的時間序列資料分析功能，同時簡化資料擷取管道。

將 EventBridge 管道與 搭配使用 Timestream 可提供下列優點：

- 即時資料擷取：將資料直接從 Kinesis 串流到 的 Timestream LiveAnalytics，啟用即時分析和監控。
- 無縫整合：使用 EventBridge 管道管理資料流程，而不需要複雜的自訂整合。
- 增強型篩選和轉換：在 Kinesis 記錄存放於 之前對其進行篩選或轉換 Timestream ，以符合您的特定資料處理需求。
- 可擴展性：使用內建平行處理和批次處理功能處理高輸送量資料串流，並確保高效率資料處理。

組態

若要設定 EventBridge 管道，將資料從 Kinesis 串流到 Timestream，請依照下列步驟進行：

1. 建立 Kinesis 串流

   確定您具有要從中擷取資料的作用中 Kinesis 資料串流。

2. 建立 Timestream 資料庫和資料表

   設定要存放資料的 Timestream 資料庫和資料表。

3. 設定 EventBridge 管道：
   - 來源：選取 Kinesis 串流作為來源。
   - 目標：選擇 Timestream 作為目標。
   - 批次處理設定：定義批次處理時段和批次大小，以最佳化資料處理並減少延遲。

> **Important**
>
> 設定管道時，建議您擷取一些記錄，以測試所有組態的正確性。請注意，成功建立管道並不保證管道正確，且資料會順利流動。可能會有執行期錯誤，例如不正確的資料表、不正確的動態路徑參數，或套用映射後的無效 Timestream 記錄，這些錯誤會在實際資料流過管道時發現。

下列組態會決定擷取資料的速率：

- BatchSize：將傳送至 Timestream for 的批次大小上限 LiveAnalytics。範圍：0 - 100。建議將此值保留為 100，以取得最大輸送量。
- MaximumBatchingWindowInSeconds：在批次傳送至 LiveAnalytics 目標的 Timestream batchSize 之前，等待填充 的時間上限。根據傳入事件的速率，此組態會決定擷取延遲，建議將此值保留 < 10 秒，以近乎即時 Timestream 地繼續將資料傳送至 。
- ParallelizationFactor：從每個碎片同時處理的批次數量。建議使用最大值 10，以取得最大輸送量和近乎即時的擷取。

  如果您的串流由多個目標讀取，請使用增強型扇出，為您的管道提供專用取用者，以實現高輸送量。如需詳細資訊，請參閱 Kinesis Data Streams 使用者指南 中的使用 開發增強型扇出消費者 Kinesis Data Streams API。

> **Note**
>
> 可達到的最大輸送量取決於每個帳戶的並行管道執行。

下列組態可確保防止資料遺失：

- DeadLetterConfig：建議一律設定 DeadLetterConfig ，以避免因使用者錯誤 LiveAnalytics 而無法擷取事件至 Timestream 的情況遺失任何資料。

使用下列組態設定最佳化管道的效能，這有助於防止記錄造成減慢或阻塞。

- MaximumRecordAgeInSeconds：不會處理超過此時間的記錄，並將直接移至 DLQ。建議將此值設定為不高於目標 Timestream 資料表的已設定記憶體存放區保留期。

- MaximumRetryAttempts：在記錄傳送至 之前，記錄的重試嘗試次數 DeadLetterQueue。建議將此設定為 10。這應該能夠協助解決任何暫時性問題，如果是持續性問題，記錄會移至串流的其餘部分 DeadLetterQueue 並解除封鎖。
- OnPartialBatchItemFailure：對於支援部分批次處理的來源，我們建議您啟用此功能，並將其設定為 AUTOMATIC_BISECT，以便在捨棄/傳送至 之前，進行失敗記錄的額外重試DLQ。

組態範例

以下是如何設定 EventBridge 管道將資料從 Kinesis 串流串流到 Timestream 資料表的範例：

Example IAM 的政策更新 Timestream

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:WriteRecords"
            ],
            "Resource": [
                "arn:aws:timestream:us-east-1:123456789012:database/my-database/table/
my-table"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*"
        }
    ]
}
```

Example Kinesis 串流組態

```
{
  "Source": "arn:aws:kinesis:us-east-1:123456789012:stream/my-kinesis-stream",
  "SourceParameters": {
    "KinesisStreamParameters": {
```

```
        "BatchSize": 100,
        "DeadLetterConfig": {
            "Arn": "arn:aws:sqs:us-east-1:123456789012:my-sqs-queue"
        },
       "MaximumBatchingWindowInSeconds": 5,
        "MaximumRecordAgeInSeconds": 1800,
        "MaximumRetryAttempts": 10,
        "StartingPosition": "LATEST",
       "OnPartialBatchItemFailure": "AUTOMATIC_BISECT"
    }
  }
}
```

Example Timestream 目標組態

```
{
    "Target": "arn:aws:timestream:us-east-1:123456789012:database/my-database/table/my-
table",
    "TargetParameters": {
        "TimestreamParameters": {
            "DimensionMappings": [
                {
                    "DimensionName": "sensor_id",
                    "DimensionValue": "$.data.device_id",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "DimensionName": "sensor_type",
                    "DimensionValue": "$.data.sensor_type",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "DimensionName": "sensor_location",
                    "DimensionValue": "$.data.sensor_loc",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": [
                {
                    "MultiMeasureName": "readings",
                    "MultiMeasureAttributeMappings": [
                        {
                            "MultiMeasureAttributeName": "temperature",
```

```
                            "MeasureValue": "$.data.temperature",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "MultiMeasureAttributeName": "humidity",
                            "MeasureValue": "$.data.humidity",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "MultiMeasureAttributeName": "pressure",
                            "MeasureValue": "$.data.pressure",
                            "MeasureValueType": "DOUBLE"
                        }
                    ]
                }
            ],
            "SingleMeasureMappings": [],
            "TimeFieldType": "TIMESTAMP_FORMAT",
            "TimestampFormat": "yyyy-MM-dd HH:mm:ss.SSS",
            "TimeValue": "$.data.time",
            "VersionValue": "$.approximateArrivalTimestamp"
        }
    }
}
```

事件轉換

EventBridge 管道可讓您在資料達到 之前轉換資料 Timestream。您可以定義轉換規則來修改傳入 Kinesis 的記錄，例如變更欄位名稱。

假設串流 Kinesis 包含溫度和濕度資料。您可以使用 EventBridge 轉換來重新命名這些欄位，然後再將其插入 Timestream。

最佳實務

批次處理和緩衝

- 設定批次處理時段和大小，以平衡寫入延遲和處理效率。
- 使用批次處理時段，在處理之前累積足夠的資料，減少頻繁的小批次的額外負荷。

平行處理

使用 ParallelizationFactor設定來增加並行，特別是對於高輸送量串流。這可確保可以同時處理每個碎片的多個批次。

資料轉換

利用 EventBridge 管道的轉換功能來篩選和增強記錄，然後再將它們儲存在 中 Timestream。這有助於使資料與您的分析需求保持一致。

安全性

- 確保用於 EventBridge 管道IAM的角色具有從 讀取 Kinesis 和寫入 的必要許可 Timestream。
- 使用加密和存取控制措施來保護傳輸中和靜態資料。

除錯失敗

- 自動停用管道

  如果目標不存在或有許可問題，管道將在約 2 小時內自動停用
- 限流

  管道能夠自動關閉並重試，直到節流減少為止。
- 啟用日誌

  我們建議您在 ERROR層級啟用日誌，並包含執行資料，以進一步了解失敗。發生任何失敗時，這些日誌將包含request/response sent/received來自 的 Timestream。這可協助您了解相關的錯誤，並視需要在修正記錄後重新處理記錄。

監控

我們建議您設定下列警示，以偵測資料流程的任何問題：

- 來源中記錄的最長期間
  - `GetRecords.IteratorAgeMilliseconds`
- 管道中的失敗指標
  - `ExecutionFailed`
  - `TargetStageFailed`
- Timestream 寫入API錯誤

- `UserErrors`

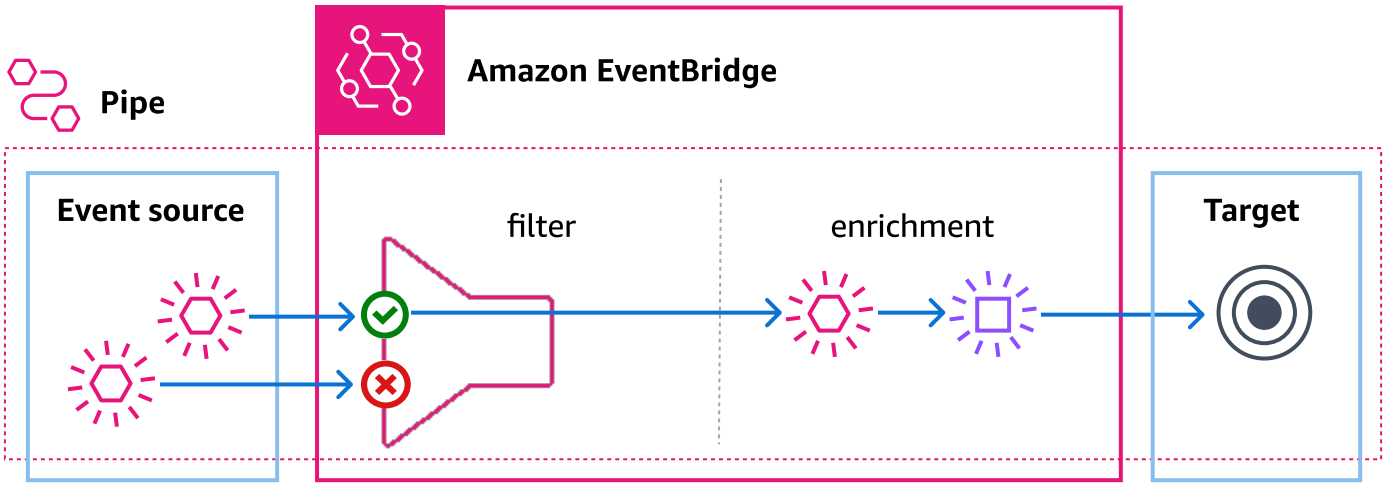
如需其他監控指標，請參閱 EventBridge 使用者指南 中的監控 EventBridge。

Amazon MQ

使用 EventBridge 管道將 Amazon MQ 資料傳送至 Timestream

您可以使用 EventBridge 管道將資料從 Amazon MQ 代理程式傳送至 Amazon Timestream 的 LiveAnalytics 資料表。

管道適用於 point-to-point支援來源和目標之間的整合，並支援進階轉換和擴充。管道可減少開發事件驅動架構時對專業知識和整合程式碼的需求。若要設定管道，您可以選擇來源、新增可選篩選、定義可選的擴充，以及選擇事件資料的目標。



如需 EventBridge 管道的詳細資訊，請參閱 EventBridge 使用者指南 中的EventBridge 管道。如需設定管道以將事件交付至 LiveAnalytics Amazon Timestream for Table 的相關資訊，請參閱EventBridge 管道目標詳細資訊 。

Amazon MSK

使用 Managed Service for Apache Flink 將 Amazon MSK 資料傳送至 Timestream LiveAnalytics

您可以透過 Amazon MSK Timestream 建置類似 Managed Service for Apache Flink 範例 Timestream 資料連接器的資料連接器，將資料從 傳送至 。請參閱 Amazon Managed Service for Apache Flink，了解詳細資訊。

使用 Kafka Connect 將 Amazon MSK資料傳送至 Timestream LiveAnalytics

您可以使用 Kafka Connect，將時間序列資料 Amazon MSK 直接從 擷取到 Timestream for 中 LiveAnalytics。

我們已為 建立 Kafka Sink Connector 範例 Timestream。我們也建立了一個範例 Apache jMeter 測試計畫，用於將資料發佈至 Kafka 主題，以便資料可以從主題流經 Timestream Kafka Sink Connector，到資料表的 LiveAnalytics Timestream。所有這些成品都可在 上取得 GitHub。

> **Note**
>
> Java 11 是使用 Timestream Kafka Sink Connector 的建議版本。如果您有多個 Java 版本，請務必將 Java 11 匯出到 JAVA_HOME 環境變數。

建立範例應用程式

若要開始使用，請遵循下列程序。

1. 在 的 Timestream 中 LiveAnalytics，建立名為 的資料庫`kafkastream`。

   如需詳細說明，???請參閱 程序。
2. 在 的 Timestream 中 LiveAnalytics，建立名為 的資料表`purchase_history`。

   如需詳細說明，???請參閱 程序。
3. 遵循 中共用的指示來建立下列項目：、 和 。

   - Amazon MSK 叢集
   - 設定為 Kafka 生產者用戶端機器的 Amazon EC2 執行個體
   - Kafka 主題

   如需詳細說明，請參閱 kafka_ingestor 專案的先決條件。
4. 複製 Timestream Kafka Sink Connector 儲存庫。

   如需 GitHub 詳細說明，請參閱在 上複製儲存庫。
5. 編譯外掛程式程式碼。

   如需詳細說明 GitHub ，請參閱 上的連接器 - 從來源建置。
6. 將下列檔案上傳至 S3 儲存貯體：遵循 中所述的指示。

- `/target` 目錄中的 jar 檔案 （ kafka-connector-timestream->VERSION<-jar-with-dependencies.jar ）
- json 結構描述檔案範例，`purchase_history.json`。

如需詳細說明，請參閱 Amazon S3 使用者指南中的上傳物件。

7. 建立兩個VPC端點。MSK Connector 會使用這些端點來存取使用 的資源 AWS PrivateLink。
   - 一個用於存取 Amazon S3 儲存貯體
   - 一個用於存取 LiveAnalytics 資料表的 Timestream。

   如需詳細說明，請參閱VPC端點。
8. 使用上傳的 jar 檔案建立自訂外掛程式。

   如需詳細說明，請參閱 Amazon MSK 開發人員指南中的外掛程式。
9. 使用 Worker Configuration 參數 中描述JSON的內容建立自訂工作者組態。 請遵循 中所述的指示 https://github.com/awslabs/amazon-timestream-tools/tree/mainline/integrations/kafka_connector#worker-configuration-parameters

   如需詳細說明，請參閱 Amazon MSK 開發人員指南中的建立自訂工作者組態。
10. 建立服務執行 IAM 角色。

    如需詳細說明，請參閱IAM 服務角色。
11. Amazon MSK 使用在先前步驟中建立的自訂外掛程式、自訂工作者組態和服務執行 IAM 角色，以及使用範例連接器組態 來建立連接器。

    如需詳細說明，請參閱 Amazon MSK 開發人員指南中的建立連接器。

    請務必使用各自的值更新下列組態參數的值。如需詳細資訊，請參閱 Connector Configuration 參數。
    - `aws.region`
    - `timestream.schema.s3.bucket.name`
    - `timestream.ingestion.endpoint`

    連接器建立需要 5–10 分鐘才能完成。當管道的狀態變更為 時，管道已準備就緒`Running`。

12. 發佈持續的訊息串流，用於將資料寫入建立的 Kafka 主題。

    如需詳細說明，請參閱如何使用。

13. 執行一或多個查詢，以確保資料從 Amazon MSK 傳送至 MSK 連線至 LiveAnalytics 資料表的 Timestream。

    如需詳細說明，???請參閱 程序。

其他資源

部落格：從 Kafka 叢集即時無伺服器資料擷取到 Timestream，以便 LiveAnalytics 使用 Kafka Connect，說明使用 Timestream for LiveAnalytics Kafka Sink Connector 設定 end-to-end管道，從使用 Apache jMeter 測試計畫將數千則範例訊息發佈至 Kafka 主題的 Kafka 生產者用戶端機器開始，以驗證在 Timestream 中擷取的 LiveAnalytics 資料表記錄。

Amazon QuickSight

您可以使用 Amazon QuickSight 來分析和發佈包含 Amazon Timestream 資料的資料儀表板。本節說明如何建立新的 QuickSight 資料來源連線、修改許可、建立新的資料集，以及執行分析。本影片教學課程說明如何使用 Timestream 和 Amazon QuickSight。

> **Note**
>
> Amazon 中的所有資料集 QuickSight 都是唯讀的。您無法透過使用 Amazon 移除資料來源、資料集或欄位 QuickSight ，對 Timestream 中的實際資料進行任何變更。

主題

- 從 存取 Amazon Timestream QuickSight
- 為 Timestream 建立新的 QuickSight 資料來源連線
- 編輯 Timestream QuickSight 資料來源連線的許可
- 為 Timestream 建立新的 QuickSight 資料集
- 建立 Timestream 的新分析
- 影片教學課程

從 存取 Amazon Timestream QuickSight

在您可以繼續之前，Amazon QuickSight 需要獲得連接至 Amazon Timestream 的授權。如果未啟用連線，當您嘗試連線時，會收到錯誤。 QuickSight 管理員可以授權與 AWS 資源的連線。若要授權從 QuickSight 到 Timestream 的連線，請依照步驟 5 中的使用 AWS 其他服務：縮小存取範圍 中的程序選擇 Amazon Timestream。

為 Timestream 建立新的 QuickSight 資料來源連線

> **Note**
>
> Amazon QuickSight 和 Amazon Timestream 之間的連線會在傳輸過程中使用 SSL（TLS 1.2）加密。您無法建立未加密的連線。

1. 請確定您已設定適當的許可，讓 Amazon QuickSight 存取 Amazon Timestream，如 中所述 從 存取 Amazon Timestream QuickSight。
2. 首先建立新的資料集。從導覽窗格中選擇資料集，然後選擇新資料集 。
3. 選取 Timestream 資料來源卡。
4. 針對資料來源名稱 ，輸入 Timestream 資料來源連線的名稱，例如 `US Timestream Data`。

   > **Note**
   >
   > 因為您可以透過與 Timestream 的連線建立許多資料集，因此最好保持名稱簡單。

5. 選擇驗證連線，檢查您是否可以成功連線至 Timestream。

   > **Note**
   >
   > 驗證連線只會驗證您可以連線。不過，它不會驗證特定資料表或查詢。

6. 選擇建立資料來源。
7. 針對資料庫 ，選擇選取... 以檢視可用選項的清單。選擇您要使用的項目。
8. 選擇選取以繼續。
9. 選擇下列其中一項：

   - 若要將資料匯入 QuickSight的記憶體內引擎（稱為 SPICE），請選擇匯入 SPICE以加快分析速度。

- 若要 QuickSight 允許在每次重新整理資料集或使用分析或儀表板時針對資料執行查詢，請選擇直接查詢您的資料。

10. 選擇編輯/預覽，然後選擇儲存以儲存資料集並將其關閉。

編輯 Timestream QuickSight 資料來源連線的許可

下列程序說明如何檢視、新增和撤銷其他 QuickSight 使用者的許可，以便他們可以存取相同的 Timestream 資料來源。人員必須是 中的作用中使用者 QuickSight ，才能新增它們。

> **Note**
>
> 在 中 QuickSight，資料來源有兩個許可層級：使用者和擁有者。
>
> - 選擇使用者以允許讀取存取。
> - 選擇擁有者，以允許該使用者編輯、共用或刪除此 QuickSight 資料來源。

1. 請確定您已設定適當的許可，讓 Amazon QuickSight 存取 Amazon Timestream，如 中所述 從 存取 Amazon Timestream QuickSight。
2. 選擇左側的資料集，然後向下捲動以尋找適用於 Timestream 連線的資料來源卡片。例如 US Timestream Data。
3. 選擇Timestream資料來源卡。
4. 選擇 Share data source。此時會顯示目前許可的清單。
5. （選用）若要編輯許可，您可以選擇 user或 owner。
6. （選用）若要撤銷許可，請選擇 Revoke access。您撤銷的人員無法從此資料來源建立新的資料集。不過，其現有的資料集仍然可以存取此資料來源。
7. 若要新增許可，請選擇 Invite users，然後依照下列步驟新增使用者：
   a. 新增人員以允許他們使用相同的資料來源。
   b. 針對每個 ，選擇您要套用Permission的 。
8. 完成後，請選擇 Close。

為 Timestream 建立新的 QuickSight 資料集

1. 請確定您已設定適當的許可，讓 Amazon QuickSight 存取 Amazon Timestream，如 中所述 從 存取 Amazon Timestream QuickSight。
2. 選擇左側的資料集，然後向下捲動以尋找適用於 Timestream 連線的資料來源卡片。如果您有許多資料來源，您可以使用頁面頂端的搜尋列，在名稱上尋找部分相符的資料來源。
3. 選擇 Timestream 資料來源卡。然後選擇建立資料集 。
4. 對於資料庫，選擇選取以檢視可用選項的清單。選擇您要使用的資料庫。
5. 接下來，在資料表中選擇您要使用的資料表。
6. 選擇編輯/預覽。
7. （選用）若要新增更多資料，請選擇右上角的新增資料。

    a. 選擇 切換資料來源 ，然後選擇不同的資料來源。

    b. 依照 UI 提示完成新增資料。

    c. 將新資料新增至相同的資料集後，選擇設定此聯結 (兩個紅點)。為每個額外的資料表設定一個聯結。

    d. 若要新增計算欄位，選擇新增計算欄位。

    e. 若要使用 Sagemaker，請選擇 Augment 搭配 SageMaker。此選項僅適用於 QuickSight Enterprise Edition。

    f. 取消核取您要省略的任何欄位。

    g. 更新您要變更的任何資料類型。
8. 完成後，選擇儲存以儲存並關閉資料集。

建立 Timestream 的新分析

1. 請確定您已設定適當的許可，讓 Amazon QuickSight 存取 Amazon Timestream，如 中所述 從 存取 Amazon Timestream QuickSight。
2. 選擇左側的分析。
3. 選擇下列其中一項：

    - 若要建立新分析，選擇右側的新建分析。
    - 若要將 Timestream 資料集新增至現有分析，請開啟您要編輯的分析。選擇左上角附近的鉛筆圖示，然後新增資料集 。
4. 選擇左側的欄位，啟動第一個資料視覺化。

5. 如需詳細資訊，請參閱使用分析 - Amazon QuickSight

影片教學課程

本影片說明 Amazon 如何與 Timestream QuickSight 搭配使用。

Amazon SageMaker

您可以使用 Amazon SageMaker Notebooks 將機器學習模型與 Amazon Timestream 整合。為了協助您開始使用，我們建立了一個範例 SageMaker 筆記本，用於處理來自 Timestream 的資料。資料會從多執行緒 Python 應用程式插入 Timestream，持續傳送資料。範例 SageMaker 筆記本和範例 Python 應用程式的原始程式碼可在 中使用 GitHub。

1. 按照 和 中所述的指示建立資料庫建立 資料庫和資料表 建立資料表
2. 依照 的指示複製多執行緒 Python 範例應用程式的 GitHub 儲存庫 GitHub
3. 依照 的指示複製範例 Timestream SageMaker Notebook 的 GitHub 儲存庫 GitHub。
4. 執行應用程式，依照 中的指示，持續將資料擷取至 Timestream README
5. 請依照此處 SageMaker 所述的指示為 Amazon 建立 Amazon S3 儲存貯體。 https://docs.aws.amazon.com/sagemaker/latest/dg/gs-config-permissions.html
6. 建立已安裝最新 boto3 的 Amazon SageMaker 執行個體：除了此處所述的指示之外，請遵循下列步驟：

   a. 在建立筆記本執行個體頁面上，按一下其他組態

   b. 按一下生命週期組態 - 選用，然後選取建立新的生命週期組態

   c. 在建立生命週期組態精靈方塊中，執行下列動作：

      i. 將所需的名稱填入組態，例如 `on-start`

      ii. 在啟動筆記本指令碼中，從 Github 複製貼上指令碼內容

      iii. 在貼上的指令碼`PACKAGE=boto3`中將 取代`PACKAGE=scipy`為 。
7. 按一下建立組態
8. 前往 AWS 管理主控台中的 IAM服務，並尋找筆記本執行個體的新建立 SageMaker執行角色。
9. 將 IAM的政策`AmazonTimestreamFullAccess`連接至執行角色。

> **Note**
>
> 此AmazonTimestreamFullAccessIAM政策不限於特定資源，且不適合用於生產。對於生產系統，請考慮使用限制存取特定資源的政策。

10. 當筆記本執行個體的狀態為 時InService，選擇開啟 Jupyter 為執行個體啟動 SageMaker 筆記本
11. 選取上傳按鈕，將檔案 timestreamquery.py和 上傳至Timestream_SageMaker_Demo.ipynb筆記本
12. 選擇 Timestream_SageMaker_Demo.ipynb

> **Note**
>
> 如果您看到一個帶有找不到核心 的快顯視窗，請選擇 conda_python3 並按一下設定核心。

13. 修改 DB_NAME、TABLE_NAME、 bucket和 ENDPOINT，以符合訓練模型的資料庫名稱、資料表名稱、S3 儲存貯體名稱和區域。
14. 選擇播放圖示以執行個別儲存格
15. 當您到達儲存格 時Leverage Timestream to find hosts with average CPU utilization across the fleet，請確定輸出至少傳回 2 個主機名稱。

> **Note**
>
> 如果輸出中少於 2 個主機名稱，您可能需要使用更多執行緒和主機規模重新執行將資料擷取到 Timestream 的範例 Python 應用程式。

16. 當您到達儲存格 時Train a Random Cut Forest (RCF) model using the CPU utilization history，train_instance_type請根據訓練任務的資源需求變更
17. 當您到達儲存格 時Deploy the model for inference，instance_type請根據推論任務的資源需求變更

> **Note**
>
> 可能需要幾分鐘的時間來訓練模型。訓練完成後，您會在儲存格的輸出中看到已完成訓練任務的訊息。

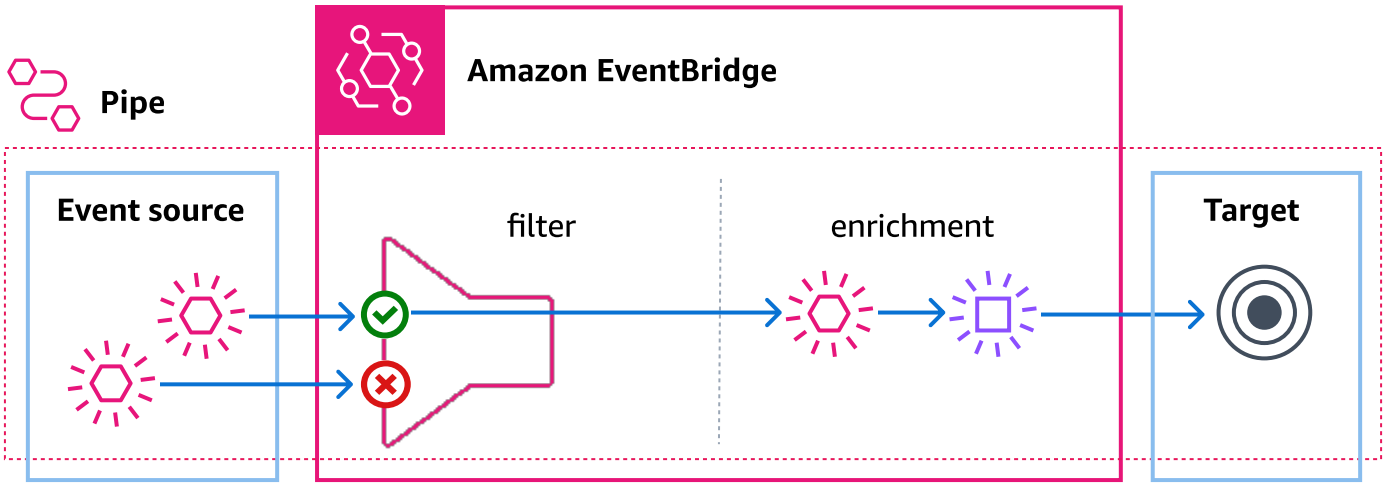
18. 執行儲存格`Stop and delete the endpoint`以清除資源。您也可以從 SageMaker 主控台停止和刪除執行個體

Amazon SQS

使用 EventBridge 管道將 Amazon SQS資料傳送至 Timestream

您可以使用 EventBridge 管道，將資料從 Amazon SQS佇列傳送至 Amazon Timestream 的 LiveAnalytics 資料表。

管道適用於 point-to-point支援來源和目標之間的整合，並支援進階轉換和擴充。管道可減少開發事件驅動架構時對專業知識和整合程式碼的需求。若要設定管道，您可以選擇來源、新增可選篩選、定義可選的擴充，以及選擇事件資料的目標。



如需 EventBridge 管道的詳細資訊，請參閱 EventBridge 使用者指南 中的EventBridge 管道。如需設定管道以將事件交付至 LiveAnalytics Amazon Timestream for Table 的相關資訊，請參閱EventBridge 管道目標詳細資訊 。

使用 DBeaver 來使用 Amazon Timestream

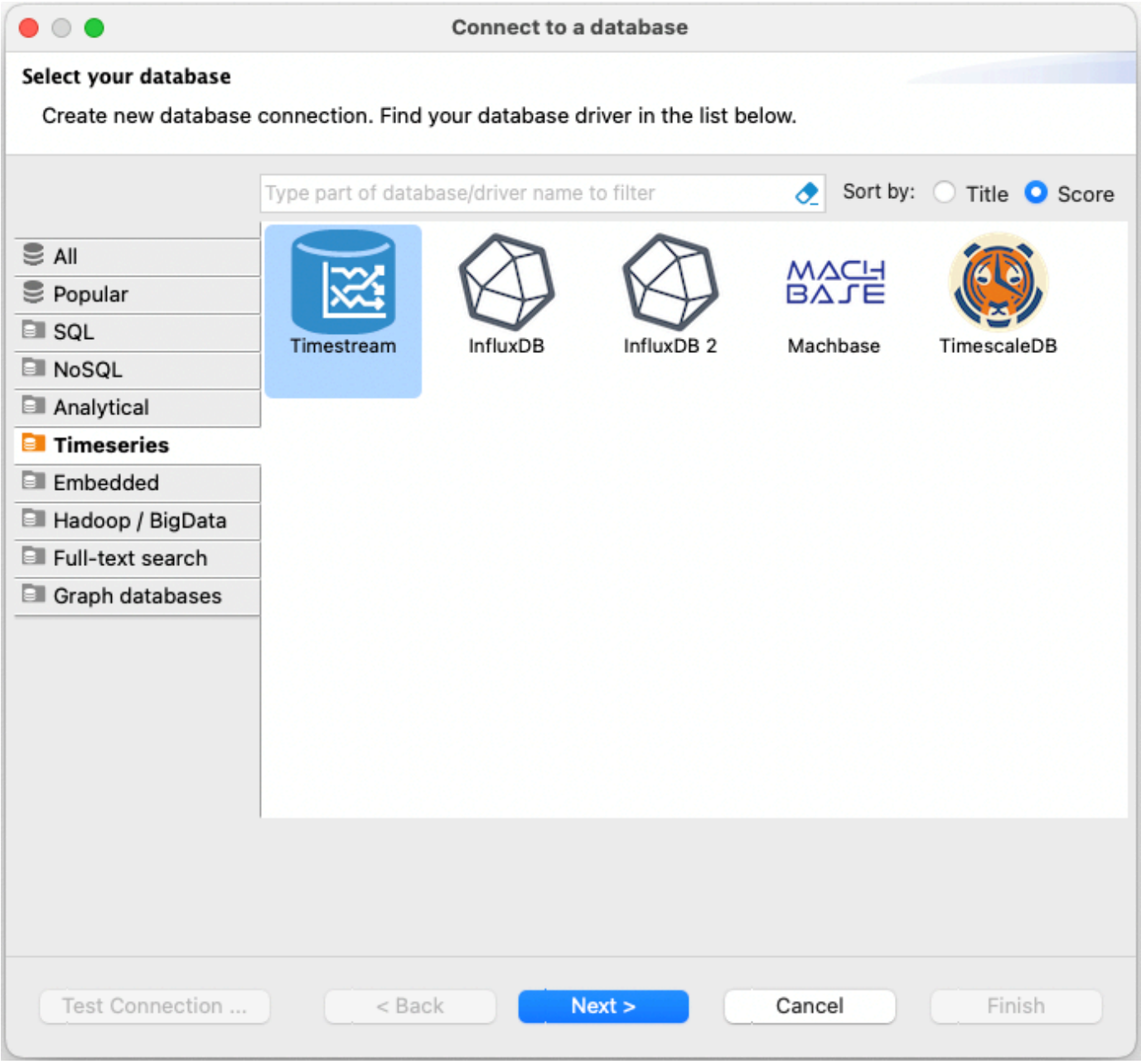
DBeaver 是免費的通用SQL用戶端，可用於管理具有JDBC驅動程式的任何資料庫。由於其強大的資料檢視、編輯和管理功能，因此廣泛用於開發人員和資料庫管理員。

使用 DBeaver的雲端連線選項，您可以原生DBeaver連線至 Amazon Timestream。DBeaver 提供全面且直覺的界面，可直接在DBeaver應用程式中使用時間序列資料。使用您的憑證，它也可讓您完整存取可從另一個查詢介面執行的任何查詢。它甚至可讓您建立圖形，以更好地了解和視覺化查詢結果。

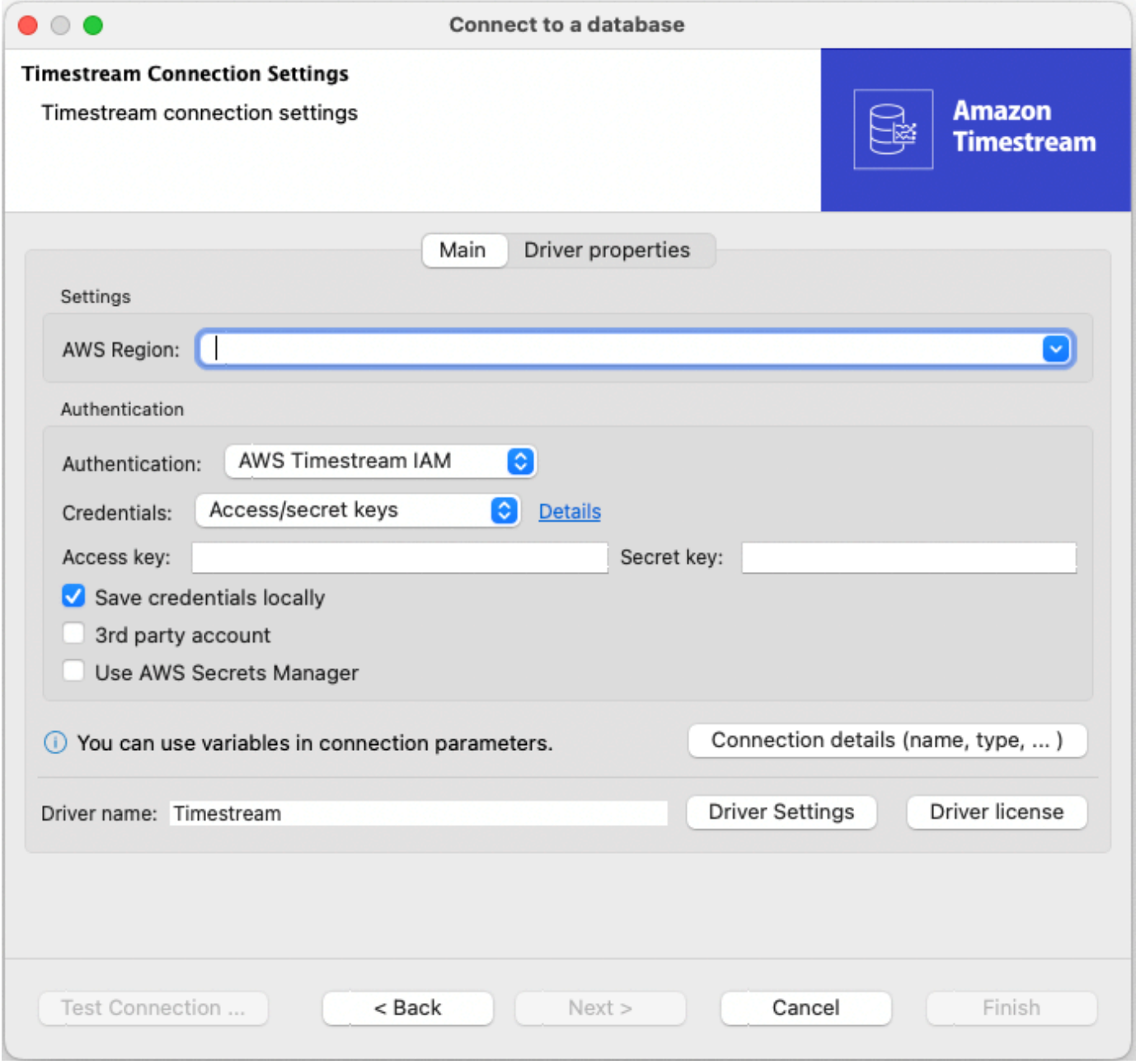
設定 DBeaver以使用 Timestream

採取下列步驟來設定 DBeaver 以使用 Timestream：

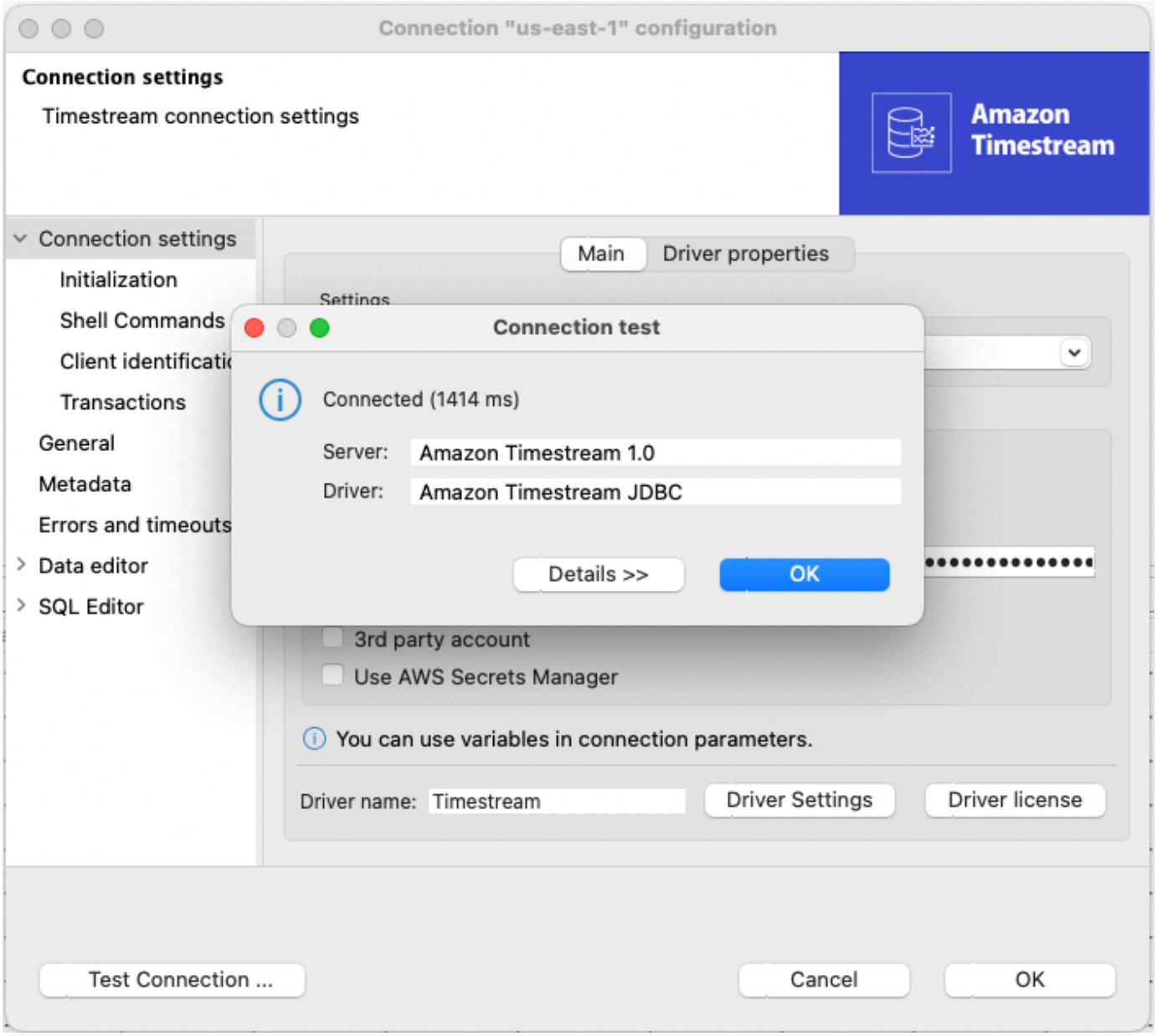
1. 在本機電腦上下載並安裝 DBeaver 。
2. 啟動 DBeaver，導覽至資料庫選取區域，在左側窗格中選擇 Timeseries，然後在右側窗格中選取 Timestream 圖示：



3. 在 Timestream Connection Settings 視窗中，輸入連線至 Amazon Timestream 資料庫所需的所有必要資訊。請確定您輸入的使用者金鑰具有存取 Timestream 資料庫所需的許可。此外，請務必確保您輸入的資訊和金鑰DBeaver安全且私密，如同任何敏感資訊一樣。



4. 測試連線以確保一切設定正確：



5. 如果連線測試成功，您現在可以與 Amazon Timestream 資料庫互動，就像與 中任何其他資料庫一樣DBeaver。例如，您可以導覽至SQL編輯器或 ER Diagram 檢視以執行查詢：

6. DBeaver 也提供強大的資料視覺化工具。若要使用它們，請執行查詢，然後選擇圖形圖示以視覺化結果集。圖形工具可協助您更了解一段時間內的資料趨勢。

將 Amazon Timestream 與 配對會DBeaver建立有效的環境來管理時間序列資料。您可以將其無縫整合到現有的工作流程中，以提高生產力和效率。

格拉法納

您可以使用 Grafana 視覺化時間序列資料並建立提醒。為了協助您開始使用資料視覺化，我們在 Grafana 中建立了範例儀表板，可將從 Python 應用程式傳送至 Timestream 的資料視覺化，並建立一段描述設定的影片教學課程。

主題

- 範例應用程式
- 影片教學課程

範例應用程式

1. 如需建立 資料庫詳細資訊，請依照 中所述的指示，在 Timestream 中建立資料庫和資料表。

   > **Note**
   >
   > Grafana 儀表板的預設資料庫名稱和資料表名稱 grafanaTable 分別設定為 grafanaDB 和 。使用這些名稱將設定降至最低。

2. 安裝 Python 3.7 或更新版本
3. 安裝和設定 Timestream Python SDK
4. 依照 的指示，將多執行緒 Python 應用程式的 GitHub 儲存庫持續擷取資料至 Timestream GitHub
5. 執行應用程式，依照 中的指示持續將資料擷取至 Timestream README
6. 完成 Amazon Managed Grafana 入門或完成安裝 Grafana 。
7. 如果安裝 Grafana 而非使用 Amazon Managed Grafana，請完成安裝 Grafana 的 Timestream 外掛程式。
8. 使用您選擇的瀏覽器開啟 Grafana 儀表板。如果您已在本機安裝 Grafana，您可以依照 Grafana 文件中所述的指示登入
9. 啟動 Grafana 後，前往資料來源，按一下新增資料來源，搜尋 Timestream，然後選擇 Timestream 資料來源
10. 設定 Auth Provider 和 區域，然後按一下儲存並測試
11. 設定預設巨集

    a. 將 $__database 設定為 Timestream 資料庫的名稱 （例如 grafanaDB ）

b. 將 $__table 設定為 Timestream 資料表的名稱 （ 例如 grafanaTable ）

c. 從 索引標籤將 $__measure 設定為最常用的量值

12. 按一下儲存並測試
13. 按一下儀表板索引標籤
14. 按一下匯入以匯入儀表板
15. 按兩下範例應用程式儀表板
16. 按一下儀表板設定
17. 選取變數
18. 變更 dbName 和 tableName 以符合 Timestream 資料庫和資料表的名稱
19. 按一下儲存
20. 重新整理儀表板
21. 若要建立警示，請依照 Grafana 文件中所述的指示建立 Grafana 受管警示規則
22. 若要對警示進行故障診斷，請依照 Grafana 文件中所述的說明進行故障診斷
23. 如需詳細資訊，請參閱 Grafana 文件

影片教學課程

本影片說明 Grafana 如何使用 Timestream。

使用 SquaredUp 搭配 Amazon Timestream

SquaredUp 是與 Amazon Timestream 整合的可觀測性平台。您可以使用 SquaredUp的直覺式儀表板設計工具來視覺化、分析和監控時間序列資料。儀表板可以公開或私下共用，而且可以建立通知管道，以便在監視器的運作狀態變更時提醒您。

SquaredUp 搭配 Amazon Timestream 使用

1. 註冊 SquaredUp 並免費開始使用。
2. 新增AWS 資料來源 。
3. 建立使用 Timestream Query 資料串流的儀表板動態磚。
4. 或者，啟用動態磚的監控、建立通知頻道，或公開或私下共用儀表板。
5. 選擇性地建立其他動態磚，以與其他監控和可觀測性工具的資料一起查看 Timestream 資料。

開放原始碼 Telegraf

您可以使用 Timestream for Telegraf 的 LiveAnalytics 輸出外掛程式，直接從開放原始碼 Telegraf 將指標寫入 Timestream for LiveAnalytics。

本節說明如何使用 LiveAnalytics 輸出外掛程式的 Timestream 安裝 Telegraf、如何使用 LiveAnalytics 輸出外掛程式的 Timestream 執行 Telegraf，以及開放原始碼 Telegraf 如何與 的 Timestream 搭配使用 LiveAnalytics。

主題

使用輸出外掛程式的 LiveAnalytics Timestream 安裝 Telegraf

自 1.16 版起，正式 Telegraf 版本中會提供 LiveAnalytics 輸出外掛程式的 Timestream。若要在大多數主要作業系統上安裝輸出外掛程式，請遵循 InfluxData Telegraf 文件 中概述的步驟。若要在 Amazon Linux 2 作業系統上安裝 ，請遵循下列指示。

使用 Amazon Linux 2 上的 LiveAnalytics 輸出外掛程式 Timestream 安裝 Telegraf

若要使用 Amazon Linux 2 上的 Timestream 輸出外掛程式安裝 Telegraf，請執行下列步驟。

1. 使用yum套件管理員安裝 Telegraf。

```
cat <<EOF | sudo tee /etc/yum.repos.d/influxdb.repo
[influxdb]
name = InfluxDB Repository - RHEL \$releasever
baseurl = https://repos.influxdata.com/rhel/\$releasever/\$basearch/stable
enabled = 1
gpgcheck = 1
gpgkey = https://repos.influxdata.com/influxdb.key
EOF
```

2. 執行下列命令。

```
sudo sed -i "s/\$releasever/$(rpm -E %{rhel})/g" /etc/yum.repos.d/influxdb.repo
```

3. 安裝和啟動 Telegraf。

```
sudo yum install telegraf
sudo service telegraf start
```

使用 LiveAnalytics 輸出外掛程式的 Timestream 執行 Telegraf

您可以依照下列指示，使用 LiveAnalytics 適用於外掛程式的 Timestream 執行 Telegraf。

1. 使用 Telegraf 產生範例組態。

```
telegraf --section-filter agent:inputs:outputs --input-filter cpu:mem --output-
filter timestream config > example.config
```

2. 使用管理主控台 、 CLI或 在 Timestream 中建立資料庫SDKs。
3. 在 example.config 檔案中，透過編輯 [[outputs.timestream]] 區段下的下列金鑰來新增資料庫名稱。

```
database_name = "yourDatabaseNameHere"
```

4. 根據預設，Telegraf 會建立資料表。如果您想要手動建立資料表，請將 create_table_if_not_exists設定為 false，並遵循指示使用管理主控台 、 CLI或 建立資料表SDKs。
5. 在 example.config 檔案中，在 [[outputs.timestream]] 區段下設定憑證。憑證應允許下列操作。

```
timestream:DescribeEndpoints
timestream:WriteRecords
```

> **Note**
>
> 如果您將 create_table_if_not_exists 設為 true，請包含：
>
> ```
> timestream:CreateTable
> ```

> **Note**
>
> 如果您將 `describe_database_on_start`設定為 `true`，請包含下列項目。
>
> ```
> timestream:DescribeDatabase
> ```

6. 您可以根據您的偏好設定編輯其餘組態。
7. 完成組態檔案的編輯後，請執行 Telegraf 並執行下列操作。

```
./telegraf --config example.config
```

8. 指標應該會在幾秒內顯示，具體取決於您的客服人員組態。您也應該會在 Timestream 主控台中看到新的資料表、cpu 和 mem 。

將 Telegraf/InfluxDB 指標對應至模型的 LiveAnalytics Timestream

將資料從 Telegraf 寫入 的 Timestream 時 LiveAnalytics，資料會對應如下。

- 時間戳記會寫入為時間欄位。
- 標籤會寫入維度。
- 欄位會寫入為量值。
- 測量大部分以資料表名稱寫入（下文將詳細說明）。

Telegraf LiveAnalytics 輸出外掛程式的 Timestream 提供多種選項，可用於組織和儲存 的 Timestream 中的資料 LiveAnalytics。這可以用以行通訊協定格式的資料開頭的範例來描述。

```
weather,location=us-midwest,season=summer temperature=82,humidity=71
1465839830100400200 airquality,location=us-west no2=5,pm25=16
1465839830100400200
```

下列說明資料。

- 測量名稱為 `weather`和 `airquality`。
- 標籤為 `location`和 `season`。
- 欄位為 `temperature`、`humidity`、 `no2`和 `pm25`。

主題

將資料存放在多個資料表中

您可以選擇為每個測量建立單獨的資料表，並將每個欄位存放在每個資料表的個別資料列中。

組態為 `mapping_mode = "multi-table"`。

- LiveAnalytics 轉接器的 Timestream 將建立兩個資料表，即 `weather`和 `airquality`。
- 每個資料表列只會包含單一欄位。

LiveAnalytics 資料表 `weather`和 產生的 Timestream `airquality`會如下所示。

`weather`

time	location	季節	measure_name	measure_value::bigint
2016-06-13 17：43：50	us-中西部	夏季	溫度	82
2016-06-13 17：43：50	us-中西部	夏季	濕度	71

`airquality`

time	location	measure_name	measure_value::bigint
2016-06-13 17：43：50	us-中西部	no2	5
2016-06-13 17：43：50	us-中西部	pm25	16

將資料存放在單一資料表中

您可以選擇將所有測量存放在單一資料表中，並將每個欄位存放在個別資料表列中。

組態為 mapping_mode = "single-table"。使用 single-table、 single_table_name和時，有兩個額外的組態single_table_dimension_name_for_telegraf_measurement_name。

- LiveAnalytics 輸出外掛程式的 Timestream 會建立名為 的單一資料表 *<single_table_name>* 其中包括 *<single_table_dimension_name_for_telegraf_measurement_name>* 資料欄。
- 資料表可在單一資料表列中包含多個欄位。

LiveAnalytics 資料表產生的 Timestream 會如下所示。

weather

time	location	季節	***<single_table_dimension_name_for_telegraf_measurement_name>***	measure_name	measure_value::bigint
2016-06-13 17：43：50	us-中西部	夏季	天氣	溫度	82
2016-06-13 17：43：50	us-中西部	夏季	天氣	濕度	71
2016-06-13 17：43：50	us-中西部	夏季	空氣品質	no2	5
2016-06-13 17：43：50	us-中西部	夏季	天氣	pm25	16

JDBC

您可以使用 Java Database Connectivity （JDBC） 連線，將 的 Timestream LiveAnalytics 連線至您的商業智慧工具和其他應用程式，例如 SQL Workbench 。驅動程式的 LiveAnalytics JDBC Timestream 目前SSO支援 Okta 和 Microsoft Azure AD。

主題

- 為 設定 Timestream 的JDBC驅動程式 LiveAnalytics
- 連線屬性
- JDBC URL 範例
- 使用 Okta 設定單一登入身分驗證的 LiveAnalytics JDBC Timestream
- 使用 Microsoft Azure AD 設定單一登入身分驗證的 LiveAnalytics JDBC Timestream

為 設定 Timestream 的JDBC驅動程式 LiveAnalytics

請依照下列步驟設定JDBC驅動程式。

主題

- 驅動程式的時間 LiveAnalytics JDBC串流 JARs
- 驅動程式類別和URL格式的時間串流 LiveAnalytics JDBC
- 範例應用程式

驅動程式的時間 LiveAnalytics JDBC串流 JARs

您可以透過直接下載或將驅動程式新增為 Maven 相依性來取得驅動程式的 LiveAnalytics JDBC Timestream。

- 直接下載：。若要直接下載JDBC驅動程式的 LiveAnalytics Timestream，請完成下列步驟：

  1. 導覽至 https://github.com/awslabs/amazon-timestream-driver-jdbc/版本
  2. 您可以`amazon-timestream-jdbc-1.0.1-shaded.jar`直接搭配商業智慧工具和應用程式使用
  3. 下載`amazon-timestream-jdbc-1.0.1-javadoc.jar`至您選擇的目錄。
  4. 在您下載 的目錄中`amazon-timestream-jdbc-1.0.1-javadoc.jar`，執行下列命令以擷取 Javadoc HTML 檔案：

```
jar -xvf amazon-timestream-jdbc-1.0.1-javadoc.jar
```

- 作為 Maven 相依性：若要將驅動程式的 LiveAnalytics JDBC Timestream 新增為 Maven 相依性，請完成下列步驟：

  1. 在您選擇的編輯器中導覽並開啟應用程式pom.xml的檔案。
  2. 將JDBC驅動程式新增為應用程式的 pom.xml 檔案的相依性：

```
<!-- https://mvnrepository.com/artifact/software.amazon.timestream/amazon-
timestream-jdbc -->
<dependency>
    <groupId>software.amazon.timestream</groupId>
    <artifactId>amazon-timestream-jdbc</artifactId>
    <version>1.0.1</version>
</dependency>
```

驅動程式類別和URL格式的時間串流 LiveAnalytics JDBC

Timestream for driver 的 LiveAnalytics JDBC驅動程式類別為：

```
software.amazon.timestream.jdbc.TimestreamDriver
```

Timestream JDBC驅動程式需要下列JDBCURL格式：

```
jdbc:timestream:
```

若要透過 JDBC 指定資料庫屬性URL，請使用下列URL格式：

```
jdbc:timestream://
```

範例應用程式

為了協助您開始使用 的 Timestream LiveAnalytics 搭配 JDBC，我們已在 中建立全功能範例應用程式 GitHub。

1. 按照此處所述的說明，使用範例資料建立資料庫。
2. 複製範例應用程式的 GitHub 儲存庫，以遵循 的指示GitHub。 JDBC

3. 請遵循 中的指示README，開始使用範例應用程式。

連線屬性

驅動程式的 LiveAnalytics JDBC Timestream 支援下列選項：

主題

> **Note**
>
> 如果未提供任何屬性，驅動程式的 LiveAnalytics JDBC Timestream 將使用預設憑證鏈載入憑證。

> **Note**
>
> 所有屬性金鑰都區分大小寫。

基本身分驗證選項

下表說明可用的基本身分驗證選項。

選項	描述	預設
AccessKeyId	AWS 使用者存取金鑰 ID。	NONE

選項	描述	預設
SecretAccessKey	AWS 使用者秘密存取金鑰。	NONE
SessionToken	存取已啟用多重要素身分驗證（MFA）的資料庫所需的臨時工作階段權杖。	NONE

標準用戶端資訊選項

下表說明標準用戶端資訊選項。

選項	描述	預設
ApplicationName	目前使用連線的應用程式名稱。 `ApplicationName` 用於偵錯目的，不會與 Timestream 通訊以進行 LiveAnalytics服務。	驅動程式偵測到的應用程式名稱。

驅動程式組態選項

下表說明驅動程式組態選項。

選項	描述	預設
EnableMetaDataPreparedStatement	讓驅動程式能夠 LiveAnalytics JDBC傳回 的中繼資料`PreparedStatements`，但在擷取中繼資料 LiveAnalytics 時，使用 Timestream 會產生額外的成本。	FALSE
區域	資料庫的區域。	us-east-1

SDK 選項

下表說明 SDK選項。

選項	描述	預設
RequestTimeout	將在 AWS SDK逾時之前等待查詢請求的時間，以毫秒為單位。非正值會停用請求逾時。	0
SocketTimeout	將在逾時之前等待透過開放連線傳輸資料的時間， AWS SDK以毫秒為單位。值必須為非負值。的值會0停用通訊端逾時。	50000
MaxRetryCountClient	中具有 5XX 錯誤碼的可重試錯誤重試次數上限SDK。值必須為非負值。	NONE
MaxConnections	LiveAnalytics 服務與 Timestream 的允許並行開啟HTTP連線數目上限。值必須為正數。	50

端點組態選項

下表說明 Endpoint Configuration 選項。

選項	描述	預設
端點	Timestream for LiveAnalytics Service 的端點。	NONE

憑證提供者選項

下表說明可用的憑證提供者選項。

選項	描述	預設
AwsCredentialsProviderClass	InstanceProfileCredentialsProvider 用於身分驗證的 PropertiesFileCredentialsProvider 或 之一。	NONE
CustomCredentialsFilePath	屬性檔案的路徑，其中包含 AWS 安全憑證accessKey 和 secretKey 。只有在 指定為 PropertiesFileCredentialsProvider 時AwsCredentialsProviderClass ，才需要這樣做。	NONE

SAMLOkta 的 型身分驗證選項

下表說明 Okta 可用的 SAML型身分驗證選項。

選項	描述	預設
IdpName	用於 SAML型身分驗證的 Identity Provider （ Idp ） 名稱。Okta 或 之一AzureAD。	NONE
IdpHost	指定 Idp 的主機名稱。	NONE
IdpUserName	指定 Idp 帳戶的使用者名稱。	NONE
IdpPassword	指定 Idp 帳戶的密碼。	NONE
OktaApplicationID	與應用程式 Timestream LiveAnalytics 相關聯的唯一 Okta 提供的 ID。 AppId可在應用程式中繼資料中提供的entityID欄位中找到。請	NONE

選項	描述	預設
	考慮下列範例： `entityID = http://www.okta.com//IdpAppID`	
角色ARN	呼叫者所擔任角色的 Amazon Resource Name （ ARN ）。	NONE
IdpARN	描述 Idp 的 中SAML提供者的 IAM Amazon Resource Name （ ARN ）。	NONE

SAMLAzure AD 的 型身分驗證選項

下表說明 Azure AD 可用的 SAML型身分驗證選項。

選項	描述	預設
IdpName	用於 SAML型身分驗證的 Identity Provider （ Idp ） 名稱。`Okta` 或 `AzureAD` 之一。	NONE
IdpHost	指定 Idp 的主機名稱。	NONE
IdpUserName	指定 Idp 帳戶的使用者名稱。	NONE
IdpPassword	指定 Idp 帳戶的密碼。	NONE
AADApplicationID	Azure AD 上已註冊應用程式的唯一 ID。	NONE
AADClientSecret	與 Azure AD 上註冊應用程式相關聯的用戶端秘密，用於授權擷取權杖。	NONE
AADTenant	Azure AD 租用戶 ID。	NONE

選項	描述	預設
IdpARN	描述 Idp 的 中SAML提供者的 IAM Amazon Resource Name （ARN）。	NONE

JDBC URL 範例

本節說明如何建立JDBC連線 URL，並提供範例。若要指定選用的連線屬性 ，請使用下列URL格式：

```
jdbc:timestream://PropertyName1=value1;PropertyName2=value2...
```

> **Note**
>
> 所有連線屬性皆為選用。所有屬性金鑰都區分大小寫。

以下是JDBC連線 的一些範例URLs。

具有基本身分驗證選項和區域的範例：

```
jdbc:timestream://
AccessKeyId=<myAccessKeyId>;SecretAccessKey=<mySecretAccessKey>;SessionToken=<mySessionToken
east-1
```

用戶端資訊、區域和SDK選項的範例：

```
jdbc:timestream://ApplicationName=MyApp;Region=us-
east-1;MaxRetryCountClient=10;MaxConnections=5000;RequestTimeout=20000
```

使用環境變數中具有憑證集的預設 AWS 憑證提供者鏈進行連線：

```
jdbc:timestream
```

在連線 中使用具有憑證集的預設 AWS 憑證提供者鏈進行連線URL：

```
jdbc:timestream://
AccessKeyId=<myAccessKeyId>;SecretAccessKey=<mySecretAccessKey>;SessionToken=<mySessionToken
```

使用 PropertiesFileCredentialsProvider 作為身分驗證方法進行連線：

```
jdbc:timestream://
AwsCredentialsProviderClass=PropertiesFileCredentialsProvider;CustomCredentialsFilePath=<pat
 to properties file>
```

使用 InstanceProfileCredentialsProvider 作為身分驗證方法進行連線：

```
jdbc:timestream://AwsCredentialsProviderClass=InstanceProfileCredentialsProvider
```

使用 Okta 憑證做為身分驗證方法進行連線：

```
jdbc:timestream://
IdpName=Okta;IdpHost=<host>;IdpUserName=<name>;IdpPassword=<password>;OktaApplicationID=<id>
```

使用 Azure AD 憑證做為身分驗證方法進行連線：

```
jdbc:timestream://
IdpName=AzureAD;IdpUserName=<name>;IdpPassword=<password>;AADApplicationID=<id>;AADClientSec
```

與特定端點連線：

```
jdbc:timestream://Endpoint=abc.us-east-1.amazonaws.com;Region=us-east-1
```

使用 Okta 設定單一登入身分驗證的 LiveAnalytics JDBC Timestream

的 Timestream LiveAnalytics 支援使用 Okta 進行 LiveAnalytics JDBC單一登入身分驗證的 Timestream。若要使用 Timestream 搭配 Okta 進行 LiveAnalytics JDBC單一登入身分驗證，請完成下列每個區段。

主題

- 必要條件
- AWS Okta 中的帳戶聯合
- 為 設定 Okta SAML

必要條件

在透過 Okta 使用 Timestream 進行 LiveAnalyticsJDBC單一登入身分驗證之前，請確定您符合下列先決條件：

- 中的管理員許可， AWS 以建立身分提供者和角色 。
- Okta 帳戶 （ 前往 https://www.okta.com/login/建立帳戶 ）。
- 存取適用於 的 Amazon Timestream LiveAnalytics。

現在您已完成先決條件，您可以繼續 AWS Okta 中的帳戶聯合。

AWS Okta 中的帳戶聯合

驅動程式的 LiveAnalytics JDBC Timestream 支援 Okta 中的 AWS 帳戶聯合。若要在 Okta 中設定 AWS 帳戶聯合，請完成下列步驟：

1. 使用下列 登入 Okta Admin 儀表板URL：

```
https://<company-domain-name>-admin.okta.com/admin/apps/active
```

> **Note**
>
> 將 < company-domain-name > 取代為您的網域名稱。

2. 成功登入後，請選擇新增應用程式並搜尋AWS 帳戶聯合 。
3. 選擇新增
4. 將登入變更為URL適當的 URL。
5. 選擇下一步
6. 選擇 SAML 2.0 作為登入方法
7. 選擇身分提供者中繼資料以開啟中繼資料XML檔案。在本機儲存檔案。
8. 將所有其他組態選項保留空白。
9. 選擇完成

現在您已完成 Okta 的帳戶 AWS 聯合，您可以繼續 為 設定 Okta SAML。

為 設定 Okta SAML

1. 選擇 Sign On (登入) 標籤。選擇檢視 。
2. 在設定區段中選擇設定指示按鈕。

尋找 Okta 中繼資料文件

1. 若要尋找文件，請前往：

```
https://<domain>-admin.okta.com/admin/apps/active
```

> **Note**
>
> <domain> 是您 Okta 帳戶的唯一網域名稱。

2. 選擇AWS 帳戶聯合應用程式
3. 選擇登入索引標籤

使用 Microsoft Azure AD 設定單一登入身分驗證的 LiveAnalytics JDBC Timestream

的 Timestream LiveAnalytics 支援使用 Microsoft Azure AD 進行 LiveAnalytics JDBC單一登入身分驗證的 Timestream。若要使用 Timestream 搭配 Microsoft Azure AD 進行 LiveAnalytics JDBC單一登入身分驗證，請完成下列每個區段。

主題

- 必要條件
- 設定 Azure AD
- 在 中設定IAM身分提供者和角色 AWS

必要條件

在搭配 Microsoft Azure AD 使用 Timestream 進行 LiveAnalyticsJDBC單一登入身分驗證之前，請確定您符合下列先決條件：

- 中的管理員許可， AWS 以建立身分提供者和角色 。
- Azure Active Directory 帳戶 （ 前往 https://azure.microsoft.com/en-ca/services/active-directory/ 建立帳戶 ）

- 存取適用於 的 Amazon Timestream LiveAnalytics。

設定 Azure AD

1. 登入 Azure 入口網站
2. 在 Azure 服務清單中選擇 Azure Active Directory。這將重新導向至預設目錄頁面。
3. 在側邊欄的管理區段下選擇企業應用程式
4. 選擇 + 新應用程式 。
5. 尋找並選取 Amazon Web Services 。
6. 在側邊欄中的管理區段下選擇單一登入
7. 選擇 SAML作為單一登入方法
8. 在基本SAML組態區段中，URL針對識別符和回覆 輸入下列項目URL：

```
https://signin.aws.amazon.com/saml
```

9. 選擇儲存
10. 在SAML簽署憑證區段XML中下載聯合中繼資料。這將在稍後建立 IAM Identity Provider 時使用
11. 返回預設目錄頁面，並在管理 下選擇應用程式註冊。
12. 從所有應用程式區段中選擇 的時間串流 LiveAnalytics。頁面將重新導向至應用程式的概觀頁面

> **Note**
>
> 請注意應用程式（用戶端）ID 和目錄（租戶）ID。建立連線時，需要這些值。

13. 選擇憑證和秘密
14. 在用戶端秘密 下，使用 + 新用戶端秘密 建立新的用戶端秘密。

> **Note**
>
> 請注意產生的用戶端秘密，因為在建立 的 Timestream 連線時，這是必要的 LiveAnalytics。

15. 在 管理 下的側邊列上，選取API許可
16. 在已設定許可 中，使用新增許可授予 Azure AD 登入 Timestream for 的許可 LiveAnalytics。在請求API許可頁面上選擇 Microsoft Graph。
17. 選擇委派許可，然後選擇 User.Read 許可

18選擇新增許可

19選擇預設目錄的授予管理員同意

在 中設定IAM身分提供者和角色 AWS

完成以下每個區段，以設定 IAM Timestream 使用 Microsoft Azure AD 進行 LiveAnalytics JDBC單一登入身分驗證：

主題

- 建立SAML身分提供者
- 建立 IAM 角色
- 建立IAM政策
- 佈建中

建立SAML身分提供者

若要為使用 Microsoft Azure AD 進行 LiveAnalytics JDBC單一登入身分驗證的 Timestream 建立SAML身分提供者，請完成下列步驟：

1. 登入 AWS 管理主控台
2. 選擇 服務，並在安全、身分和合規IAM下選取
3. 在存取管理下選擇身分提供者
4. 選擇建立提供者，然後選擇 SAML作為提供者類型。輸入供應商名稱 。此範例將使用 AzureADProvider。
5. 上傳先前下載的聯合中繼資料XML檔案
6. 選擇下一個 ，然後選擇建立 。
7. 完成後，頁面將重新導向回 Identity Provider 頁面

建立 IAM 角色

若要為使用 Microsoft Azure AD 進行單一登入身分驗證的 LiveAnalytics JDBC Timestream 建立IAM角色，請完成下列步驟：

1. 在側邊列上選取存取管理下的角色

2. 選擇 Create role (建立角色)。
3. 選擇 SAML 2.0 聯合作為受信任實體
4. 選擇 Azure AD 供應商
5. 選擇允許程式設計 AWS 和管理主控台存取
6. 選擇 Next: Permissions (下一步：許可)
7. 附加許可政策或繼續至下一步：標籤
8. 新增選用標籤或繼續下一步：檢閱
9. 輸入 Role name (角色名稱)。此範例將使用 AzureSAMLRole
10提供角色描述
11選擇建立角色以完成

建立IAM政策

若要為使用 Microsoft Azure AD 進行 LiveAnalytics JDBC單一登入身分驗證的 Timestream 建立IAM政策，請完成下列步驟：

1. 在側邊列中，選擇存取管理下的政策
2. 選擇建立政策，然後選擇JSON索引標籤
3. 新增下列政策

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

4. 選擇 Create policy (建立政策)
5. 輸入政策名稱。此範例將使用 TimestreamAccessPolicy。

6. 選擇建立政策
7. 在側邊列中，選擇存取管理下的角色。
8. 選擇先前建立的 Azure AD 角色，然後選擇許可下的附加政策。
9. 選取先前建立的存取政策。

佈建中

若要為使用 Microsoft Azure AD 進行單一登入身分驗證的 LiveAnalytics JDBC Timestream 佈建身分提供者，請完成下列步驟：

1. 返回 Azure 入口網站
2. 在 Azure 服務清單中選擇 Azure Active Directory。這將重新導向至預設目錄頁面
3. 在側邊欄的管理區段下選擇企業應用程式
4. 選擇佈建
5. 選擇佈建方法的自動模式
6. 在管理憑證下，輸入用戶端秘密的 AwsAccessKeyID，以及秘密權杖SecretAccessKey的 ID
7. 將佈建狀態設定為開啟
8. 選擇儲存 。這可讓 Azure AD 載入必要的IAM角色
9. 完成目前週期狀態後，選擇側邊欄上的使用者和群組
10選擇 + 新增使用者
11選擇 Azure AD 使用者，以提供 的 Timestream 存取權 LiveAnalytics
12選擇在 中建立的 IAM Azure AD 角色和對應的 Azure Identity Provider AWS
13選擇指派

ODBC

的 Amazon Timestream 開放原始碼ODBC驅動程式 LiveAnalytics 為開發人員 LiveAnalytics 提供 Timestream 的 SQL相關介面，並啟用 Power BI Desktop 和 Microsoft Excel 等商業智慧（BI）工具的連線。驅動程式的 LiveAnalytics ODBC Timestream 目前可在 Windows、macOS 和 Linux 上使用，也SSO支援 Okta 和 Microsoft Azure Active Directory（AD）。

如需詳細資訊，請參閱 上的 ODBC Amazon Timestream 驅動程式 LiveAnalytics文件 GitHub。

主題

設定驅動程式的 LiveAnalytics ODBC Timestream

設定您 AWS 帳戶中 LiveAnalytics 的 Timestream 存取權

如果您尚未設定 AWS 帳戶以使用 的 Timestream LiveAnalytics，請遵循 中的 insructions存取 的 Timestream LiveAnalytics。

在系統上安裝ODBC驅動程式

從ODBC GitHub儲存庫 下載適用於您系統的 Timestream ODBC驅動程式安裝程式，並遵循適用於您系統的安裝指示：

- Windows 安裝指南
- MacOS 安裝指南
- Linux 安裝指南

設定ODBC驅動程式的資料來源名稱 （DSN）

請遵循 系統的DSN組態指南中的指示：

- Windows DSN組態
- MacOS DSN組態
- Linux DSN組態

設定您的商業智慧 （BI） 應用程式以與ODBC驅動程式搭配使用

以下是設定多個常用 BI 應用程式以與ODBC驅動程式搭配使用的指示：

- 設定 Microsoft Power BI。
- 設定 Microsoft Excel

- 設定 Tableau

對於其他應用程式

ODBC 驅動程式的連線字串語法和選項

指定ODBC驅動程式連線字串選項的語法如下：

```
DRIVER={Amazon Timestream ODBC Driver};(option)=(value);
```

可用選項如下：

驅動程式連線選項

- **Driver**（必要）– 與 搭配使用的驅動程式ODBC。

  預設值為 Amazon Timestream。
- **DSN** – 用於設定連線的資料來源名稱（DSN）。

  預設值為 NONE。
- **Auth** – 身分驗證模式。這必須是下列其中一項：
  - AWS_PROFILE – 使用預設憑證鏈。
  - IAM – 使用 AWS IAM憑證。
  - AAD – 使用 Azure Active Directory（AD）身分提供者。
  - OKTA – 使用 Okta 身分提供者。

  預設值為 AWS_PROFILE。

端點組態選項

- **EndpointOverride** – Timestream for LiveAnalytics Service 的端點覆寫。這是覆寫區域的進階選項。例如：

  ```
  query-cell2.timestream.us-east-1.amazonaws.com
  ```
- **Region** – LiveAnalytics 服務端點 Timestream 的簽署區域。

  預設值為 us-east-1。

憑證提供者選項

- **ProfileName** – AWS 組態檔案中的設定檔名稱。

  預設值為 NONE。

AWS IAM 身分驗證選項

- **UID** 或 **AccessKeyId** – AWS 使用者存取金鑰 ID。如果連線字串中同時提供 AccessKeyId UID和 ，除非值為空，否則將使用 UID值。

  預設值為 NONE。
- **PWD** 或 **SecretKey** – AWS使用者秘密存取金鑰。如果連線字串中同時提供 SecretKey PWD和 ，則除非值為空白，否則將使用 搭配 PWD的值。

  預設值為 NONE。
- **SessionToken** – 存取已啟用多重要素身分驗證 （ MFA ） 的資料庫所需的臨時工作階段權杖。請勿在輸入 = 中包含尾隨。

  預設值為 NONE。

SAMLOkta 的 型身分驗證選項

- **IdPHost** – 指定 IdP 的主機名稱。

  預設值為 NONE。
- **UID** 或 **IdPUserName** – 指定 IdP 帳戶的使用者名稱。如果連線字串中同時提供 IdPUserName UID和 ，則除非值為空白，否則將使用 UID值。

  預設值為 NONE。
- **PWD** 或 **IdPPassword** – 指定 IdP 帳戶的密碼。如果連線字串中同時提供 IdPPassword PWD和 ，則除非值為空白，否則將使用 PWD值。

  預設值為 NONE。
- **OktaApplicationID** – 與 LiveAnalytics 應用程式 Timestream 相關聯的唯一 Okta 提供的 ID。尋找應用程式 ID （ AppId ） 的位置位於應用程式中繼資料中提供的entityID欄位中。例如：

```
entityID="http://www.okta.com//(IdPAppID)
```

預設值為 NONE。

- **RoleARN** – 呼叫者所擔任角色的 Amazon Resource Name （ARN）。

  預設值為 NONE。

- **IdPARN** – IAM描述 IdP 的 中SAML提供者的 Amazon Resource Name （ARN）。

  預設值為 NONE。

SAMLAzure Active Directory 的 型身分驗證選項

- **UID** 或 **IdPUserName** – 指定 IdP 帳戶的使用者名稱。

  預設值為 NONE。

- **PWD** 或 **IdPPassword** – 指定 IdP 帳戶的密碼。

  預設值為 NONE。

- **AADApplicationID** – Azure AD 上已註冊應用程式的唯一 ID。

  預設值為 NONE。

- **AADClientSecret** – 與 Azure AD 上註冊應用程式相關聯的用戶端秘密，用於授權擷取權杖。

  預設值為 NONE。

- **AADTenant** – Azure AD 租用戶 ID。

  預設值為 NONE。

- **RoleARN** – 呼叫者所擔任角色的 Amazon Resource Name （ARN）。

  預設值為 NONE。

- **IdPARN** – IAM描述 IdP 的 中SAML提供者的 Amazon Resource Name （ARN）。

  預設值為 NONE。

AWS SDK （進階） 選項

- **RequestTimeout** – 在逾時之前等待查詢請求的時間， AWS SDK以毫秒為單位。任何非正值都會停用請求逾時。

預設值為 3000。

- **ConnectionTimeout** – 在逾時之前， 等待透過開放連線傳輸資料的時間， AWS SDK以毫秒為單位。值 0 會停用連線逾時。此值不得為負數。

  預設值為 1000。

- **MaxRetryCountClient** – 在 中使用 5xx 錯誤碼重試可重試錯誤的次數上限SDK。值不得為負。

  預設值為 0。

- **MaxConnections** – Timestream 服務允許的同時開啟HTTP連線數目上限。值必須為正數。

  預設值為 25。

ODBC 驅動程式記錄選項

- **LogLevel** – 驅動程式記錄的日誌層級。必須是下列其中一個：
  - 0（OFF）。
  - 1（ERROR）。
  - 2（WARNING）。
  - 3（INFO）。
  - 4（DEBUG）。

  預設值為 1（ERROR）。

  警告：使用記錄模式時，驅動程式可以DEBUG記錄個人資訊。

- **LogOutput** – 存放日誌檔案的資料夾。

  預設值為：

  - Windows：%USERPROFILE%，如果無法使用，則為 %HOMEDRIVE%%HOMEPATH%。
  - macOS 和 Linux：$HOME，如果無法使用，則為函數getpwuid(getuid())傳回值pw_dir中的欄位。

SDK 記錄選項

AWS SDK 日誌層級與驅動程式日誌層級的 LiveAnalytics ODBC Timestream 分開。設定一個不會影響另一個。

SDK 日誌層級是使用環境變數 設定TS_AWS_LOG_LEVEL。有效的 值如下：

- OFF
- ERROR
- WARN
- INFO
- DEBUG
- TRACE
- FATAL

如果 TS_AWS_LOG_LEVEL 未設定，則SDK日誌層級會設為預設值，即 WARN。

透過代理連線

ODBC 驅動程式支援 LiveAnalytics 透過代理連線至 的 Amazon Timestream。若要使用此功能，請根據您的代理設定設定下列環境變數：

- **TS_PROXY_HOST** – 代理主機。
- **TS_PROXY_PORT** – 代理連接埠號碼。
- **TS_PROXY_SCHEME** – 代理結構描述，或 http https。
- **TS_PROXY_USER** – 代理身分驗證的使用者名稱。
- **TS_PROXY_PASSWORD** – 代理身分驗證的使用者密碼。
- **TS_PROXY_SSL_CERT_PATH** – 用於連線至HTTPS代理的SSL憑證檔案。
- **TS_PROXY_SSL_CERT_TYPE** – 代理用戶端SSL憑證的類型。
- **TS_PROXY_SSL_KEY_PATH** – 用來連線至HTTPS代理的私有金鑰檔案。
- **TS_PROXY_SSL_KEY_TYPE** – 用來連線至HTTPS代理的私有金鑰檔案類型。
- **TS_PROXY_SSL_KEY_PASSWORD** – 用於連線至HTTPS代理的私有金鑰檔案的密碼。

Timestream for LiveAnalytics ODBC driver 的連線字串範例

使用 IAM 憑證連線至ODBC驅動程式的範例

```
Driver={Amazon Timestream ODBC Driver};Auth=IAM;AccessKeyId=(your access key
 ID);secretKey=(your secret key);SessionToken=(your session token);Region=us-east-2;
```

使用設定檔連線至ODBC驅動程式的範例

```
Driver={Amazon Timestream ODBC Driver};ProfileName=(the profile name);region=us-west-2;
```

驅動程式會嘗試使用 中提供的憑證進行連線~/.aws/credentials，或者如果環境變數 中指定了檔案AWS_SHARED_CREDENTIALS_FILE，則會使用該檔案中的憑證進行連線。

使用 Okta 連線至ODBC驅動程式的範例

```
driver={Amazon Timestream ODBC Driver};auth=okta;region=us-west-2;idPHost=(your host at
 Okta);idPUsername=(your user name);idPPassword=(your password);OktaApplicationID=(your
 Okta AppId);roleARN=(your role ARN);idPARN=(your Idp ARN);
```

使用 Azure Active Directory （ AAD ） 連線至ODBC驅動程式的範例

```
driver={Amazon Timestream ODBC Driver};auth=aad;region=us-west-2;idPUsername=(your
 user name);idPPassword=(your password);aadApplicationID=(your AAD
 AppId);aadClientSecret=(your AAD client secret);aadTenant=(your AAD
 tenant);roleARN=(your role ARN);idPARN=(your idP ARN);
```

使用指定的端點和 2 （ WARNING ） 的日誌層級連線到ODBC驅動程式的範例

```
Driver={Amazon Timestream ODBC Driver};Auth=IAM;AccessKeyId=(your access
 key ID);secretKey=(your secret key);EndpointOverride=ingest.timestream.us-
west-2.amazonaws.com;Region=us-east-2;LogLevel=2;
```

對與ODBC驅動程式的連線進行故障診斷

> **Note**
>
> 當使用者名稱和密碼已在 中指定時DSN，當ODBC驅動程式管理員要求使用者名稱和密碼時，就不需要再次指定使用者名稱和密碼。

當連線字串選項在連線字串中傳遞超過一次時，Re-writing *(connection string option)* (have you specified it several times?會發生01S02具有訊息的 錯誤碼。多次指定選項會導致錯誤。與 DSN和 連線字串建立連線時，如果已在 中指定連線選項DSN，請勿在連線字串中再次指定。

VPC 端點（AWS PrivateLink）

您可以建立介面VPC端點 ，LiveAnalytics 在 VPC與 Amazon Timestream 之間建立私有連線。如需詳細資訊，請參閱VPC 端點（AWS PrivateLink）。

最佳實務

若要完全實現 Amazon Timestream 對 的好處 LiveAnalytics，請遵循下列最佳實務。

> **Note**
>
> 執行 proof-of-concept應用程式時，請考慮應用程式在評估 的 Timestream 效能和規模時，會在幾個月或幾年內累積的資料量 LiveAnalytics。隨著資料隨著時間的增長，您會注意到 的 Timestream 效能 LiveAnalytics 大多保持不變，因為其無伺服器架構可以利用大量平行處理來處理較大的資料磁碟區，並自動擴展以符合應用程式的需求。

主題

建立資料模型

的 Amazon Timestream LiveAnalytics 旨在從應用程式和裝置收集、儲存和分析時間序列資料，以時間戳記發出一系列資料。為了獲得最佳效能，傳送至 Timestream 的資料 LiveAnalytics 必須具有時間特性，且時間必須是資料的典型元件。

的 Timestream LiveAnalytics 可讓您靈活地以不同的方式建立資料模型，以符合應用程式的需求。在本節中，我們涵蓋了多種模式，並提供指導方針以最佳化成本和效能。熟悉關鍵的 Timestream，了解維度和測量等 LiveAnalytics 概念。在本節中，您將進一步了解：

決定建立單一資料表或多個資料表以存放資料時，請考慮下列事項：

- 要在多個資料表和資料庫之間分隔資料時，要放在相同資料表中的資料與 。
- 如何在 Timestream 之間選擇 LiveAnalytics 多測量記錄與單測量記錄，以及使用多測量記錄建立模型的優點，特別是當您的應用程式同時追蹤多個測量時。
- 要建模為維度或量值的屬性。
- 如何有效地使用量值名稱屬性來最佳化查詢延遲。

主題

單一資料表與多個資料表

當您在應用程式中建立資料模型時，另一個重要的層面是如何將資料建模為資料表和資料庫。Timestream 中的資料庫和資料表 LiveAnalytics 是存取控制的摘要，指定KMS金鑰、保留期間等。自動 LiveAnalytics分割資料的時間串流，旨在擴展資源以符合應用程式的擷取、儲存和查詢載入和需求。

Timestream 中的資料表 LiveAnalytics 可以擴展到儲存的 PB 資料、數十 GB/秒的資料寫入，而查詢可以TBs每小時處理數百個資料。Timestream 中的查詢 LiveAnalytics 可以跨越多個資料表和資料庫，提供聯結和聯合，以跨多個資料表和資料庫無縫存取您的資料。因此，在決定如何在 Timestream 中組織資料時，資料規模或請求磁碟區通常不是主要問題 LiveAnalytics。以下是決定要在相同資料表中與不同資料表或不同資料庫中的資料表中共同放置哪些資料時的一些重要考量。

- 資料表的精細程度支援資料保留政策（記憶體存放區保留、磁性存放區保留等）。因此，需要不同保留政策的資料必須位於不同的資料表中。
- AWS KMS 用於加密資料的金鑰是在資料庫層級設定。因此，不同的加密金鑰需求表示資料需要位於不同的資料庫中。
- 的 Timestream LiveAnalytics 支援資料表和資料庫精細度的資源型存取控制。在決定寫入至相同資料表與不同資料表的資料時，請考慮您的存取控制需求。
- 在決定儲存在哪個資料表中的資料時，請注意維度、量值名稱和多量值屬性名稱的限制。

- 在決定如何組織資料時，請考慮您的查詢工作負載和存取模式，因為查詢延遲和撰寫查詢的便利性取決於此。
  - 如果您將經常查詢的資料儲存在相同的資料表中，這通常會簡化您撰寫查詢的方式，因此您通常可以避免寫入聯結、聯合或常用資料表表達式。這通常也會降低查詢延遲。您可以在維度和量值名稱上使用述詞，以篩選與查詢相關的資料。

    例如，請考慮一個案例，您存放來自位於六大洲的裝置的資料。如果您的查詢經常從各洲存取資料以取得全域彙總檢視，則將來自這些洲的資料儲存在相同資料表中將更容易寫入查詢。另一方面，如果您將資料存放在不同的資料表上，您仍然可以在相同的查詢中合併資料，但您需要撰寫查詢，以將資料從跨資料表聯結起來。
  - Timestream for LiveAnalytics 會在您的資料上使用自適應分割和索引。因此，查詢只會針對與您的查詢相關的資料收取費用。例如，如果您有資料表儲存來自六大洲上百萬個裝置的資料，如果您的查詢具有 表單`WHERE device_id = 'abcdef'`或 的述詞`WHERE continent = 'North America'`，則查詢只會針對裝置或 大洲的資料收取費用。
  - 在可能的情況下，如果您使用量值名稱來分隔相同資料表中未同時發出或未經常提出查詢的資料，則使用查詢`WHERE measure_name = 'cpu'`中的 等述詞，不僅可以獲得計量優勢， 的 Timestream LiveAnalytics 還可以有效地消除沒有查詢述詞中使用的量值名稱的分割區。這可讓您將具有不同量值名稱的相關資料存放在相同資料表中，而不會影響查詢延遲或成本，並避免將資料分散到多個資料表。量值名稱基本上用於分割資料和剪除分割區，與查詢無關。

多測量記錄與單一測量記錄

的 Timestream LiveAnalytics 可讓您寫入每個記錄具有多個量值的資料 （多量值），或每個記錄具有單一量值 （單一量值）。

多量值記錄

在許多使用案例中，您正在追蹤的裝置或應用程式可能會在同一時間戳發出多個指標或事件。在這種情況下，您可以將在相同時間戳記發出的所有指標存放在相同的多測量記錄中。也就是說，儲存在相同多測量記錄中的所有量值都會顯示為相同資料列中的不同資料欄。

例如，請考慮您的應用程式正在從同時即時測量的裝置發出指標，例如 cpu、記憶體、 disk_iops。以下是此類資料表的範例，其中同時發出多個指標會立即儲存在相同的資料列中。您將看到兩個主機每秒發出一次指標。

Hostname (主機名稱)	measure_name	時間	cpu	記憶體	disk_iops
host-24Gju	指標	2021-12-01 19：00：00	35	54.9	38.2
host-24Gju	指標	2021-12-01 19：00：01	36	58	39
host-28Gju	指標	2021-12-01 19：00：00	15	55	92
host-28Gju	指標	2021-12-01 19：00：01	16	50	40

單一測量記錄

當您的裝置在不同時段發出不同的指標，或您使用發出metrics/events at different time periods (for instance, when a device's reading/state變更的自訂處理邏輯時，單一量值記錄就很合適）。由於每個量值都有唯一的時間戳記，因此這些量值可以在 Timestream 中存放在自己的記錄中 LiveAnalytics。例如，請考慮 IoT 感應器，其會追蹤土壤溫度和濕度，只有在偵測到與先前報告項目的變更時才會發出記錄。下列範例提供使用單一量值記錄發出此類資料的範例。

device_id	measure_name	時間	measure_value::double	measure_value::bigint
sensor-sea478	溫度	2021-12-01 19：22：32	35	NULL
sensor-sea478	溫度	2021-12-01 18：07：51	36	NULL
sensor-sea478	濕度	2021-12-01 19：05：30	NULL	21
sensor-sea478	濕度	2021-12-01 19：00：01	NULL	23

比較單一量值和多量值記錄

的 Timestream LiveAnalytics 可讓您靈活地根據應用程式的需求和特性，將資料建模為單一量值或多量值記錄。如果您的應用程式需求需要，單一資料表可以同時儲存單一量值和多量值記錄。一般而言，當您的應用程式同時發出多個量值/事件時，通常建議將資料建模為多量值記錄，以便執行資料存取和符合成本效益的資料儲存。

例如，如果您考慮DevOps 使用案例追蹤指標和來自數十萬個伺服器的事件，則每個伺服器都會定期發出 20 個指標和 5 個事件，其中事件和指標會同時發出。該資料可以使用單一量值記錄或使用多量值記錄建立模型（請參閱開放原始碼資料產生器以取得產生的結構描述）。對於此使用案例，使用多量值記錄與單一量值記錄建立資料模型會導致：

- 擷取計量 - 多測量記錄會導致寫入的擷取位元組減少約 40%。
- 擷取批次 - 多測量記錄會導致傳送的資料批次較大，這表示用戶端需要較少的執行緒和較少的執行緒 CPU來處理擷取。
- 儲存計量 - 多測量記錄導致儲存量降低約 8X，進而大幅節省記憶體和磁性儲存體的儲存。
- 查詢延遲 - 與單一測量記錄相比，多測量記錄會導致大多數查詢類型的查詢延遲較低。
- 查詢計量位元組 - 對於掃描小於 10MB 資料的查詢，單一量值和多量值記錄都是相當的。對於存取單一量值和掃描 > 10MB 資料的查詢，單一量值記錄通常會產生較低的位元組計量。對於參考 3 個或更多量值的查詢，多量值記錄會產生較低的位元組計量。
- 易於表達多測量查詢 - 當您的查詢參考多個測量時，使用多測量記錄建立資料模型會導致更輕鬆地撰寫更精簡的查詢。

先前的因素會因您要追蹤的指標數量、資料擁有的維度等而異。雖然上述範例提供一些具體資料作為一個範例，但我們在許多應用程式案例和使用案例中看到，如果應用程式同時發出多個量值，則將資料儲存為多量值記錄會更有效率。此外，多測量記錄可為您提供資料類型的彈性，並將多個其他值儲存為內容（例如，儲存請求 IDs和其他時間戳記，稍後會討論）。

請注意，多測量記錄也可以建立稀疏測量的模型，例如用於單一測量記錄的先前範例：您可以使用 measure_name 來存放測量的名稱，並使用一般多測量屬性名稱，例如 value_double to store DOUBLE measures、value_bigint to store BIGINT measures、value_timestamp 來存放其他TIMESTAMP值等。

維度和量值

Timestream 中的資料表 LiveAnalytics 可讓您儲存維度（識別您要儲存之裝置/資料的屬性）和量值（您正在追蹤的指標/值），如需更多詳細資訊，請參閱 Timestream 中的 LiveAnalytics 概念。當您在 Timestream 上建立應用程式模型時 LiveAnalytics，將資料映射到維度和量值的方式會影響擷取和查詢延遲。以下是如何將資料建模為可套用至使用案例的維度和措施的指導方針。

選擇維度

識別傳送時間序列資料之來源的資料是維度的自然擬合，這些維度是不會隨時間變更的屬性。例如，如果您有伺服器發出指標，則識別伺服器的屬性，例如主機名稱、區域、機架、可用區域，都是維度的候選項目。同樣地，對於具有多個感應器的 IoT 裝置，報告時間序列資料、裝置 ID、感應器 ID 等是維度的候選項目。

如果您要將資料寫入為多測量記錄，當您在資料表上執行 DESCRIBE或執行SELECT陳述式時，維度和多測量屬性會以資料欄的形式出現在資料表中。因此，在撰寫查詢時，您可以自由地在相同的查詢中使用維度和量值。不過，當您建構寫入記錄以擷取資料時，當您選擇哪些屬性指定為維度，以及哪些屬性為測量值時，請記住下列事項：

- 維度名稱、值、量值名稱和時間戳記可唯一識別時間序列資料。的 Timestream LiveAnalytics 使用此唯一識別碼自動刪除重複資料。也就是說，如果 的 Timestream LiveAnalytics 收到兩個具有相同維度名稱、維度值、量值名稱和時間戳記值的資料點，如果值具有相同的版本編號，則表示刪除重複資料的 Timestream LiveAnalytics。如果新的寫入請求版本低於 的 Timestream 中已存在的資料 LiveAnalytics，則會拒絕寫入請求。如果新的寫入請求具有較高的版本，則新值會覆寫舊值。因此，您選擇維度值的方式會影響此重複資料刪除行為。
- 維度名稱和值無法更新，測量值可以是 。因此，任何可能需要更新的資料都會以更好的模型作為測量值。例如，如果您的機器位於工廠現場，其顏色可能會變更，則您可以將顏色建模為測量值，除非您也想要使用顏色來識別重複資料刪除所需的屬性。也就是說，測量值可用來儲存屬性，這些屬性只會隨著時間慢慢變更。

請注意，Timestream 中的資料表 LiveAnalytics 不會限制維度名稱和值的唯一組合數目。例如，您可以在資料表中存放數十億個此類唯一值組合。不過，如您在下列範例中所見，仔細選擇維度和量值可以大幅最佳化您的請求延遲，特別是查詢。

維度IDs唯一

如果您的應用程式案例需要您為每個資料點儲存唯一識別符（例如請求 ID、交易 ID 或關聯 ID），則將 ID 屬性建模為測量值將導致顯著更好的查詢延遲。使用多量值記錄建立資料模型時，ID 會與您的其他維度和時間序列資料顯示在相同的資料列中，因此您的查詢可以繼續使用它們。例如，考慮DevOps使用案例，其中伺服器發出的每個資料點都有唯一的請求 ID 屬性，將請求 ID 建模為測量值會導致不同查詢類型的查詢延遲降低高達 4 倍，而不是將唯一請求 ID 建模為維度。

您可以針對不是每個資料點完全唯一的屬性使用類似的比喻，但具有數十萬或數百萬個唯一值。您可以將這些屬性建模為維度或測量值。如果值是先前討論在寫入路徑上刪除重複所必需，或者您經常在查詢中使用它作為述詞（例如，在 WHERE子句中，具有該屬性值的等式述詞，例如應用程式正在追蹤數百萬個裝置`device_id = 'abcde'`的位置），則您想要將其作為維度模型。

具有多測量記錄的資料類型的豐富度

多測量記錄可讓您靈活地有效地建立資料模型。您在多測量記錄中儲存的資料會以類似維度的資料列形式出現在資料表中，因此提供相同的易查詢維度和測量值的便利性。您在前面討論的範例中看到了其中一些模式。您可以在下方找到其他模式，以有效使用多測量記錄來滿足應用程式的使用案例。

多測量記錄支援資料類型 DOUBLE、BIGINT、BOOLEAN、 VARCHAR和 的屬性TIMESTAMP。因此，它們自然適合不同類型的屬性：

- 位置資訊 ：例如，如果您想要追蹤位置 （以緯度和經度表示），則將其建模為多測量屬性會導致查詢延遲比將其儲存為VARCHAR維度更低，特別是當您在緯度和經度上有述詞時。
- 記錄 中的多個時間戳記：如果您的應用程式案例需要您追蹤時間序列記錄的多個時間戳記，您可以將它們建模為多量值記錄中的其他屬性。此模式可用來儲存具有未來時間戳記或過去時間戳記的資料。請注意，每個記錄仍會使用時間欄中的時間戳記來分割、索引和唯一識別記錄。

特別是，如果您在查詢中具有引數的數值資料或時間戳記，則將這些屬性建模為多測量屬性，而不是維度，將導致較低的查詢延遲。這是因為當您使用多測量記錄支援的豐富資料類型建立此類資料的模型時，如果您將此類資料建模為維度，您可以使用原生資料類型來表達述詞，而不是將值從 轉換為VARCHAR其他資料類型。

將量值名稱與多量值記錄搭配使用

Timestream 中的資料表 LiveAnalytics 支援稱為量值名稱的特殊屬性 （或資料欄）。您可以為寫入 Timestream for 的每個記錄指定此屬性的值 LiveAnalytics。對於單一量值記錄，自然會使用指標的名稱 （例如 cpu、伺服器指標的記憶體或溫度、感應器指標的壓力）。使用多測量記錄時，由於多測量記錄中的屬性已命名 （且這些名稱會成為資料表中的資料欄名稱），Cpu、記憶體或溫度時，壓力可能會成為多測量屬性名稱。因此，自然問題是如何有效地使用量值名稱。

的 Timestream LiveAnalytics 使用量值名稱屬性中的值來分割和索引資料。因此，如果資料表具有多個不同的量值名稱，且如果查詢使用這些值作為查詢述詞，則 的 Timestream LiveAnalytics 可以使用其自訂分割和索引來刪除與查詢無關的資料。例如，如果您的資料表具有 CPU 和記憶體量值名稱，且您的查詢具有述詞 `WHERE measure_name = 'cpu'`，則 的 Timestream LiveAnalytics 可以有效地為與查詢無關的量值名稱剪除資料，例如此範例中具有量值名稱記憶體的列。即使將量值名稱與多量值記錄搭配使用，此剪除也適用。您可以有效地使用量值名稱屬性作為資料表的分割屬性。測量名稱以及維度名稱和值，以及時間用於分割 LiveAnalytics 資料表時間串流中的資料。請注意 LiveAnalytics 資料表時間串流中允許的唯一量值名稱數量限制。另請注意，測量名稱也與測量值資料類型相關聯，例如，單一測量名稱只能與一種類型的測量值相關聯。該類型可以是 DOUBLE、BIGINT、VARCHAR、BOOLEAN和 之一MULTI。以量值名稱儲存的多量值記錄，其資料類型為 MULTI。由於單一多量值

記錄可以儲存具有不同資料類型（DOUBLE、BIGINT、VARCHAR、 和 TIMESTAMP）的多個指標BOOLEAN，因此您可以在多量值記錄中關聯不同類型的資料。

下列各節說明幾個不同的範例，說明如何有效地使用量值名稱屬性來將相同資料表中的不同資料類型分組。

IoT 感應器報告品質和價值

請考慮您擁有來自 IoT 感應器的應用程式監控資料。每個感應器都會追蹤不同的量值，例如溫度、壓力。除了實際值之外，感應器還會報告測量的品質，這是測量讀數有多準確，以及測量單位的指標。由於品質、單位和值會一起發出，因此它們可以建模為多測量記錄，如以下範例資料所示，其中 device_id 是維度、品質、值和單位是多測量屬性：

device_id	measure_name	時間	品質	Value	單位
sensor-sea478	溫度	2021-12-01 19：22：32	92	35	c
sensor-sea478	溫度	2021-12-01 18：07：51	93	34	c
sensor-sea478	pressure	2021-12-01 19：05：30	98	31	psi
sensor-sea478	pressure	2021-12-01 19：00：01	24	132	psi

此方法可讓您使用量值名稱，結合多量值記錄的優點，以及分割和剪除資料。如果查詢參考單一量值，例如溫度，您可以在查詢中包含量值名稱述詞。以下是此類查詢的範例，該查詢也會針對品質高於 90 的測量，投影單位。

```
SELECT device_id, time, value AS temperature, unit
FROM db.table
WHERE time > ago(1h)
    AND measure_name = 'temperature'
    AND quality > 90
```

在查詢上使用 measure_name 述詞 LiveAnalytics ，可讓 的 Timestream 有效地剪除與查詢無關的分割區和資料，進而改善您的查詢延遲。

如果在同一時間戳發出所有指標和/或在同一查詢中同時查詢多個指標，也可以將所有指標存放在相同的多測量記錄中。例如，您可以建構為具有屬性 temperature_quality、perme_value、perme_unit、pressure_quality、 pressure_value、 pressure_unit 等的多測量記錄。先前討論使用單一量值與多量值記錄建立資料模型的許多要點，都適用於您如何建立資料模型的決定。請考慮您的查詢存取模式，以及您的資料如何產生，以選擇可最佳化成本、擷取和查詢延遲的模型，以及寫入查詢的簡易性。

相同資料表中的不同類型的指標

另一個您可以結合多測量記錄與測量名稱值的使用案例，是建立從相同裝置獨立發出的不同資料類型的模型。請考慮 DevOps 監控使用案例伺服器會發出兩種類型的資料：定期發出指標和不規則事件。此方法的範例是建立 DevOps 使用案例模型時，在資料產生器中討論的結構描述。在此情況下，您可以使用不同的量值名稱，將從相同伺服器發出的不同類型的資料儲存在相同資料表中。例如，同時發出的所有指標都會與量值名稱指標一起存放。所有在與指標不同的時間立即發出的事件，都會與量值名稱事件一起存放。資料表的量值結構描述（例如SHOW MEASURES查詢的輸出）為：

measure_name	data_type	維度
事件	多重	【{"data_type"："varchar"，"dimension_name"："availability_zone"}，{"data_type"："varchar"，"dimension_name"："microservice_name"}，{"data_type"："varchar"，"dimension_name"："instance_name"}，{"data_type"："varchar"，"dimension_name"："process_name"}，{"data_type"："varchar"，"dimension_name"："jdk_version"}，{"data_type"："varchar"，"dimension_name"name"："" ，{"
指標	多重	【{"data_type"："varchar"，"dimension_name"："availability

measure_name	data_type	維度
		_zone"}，{"data_type"："varchar"，"dimension_name"："microservice_name"}，{"data_type"："varchar"，"dimension_name"："instance_name"}，{"data_type"："varchar"，"dimension_name"："os_version"}，{"data_type"："varchar"，"dimension_name"："cell"}，{"data_type"："varchar"，"dimension_name"："""，{data_type"："，"dimension"

在這種情況下，您可以看到事件和指標也具有不同的維度集，其中事件具有不同的維度 jdk_version 和 process_name，而指標具有維度 instance_type 和 os_version。

使用不同的量值名稱可讓您使用述詞撰寫查詢`WHERE measure_name = 'metrics'`，例如只取得指標。同時，在相同資料表中從相同執行個體發出所有資料，表示您也可以使用 instance_name 述詞撰寫更簡單的查詢，以取得該執行個體的所有資料。例如，`WHERE instance_name = ‘instance-1234’`沒有 measure_name 述詞的表單述詞會傳回特定伺服器執行個體的所有資料。

分割多測量記錄的建議

> **Important**
>
> 本節已棄用！
> 這些建議已過期。現在可以使用客戶定義的分割金鑰 來更好地控制分割。

我們看到時間序列生態系統中越來越多的工作負載需要擷取和儲存大量資料，同時在透過高維度資料集存取資料時需要低延遲查詢回應。

由於這類特性，本節中的建議對於具有下列內容的客戶工作負載很有用。

- 已採用或想要採用多測量記錄。
- 預期有大量資料進入系統，而這些資料將長期儲存。
- 主要存取（查詢）模式需要低延遲回應時間。
- 知道最重要的查詢模式涉及述詞中某種類型的篩選條件。此篩選條件是以高基數維度為基礎。例如，考慮 UserId、 DeviceId、ServerID、主機名稱等的事件或彙總。

在這些情況下，所有多量值測量的單一名稱將沒有幫助，因為我們的引擎使用多量值名稱來分割資料，且具有單一值會限制您取得的分割區優勢。這些記錄的分割主要是以兩個維度為基礎。假設時間在 x 軸上，以及 y 軸measure_name上一個維度名稱和 的雜湊。measure_name 在這些情況下， 的運作方式幾乎與分割金鑰類似。

我們的建議如下。

- 為類似我們提到的使用案例建立資料模型時，請使用 measure_name 作為主要查詢存取模式的直接衍生。例如：
  - 您的使用案例需要從最終使用者的觀點追蹤應用程式效能和 QoE。這也可能是追蹤單一伺服器或 IoT 裝置的測量。
  - 如果您要依 查詢和篩選， UserId則需要在擷取時間找到measure_name與 建立關聯的最佳方法 UserId。
  - 由於多量值資料表只能保留 8192 個不同的量值名稱，因此採用的任何公式都不應產生超過 8192 個不同的值。
- 我們成功套用字串值的一個方法是將雜湊演算法套用至字串值。然後使用雜湊結果的絕對值 和 8192 執行模數操作。

```
measure_name = getMeasureName(UserId)
int getMeasureName(value) {
    hash_value =  abs(hash(value))
    return hash_value % 8192
}
```

- 我們也新增了 abs() 以移除 符號，消除了值從 -8192 到 8192 的可能性。這應在模數操作之前執行。
- 透過使用此方法，您的查詢可以在未分割資料模型上執行所需的一小部分時間上執行。
- 查詢資料時，請確定您在使用 measure_name 新衍生值的述詞中包含篩選條件。例如：
  -
```
SELECT * FROM your_database.your_table
WHERE host_name = 'Host-1235' time BETWEEN '2022-09-01'
```

```
   AND '2022-09-18'
   AND measure_name = (SELECT
cast(abs(from_big_endian_64(xxhash64(CAST('HOST-1235' AS varbinary))))%8192 AS
varchar))
```

- 這將最大限度地減少掃描的分割區總數，以取得資料，這些資料將隨著時間的推移更快地轉換查詢。

請記住，如果您想要從此分割區結構描述中取得利益，則需要在用戶端計算雜湊，並將其作為 LiveAnalytics 靜態值傳遞給查詢引擎的 Timestream。上述範例提供一種方法來驗證產生的雜湊是否可以在需要時由引擎解決。

time	host_name	location	server_type	cpu_usage	available_memory	cpu_temp
2022-09-07 21：48：44 .000000000	host-1235	us-east1	5.8xl	55	16.2	78
R2022-09-07 21：48：44 .000000000	host-3587	us-west1	5.8xl	62	18.1	81
2022-09-07 21：48：45.000000000	host-258743	eu-central	5.8xl	88	9.4	91
2022-09-07 21：48：45 .000000000	host-35654	us-east2	5.8xl	29	24	54
R2022-09-07 21：48：	host-254	us-west1	5.8xl	44	32	48

time	host_name	location	server_type	cpu_usage	available_memory	cpu_temp
45 .00000000						

若要measure_name依照我們的建議產生相關聯的 ，有兩個路徑取決於您的擷取模式。

1. 對於歷史資料的批次擷取：如果您將自己的程式碼用於批次程序，則可以將轉換新增至寫入程式碼。

   在上述範例之上建置 。

```
        List<String> hosts = new ArrayList<>();

        hosts.add("host-1235");
        hosts.add("host-3587");
        hosts.add("host-258743");
        hosts.add("host-35654");
        hosts.add("host-254");

        for (String h: hosts){
            ByteBuffer buf2 = ByteBuffer.wrap(h.getBytes());
            partition = abs(hasher.hash(buf2, 0L)) % 8192;
            System.out.println(h + " - " + partition);

        }
```

   輸出

```
host-1235 - 6445
host-3587 - 6399
host-258743 - 640
host-35654 - 2093
host-254 - 7051
```

   產生的資料集

time	host_name	location	measure_name	server_type	cpu_usage	available_memory	cpu_temp
2022-09-07 21：48：44 .000000000	host-1235	us-east1	6445	5.8xl	55	16.2	78
R2022-09-07 21：48：44 .000000000	host-3587	us-west1	6399	5.8xl	62	18.1	81
2022-09-07 21：48：45.000000000	host-258743	eu-central	640	5.8xl	88	9.4	91
2022-09-07 21：48：45 .000000000	host-35654	us-east2	2093	5.8xl	29	24	54
R2022-09-07 21：48：45 .000000000	host-254	us-west1	7051	5.8xl	44	32	48

2. 對於即時擷取：您需要在資料傳入時產生傳輸`measure_name`中。

在這兩種情況下，我們建議您測試兩端的雜湊產生演算法（擷取和查詢），以確保您獲得相同的結果。

以下是根據 產生雜湊值的一些程式碼範例host_name。

Example Python

```
>>> import xxhash
>>> from bitstring import BitArray
>>> b=xxhash.xxh64('HOST-ID-1235').digest()
>>> BitArray(b).int % 8192
### 3195
```

Example Go

```
package main

import (
    "bytes"
    "fmt"
    "github.com/cespare/xxhash"
)

func main() {
    buf := bytes.NewBufferString("HOST-ID-1235")
    x := xxhash.New()
    x.Write(buf.Bytes())
    // convert unsigned integer to signed integer before taking mod
    fmt.Printf("%f\n", abs(int64(x.Sum64())) % 8192)
}

func abs(x int64) int64 {
    if (x < 0) {
        return -x
    }
    return x
}
```

Example Java

```
import java.nio.ByteBuffer;

import net.jpountz.xxhash.XXHash64;

public class test {
```

```
    public static void main(String[] args) {
        XXHash64 hasher = net.jpountz.xxhash.XXHashFactory.fastestInstance().hash64();

        String host = "HOST-ID-1235";
        ByteBuffer buf = ByteBuffer.wrap(host.getBytes());

        Long result = Math.abs(hasher.hash(buf, 0L));
        Long partition = result % 8192;

        System.out.println(result);
        System.out.println(partition);
    }
}
```

Example Maven 中的相依性

```
        <dependency>
            <groupId>net.jpountz.lz4</groupId>
            <artifactId>lz4</artifactId>
            <version>1.3.0</version>
        </dependency>
```

安全

- 若要持續存取 的 Timestream LiveAnalytics，請確保加密金鑰受到保護，且不會遭到撤銷或無法存取。
- 從 監控API存取日誌 AWS CloudTrail。從未經授權的使用者稽核並撤銷任何異常存取模式。
- 請遵循中所述的其他準則適用於 的 Amazon Timestream 安全最佳實務 LiveAnalytics。

為 設定 Amazon Timestream LiveAnalytics

設定記憶體存放區和磁性存放區的資料保留期間，以符合資料處理、儲存、查詢效能和成本需求。

- 設定記憶體存放區的資料保留，以符合應用程式處理延遲抵達資料的需求。延遲抵達資料是傳入的資料，其時間戳記早於目前時間。它會從將資料傳送至 Timestream for 之前批次事件一段時間的資源發出 LiveAnalytics，以及從具有間歇性連線的資源發出，例如間歇性連線的 IoT 感應器。
- 如果您預期延遲到達的資料偶爾會比記憶體存放區保留時間更早出現時間戳記，則應為資料表啟用磁性存放區寫入。 MagneticStoreWritesProperties 將 EnableMagneticStoreWrites 設定為資料表的後，資料表將接受時間戳記早於記憶體儲存體保留，但在磁性儲存體保留期間內的資料。

- 考慮您計劃在 Timestream 上執行的查詢特性， LiveAnalytics 例如查詢類型、頻率、時間範圍和效能需求。這是因為記憶體存放區和磁性存放區已針對不同的案例進行最佳化。記憶體存放區針對處理傳送至 Timestream for 之少量近期資料的快速 point-in-time查詢進行最佳化 LiveAnalytics。磁性存放區已針對處理傳送至 Timestream for 的中大量資料之快速分析查詢進行最佳化 LiveAnalytics。
- 您的資料保留期也應該受到系統成本需求的影響。

  例如，請考慮一個案例，即應用程式的延遲到達資料閾值為 2 小時，而您的應用程式會傳送許多查詢，以處理一天值、一週值或月份值的資料。在這種情況下，您可能想要為記憶體存放區設定更短的保留期 （ 2-3 小時），並允許更多資料流向磁性存放區，因為磁性存放區已針對快速分析查詢進行最佳化。

了解增加或減少記憶體存放區的資料保留期和現有資料表的磁性存放區的影響。

- 當您減少記憶體存放區的保留期間時，資料會從記憶體存放區移至磁性存放區，而此資料傳輸是永久的。的 Timestream LiveAnalytics 不會從磁性存放區擷取資料以填入記憶體存放區。當您減少磁性存放區的保留期間時，資料會從系統中刪除，而資料刪除是永久的。
- 當您增加記憶體存放區或磁性存放區的保留期間時， LiveAnalytics 從該時間點開始傳送至 Timestream 的資料變更會生效。的 Timestream LiveAnalytics 不會從磁性存放區擷取資料以填入記憶體存放區。例如，如果記憶體存放區的保留期間最初設定為 2 小時，然後增加至 24 小時，則記憶體存放區需要 22 小時才能包含 24 小時的資料。

寫入

- 確保傳入資料的時間戳記不早於為記憶體存放區設定的資料保留，且不晚於 中定義的未來擷取期間配額。將具有時間戳記的資料傳送到這些界限之外，將導致 Timestream 拒絕 的資料， LiveAnalytics 除非您為資料表啟用磁性存放區寫入。如果您啟用磁性存放區寫入，請確保傳入資料的時間戳記不早於為磁性存放區設定的資料保留。
- 如果您預期延遲到達資料，請開啟資料表的磁性存放區寫入。這將允許擷取時間戳在記憶體儲存保留期間之外，但仍在磁性儲存保留期間內的資料。您可以更新資料表中`MagneticStoreWritesProperties`的`EnableMagneticStoreWrites`旗標來設定此項目。此屬性預設為 false。請注意，寫入磁性存放區將無法立即用於查詢。它們將在 6 小時內提供。
- 透過確保擷取資料的時間戳記落在記憶體存放區保留界限內，將高輸送量工作負載鎖定至記憶體存放區。寫入磁性存放區限制為可接收資料庫並行擷取的作用中磁性存放區分割區數量上限。您可以在中查看此`ActiveMagneticStorePartitions`指標 CloudWatch。為了減少作用中磁性存放區分割區，目標是減少您同時擷取磁性存放區擷取的序列數和持續時間。

- 將資料傳送至 的 Timestream 時 LiveAnalytics，請在單一請求中批次處理多個記錄，以最佳化資料擷取效能。
  - 將來自相同時間序列的記錄和具有相同測量名稱的記錄批次合併是有利的。
  - 只要請求在 中定義的服務限制內，在單一請求中盡可能批次處理更多記錄配額。
  - 盡可能使用常見屬性來降低資料傳輸和擷取成本。如需詳細資訊，請參閱 WriteRecords API。
- 如果您在將資料寫入 的 Timestream 時遇到部分用戶端失敗 LiveAnalytics，您可以在解決拒絕原因之後重新傳送擷取失敗的記錄批次。
- 時間戳記排序的資料具有更好的寫入效能。
- 的 Amazon Timestream LiveAnalytics 旨在自動擴展以滿足您的應用程式需求。當 LiveAnalytics 通知的 Timestream 在來自應用程式的寫入請求中暴增時，您的應用程式可能會遇到某種程度的初始記憶體儲存限流。如果您的應用程式遇到記憶體儲存限流，請繼續 LiveAnalytics 以相同 （或增加） 速率將資料傳送至 Timestream，讓 Timestream LiveAnalytics 自動擴展以滿足應用程式的需求。如果您看到磁性存放區調節，您應該降低磁性存放區擷取速率，直到發生次數`ActiveMagneticStorePartitions`下降為止。

批次載入

批次載入的最佳實務如 所述批次載入最佳實務。

查詢

以下是使用 Amazon Timestream 進行查詢的建議最佳實務 LiveAnalytics。

- 僅包含查詢所需的量值和維度名稱。新增外部資料欄會增加資料掃描，這會影響查詢的效能。
- 在生產中部署查詢之前，建議您檢閱查詢洞察，以確保空間和時間修剪是最佳的。如需詳細資訊，請參閱使用查詢洞察功能來最佳化 Amazon Timestream 中的查詢。
- 可能的話，將資料運算推送至 Timestream，以 LiveAnalytics 使用SELECT子句和WHERE子句中的內建彙總和純量函數，以改善查詢效能並降低成本。請參閱 SELECT 和 彙總函數。
- 盡可能使用近似函數。例如，使用 APPROX_DISTINCT 而非 COUNT（DISTINCT column_name）來最佳化查詢效能並降低查詢成本。請參閱 彙總函數。
- 使用 CASE運算式來執行複雜的彙總，而不是多次從相同的資料表中選取 。請參閱 CASE 陳述式。
- 如果可能，請在查詢的WHERE子句中包含時間範圍。這可最佳化查詢效能和成本。例如，如果您只需要資料集的最後一小時資料，則請包含時間述詞，例如 > 前 （1h）。請參閱 SELECT 和 間隔和持續時間。

- 當查詢存取資料表中的量值子集時， 一律會在查詢的WHERE子句中包含量值名稱。
- 在可能的情況下，在比較查詢子WHERE句中的維度和量值時，請使用平等運算子。維度和量值名稱的平等原則可改善查詢效能並降低查詢成本。
- 盡可能避免在 WHERE子句中使用函數來最佳化成本。
- 避免多次使用LIKE子句。相反地，當您篩選字串欄上的多個值時，請使用規則運算式。請參閱 規則運算式函數。
- 僅使用查詢 GROUP BY 子句中的必要資料欄。
- 如果查詢結果需要以特定順序排列，請在最外部查詢的 ORDER BY 子句中明確指定該順序。如果您的查詢結果不需要排序，請避免使用 ORDER BY 子句來改善查詢效能。
- 如果您只需要查詢中的前 N 列，請使用 LIMIT子句。
- 如果您使用 ORDER BY 子句來查看頂端或底部 N 值，請使用LIMIT子句來降低查詢成本。
- 使用傳回回應中的分頁權杖來擷取查詢結果。如需詳細資訊，請參閱查詢。
- 如果您已開始執行查詢，並發現查詢不會傳回您要尋找的結果，請取消查詢以節省成本。如需詳細資訊，請參閱 CancelQuery。
- 如果您的應用程式遇到限流，請繼續以相同的速率將資料傳送至 Amazon Timestream LiveAnalytics，讓 Amazon Timestream LiveAnalytics 自動擴展，以滿足應用程式的查詢輸送量需求。
- 如果應用程式的查詢並行需求超過 的 Timestream 預設限制 LiveAnalytics，請聯絡 AWS Support 以取得限制增加。

排程查詢

排程查詢透過預先計算一些機群範圍的彙總統計資料，協助您最佳化儀表板。因此，要問的一個自然問題是，如何採用您的使用案例，並識別要預先計算的結果，以及如何使用這些結果儲存在衍生資料表中來建立儀表板。此程序的第一步是識別要預先運算的面板。以下是一些高階準則：

- 請考慮用於填入面板的查詢掃描的位元組、儀表板重新載入的頻率，以及載入這些儀表板的並行使用者數量。您應該從最常載入的儀表板開始，並掃描大量資料。彙總儀表板範例中的前兩個儀表板，以及向下切入範例中的彙總儀表板，都是這類儀表板的良好範例。
- 考慮哪些運算重複使用 。雖然可以為每個面板和面板中使用的每個變數值建立排程查詢，但您可以透過尋找使用一個運算來預先計算多個面板所需的資料的方法，大幅最佳化成本和排程查詢的數量。
- 考慮排程查詢的頻率，以重新整理衍生資料表中的具體化結果。您想要分析儀表板重新整理的頻率，相較於儀表板中查詢的時段，相較於預先運算中使用的時間分隔，以及儀表板中的面板。例如，如果正在繪製過去幾天每小時彙總的儀表板只會在幾個小時內重新整理一次，您可能想要將排程查詢設定

為每 30 分鐘或一小時重新整理一次。另一方面，如果您有一個儀表板，可繪製每分鐘彙總並每分鐘重新整理一次，則您希望排程查詢每分鐘或每分鐘重新整理一次結果。

- 考慮哪些查詢模式可以使用排程查詢進一步最佳化（從查詢成本和查詢延遲角度來看）。例如，在計算儀表板中經常用作變數的唯一維度值時，或傳回從感應器發出的最後一個資料點，或在特定日期後從感應器發出的第一個資料點等。本指南討論了其中一些範例模式。

當您移動儀表板來查詢衍生的資料表、儀表板中資料的新鮮度，以及排程查詢所產生的成本時，上述考量將對您的節省產生重大影響。

用戶端應用程式和支援的整合

從與 Timestream 相同的區域執行用戶端應用程式 LiveAnalytics ，以減少網路延遲和資料傳輸成本。如需使用其他 服務的詳細資訊，請參閱 使用其他 服務。以下是一些其他有用的連結。

- 使用 AWS 開發的最佳實務 AWS SDK for Java
- 使用 AWS Lambda 函數的最佳實務
- Amazon Managed Service for Apache Flink 的最佳實務
- 在 Grafana 中建立儀表板的最佳實務

一般

- 使用 Timestream for 時，請務必遵循 AWS Well-Architected 架構 LiveAnalytics。本白皮書提供有關卓越營運、安全性、可靠性、效能效率和成本最佳化最佳實務的指導。

計量和成本最佳化

使用適用於 的 Amazon Timestream LiveAnalytics，您只需為使用的項目付費。用於寫入、儲存資料和查詢掃描資料之儀 LiveAnalytics 表的個別時間串流。每個計量維度的價格會在定價頁面上指定。您可以使用 Amazon Timestream for LiveAnalytics Pricing Calculator 預估每月帳單。

本節說明 的 Timestream 中，計量如何用於寫入、儲存和查詢 LiveAnalytics。也提供範例案例和計算。此外，還包含成本最佳化的最佳實務清單。您可以選取以下主題：

主題

- 寫入
- 儲存

- 查詢
- 成本最佳化
- 使用 Amazon 監控 CloudWatch

寫入

每個時間序列事件的寫入大小計算為時間戳記大小和一或多個維度名稱、維度值、量值名稱和量值的總和。時間戳記的大小為 8 個位元組。維度名稱、維度值和量值名稱的大小是字串 UTF-8 編碼位元組的長度，代表每個維度名稱、維度值和量值名稱。測量值的大小取決於資料類型。布林值資料類型為 1 位元組，Butint 和雙位元組為 8 位元組，字串為 UTF-8 編碼位元組的長度。每次寫入都會以 1 KiB 的單位計算。

以下提供兩個範例計算：

主題

- 計算時間序列事件的寫入大小
- 計算寫入次數

計算時間序列事件的寫入大小

考慮代表EC2執行個體CPU使用率的時間序列事件，如下所示：

時間	region	az	vpc	Hostname (主機名稱)	measure_name	measure_value::double
1602983435238563000	us-east-1	1d	vpc-1a2b3c4d	host-24Gju	cpu_utilization	35.0

時間序列事件的寫入大小可以計算為：

- time = 8 個位元組
- 第一個維度 = 15 位元組 （ `region` + `us-east-1` ）
- 第二個維度 = 4 位元組 （ `az` + `1d` ）

- 第三個維度 = 15 位元組（vpc + vpc-1a2b3c4d）
- 第四個維度 = 18 位元組（hostname + host-24Gju）
- 量值名稱 = 15 位元組（cpu_utilization）
- 量值 = 8 位元組

時間序列事件的寫入大小 = 83 位元組

計算寫入次數

現在，請考慮 100 個EC2執行個體，類似於 中所述的執行個體計算時間序列事件的寫入大小，每 5 秒發出一次指標。EC2 執行個體的每月寫入總數會根據每次寫入存在多少時間序列事件，以及在批次處理時間序列事件時使用的一般屬性而有所不同。針對下列每個案例提供計算每月寫入總數的範例：

主題

- 每次寫入的一次性序列事件
- 批次寫入中的時間序列事件
- 批次處理時間序列事件，並在寫入中使用常見屬性

每次寫入的一次性序列事件

如果每個寫入僅包含一次序列事件，每月寫入總數計算為：

- 100 個時間序列事件 = 每 5 秒 100 次寫入
- x 12 次寫入/分鐘 = 1，200 次寫入
- x 60 分鐘/小時 = 72，000 次寫入
- x 24 小時/天 = 1，728，000 次寫入
- x 30 天/月 = 51，840，000 次寫入

每月寫入總數 = 51，840，000

批次寫入中的時間序列事件

由於每個寫入的測量單位為 1 KB，寫入可以包含一批 12 個時間序列事件（998 位元組），每月寫入總數計算為：

- 100 次時間序列事件 = 每 5 秒 9 次寫入（每次寫入 12 次時間序列事件）

- x 12 次寫入/分鐘 = 108 次寫入
- x 60 分鐘/小時 = 6，480 次寫入
- x 24 小時/天 = 155，520 次寫入
- x 30 天/月 = 4，665，600 次寫入

每月寫入總數 = 4，665，600

批次處理時間序列事件，並在寫入中使用常見屬性

如果區域、az、vpc 和量值名稱在 100 個EC2執行個體中通用，則每次寫入只能指定一次通用值，並稱為通用屬性。在此情況下，常見屬性的大小為 52 個位元組，時間序列事件的大小為 27 個位元組。由於每個寫入的測量單位為 1 KiB，寫入可以包含 36 個時間序列事件和常見屬性，每月寫入總數計算為：

- 100 次時間序列事件 = 每 5 秒 3 次寫入 （每次寫入 36 次時間序列事件）
- x 12 次寫入/分鐘 = 36 次寫入
- x 60 分鐘/小時 = 2，160 次寫入
- x 24 小時/天 = 51，840 次寫入
- x 30 天/月 = 1，555，200 次寫入

每月寫入總數 = 1，555，200

> **Note**
>
> 由於使用批次處理、一般屬性，並將寫入四捨五入為 1KB 的單位，時間序列事件的儲存大小可能與寫入大小不同。

儲存

記憶體存放區和磁性存放區中每個時間序列事件的儲存體大小，計算方式為時間戳記、維度名稱、維度值、量值名稱和量值的總和。時間戳記的大小為 8 個位元組。維度名稱、維度值和量值名稱的大小是每個字串 UTF-8 編碼位元組的長度，代表維度名稱、維度值和量值名稱。測量值的大小取決於資料類型。布林值資料類型為 1 位元組，Butint 和雙位元組為 8 位元組，字串為 UTF-8 編碼位元組的長度。每個量值都會以個別記錄的形式儲存在 Amazon Timestream 中 LiveAnalytics，也就是說，如果您的時間序列事件有四個量值，儲存中會有該時間序列事件的四個記錄。

考量代表EC2執行個體CPU使用率的時間序列事件範例（請參閱 計算時間序列事件的寫入大小），時間序列事件的儲存體大小計算為：

- time = 8 個位元組
- 第一個維度 = 15 位元組（`region` + `us-east-1`）
- 第二個維度 = 4 位元組（`az` + `1d`）
- 第三個維度 = 15 位元組（`vpc` + `vpc-1a2b3c4d`）
- 第四個維度 = 18 位元組（`hostname` + `host-24Gju`）
- 量值名稱 = 15 位元組（`cpu_utilization`）
- 量值 = 8 位元組

時間序列事件的儲存大小 = 83 位元組

> **Note**
>
> 記憶體存放區以 GB 小時為單位進行計量，磁性存放區以 GB 個月為單位進行計量。

查詢

查詢是根據應用程式在 TCUAmazon Timestream 定價頁面上指定的小時內使用的時間串流運算單位（TCUs）持續時間收費。 https://aws.amazon.com/timestream/pricing/ LiveAnalytics查詢引擎的 Amazon Timestream 在處理查詢時剪除不相關的資料。具有預測和述詞的查詢，包括時間範圍、測量名稱和/或維度名稱，可讓查詢處理引擎縮減大量資料，並協助降低查詢成本。

成本最佳化

若要最佳化寫入、儲存和查詢的成本，請在 Amazon Timestream 中使用下列最佳實務 LiveAnalytics：

- 每次寫入批次多個時間序列事件，以減少寫入請求的數量。
- 考慮使用多量值記錄，這可讓您以單一寫入請求寫入多個時間序列量值，並以更精簡的方式存放資料。這可減少寫入請求的數量，以及資料儲存成本和查詢成本。
- 使用具有批次處理的常見屬性，在每次寫入時批次處理更多時間序列事件，以進一步減少寫入請求的數量。
- 設定記憶體存放區的資料保留，以符合應用程式處理延遲抵達資料的需求。延遲抵達資料是指傳入的資料，其時間戳記早於目前時間，且超出記憶體存放區保留期。

- 設定磁性存放區的資料保留，以符合您的長期資料儲存需求。
- 撰寫查詢時，請僅包含查詢所需的量值和維度名稱。新增外部資料欄會增加資料掃描，因此也會增加查詢成本。建議您檢閱查詢洞察，以評估包含維度和量值的截斷效率。
- 如果可能，請在查詢的WHERE子句中包含時間範圍。例如，如果您只需要資料集的最後一小時資料，請包含時間述詞，例如 `time > ago(1h)`。
- 當查詢存取資料表中的量值子集時，請務必在查詢的WHERE子句中包含量值名稱。
- 如果您已開始執行查詢，並發現查詢不會傳回您要尋找的結果，請取消查詢以節省成本。

使用 Amazon 監控 CloudWatch

您可以監控 Timestream 是否 LiveAnalytics 使用 Amazon CloudWatch，其會將來自 Timestream 的 LiveAnalytics原始資料收集並處理為可讀取的 near-real-time指標。它會記錄這些統計資料兩週，以便您可以存取歷史資訊，並更好地了解 Web 應用程式或服務的表現。依預設， LiveAnalytics 指標資料的時間串流會在 CloudWatch 1 分鐘或 15 分鐘的期間內自動傳送至 。如需詳細資訊，請參閱 Amazon 使用者指南 CloudWatch中的什麼是 Amazon？。 CloudWatch

主題

- 如何將 Timestream 用於 LiveAnalytics 指標？
- LiveAnalytics 指標和維度的時間串流
- 建立 CloudWatch 警示以監控 的 Timestream LiveAnalytics

如何將 Timestream 用於 LiveAnalytics 指標？

Timestream 針對 報告的指標 LiveAnalytics 提供您可以不同方式分析的資訊。下列清單顯示一些常見的指標用途。這些是協助您開始的建議，而不是完整清單。

如何？	相關指標
How can I determine if any system errors occurred?	您可以監控 SystemErrors 來判斷任何請求是否導致伺服器錯誤碼。這項指標通常應該等於零。如果不是，您可能想要調查。
How can I monitor the amount of data in the memory store?	您可以在MemoryCumulativeBytesMetered 指定的期間內監控，以監控以位元組為單位儲存在記憶體儲存體中的資料量。此指標每小時發出一次，您可以追蹤儲存在帳戶和資料庫

如何？	相關指標
	精細度的位元組。記憶體存放區以 GB 小時為單位計量（儲存 1GB 資料一小時的成本）。因此，將 的每小時值乘MemoryCumulativeBytesMetered 以您區域中的 GB 小時定價，將可為您提供每小時產生的成本。 維度：操作（儲存）DatabaseName、、指標名稱
How can I monitor the amount of data in the magnetic store?	您可以在MagneticCumulativeBytesMetered 指定的期間內監控 ，以監控以位元組為單位儲存在磁性存放區中的資料量。此指標每小時發出一次，您可以追蹤儲存在帳戶和資料庫精細度的位元組。記憶體存放區以 GB 為單位計量（一個月內儲存 1GB 資料的成本）。因此，將 的每小時值乘MagneticCumulativeBytesMetered 以您區域中的 GB 月定價，將可為您提供每小時產生的成本。例如，如果 的值MagneticCumulativeBytesMetered 是 107374182400 位元組（100GB），則磁性儲存體中 1GB 資料的每小時費用 =（0.03）（us-east-1 定價）/（30.4*24）。將此值與 GB MagneticCumulativeBytesMetered 的 相乘，將為該小時提供 ~$0.004。 維度：操作（儲存）DatabaseName、、指標名稱
How can I monitor the data scanned by queries?	您可以在CumulativeBytesMetered 指定的期間內監控，以監控透過傳送至 Timestream for 的查詢（以位元組為單位）掃描的資料 LiveAnalytics。此指標會在查詢執行後發出，您可以追蹤帳戶和資料庫精細度掃描的資料。您可以將指標的值乘以您區域中每個 GB 掃描定價，以計算特定期間的查詢成本。排程查詢掃描的位元組會計入此指標。 維度：操作（查詢）DatabaseName、、指標名稱

如何？	相關指標
How can I monitor the data scanned by scheduled queries?	您可以在CumulativeBytesMetered 指定的期間內監控，以監控 Timestream 為 執行的排程查詢（以位元組為單位）掃描的資料 LiveAnalytics。此指標會在查詢執行後發出，您可以追蹤帳戶和資料庫精細度掃描的資料。您可以將指標的值乘以您區域中每個 GB 掃描定價，以計算特定期間的查詢成本。 **Note** 所計量的位元組也會在查詢 中說明 CumulativeBytesMetered 。 維度：操作（TriggeredScheduledQuery）DatabaseName、、指標名稱
How can I monitor the number of records ingested?	您可以在NumberOfRecords 指定的期間內監控 ，以監控擷取的記錄數目。您可以追蹤儲存在 帳戶和資料庫精細度的位元組。您也可以使用此指標來監控在將查詢結果寫入個別資料表時，排程查詢所做的寫入。 使用 WriteRecords 時API，會為每個WriteRecords 請求發出指標， CloudWatch 操作維度為 WriteRecords 。使用 BatchLoad 或 ScheduledQuery 時APIs，會依服務決定的間隔發出指標，直到任務完成為止。此指標 CloudWatch 的操作維度為 BatchLoad 或 ScheduledQuery ，取決於使用的API維度。 維度：操作（WriteRecords BatchLoad、 或 ScheduledQuery）DatabaseName、、指標名稱

如何？	相關指標
How can I monitor the cost of records ingested?	您可以監控 CumulativeBytesMetered 來監控累積成本的擷取位元組數量。您可以追蹤儲存在 帳戶和資料庫精細度的位元組。擷取的記錄會以累積位元組計量。將 的值乘以區域中CumulativeBytesMetered 的 By Writes 定價，即可產生擷取成本。 使用 WriteRecords 時API，會針對每個WriteRecords 請求發出此指標，操作 CloudWatch 維度為 WriteRecords 。使用 BatchLoad 或 ScheduledQuery 時API，會依服務決定的間隔發出指標，直到任務完成為止。此指標 CloudWatch 的操作維度是 BatchLoad 或 ScheduledQuery ，取決於使用的 API 。 維度：操作 （ WriteRecords BatchLoad、 或 ScheduledQuery ） DatabaseName、、指標名稱
How can I monitor the Timestream Compute Units (TCUs) used in my account?	您可以在QueryTCU指定的期間內監控，以監控帳戶中查詢工作負載消耗的運算單位。此指標會在帳戶作用中查詢工作負載期間，每分鐘發出最大和最小運算單位。 單位：Count 有效統計資料：最小值、最大值 指標： ResourceCount 維度：Service: Timestream 、Resource: QueryTCU、Type: Resource、 Class: OnDemand

LiveAnalytics 指標和維度的時間串流

當您與 的 Timestream 互動時 LiveAnalytics，它會將下列指標和維度傳送至 Amazon CloudWatch。所有指標都會每分鐘彙總和報告。您可以使用下列程序來檢視 的 Timestream 指標 LiveAnalytics。

使用 CloudWatch 主控台檢視指標

指標會先依服務命名空間分組，再依各命名空間內不同的維度組合分類。

1. 在 開啟 CloudWatch 主控台https://console.aws.amazon.com/cloudwatch/。
2. 如有必要請變更區域。在導覽列上，選擇AWS資源所在的區域。如需詳細資訊，請參閱 AWS 服務端點。
3. 在導覽窗格中，選擇 指標。
4. 在所有指標索引標籤下，選擇 AWS/Timestream for LiveAnalytics.

若要使用 檢視指標 AWS CLI

- 在命令提示中，使用下列命令。

```
aws cloudwatch list-metrics --namespace "AWS/Timestream"
```

LiveAnalytics 指標的 Timestream 維度

的 Timestream 指標 LiveAnalytics 由帳戶、資料表名稱或操作的值來驗證。您可以使用 CloudWatch 主控台來擷取 Timestream，以取得下表中任何維度 LiveAnalytics 的資料：

維度	描述
DatabaseName	此維度會將資料限制為 LiveAnalytics 資料庫的特定 Timestream。此值可以是目前區域中的任何資料庫和目前 AWS 帳戶
Operation	此維度將資料限制為其中一個 Timestream 進行 LiveAnalytics 操作，例如 Storage、WriteRecords 、 BatchLoad 或 ScheduledQuery 。如需可用值的清單，請參閱 LiveAnalytics 查詢API參考的時間串流。
TableName	此維度會將資料限制在 LiveAnalyticss 資料庫的 Timestream 中的特定資料表。

Important

CumulativeBytesMetered、 UserErrors和 SystemErrors指標只有 Operation 維度。 SuccessfulRequestLatency指標一律具有 Operation維度，但也可能具有 DatabaseName和 TableName維度，具體取決於 的值Operation。這是因為資料表層級操

作的時間 LiveAnalytics串流具有 DatabaseName和 TableName作為維度，但帳戶層級操作則不會。

LiveAnalytics 指標的時間串流

> **Note**
>
> Amazon 會以一分鐘的間隔 CloudWatch 彙總下列所有 Timestream 的 LiveAnalytics 指標。

一般指標

指標	描述
SuccessfulRequestLatency	在 LiveAnalytics 指定期間內向 Timestream 提出的成功請求。 SuccessfulRequestLatency 可以提供兩種不同類型的資訊： • 成功請求的經過時間（最小值、最大值、總和或平均值）。 • 成功請求的數量 (SampleCount)。 SuccessfulRequestLatency 僅反映 的 Timestream 內的活動 LiveAnalytics ，不會考慮網路延遲或用戶端活動。 單位：Milliseconds 維度 • DatabaseName • TableName • Operation 有效的統計數字： • Minimum

指標	描述
	• Maximum • Average • SampleCount • P10 • p50 • p90 • p95 • p99

寫入和儲存指標

指標	描述
MagneticStoreRejectedRecordCount	非同步拒絕的磁性存放區寫入記錄數目。如果新記錄的版本低於目前版本，或新記錄的版本等於目前版本，但具有不同的資料，則可能會發生這種情況。 單位：Count 維度 • DatabaseName • TableName • Operation 有效的統計數字： • Sum • SampleCount
MagneticStoreRejectedUploadUserFailures	因使用者錯誤而未上傳的磁性存放區拒絕記錄報告數目。這可能是由於IAM許可未正確設定或已刪除 S3 儲存貯體。

指標	描述
	單位：Count 維度 • DatabaseName • TableName • Operation 有效的統計數字： • Sum • SampleCount
MagneticStoreRejectedUpload SystemFailures	因系統錯誤而未上傳的磁性存放區拒絕記錄報告數目。 單位：Count 維度 • DatabaseName • TableName • Operation 有效的統計數字： • Sum • SampleCount

指標	描述
ActiveMagneticStorePartitions	在特定時間主動擷取資料的磁性存放區分割區數量。 單位：Count 維度 • DatabaseName • Operation 有效的統計數字： • Sum • SampleCount

指標	描述
MagneticStorePendingRecordsLatency	寫入磁性存放區的最舊寫入，無法用於查詢。寫入磁性存放區的記錄將在 6 小時內可供查詢。 單位：Milliseconds 維度 • DatabaseName • TableName • Operation 有效的統計數字： • Minimum • Maximum • Average • SampleCount • P10 • p50 • p90 • p95 • p99
MemoryCumulativeBytesMetered	儲存在記憶體存放區中的資料量，以位元組為單位 單位：Bytes 維度：Operation 有效的統計數字： • Average

指標	描述
MagneticCumulativeBytesMetered	儲存在磁性儲存體中的資料量，以位元組為單位 單位：Bytes 維度：Operation 有效的統計數字： • Average
CumulativeBytesMetered	擷取至 Timestream 以位元組 LiveAnalytics為單位的 資料量。 單位：Bytes 維度：Operation 有效統計資料： Sum
NumberOfRecords	擷取至 Timestream 中的記錄數目 LiveAnalytics。 單位：Count 維度：Operation 有效統計資料： Sum

查詢指標

指標	描述
CumulativeBytesMetered	傳送至 Timestream for 的查詢掃描的資料量 LiveAnalytics，以位元組為單位。 單位：Bytes 維度：Operation

指標	描述
	有效的統計數字： • Sum
ResourceCount	帳戶中查詢工作負載耗用的 Timestream Compute Units （TCUs）。此指標會在帳戶作用中查詢工作負載期間，每分鐘發出最大和最小運算單位。 單位：Count 有效統計資料：最小值、最大值 維度：Service: Timestream 、Resource: QueryTCU、Type: Resource、 Class: OnDemand

錯誤指標

指標	描述
SystemErrors	在指定期間內，對 產生 LiveAnalytics 的 Timestream SystemError 請求。 SystemError 通常表示內部服務錯誤。 單位：Count 維度：Operation 有效的統計數字： • Sum • SampleCount
UserErrors	在指定期間內，對 LiveAnalytics 產生 InvalidRequest 錯誤的 Timestream 請求。 InvalidRequest 通常表示用戶端錯誤，例如參數的無效組合、嘗試更新不存在的資料表或不正確的請求

指標	描述
	簽章。 UserErrors 代表目前 AWS 區域和目前 AWS 帳戶的無效請求彙總。 單位：Count 維度：Operation 有效的統計數字： • Sum • SampleCount

> **Important**
>
> 並非所有統計數字，例如 Average 或 Sum，皆適用於所有指標。不過，所有這些值都可以透過適用於 LiveAnalytics 主控台的 Timestream，或使用 CloudWatch 主控台 AWS CLI，或 AWS SDKs適用於所有指標。

建立 CloudWatch 警示以監控 的 Timestream LiveAnalytics

您可以為 Timestream 建立 Amazon CloudWatch 警示 LiveAnalytics ，在警示變更狀態時傳送 Amazon Simple Notification Service （AmazonSNS） 訊息。警示會在您指定的期間監看單一指標。警示會根據在數個期間與指定閾值相關的指標值，來執行一個或多個動作。動作是傳送至 Amazon SNS主題或 Auto Scaling 政策的通知。

警示只會針對持續狀態變更叫用動作。 CloudWatch alarms 不會單純因為動作處於特定狀態而叫用動作。狀態必須已變更，且在指定的期間數內維持此狀態。

如需建立 CloudWatch 警示的詳細資訊，請參閱 Amazon 使用者指南 中的使用 Amazon CloudWatch 警示。 CloudWatch

故障診斷

本節包含 的 Timestream 疑難排解資訊 LiveAnalytics。

主題

處理 WriteRecords 限流

隨著 Timestream 擴展以適應應用程式的資料擷取需求，您的記憶體存放區寫入請求可能會受到限流。如果您的應用程式遇到限流例外狀況，您必須繼續以相同 （或更高） 的輸送量傳送資料，以允許 Timestream 自動擴展到應用程式的需求。

如果接收擷取的磁性存放區分割區上限，您的磁性存放區寫入請求可能會受到限制。您將看到提示訊息，指示您檢查此資料庫的 `ActiveMagneticStorePartitions` Cloudwatch 指標。此限流可能需要長達 6 小時才能解決。若要避免此限流，您應該將記憶體存放區用於任何高輸送量擷取工作負載。對於磁性存放區擷取，您可以透過限制擷取的序列數量和持續時間，將擷取目標設定為較少分割區

如需有關資料擷取最佳實務的詳細資訊，請參閱 寫入。

處理被拒絕的記錄

如果 Timestream 拒絕記錄，您將收到`RejectedRecordsException`有關拒絕的詳細資訊。如需如何從 WriteRecords 回應中擷取此資訊的詳細資訊，請參閱處理寫入失敗。

所有拒絕都會包含在此回應中，但新記錄版本小於或等於現有記錄版本 的磁性存放區更新除外。在此情況下，Timestream 不會更新具有更高版本的現有記錄。Timestream 會拒絕較低或相等版本的新記錄，並以非同步方式將這些錯誤寫入 S3 儲存貯體。若要接收這些非同步錯誤報告，您應該在資料表`MagneticStoreWriteProperties`中設定 `MagneticStoreRejectedDataLocation` 屬性。

UNLOAD 從 Timestream 針對 進行故障診斷 LiveAnalytics

以下是與UNLOAD命令相關的疑難排解指南。

類別	錯誤訊息	如何故障診斷
S3 金鑰長度	UNLOAD 使用目的地中提供的 S3 字首 【%s】 時，結果檔案金鑰會超過 S3 允許的金鑰長	使用 UNLOAD陳述式匯出查詢結果時，包含 S3 儲存貯體名稱和字首長度總和的 S3 金鑰長度 超過支援的 S3 金鑰長度

類別	錯誤訊息	如何故障診斷
	度。如需更多詳細資訊，請參閱 文件。	上限。 S3 建議您減少字首或儲存貯體名稱長度。
	UNLOAD 使用 partitioned_by【%s】 時的結果檔案金鑰將超過 S3 允許的金鑰長度。如需更多詳細資訊，請參閱 文件。	使用 UNLOAD 陳述式匯出查詢結果時，使用 partitioned_by 資料欄的 S3 金鑰長度超過支援的 S3 金鑰長度上限 。建議您使用替代資料欄進行分割，或縮短分割區_資料欄的長度（如果可行）。
	UNLOAD 使用 S3 字首【%s】 搭配 partitioned_by【%s】 時的結果檔案金鑰將超過 S3 允許的金鑰長度。如需更多詳細資訊，請參閱 文件。	使用 UNLOAD陳述式匯出查詢結果時，S3 金鑰長度，包含 S3 儲存貯體名稱、字首和 partitioned_by 資料欄名稱的總和超過支援的 S3 金鑰長度上限。建議您減少字首、儲存貯體名稱長度，或使用替代資料欄來分割資料。
	產生的 S3 物件金鑰：%s 太長。如需更多詳細資訊，請參閱 文件。	使用 UNLOAD 陳述式處理查詢時，分割欄中的其中一個值超過支援的 S3 金鑰長度上限。您可以在產生的物件金鑰中找到分割區資料欄和值。

類別	錯誤訊息	如何故障診斷
S3 限流	我們偵測到 Amazon S3 正在調節來自UNLOAD命令的寫入。如需詳細資訊，請參閱 Amazon Timestream 文件	請參閱此處的 S3 文件 。https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html當多個讀取器/寫入器存取相同的資料夾時，S3 API呼叫率可能會受限。請將通話量稽核至提供的儲存貯體。如果您將相同的儲存貯體用於多個並行UNLOAD查詢，請嘗試將不同的儲存貯體用於相同的查詢。如果您針對 的 Timestream 以外的多個操作使用相同的儲存貯體 LiveAnalytics UNLOAD，請考慮將UNLOAD結果移至個別儲存貯體。

LiveAnalytics 特定錯誤碼的時間串流

本節包含 的 Timestream 特定錯誤代碼 LiveAnalytics。

LiveAnalytics 寫入API錯誤的時間串流

InternalServerException

HTTP 狀態碼：500

ThrottlingException

HTTP 狀態碼：429

ValidationException

HTTP 狀態碼：400

ConflictException

HTTP 狀態碼：409

AccessDeniedException

您沒有足夠存取權可執行此動作。

HTTP 狀態碼：403

ServiceQuotaExceededException

HTTP 狀態碼：402

ResourceNotFoundException

HTTP 狀態碼：404

RejectedRecordsException

HTTP 狀態碼：419

InvalidEndpointException

HTTP 狀態碼：421

LiveAnalytics 查詢API錯誤的時間串流

ValidationException

HTTP 狀態碼：400

QueryExecutionException

HTTP 狀態碼：400

ConflictException

HTTP 狀態碼：409

ThrottlingException

HTTP 狀態碼：429

InternalServerException

HTTP 狀態碼：500

InvalidEndpointException

HTTP 狀態碼：421

配額

本主題說明 Amazon Timestream 中目前配額，也稱為限制 LiveAnalytics。各項配額除非另有說明，否則都是區域特定規定。

主題

預設配額

下表包含 LiveAnalytics 配額的 Timestream 和預設值。

displayName	描述	defaultValue
每個帳戶的資料庫	您可以為每個 建立的資料庫數目上限 AWS 帳戶。	500
每個帳戶的資料表	您可以為每個 建立的資料表數目上限 AWS 帳戶。	50000
的限流率 CRUD APIs	在目前區域中，每個帳戶每秒允許的Create/Update/List/Describe/Delete database/table/scheduled查詢API請求數目上限。	1
每個帳戶的排程查詢	您可以為每個 建立的排程查詢數目上限 AWS 帳戶。	10000

displayName	描述	defaultValue
作用中磁性存放區分割區的最大計數	每個資料庫的作用中磁性存放區分割區數量上限。分割區在收到擷取後最多可能會保持作用中狀態 6 小時。	250
maxQueryTCU	TCUs 您可以為帳戶設定的最大查詢數。	1000

服務限制

下表包含 LiveAnalytics 服務限制的 Timestream 和預設值。若要從主控台編輯資料表的資料保留，請參閱編輯資料表 。

displayName	描述	defaultValue
以分鐘為單位的未來擷取期間	相較於目前系統時間，時間序列資料的最大前置時間（分鐘）。例如，如果未來擷取期間為 15 分鐘，則 的 Timestream LiveAnalytics 將接受比目前系統時間最多 15 分鐘的資料。	15
記憶體儲存的最短保留期，以小時為單位	每個資料表的記憶體存放區中必須保留資料的最短持續時間（以小時為單位）。	1
記憶體儲存的最長保留期間，以小時為單位	每個資料表的記憶體存放區中可保留資料的持續時間上限（以小時為單位）。	8766
磁性存放區的最短保留期間，以天為單位	每個資料表的磁性存放區中必須保留資料的最短持續時間（天數）。	1

displayName	描述	defaultValue
磁性存放區的最大保留期間，以天為單位	資料可以保留在磁性存放區中的最長持續時間（以天為單位）。此值相當於 200 年。	73000
磁性存放區的預設保留期間，以天為單位	每個資料表的磁性存放區中保留資料的預設值（以天為單位）。此值相當於 200 年。	73000
記憶體存放區的預設保留期，以小時為單位	資料保留在記憶體存放區的預設持續時間（小時）。	6
每個資料表的維度	每個資料表的維度上限。	128
測量每個資料表的名稱	每個資料表的唯一量值名稱數目上限。	8192
每個系列的維度名稱維度值對大小	每個序列的維度名稱和維度值對的大小上限。	2 KB
記錄大小上限	記錄的大小上限。	2 KB
每個 WriteRecords API請求的記錄	請求中的 WriteRecords API記錄數目上限。	100
維度名稱長度	Dimension 名稱的位元組數上限。	60 個位元組
測量名稱長度	量值名稱的位元組數上限。	256 位元組
資料庫名稱長度	資料庫名稱的位元組數目上限。	256 位元組
資料表名稱長度	資料表名稱的位元組數目上限。	256 位元組
QueryString KiB 的長度	查詢字串的長度上限（以 KiB 為單位），以 UTF-8 編碼字元表示。	256

displayName	描述	defaultValue
查詢的執行持續時間，以小時為單位	查詢的執行持續時間上限（小時）。花費較長時間的查詢將會逾時。	1
Query Insights	在目前區域中，每個帳戶每秒啟用查詢洞察的允許查詢API請求數量上限。	1
查詢結果的中繼資料大小	查詢結果的中繼資料大小上限。	100 KB
查詢結果的資料大小	查詢結果的資料大小上限。	5 GB
每個多量值記錄的量值	每個多量值記錄的量值數量上限。	256
測量每個多測量記錄的值大小	每個多量值記錄的量值大小上限。	2048
每個資料表跨多測量記錄的唯一測量	單一資料表中定義的所有多測量記錄中的唯一量值。	1024
每個帳戶的 Timestream Compute Units（TCUs）	TCUs 每個帳戶的預設上限。	200

支援的資料類型

下表說明測量和維度值支援的資料類型。

描述	LiveAnalytics 值的時間串流
測量值支援的資料類型。	Big int、雙、字串、布林值、MULTI、時間戳記
維度值支援的資料類型。	字串

批次載入

在批次負載內，目前的配額也稱為限制，如下所示。

描述	LiveAnalytics 值的時間串流
批次載入任務大小上限	批次載入任務大小上限不得超過 100 GB。
檔案數量	批次載入任務不能有超過 100 個檔案。
檔案大小上限	批次載入任務中的檔案大小上限不得超過 5 GB。
CSV 檔案資料列大小	檔案中的資料列CSV不能超過 16 MB。這是無法增加的硬限制。
作用中批次載入任務	資料表不能具有超過 5 個作用中批次載入任務，且帳戶不能具有超過 10 個作用中批次載入任務。的 Timestream LiveAnalytics 會限制新的批次載入任務，直到有更多資源可用為止。

命名限制條件

下表說明命名限制。

描述	LiveAnalytics 值的時間串流
維度名稱的長度上限。	60 個位元組
量值名稱的長度上限。	256 位元組
資料表名稱或資料庫名稱的長度上限。	256 位元組
資料表和資料庫名稱	• 建議您不要使用 系統識別碼。 • 可包含 a-z A-Z 0-9 _（底線）-（破折號）.（點）。 • 所有名稱都必須編碼為 UTF-8，且區分大小寫。

描述	LiveAnalytics 值的時間串流
	Note 使用 UTF-8 二進位表示法比較資料表和資料庫名稱。這表示ASCII字元的比較區分大小寫。
測量名稱	• 不得包含 系統識別碼或冒號 ':'。 • 不得以預留字首 （ `ts_`、 ） 開頭`measure_value` 。 **Note** 使用 UTF-8 二進位表示法比較資料表和資料庫名稱。這表示ASCII字元的比較區分大小寫。
維度名稱	• 不得包含 系統識別碼、冒號 ':' 或雙引號 （ " ）。 • 不得以預留字首 （ `ts_`、 ） 開頭`measure_value` 。 • 不得包含此處列出的 Unicode 字元 【0，31】 或 "\u2028" 或 "\u2029"。 **Note** 使用 UTF-8 二進位表示法比較維度和量值名稱。這表示ASCII字元的比較區分大小寫。
所有資料欄名稱	資料欄名稱無法重複。由於多量值記錄代表維度和量值作為資料欄，因此維度的名稱不能與量值的名稱相同。名稱區分大小寫。

保留的關鍵字

下列所有項目都是預留關鍵字：

- ALTER
- AND

- AS
- BETWEEN
- BY
- CASE
- CAST
- CONSTRAINT
- CREATE
- CROSS
- CUBE
- CURRENT_DATE
- CURRENT_TIME
- CURRENT_TIMESTAMP
- CURRENT_USER
- DEALLOCATE
- DELETE
- DESCRIBE
- DISTINCT
- DROP
- ELSE
- END
- ESCAPE
- EXCEPT
- EXECUTE
- EXISTS
- EXTRACT
- FALSE
- FOR
- FROM
- FULL

- GROUP
- GROUPING
- HAVING
- IN
- INNER
- INSERT
- INTERSECT
- INTO
- IS
- JOIN
- LEFT
- LIKE
- LOCALTIME
- LOCALTIMESTAMP
- NATURAL
- NORMALIZE
- NOT
- NULL
- ON
- 或
- ORDER
- OUTER
- PREPARE
- RECURSIVE
- RIGHT
- ROLLUP
- SELECT
- TABLE
- THEN
- TRUE

- UESCAPE
- UNION
- UNNEST
- USING
- VALUES
- WHEN
- WHERE
- WITH

系統識別碼

我們保留資料欄名稱「measure_value」、「ts_non_existent_col」和「time」作為系統識別符的時間串流 LiveAnalytics。此外，資料欄名稱不能以 "ts_" 或 "measure_name" 開頭。系統識別碼區分大小寫。使用 UTF-8 二進位表示法比較的識別符。這表示識別碼的比較區分大小寫。

> **Note**
>
> 系統識別符不得用於維度或量值名稱。建議您不要將系統識別碼用於資料庫或資料表名稱。

UNLOAD

如需與UNL0AD命令相關的限制，請參閱使用 UNLOAD 將查詢結果從 Timestream 匯出至 S3。

查詢語言參考

> **Note**
>
> 此查詢語言參考包括下列來自 Trino Software Foundation （先前稱為 Presto Software Foundation） 的第三方文件，其根據 Apache License 2.0 版授權。除非符合此授權，否則您無法使用此檔案。若要取得 Apache 授權 2.0 版的副本，請參閱 Apache 網站 。

的 Timestream LiveAnalytics 支援豐富的查詢語言，以使用您的資料。您可以在下方查看可用的資料類型、運算子、函數和建構。

您也可以立即開始使用 範例查詢 區段中的 Timestream 查詢語言。

主題

支援的資料類型

LiveAnalytics查詢語言的時間串流支援下列資料類型。

> **Note**
>
> 支援寫入的資料類型會在資料類型 中說明。

資料類型	描述
`int`	代表 32 位元整數。
`bigint`	代表 64 位元已簽署整數。
`boolean`	邏輯的兩個事實值之一，`True`以及 `False`。
`double`	代表 64 位元變數精確度資料類型。實作IEEE二進位浮點運算的標準 754。 **Note** 查詢語言用於讀取資料。`Infinity` 和`NaN`雙值的函數可用於查詢。但您無法將這些值寫入 Timestream。
`varchar`	大小上限為 2KB 的變數長度字元資料。
`array[T,...]`	包含指定資料類型的一或多個元素 *T*其中，*T* 可以是 Timestream 中支援的任何資料類型。
`row(T,...)`	包含資料類型的一或多個具名欄位 *T*。 這些欄位可以是 Timestream 支援的任何資料類型，並使用點欄位參考運算子存取： `.`
`date`	代表格式為 的日期*YYYY*-*MM*-*DD*。其中 *YYYY* 是年份 *MM* 是月份，且 *DD* 分別是天。支援的範圍是 `1970-01-01` 到 `2262-04-11` 。 範例： `1971-02-03`
`time`	代表 中一天的時間UTC。`time` 資料類型以*HH*.*MM*.*SS*.*sssssssss* .支援奈秒精確度的形式表示。

資料類型	描述
	範例： `17:02:07.496000000`
`timestamp`	代表在 中使用奈秒精確度時間的執行個體UTC。 *YYYY*-*MM*-*DD* *hh*:*mm*:*ss*.*sssssssss* 查詢支援 1677-09-21 00:12:44.000000000 到 範圍內的時間戳記2262-04-11 23:47:16.854775807 。

資料類型	描述
interval	表示字串常值 的時間間隔*X**t*，由兩個部分組成，*X* 以及 *t*. *X* 是大於或等於 的數值0，且 *t* 是秒或小時等時間的單位。裝置未分段。時間單位 *t* 必須是下列其中一個字串常值： • nanosecond • microsecond • millisecond • second • minute • hour • day • ns（與 相同nanosecond ） • us（與 相同microsecond ） • ms（與 相同millisecond ） • s（與 相同second） • m（與 相同minute） • h（與 相同hour） • d（與 相同day） 範例： `17s` `12second` `21hour` `2d`

資料類型	描述
timeseries[row(timestamp, *T*,...)]	表示以row物件array組成的時間間隔內記錄的量值值。每個 row都包含資料類型的 timestamp 和一或多個量值 *T*其中，*T* 可以是 bigint、boolean、 double或 中的任何一個varchar。資料列會依 遞增排序timestamp 。Timeeries 資料類型代表一段時間內量值的值。
unknown	代表 null 資料。

內建時間序列功能

的 Timestream LiveAnalytics 提供內建的時間序列功能，可將時間序列資料視為一流的概念。

內建時間序列功能可分為兩個類別：檢視和函數。

您可以閱讀以下每個建構的相關資訊。

主題

- Timeseries 檢視
- 時間序列函數

Timeseries 檢視

的 Timestream LiveAnalytics 支援下列函數，可將您的資料轉換為timeseries資料類型：

主題

- CREATE_TIME_SERIES
- UNNEST

CREATE_TIME_SERIES

CREATE_TIME_SERIES 是彙總函數，會取得時間序列的所有原始測量 （ 時間和測量值 ），並傳回時脈資料類型。此函數的語法如下：

```
CREATE_TIME_SERIES(time, measure_value::<data_type>)
```

其中 <data_type>是測量值的資料類型，可以是 bigint、布林值、雙值或 varchar 的其中一個。第二個參數不能為 null。

考慮CPU使用儲存在名為指標的資料表中的EC2執行個體，如下所示：

時間	region	az	vpc	instance_id	measure_name	measure_value::double
2019-12-04 19:00:00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	35.0
2019-12-04 19:00:01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	38.2
2019-12-04 19:00:02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	45.3
2019-12-04 19:00:00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef1	cpu_utilization	54.1
2019-12-04 19:00:01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef1	cpu_utilization	42.5
2019-12-04 19:00:02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef1	cpu_utilization	33.7

執行查詢：

```
SELECT region, az, vpc, instance_id, CREATE_TIME_SERIES(time, measure_value::double) as
 cpu_utilization FROM metrics
    WHERE measure_name='cpu_utilization'
    GROUP BY region, az, vpc, instance_id
```

將傳回所有具有 cpu_utilization作為測量值的序列。在這種情況下，我們有兩個系列：

region	az	vpc	instance_id	cpu_utilization
us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	【{時間：2019-12-04 19：00：00.000000000 ， measure_value： ：double： 35.0} , {時間：2019-12-04 19：00：01.000000000 ， measure_value： ：double： 38.2} , {時間：2019-12-04 19：00：02.000000000 ， measure_value： ： ：double： 45.3}】
us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef1	【{時間：2019-12-04 19：00：00.000000000 ， measure_value： ：double： 35.1} , {時間：2019-12-04 19：00：

region	az	vpc	instance_id	cpu_utilization
				01.000000000，measure_value：：double：38.5}，{時間：2019-12-04 19：00：02.000000000，measure_value：：：double：45.7}】

UNNEST

UNNEST 是資料表函數，可讓您將timeseries資料轉換為平面模型。語法如下：

UNNEST 將 timeseries 轉換為兩個資料欄，即 time和 value。您也可以將別名與 搭配使用 UNNEST，如下所示：

```
UNNEST(timeseries) AS <alias_name> (time_alias, value_alias)
```

其中 <alias_name>是平面資料表的別名， time_alias 是time資料欄的別名， value_alias是資料value欄的別名。

例如，請考慮您機群中的某些EC2執行個體設定為以 5 秒間隔發出指標的情況，其他執行個體則以 15 秒間隔發出指標，而您需要過去 6 小時內以 10 秒精細度發出所有執行個體的平均指標。若要取得此資料，您可以使用 CREATE_TIME_SERIES 將指標轉換為時間序列模型。然後，您可以使用 INTERPOLATE_LINEAR 取得 10 秒精細度的遺失值。接下來，您可以使用 將資料轉換回一般模型UNNEST，然後使用 AVG取得所有執行個體的平均指標。

```
WITH interpolated_timeseries AS (
    SELECT region, az, vpc, instance_id,
        INTERPOLATE_LINEAR(
            CREATE_TIME_SERIES(time, measure_value::double),
                SEQUENCE(ago(6h), now(), 10s)) AS interpolated_cpu_utilization
    FROM timestreamdb.metrics
    WHERE measure_name= 'cpu_utilization' AND time >= ago(6h)
```

```
    GROUP BY region, az, vpc, instance_id
)
SELECT region, az, vpc, instance_id, avg(t.cpu_util)
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_cpu_utilization) AS t (time, cpu_util)
GROUP BY region, az, vpc, instance_id
```

上述查詢示範UNNEST搭配別名使用 。以下是未使用 別名的相同查詢範例UNNEST：

```
WITH interpolated_timeseries AS (
    SELECT region, az, vpc, instance_id,
        INTERPOLATE_LINEAR(
            CREATE_TIME_SERIES(time, measure_value::double),
                SEQUENCE(ago(6h), now(), 10s)) AS interpolated_cpu_utilization
    FROM timestreamdb.metrics
    WHERE measure_name= 'cpu_utilization' AND time >= ago(6h)
    GROUP BY region, az, vpc, instance_id
)
SELECT region, az, vpc, instance_id, avg(value)
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_cpu_utilization)
GROUP BY region, az, vpc, instance_id
```

時間序列函數

的 Amazon Timestream LiveAnalytics 支援時脈函數，例如衍生工具、積分和關聯等，以從您的時間序列資料中取得更深入的洞見。本節提供每個函數的使用資訊，以及範例查詢。選取以下主題以進一步了解。

主題

- 插補函數
- 衍生工具函數
- 整合函數
- 關聯函數
- 篩選和減少函數

插補函數

如果您的時間序列資料在某些時間點遺失事件的值，您可以使用插補來估計遺失事件的值。Amazon Timestream 支援四個插補變體：線性插補、立方曲線插補、最後觀察值推估（locf）插補和恆定插補。本節提供 Timestream 插 LiveAnalytics補函數的使用資訊，以及範例查詢。

用量資訊

函式	輸出資料類型	描述
`interpolate_linear(timeseries, array[timestamp])`	計時	使用線性插補 填入遺失的資料。
`interpolate_linear(timeseries, timestamp)`	double	使用線性插補 填入遺失的資料。
`interpolate_spline_cubic(timeseries, array[timestamp])`	計時	使用立方 spline 插補 填入缺少的資料。
`interpolate_spline_cubic(timeseries, timestamp)`	double	使用立方括號插補 填入缺少的資料。
`interpolate_locf(timeseries, array[timestamp])`	計時	使用上次取樣的值填入遺失的資料。
`interpolate_locf(timeseries, timestamp)`	double	使用上次取樣的值填入遺失的資料。
`interpolate_fill(timeseries, array[timestamp], double)`	計時	使用常數值填入遺失的資料。

函式	輸出資料類型	描述
interpolate_fill(timeseries, timestamp, double)	double	使用常數值填入遺失的資料。

查詢範例

Example

尋找過去 2 小時內特定EC2主機以 30 秒間隔固定的平均CPU使用率，並使用線性插補填入缺少的值：

```
WITH binned_timeseries AS (
SELECT hostname, BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double),
 2) AS avg_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
SELECT hostname,
    INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
            SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
FROM binned_timeseries
GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

Example

找出過去 2 小時內特定EC2主機以 30 秒間隔固定的平均CPU使用率，根據上次轉送的觀察使用插補來填入遺失值：

```
WITH binned_timeseries AS (
SELECT hostname, BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double),
 2) AS avg_cpu_utilization
```

```
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
SELECT hostname,
    INTERPOLATE_LOCF(
        CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
            SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
FROM binned_timeseries
GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

衍生工具函數

衍生性商品會計算特定指標的變更率，並可用來主動回應事件。例如，假設您計算過去 5 分鐘內EC2執行個體CPU使用率的衍生性商品，並注意到顯著的正衍生性商品。這可能表示工作負載的需求增加，因此您可能決定分割更多EC2執行個體，以更好地處理您的工作負載。

Amazon Timestream 支援衍生函數的兩種變體。本節提供 LiveAnalytics 衍生函數 Timestream 的使用資訊，以及範例查詢。

用量資訊

函式	輸出資料類型	描述
derivative_linear(timeseries, interval)	計時	針對指定的 ，計算 中每個點timeseries 的導數interval。
non_negative_derivative_linear(timeseries, interval)	計時	與 相同derivative_linear(timeseries, interval) ，但只會傳回正值。

查詢範例

Example

找出過去 1 小時內每 5 分鐘CPU的使用率變化率：

```
SELECT DERIVATIVE_LINEAR(CREATE_TIME_SERIES(time, measure_value::double), 5m) AS
 result
FROM “sampleDB”.DevOps
WHERE measure_name = 'cpu_utilization'
AND hostname = 'host-Hovjv' and time > ago(1h)
GROUP BY hostname, measure_name
```

Example

計算一個或多個微服務產生的錯誤增加率：

```
WITH binned_view as (
    SELECT bin(time, 5m) as binned_timestamp, ROUND(AVG(measure_value::double), 2) as
 value
    FROM “sampleDB”.DevOps
    WHERE micro_service = 'jwt'
    AND time > ago(1h)
    AND measure_name = 'service_error'
    GROUP BY bin(time, 5m)
)
SELECT non_negative_derivative_linear(CREATE_TIME_SERIES(binned_timestamp, value), 1m)
 as rateOfErrorIncrease
FROM binned_view
```

整合函數

您可以使用積分來尋找時間序列事件的每單位時間曲線下面積。例如，假設您正在追蹤應用程式每單位時間收到的請求量。在此案例中，您可以使用積分函數來判斷在特定期間內每個指定間隔所服務的請求總量。

Amazon Timestream 支援整合函數的一個變體。本節提供 Timestream for LiveAnalytics integral 函數的使用資訊，以及範例查詢。

用量資訊

函式	輸出資料類型	描述
`integral_trapezoidal(timeseries(double))` `integral_trapezoidal(timeseries(double), interval day to second)` `integral_trapezoidal(timeseries(bigint))` `integral_trapezoidal(timeseries(bigint), interval day to second)` `integral_trapezoidal(timeseries(integer), interval day to second)` `integral_trapezoidal(timeseries(integer))`	double	使用梯形規則 ，根據`interval day to second`為`timeseries` 提供的 指定的 近似積分。間隔天數到第二個參數為選用，預設值為 1s。如需間隔的詳細資訊，請參閱 間隔和持續時間。

查詢範例

Example

計算特定主機在過去一小時中每五分鐘服務的請求總量：

```
SELECT INTEGRAL_TRAPEZOIDAL(CREATE_TIME_SERIES(time, measure_value::double), 5m) AS
 result FROM sample.DevOps
```

```
WHERE measure_name = 'request'
AND hostname = 'host-Hovjv'
AND time > ago (1h)
GROUP BY hostname, measure_name
```

關聯函數

假設有兩個類似的長度時間序列，關聯函數提供關聯係數，這說明了兩個時間序列如何隨時間推移而趨勢。相關係數的範圍從 -1.0到 1.0。 -1.0表示兩個時間序列以相同速率相反方向的趨勢。 而 1.0 表示兩個倍數以相同速率在相同方向的趨勢。的值0表示兩個時間序列之間沒有關聯。例如，如果油價上漲，且油公司的股價上漲，油價上漲的趨勢和油價公司的價格上漲將具有正相關係數。高正相關係數表示兩個價格的趨勢速率相似。同樣地，債券價格與債券收益率之間的關聯係數為負數，表示這兩個值會隨著時間的推移朝相反方向發展。

Amazon Timestream 支援兩種相互關聯的函數變體。本節提供 LiveAnalytics Timestream 相關函數的使用資訊，以及範例查詢。

用量資訊

函式	輸出資料類型	描述
`correlate_pearson( timeseries, timeseries)`	double	計算兩個 的 Pearson 相關係數`timeseries` 。時間記錄必須具有相同的時間戳記。
`correlate_spearman (timeseries, timeseries)`	double	計算兩個 的 Spearman 相關係數`timeseries` 。時間記錄必須具有相同的時間戳記。

查詢範例

Example

```
WITH cte_1 AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, measure_value::double),
        SEQUENCE(min(time), max(time), 10m)) AS result
    FROM sample.DevOps
    WHERE measure_name = 'cpu_utilization'
```

```
    AND hostname = 'host-Hovjv' AND time > ago(1h)
    GROUP BY hostname, measure_name
),
cte_2 AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, measure_value::double),
        SEQUENCE(min(time), max(time), 10m)) AS result
    FROM sample.DevOps
    WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv' AND time > ago(1h)
    GROUP BY hostname, measure_name
)
SELECT correlate_pearson(cte_1.result, cte_2.result) AS result
FROM cte_1, cte_2
```

篩選和減少函數

Amazon Timestream 支援執行篩選和減少時間序列資料操作的功能。本節提供用於 LiveAnalytics 篩選和減少函數的 Timestream 的使用資訊，以及範例查詢。

用量資訊

函式	輸出資料類型	描述
`filter(timeseries(T), function(T, Boolean))`	timeseries (T)	使用傳遞`function`傳回 的值，從輸入時間序列建構時間序列`true`。
`reduce(timeseries(T), initialState S, inputFunction(S, T, S), outputFunction(S, R))`	R	傳回單一值，從時間序列減少。`inputFunction` 將依序叫用在時間集中的每個元素上。除了採用目前的 元素之外， 還會 inputFunction 取得目前的狀態 (一開始是 `initialState`) ，並傳回新的狀態。`outputFunction` 將叫用 ，將最終狀態轉換為結果值。`outputFunction` 可以是身分函數。

查詢範例

Example

建立主機和篩選條件點CPU使用時間序列，其測量值大於 70：

```
WITH time_series_view AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, ROUND(measure_value::double,2)),
            SEQUENCE(ago(15m), ago(1m), 10s)) AS cpu_user
    FROM sample.DevOps
    WHERE hostname = 'host-Hovjv' and measure_name = 'cpu_utilization'
        AND time > ago(30m)
    GROUP BY hostname
)
SELECT FILTER(cpu_user, x -> x.value > 70.0) AS cpu_above_threshold
from time_series_view
```

Example

建立主機CPU使用率的時間序列，並決定測量的平方和：

```
WITH time_series_view AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, ROUND(measure_value::double,2)),
            SEQUENCE(ago(15m), ago(1m), 10s)) AS cpu_user
    FROM sample.DevOps
    WHERE hostname = 'host-Hovjv' and measure_name = 'cpu_utilization'
        AND time > ago(30m)
    GROUP BY hostname
)
SELECT REDUCE(cpu_user,
    DOUBLE '0.0',
    (s, x) -> x.value * x.value + s,
    s -> s)
from time_series_view
```

Example

建構主機CPU使用時間序列，並判斷高於CPU閾值的取樣分數：

```
WITH time_series_view AS (
    SELECT INTERPOLATE_LINEAR(
```

```
        CREATE_TIME_SERIES(time, ROUND(measure_value::double,2)),
            SEQUENCE(ago(15m), ago(1m), 10s)) AS cpu_user
    FROM sample.DevOps
    WHERE hostname = 'host-Hovjv' and measure_name = 'cpu_utilization'
        AND time > ago(30m)
    GROUP BY hostname
)
SELECT ROUND(
    REDUCE(cpu_user,
      -- initial state
      CAST(ROW(0, 0) AS ROW(count_high BIGINT, count_total BIGINT)),
      -- function to count the total points and points above a certain threshold
      (s, x) -> CAST(ROW(s.count_high + IF(x.value > 70.0, 1, 0), s.count_total + 1) AS
 ROW(count_high BIGINT, count_total BIGINT)),
      -- output function converting the counts to fraction above threshold
      s -> IF(s.count_total = 0, NULL, CAST(s.count_high AS DOUBLE) / s.count_total)),
    4) AS fraction_cpu_above_threshold
from time_series_view
```

SQL 支援

的 Timestream SQL LiveAnalytics 支援一些常見的建構。您可以在下方閱讀更多內容。

主題

- SELECT
- 子查詢支援
- SHOW 陳述式
- DESCRIBE 陳述式
- UNLOAD

SELECT

SELECT 陳述式可用來從一或多個資料表擷取資料。Timestream 的查詢語言支援下列SELECT陳述式語法：

```
[ WITH with_query [, ...] ]
            SELECT [ ALL | DISTINCT ] select_expr [, ...]
            [ function (expression) OVER (
            [ PARTITION BY partition_expr_list ]
            [ ORDER BY order_list ]
```

```
            [ frame_clause ] )
            [ FROM from_item [, ...] ]
            [ WHERE condition ]
            [ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
            [ HAVING condition]
            [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
            [ ORDER BY order_list ]
            [ LIMIT [ count | ALL ] ]
```

where

- function (expression) 是支援的視窗函數 之一。
- partition_expr_list 是：

```
expression | column_name [, expr_list ]
```

- order_list 是：

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

- frame_clause 是：

```
ROWS | RANGE
{ UNBOUNDED PRECEDING | expression PRECEDING | CURRENT ROW } |
{BETWEEN
{ UNBOUNDED PRECEDING | expression { PRECEDING | FOLLOWING } |
CURRENT ROW}
AND
{ UNBOUNDED FOLLOWING | expression { PRECEDING | FOLLOWING } |
CURRENT ROW }}
```

- from_item 是下列其中一項：

```
table_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
from_item join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]
```

- join_type 是下列其中一項：

```
[ INNER ] JOIN
LEFT [ OUTER ] JOIN
```

```
RIGHT [ OUTER ] JOIN
FULL [ OUTER ] JOIN
```

- grouping_element 是下列其中一項：

```
()
expression
```

子查詢支援

Timestream 支援 EXISTS和 IN 述詞中的子查詢。EXISTS 述詞會判斷子查詢是否傳回任何資料列。IN 述詞會判斷子查詢產生的值是否符合 IN 子句中的 值或表達式。Timestream 查詢語言支援關聯的 和其他子查詢。

```
SELECT t.c1
FROM (VALUES 1, 2, 3, 4, 5) AS t(c1)
WHERE EXISTS
(SELECT t.c2
 FROM (VALUES 1, 2, 3) AS t(c2)
 WHERE t.c1= t.c2
)
ORDER BY t.c1
```

c1
1
2
3

```
SELECT t.c1
FROM (VALUES 1, 2, 3, 4, 5) AS t(c1)
WHERE t.c1 IN
(SELECT t.c2
 FROM (VALUES 2, 3, 4) AS t(c2)
)
ORDER BY t.c1
```

c1
2
3
4

SHOW 陳述式

您可以使用 SHOW DATABASES陳述式來檢視帳戶中的所有資料庫。語法如下：

```
SHOW DATABASES [LIKE pattern]
```

其中子LIKE句可用於篩選資料庫名稱。

您可以使用 SHOW TABLES陳述式來檢視帳戶中的所有資料表。語法如下：

```
SHOW TABLES [FROM database] [LIKE pattern]
```

其中子FROM句可用於篩選資料庫名稱，子LIKE句可用於篩選資料表名稱。

您可以使用 SHOW MEASURES陳述式來檢視資料表的所有量值。語法如下：

```
SHOW MEASURES FROM database.table [LIKE pattern]
```

其中子FROM句將用於指定資料庫和資料表名稱，而LIKE子句可用於篩選量值名稱。

DESCRIBE 陳述式

您可以使用 DESCRIBE陳述式檢視資料表的中繼資料。語法如下：

```
DESCRIBE database.table
```

table 包含資料表名稱的 。描述陳述式會傳回資料表的資料欄名稱和資料類型。

UNLOAD

的 Timestream LiveAnalytics 支援 UNLOAD命令作為其SQL支援的延伸。中UNLOAD說明 支援的資料類型支援的資料類型。time 和 unknown類型不適用於 UNLOAD。

```
UNLOAD (SELECT statement)
 TO 's3://bucket-name/folder'
 WITH ( option = expression [, ...] )
```

其中選項為

```
{ partitioned_by = ARRAY[ col_name[,…] ]
 | format = [ '{ CSV | PARQUET }' ]
 | compression = [ '{ GZIP | NONE }' ]
 | encryption = [ '{ SSE_KMS | SSE_S3 }' ]
 | kms_key = '<string>'
 | field_delimiter ='<character>'
 | escaped_by = '<character>'
 | include_header = ['{true, false}']
 | max_file_size = '<value>'
}
```

SELECT 陳述式

用於從資料表的一或多個 Timestream 選取和擷取資料的查詢陳述式 LiveAnalytics 。

```
(SELECT column 1, column 2, column 3 from database.table
      where measure_name = "ABC" and timestamp between ago (1d) and now() )
```

TO 子句

```
TO 's3://bucket-name/folder'
```

或

```
TO 's3://access-point-alias/folder'
```

UNLOAD 陳述式中的TO子句指定查詢結果輸出的目的地。您需要提供完整路徑，包括 Amazon S3 儲存貯體名稱或 Amazon S3 access-point-alias，其中 的資料夾位置位於 Amazon S3，其中 Timestream for LiveAnalytics 會寫入輸出檔案物件。S3 儲存貯體應該由相同帳戶擁有，且位於相同區域。除了查詢結果集之外， 的 Timestream 也會將資訊清單和中繼資料檔案 LiveAnalytics 寫入指定的目的地資料夾。

PARTITIONED_BY 子句

```
partitioned_by = ARRAY [col_name[,…] , (default: none)
```

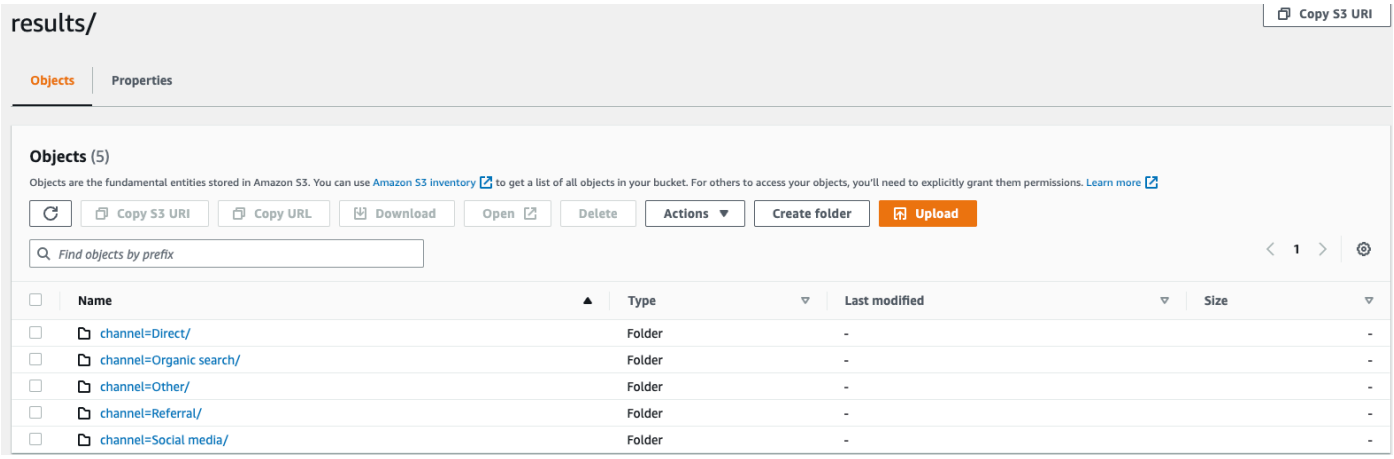
partitioned_by 子句用於查詢，以精細層級分組和分析資料。當您將查詢結果匯出至 S3 儲存貯體時，您可以選擇根據選取查詢中的一或多個資料欄來分割資料。分割資料時，匯出的資料會根據分割區資料欄分為子集，且每個子集都存放在個別的資料夾中。在包含匯出資料的結果資料夾中，folder/results/partition column = partition value/會自動建立子資料夾。不過請注意，分割的資料欄不包含在輸出檔案中。

partitioned_by 不是語法中的強制性子句。如果您選擇在沒有分割的情況下匯出資料，則可以在語法中排除 子句。

Example

假設您正在監控網站的點擊串流資料，並具有 5 個流量通道，即 direct、Social Media、Organic Search、 Other和 Referral。匯出資料時，您可以選擇使用資料欄 分割資料Channel。在資料資料夾 內s3://bucketname/results，您會有五個資料夾，每個資料夾都有各自的頻道名稱，例如，s3://bucketname/results/channel=Social Media/.在此資料夾中，您會找到透過Social Media頻道登陸網站的所有客戶的資料。同樣地，您將擁有其他資料夾用於其餘通道。

依頻道資料欄分割的匯出資料



FORMAT

```
format = [ '{ CSV | PARQUET }' , default: CSV
```

用來指定寫入 S3 儲存貯體之查詢結果格式的關鍵字。您可以使用逗號（ ，CSV ）作為預設分隔符號，以逗號分隔值（ ） 匯出資料，或以 Apache Parquet 格式匯出資料，這是用於分析的有效開放資料欄儲存格式。

COMPRESSION

```
compression = [ '{ GZIP | NONE }' ], default: GZIP
```

您可以使用壓縮演算法壓縮匯出的資料，GZIP或指定 NONE選項使其取消壓縮。

ENCRYPTION

```
encryption = [ '{ SSE_KMS | SSE_S3 }' ], default: SSE_S3
```

Amazon S3 上的輸出檔案會使用您選取的加密選項進行加密。除了您的資料之外，資訊清單和中繼資料檔案也會根據您選取的加密選項進行加密。我們目前支援 SSE_S3 和 SSE_KMS 加密。SSE_S3 是一種伺服器端加密，Amazon S3 使用 256 位元進階加密標準 （AES） 加密資料。SSE_KMS 是一種伺服器端加密，可使用客戶管理的金鑰加密資料。

KMS_KEY

```
kms_key = '<string>'
```

KMS 金鑰是客戶定義的金鑰，用於加密匯出的查詢結果。KMS 金鑰由 AWS Key Management Service （AWS KMS） 安全管理，並用於加密 Amazon S3 上的資料檔案。

FIELD_DELIMITER

```
field_delimiter ='<character>' , default: (,)
```

以 CSV 格式匯出資料時，此欄位會指定用於分隔輸出檔案中欄位的單一ASCII字元，例如管道字元（|）、逗號（ ，） 或索引標籤（/t）。CSV 檔案的預設分隔符號是逗號字元。如果資料中的值包含選取的分隔符號，則會以引號字元引述分隔符號。例如，如果您資料中的值包含 Time,stream，則此值會如匯出資料"Time,stream"中一樣引述。Timestream 使用的引號字元 LiveAnalytics 是雙引號（"）。

FIELD_DELIMITER 如果您想要在 中包含標頭，請避免指定歸位字元（ASCII 13， 十六進位 0D， 文字 '\r'） 或換行字元（ASCII 10， 十六進位 0A ， 文字 '\n'）CSV，因為這會阻止許多剖析器在產生的CSV輸出中正確剖析標頭。

ESCAPED_BY

```
escaped_by = '<character>', default: (\)
```

以 CSV 格式匯出資料時，此欄位會指定在寫入 S3 儲存貯體的資料檔案中應視為逸出字元的字元。逸出會在下列情況下發生：

1. 如果值本身包含引號字元（"），則會使用逸出字元逸出。例如，如果值為 Time"stream，其中（\）是設定的逸出字元，則會以 逸出Time\"stream。
2. 如果值包含設定的逸出字元，則會逸出。例如，如果值為 Time\stream，則會以 的形式逸出Time\\stream。

> **Note**
>
> 如果匯出的輸出包含類似 Arrays、Rows 或 Timeseries 的複雜資料類型，則會將其序列化為JSON字串。以下是範例。

資料類型	實際值	如何逸出CSV格式為 【序列化JSON字串】 的值
陣列	`[ 23,24,25 ]`	`"[23,24,25]"`
Row	`( x=23.0, y=hello )`	`"{\"x\":23.0,\"y\":\"hello\"}"`
時間序列	`[ ( time=1970-01-01 00:00:00.000000010, value=100.0 ), ( time=1970-01-01 00:00:00.000000012, value=120.0 ) ]`	`"[{\"time\":\"1970-01-01 00:00:00.000000010Z\",\"value\":100.0},{\"time\":\"1970-01-01 00:00:00.000000012Z\",\"value\":120.0}]"`

INCLUDE_HEADER

```
include_header = 'true' , default: 'false'
```

以 CSV 格式匯出資料時，此欄位可讓您將資料欄名稱包含為匯出CSV資料檔案的第一列。

接受的值為 'true' 和 'false'，預設值為 'false'。文字轉換選項，例如 `escaped_by`和也`field_delimiter`適用於標頭。

> **Note**
>
> 包含標頭時，請務必不要選取歸位字元（ASCII 13，十六進位 0D文字 '\r'）或換行字元（ASCII 10，十六進位 0A，文字 '\n'）作為 `FIELD_DELIMITER`，因為這會阻止許多剖析器在產生的CSV輸出中正確剖析標頭。

MAX_FILE_SIZE

```
max_file_size = 'X[MB|GB]' , default: '78GB'
```

此欄位指定UNLOAD陳述式在 Amazon S3 中建立的檔案大小上限。UNLOAD 陳述式可以建立多個檔案，但寫入 Amazon S3 的每個檔案的大小上限大約是此欄位中指定的大小。

欄位的值必須介於 16 MB 和 78 GB 之間，包括在內。您可以指定整數，例如 `12GB`或小數，例如 `0.5GB`或 `24.7MB`。預設值為 78 GB。

實際檔案大小是在寫入檔案時近似的，因此實際大小上限可能不等於您指定的數字。

邏輯運算子

的 Timestream LiveAnalytics 支援下列邏輯運算子。

運算子	描述	範例
AND	如果兩個值都是 true，則為 True	a AND b
或	如果任一值為 true，則為 True	a OR b

運算子	描述	範例
NOT	如果值為 false，則為 True	NOT a

- 如果表達式的一端或兩端是 ，NULL則AND比較結果可能是 NULL。
- 如果AND運算子的至少一端是 運算式FALSE，則會評估為 FALSE。
- 如果表達式的一端NULL或兩端為 ，則OR比較結果可能為 NULL。
- 如果OR運算子的至少一端是 運算式TRUE，則會評估為 TRUE。
- 的邏輯補充NULL為 NULL。

下列事實資料表示範 AND和 NULL中的 處理OR方式：

A	B	A 和 b	A 或 b
null	null	null	null
false	null	false	null
null	false	false	null
true	null	null	true
null	true	null	true
false	false	false	false
true	false	false	true
false	true	false	true
true	true	true	true

下列事實資料表示範 NULL中 的 處理NOT方式：

A	不是
null	null
true	false
false	true

比較運算子

的 Timestream LiveAnalytics 支援下列比較運算子。

運算子	描述
<	小於
>	大於
<=	小於或等於
>=	大於或等於
=	等於
<>	不等於
!=	不等於

Note

- BETWEEN 運算子會測試值是否在指定的範圍內。語法如下：

  ```
  BETWEEN min AND max
  ```

  BETWEEN 或 NOT BETWEEN陳述式NULL中的 存在將導致陳述式評估為 NULL。
- IS NULL 和 IS NOT NULL 運算子會測試值是否為 null （未定義）。NULL 搭配 使用 會IS NULL評估為 true。

- 在 中SQL，NULL值表示未知值。

比較函數

的 Timestream LiveAnalytics 支援下列比較函數。

主題

greatest ( )

greatest ( ) 函數會傳回所提供值的最大值。NULL 如果任何提供的值為 ，則會傳回 NULL。語法如下。

```
greatest(value1, value2, ..., valueN)
```

least ( )

least ( ) 函數會傳回所提供值的最小值。NULL 如果任何提供的值為 ，則會傳回 NULL。語法如下。

```
least(value1, value2, ..., valueN)
```

ALL ( )、ANY ( ) 和 SOME ( )

ALL、 ANY和 SOME 量化器可以搭配比較運算子使用，方式如下。

表達式	意義
A = ALL (...)	當 A 等於所有值時，評估為 true。
A <> ALL (...)	當 A 與任何值不相符時，評估為 true。
A < ALL (...)	當 A 小於最小值時，評估為 true。
A = ANY (...)	當 A 等於任何值時， 會評估為 true。

表達式	意義
A <> ANY (...)	當 A 不符合一或多個值時， 會評估為 true。
A < ANY (...)	當 A 小於最大值時，評估為 true。

範例和用量備註

> **Note**
>
> 使用 ALL、 ANY或 時SOME，如果比較值是常值清單，VALUES則應使用關鍵字。

範例：**ANY()**

查詢陳述式ANY()中的 範例，如下所示。

```
SELECT 11.7 = ANY (VALUES 12.0, 13.5, 11.7)
```

相同操作的替代語法如下所示。

```
SELECT 11.7 = ANY (SELECT 12.0 UNION ALL SELECT 13.5 UNION ALL SELECT 11.7)
```

在此情況下， 會ANY()評估為 True。

範例：**ALL()**

查詢陳述式ALL()中的 範例，如下所示。

```
SELECT 17 < ALL (VALUES 19, 20, 15);
```

相同操作的替代語法如下所示。

```
SELECT 17 < ALL (SELECT 19 UNION ALL SELECT 20 UNION ALL SELECT 15);
```

在此情況下， 會ALL()評估為 False。

範例：**SOME()**

查詢陳述式SOME()中的 範例，如下所示。

```
SELECT 50 >= SOME (VALUES 53, 77, 27);
```

相同操作的替代語法如下所示。

```
SELECT 50 >= SOME (SELECT 53 UNION ALL SELECT 77 UNION ALL SELECT 27);
```

在此情況下， 會SOME()評估為 True。

條件式運算式

的 Timestream LiveAnalytics 支援下列條件式表達式。

主題

- CASE 陳述式
- IF 陳述式
- COALESCE 陳述式
- NULLIF 陳述式
- TRY 陳述式

CASE 陳述式

CASE 陳述式會從左到右搜尋每個值表達式，直到找到等於 的值表達式expression。如果找到相符項目，則會傳回相符值的結果。如果找不到相符項目，則會傳回子ELSE句中存在的結果；否則null傳回。語法如下：

```
CASE expression
    WHEN value THEN result
    [ WHEN ... ]
    [ ELSE result ]
END
```

Timestream 也支援下列CASE陳述式語法。在此語法中，「搜尋」形式會從左到右評估每個布林條件，直到一個為 為止，true並傳回相符的結果。如果沒有任何條件為 true，則子ELSE句的結果會在存在時傳回；否則null會傳回。請參閱下列替代語法：

```
CASE
    WHEN condition THEN result
```

```
    [ WHEN ... ]
    [ ELSE result ]
END
```

IF 陳述式

IF 陳述式會評估條件為 true 或 false，並傳回適當的值。Timestream 支援下列兩種適用於 IF 的語法表示法：

```
if(condition, true_value)
```

`true_value` 如果條件為 ，則此語法會評估 並傳回`true`；否則`null`會傳回 且`true_value`不會評估。

```
if(condition, true_value, false_value)
```

`true_value` 如果條件為 ，則此語法會評估 並傳回 `true`，否則會評估並傳回 `false_value`。

範例

```
SELECT
  if(true, 'example 1'),
  if(false, 'example 2'),
  if(true, 'example 3 true', 'example 3 false'),
  if(false, 'example 4 true', 'example 4 false')
```

_col0	_col1	_col2	_col3
example 1	-	example 3 true	example 4 false
	null		

COALESCE 陳述式

COALESCE 會傳回引數清單中的第一個非空值。語法如下：

```
coalesce(value1, value2[,...])
```

NULLIF 陳述式

IF 陳述式會評估條件為 true 或 false，並傳回適當的值。Timestream 支援下列兩種適用於 IF 的語法表示法：

NULLIF 如果value1等於 則傳回 nullvalue2；否則傳回 value1。語法如下：

```
nullif(value1, value2)
```

TRY 陳述式

TRY 函數會評估表達式，並透過傳回 來處理特定類型的錯誤null。語法如下：

```
try(expression)
```

轉換函數

的 Timestream LiveAnalytics 支援下列轉換函數。

主題

- cast()
- try_cast ( )

cast()

將值明確轉換為類型的投放函數語法如下所示。

```
cast(value AS type)
```

try_cast ( )

的 Timestream LiveAnalytics 也支援 try_cast 函數，該函數類似於 cast，但如果 cast 失敗，則傳回 null。語法如下。

```
try_cast(value AS type)
```

數學運算子

的 Timestream LiveAnalytics 支援下列數學運算子。

運算子	描述
+	加法
-	減法
*	乘法
/	分區（整數分區執行截斷）
%	Modulus（剩餘）

數學函式

的 Timestream LiveAnalytics 支援下列數學函數。

函式	輸出資料類型	描述
abs (x)	【與輸入相同】	傳回 x 的絕對值。
cbrt (x)	double	傳回 x 的立方根。
ceiling (x) 或 ceil (x)	【與輸入相同】	傳回 x 四捨五入到最接近的整數。
degrees (x)	double	將角 x 以弧度轉換為度數。
e ( )	double	傳回常數 Euler 的數字。
exp (x)	double	傳回 Euler 提升至 x 功率的數字。
floor (x)	【與輸入相同】	傳回 x 四捨五入至最接近的整數。
from_base (string , radix)	bigint	傳回解譯為 base-radix 數字的字串值。
ln (x)	double	傳回 x 的自然對數。

函式	輸出資料類型	描述
log2（x）	double	傳回 x 的基本 2 對數。
log10（x）	double	傳回 x 的基本 10 個對數。
mod（n，m）	【與輸入相同】	傳回 n 除以 m 的模數（剩餘）。
pi（ ）	double	傳回常數 Pi。
pow（x， p） 或 power（x，p）	double	傳回 x 提升至 p 的強大功能。
弧度（x）	double	將角度 x 以度數轉換為弧度。
rand（ ） 或 random（ ）	double	傳回 0.0 1.0 範圍內的虛擬隨機值。
random（n）	【與輸入相同】	傳回介於 0 和 n（專屬）之間的虛擬隨機數字。
round（x）	【與輸入相同】	傳回 x 四捨五入至最接近的整數。
round（x，d）	【與輸入相同】	傳回 x 四捨五入至小數位數。
sign（x）	【與輸入相同】	傳回 x 的簽章函數，亦即： • 如果引數為 0，則為 0 • 如果引數大於 0，則為 1 • 如果引數小於 0，則為 -1。 對於雙引數，函數還會傳回： • 如果引數為 NaN，則為 NaN • 如果引數為 +Infinity，則為 1 • 如果引數為 -Infinity，則為 -1。

函式	輸出資料類型	描述
sqrt (x)	double	傳回 x 的平方根。
to_base (x ， radix)	varchar	傳回 x 的 base-radix 表示法。
truncate (x)	double	透過捨棄小數點後的數字，傳回 x 四捨五入到整數。
acos (x)	double	傳回 x 的弧餘弦。
asin (x)	double	傳回 x 的弧形正弦。
atan (x)	double	傳回 x 的弧切線。
atan2 (y ， x)	double	傳回 y / x 的弧切線。
cos (x)	double	傳回 x 的餘弦。
cosh (x)	double	傳回 x 的雙曲餘弦。
sin (x)	double	傳回 x 的正弦。
tan (x)	double	傳回 x 的正切。
tanh (x)	double	傳回 x 的雙曲正切。
infinity ( )	double	傳回代表正無限的常數。
is_finite (x)	boolean	判斷 x 是否為有限。
is_infinite (x)	boolean	判斷 x 是否無限。
is_nan (x)	boolean	判斷 x 是否為 not-a-number。
nan ( )	double	傳回代表 的常數 not-a-number。

字串運算子

的 Timestream LiveAnalytics 支援 | | 運算子串連一或多個字串。

字串函數

> **Note**
>
> 除非另有指定，否則這些函數的輸入資料類型會假設為 varchar。

函式	輸出資料類型	描述
chr (n)	varchar	傳回 Unicode 程式碼點 n 作為 varchar。
codepoint (x)	integer	傳回 str 唯一字元的 Unicode 程式碼點。
concat (x1 ， ... ， xN)	varchar	傳回 x1、x2、...、xN 的串連。
hamming_distance (x1 ，x2)	bigint	傳回 x1 和 x2 的 Hamming 距離，即對應字元不同的位置數目。請注意，兩個 varchar 輸入的長度必須相同。
length (x)	bigint	傳回以字元為單位的 x 長度。
levenshtein_distance (x1 ， x2)	bigint	傳回 x1 和 x2 的 Levenshtein 編輯距離，即將 x1 變更為 x2 所需的單一字元編輯 （ 插入、刪除或取代 ） 數目下限。
lower (x)	varchar	將 x 轉換為小寫。
lpad (x1 ， bigint 大小 ，x2)	varchar	左側墊 x1 大小為 x2 字元。如果大小小於 x1 的長度，則結果會截斷為大小字元。大小不得為負數，且 x2 不得空白。
ltrim (x)	varchar	從 x 移除主要空白。

函式	輸出資料類型	描述
replace (x1 ， x2)	varchar	從 x1 移除 x2 的所有執行個體。
replace (x1、x2、x3)	varchar	將 x2 的所有執行個體取代為 x1 中的 x3。
反向 （ x ）	varchar	以相反順序傳回具有字元的 x。
rpad (x1 ， bigint 大小，x2)	varchar	右側 Pad x1 大小為 x2 的字元。如果大小小於 x1 的長度，則結果會截斷為大小字元。大小不得為負數，且 x2 不得空白。
rtrim (x)	varchar	從 x 移除尾隨空格。
split (x1 ， x2)	array(varchar)	在分隔符號 x2 上分割 x1，並傳回陣列。
split (x1 ， x2 ， bigint 限制)	array(varchar)	在分隔符號 x2 上分割 x1，並傳回陣列。陣列中的最後一個元素一律包含 x1 中剩餘的所有項目。限制必須是正數。
split_part (x1 ， x2 ， bigint pos)	varchar	在分隔符號 x2 上分割 x1，並將 varchar 欄位傳回為 pos。欄位索引以 1 開頭。如果 pos 大於欄位數目，則會傳回 null。
strpos (x1 ， x2)	bigint	傳回 x1 中第一個 x2 執行個體的起始位置。位置開頭為 1。如果找不到，則會傳回 0。

函式	輸出資料類型	描述
strpos（x1， x2，bigint 執行個體）	bigint	傳回 x1 中第 N 個 x2 執行個體的位置。執行個體必須是正數。位置開頭為 1。如果找不到，則會傳回 0。
strrpos（x1， x2）	bigint	傳回 x1 中最後一個 x2 執行個體的開始位置。位置開頭為 1。如果找不到，則會傳回 0。
strrpos（x1， x2， bigint 執行個體）	bigint	從 x1 結尾開始，傳回 x1 中第 N 個執行個體 x2 的位置。執行個體必須是正數。位置開頭為 1。如果找不到，則會傳回 0。
position（x2 IN x1）	bigint	傳回 x1 中第一個 x2 執行個體的起始位置。位置開頭為 1。如果找不到，則會傳回 0。
substr（x， bigint 啟動）	varchar	從開始位置傳回其餘的 x。位置開頭為 1。負起始位置會解譯為相對於 x 結束。
substr（x， bigint start，bigint len）	varchar	從起始位置開始，從 x 的長度長度 len 傳回子字串。位置開頭為 1。負起始位置會解譯為相對於 x 結束。
trim（x）	varchar	從 x 移除前後空格。
upper（x）	varchar	將 x 轉換為大寫。

陣列運算子

的 Timestream LiveAnalytics 支援下列陣列運算子。

運算子	描述
[]	存取陣列的 元素，其中第一個索引從 1 開始。
||	將陣列與相同類型的另一個陣列或元素串連。

陣列函數

的 Timestream LiveAnalytics 支援下列陣列函數。

函式	輸出資料類型	描述
array_distinct (x)	陣列	從陣列 x 移除重複的值。 `SELECT array_distinct(ARRAY[1,2,2,3])` 範例結果：`[ 1,2,3 ]`
array_intersect (x， y)	陣列	傳回 x 和 y 交集的元素陣列，不重複。 `SELECT array_intersect(ARRAY[1,2,3], ARRAY[3,4,5])` 範例結果：`[ 3 ]`
array_union (x， y)	陣列	傳回 x 和 y 聯合中的元素陣列，不重複。 `SELECT array_union(ARRAY[1,2,3], ARRAY[3,4,5])` 範例結果：`[ 1,2,3,4,5 ]`

函式	輸出資料類型	描述
array_except (x ， y)	陣列	傳回 x 中的元素陣列，但不傳回 y 中的元素陣列，不傳回重複項目。 `SELECT array_except(ARRAY[1,2,3], ARRAY[3,4,5])` 範例結果： `[ 1,2 ]`
array_join (x ， delimiter ， null_replacement)	varchar	使用分隔符號和選用字串來串連指定陣列的元素，以取代 null。 `SELECT array_join(ARRAY[1,2,3], ';', '')` 範例結果： `1;2;3`
array_max (x)	與陣列元素相同	傳回輸入陣列的最大值。 `SELECT array_max(ARRAY[1,2,3])` 範例結果： `3`
array_min (x)	與陣列元素相同	傳回輸入陣列的最小值。 `SELECT array_min(ARRAY[1,2,3])` 範例結果： `1`

函式	輸出資料類型	描述
array_position（x， 元素）	bigint	傳回 陣列 x 中元素第一次出現的位置 （如果找不到，則為 0）。 `SELECT array_position(ARRAY[3,4,5,9], 5)` 範例結果： 3
array_remove（x， 元素）	陣列	從 x 陣列移除所有等於元素的元素。 `SELECT array_remove(ARRAY[3,4,5,9], 4)` 範例結果： [3,5,9]
array_sort（x）	陣列	排序並傳回陣列 x。x 的元素必須可排序。Null 元素將放置在傳回陣列的結尾。 `SELECT array_sort(ARRAY[6,8,2,9,3])` 範例結果： [2,3,6,8,9]

函式	輸出資料類型	描述
arrays_overlap (x ， y)	boolean	測試陣列 x 和 y 是否有任何非空元素是共同的。如果沒有一般的非空元素，但任一陣列都包含空值，則傳回空值。 `SELECT arrays_overlap(ARRAY[6,8,2,9,3], ARRAY[6,8])` 範例結果： `true`
cardinality (x)	bigint	傳回陣列 x 的大小。 `SELECT cardinality(ARRAY[6,8,2,9,3])` 範例結果： 5
concat (array1 ， array2 ， ... ， arrayN)	陣列	串連陣列陣列1、Array2、...、arrayN。 `SELECT concat(ARRAY[6,8,2,9,3], ARRAY[11,32], ARRAY[6,8,2,0,14])` 範例結果： `[ 6,8,2,9,3,11,32,6,8,2,0,14 ]`

函式	輸出資料類型	描述
element_at（array（E），索引）	E	傳回指定索引的陣列元素。如果索引 < 0， element_at 會存取從最後一個到第一個的元素。 `SELECT element_at(ARRAY[6,8,2,9,3], 1)` 範例結果： 6
repeat（元素，計數）	陣列	針對計數時間重複 元素。 `SELECT repeat(1, 3)` 範例結果： `[ 1,1,1 ]`
reverse（x）	陣列	傳回具有反向陣列 x 順序的陣列。 `SELECT reverse(ARRAY[6,8,2,9,3])` 範例結果： `[ 3,9,2,8,6 ]`
sequence（開始、停止）	array（bigint）	從開始到停止產生整數序列，如果開始小於或等於停止，則增加 1，否則為 -1。 `SELECT sequence(3, 8)` 範例結果： `[ 3,4,5,6,7,8 ]`

函式	輸出資料類型	描述
sequence（開始、停止、步驟）	array（bigint）	產生從開始到停止的整數序列，依步驟遞增。 `SELECT sequence(3, 15, 2)` 範例結果：[3,5,7,9,11,13,15]
sequence（開始、停止）	array（時間戳記）	產生從開始日期到結束日期的一系列時間戳記，以 1 天為單位遞增。 `SELECT sequence( '2023-04-02 19:26:12.941000000', '2023-04-06 19:26:12.941000000', 1d)` 範例結果：[2023-04-02 19:26:12.941000000 ,2023-04-03 19:26:12.941000000 ,2023-04-04 19:26:12.941000000 ,2023-04-05 19:26:12.941000000 ,2023-04-06 19:26:12.941000000]

函式	輸出資料類型	描述
sequence（開始、停止、步驟）	array（時間戳記）	產生從開始到停止的一系列時間戳記，依步驟遞增。步驟的資料類型是間隔。 `SELECT sequence( '2023-04-02 19:26:12.941000000', '2023-04-10 19:26:12.941000000', 2d)` 範例結果：`[ 2023-04-02 19:26:12.941000000 ,2023-04-04 19:26:12.941000000 ,2023-04-06 19:26:12.941000000 ,2023-04-08 19:26:12.941000000 ,2023-04-10 19:26:12.941000000 ]`
shuffle（x）	陣列	產生指定陣列 x 的隨機排列。 `SELECT shuffle(ARRAY[6,8,2,9,3])` 範例結果：`[ 6,3,2,9,8 ]`
slice（x、開始、長度）	陣列	子集陣列 x 從索引開頭開始（如果開頭為負值，則從結尾開始），長度為 。 `SELECT slice(ARRAY[6,8,2,9,3], 1, 3)` 範例結果：`[ 6,8,2 ]`

函式	輸出資料類型	描述
zip（array1，array2【，...】）	array（row）	將指定陣列依元素合併為單一資料列陣列。如果引數的長度不平均，則缺少的值會填入 NULL。 `SELECT zip(ARRAY [6,8,2,9,3], ARRAY[15, 24])` 範例結果：`[ ( 6, 15 ), ( 8, 24 ),( 2, - ), ( 9, - ),( 3, - ) ]`

位元函數

的 Timestream LiveAnalytics 支援下列位元函數。

函式	輸出資料類型	描述
bit_count（bigint， bigint）	bigint （兩個補充）	傳回第一個 Bigint 參數中的位元計數，其中第二個參數是位元簽署整數，例如 8 或 64。 `SELECT bit_count(19, 8)` 範例結果：`3` `SELECT bit_count(19, 2)` 範例結果：`Number must be representable with the bits specified . 19 can not be`

函式	輸出資料類型	描述
		represented with 2 bits
bitwise_and (bigint、biint)	bigint （兩個補充）	傳回 bigint 參數AND的位元。 `SELECT bitwise_and(12, 7)` 範例結果： 4
bitwise_not (bigint)	bigint （兩個補充）	傳回 bigint 參數NOT的位元。 `SELECT bitwise_not(12)` 範例結果： -13
bitwise_or (bigint、biint)	bigint （兩個補充）	傳回 Bigint 參數的位元 OR。 `SELECT bitwise_or(12, 7)` 範例結果： 15
bitwise_xor (bigint， bigint)	bigint （兩個補充）	傳回 bigint 參數XOR的位元。 `SELECT bitwise_xor(12, 7)` 範例結果： 11

規則運算式函數

Timestream 中的規則表達式函數 LiveAnalytics 支援 Java 模式語法 。的 Timestream LiveAnalytics 支援下列規則表達式函數。

函式	輸出資料類型	描述
regexp_extract_all（字串，模式）	array(varchar)	傳回字串中規則表達式模式相符的子字串（s）。 `SELECT regexp_extract_all('example expect complex', 'ex\w')` 範例結果：`[ exa,exp ]`
regexp_extract_all（字串、模式、群組）	array(varchar)	尋找字串中規則運算式模式的所有出現次數，並傳回擷取群組號碼群組。 `SELECT regexp_extract_all('example expect complex', '(ex)(\w)', 2)` 範例結果：`[ a,p ]`
regexp_extract（字串，模式）	varchar	傳回字串中規則表達式模式相符的第一個子字串。 `SELECT regexp_extract('example expect', 'ex\w')` 範例結果：`exa`
regexp_extract（字串、模式、群組）	varchar	尋找字串中規則表達式模式的第一次出現，並傳回擷取群組編號群組。 `SELECT regexp_extract('example`

函式	輸出資料類型	描述
		`expect', '(ex)(\w)', 2)` 範例結果： a
regexp_like（字串，模式）	boolean	評估規則表達式模式，並判斷是否包含在字串中。此函數類似於LIKE運算子，但僅需要將模式包含在字串中，而不需要符合所有字串。換句話說，這會執行包含 操作，而不是相符操作。您可以使用 ^ 和 $ 錨定模式，以符合整個字串。 `SELECT regexp_like('example', 'ex')` 範例結果： `true`
regexp_replace（字串，模式）	varchar	從字串中移除由規則表達式模式相符的每個子字串執行個體。 `SELECT regexp_replace('example expect', 'expect')` 範例結果： `example`

函式	輸出資料類型	描述
regexp_replace（字串、模式、取代）	varchar	將字串中 regex 模式相符的每個子字串執行個體替換為替換。擷取群組可以在替換中使用 $g 作為編號群組，或 ${name} 作為具名群組。用反斜線（\$）逸出，美元符號（$）可能會包含在替換中。 `SELECT regexp_replace('example expect', 'expect', 'surprise')` 範例結果：`example surprise`
regexp_replace（字串、模式、函數）	varchar	使用 函數取代字串中規則表達模式相符的每個子字串執行個體。每個相符項目都會叫用 lambda 表達式函數，擷取群組會以陣列形式傳遞。擷取群組編號從 1 開始；整個相符項目沒有群組（如果您需要，請以括號括住整個表達式）。 `SELECT regexp_replace('example', '(\w)', x -> upper(x[1]))` 範例結果：`EXAMPLE`

函式	輸出資料類型	描述
regexp_split（字串，模式）	array(varchar)	使用規則表達式模式分割字串並傳回陣列。保留追蹤空白字串。 `SELECT regexp_split('example', 'x')` 範例結果：`[ e,ample ]`

日期/時間運算子

> **Note**
>
> 的 Timestream LiveAnalytics 不支援負時間值。任何導致負時間的操作都會導致錯誤。

的 Timestream LiveAnalytics 支援 `timestamps`、 `dates`和 的下列操作`intervals`。

運算子	描述
+	加法
-	減法

主題

- 作業
- 加法
- 減法

作業

操作的結果類型是以運算元為基礎。`3s` 可以使用間隔常值，例如 `1day`和 。

```
SELECT date '2022-05-21' + interval '2' day
```

```
SELECT date '2022-05-21' + 2d
```

```
SELECT date '2022-05-21' + 2day
```

每個 的範例結果： 2022-05-23

間隔單位包括 second、minute、hour、day、week、 month和 year。但在某些情況下，並非所有都適用。例如，無法將秒、分鐘和小時新增至日期或從日期減去。

```
SELECT interval '4' year + interval '2' month
```

範例結果： 4-2

```
SELECT typeof(interval '4' year + interval '2' month)
```

範例結果： interval year to month

間隔操作的結果類型可能是 'interval year to month'或 'interval day to second'，取決於運算元。間隔可以新增至 或從 中減去 dates timestamps。但 date或 timestamp無法從 date或 中新增或減去timestamp。若要尋找與日期或時間戳記相關的間隔或持續時間，請參閱 中的 date_diff和 相關函數間隔和持續時間。

加法

Example

```
SELECT date '2022-05-21' + interval '2' day
```

範例結果： 2022-05-23

Example

```
SELECT typeof(date '2022-05-21' + interval '2' day)
```

範例結果： date

Example

```
SELECT interval '2' year + interval '4' month
```

範例結果：2-4

Example

```
SELECT typeof(interval '2' year + interval '4' month)
```

範例結果：interval year to month

減法

Example

```
SELECT timestamp '2022-06-17 01:00' - interval '7' hour
```

範例結果：2022-06-16 18:00:00.000000000

Example

```
SELECT typeof(timestamp '2022-06-17 01:00' - interval '7' hour)
```

範例結果：timestamp

Example

```
SELECT interval '6' day - interval '4' hour
```

範例結果：5 20:00:00.000000000

Example

```
SELECT typeof(interval '6' day - interval '4' hour)
```

範例結果：interval day to second

日期/時間函數

> **Note**
>
> 的 Timestream LiveAnalytics 不支援負時間值。任何導致負時間的操作都會導致錯誤。

的 Timestream LiveAnalytics 會使用UTC日期和時間的時區。Timestream 支援下列函數的日期和時間。

主題

一般和轉換

的 Timestream LiveAnalytics 支援日期和時間的下列一般和轉換函數。

函式	輸出資料類型	描述
current_date	date	傳回 中的目前日期UTC。未使用括號。 `SELECT current_date` 範例結果： `2022-07-07` **Note** 這也是預留的關鍵字。如需預留關鍵字的清單，請參閱 保留的關鍵字。

函式	輸出資料類型	描述
current_time	time	傳回 中的目前時間UTC。未使用括號。 `SELECT current_time` 範例結果： `17:41:52.827000000` **Note** 這也是預留的關鍵字。如需預留關鍵字的清單，請參閱 保留的關鍵字。
current_timestamp 或 now ( )	timestamp	傳回 中的目前時間戳記UTC。 `SELECT current_timestamp` 範例結果： `2022-07-07 17:42:32.939000000` **Note** 這也是預留的關鍵字。如需預留關鍵字的清單，請參閱 保留的關鍵字。

函式	輸出資料類型	描述
current_timezone（ ）	varchar 值將為 'UTC.'	Timestream 會使用UTC時區做為日期和時間。 `SELECT current_timezone()` 範例結果： UTC
date（varchar（x））， date（時間戳記）	date	`SELECT date(TIMESTAMP '2022-07-07 17:44:43.771000000')` 範例結果： 2022-07-07
last_day_of_month（時間戳記）、 last_day_of_month（日期）	date	`SELECT last_day_of_month(TIMESTAMP '2022-07-07 17:44:43.771000000')` 範例結果： 2022-07-31
from_iso8601_timestamp（string）	timestamp	將 ISO 8601 時間戳記剖析為內部時間戳記格式。 `SELECT from_iso8601_timestamp('2022-06-17T08:04:05.000000000+05:00')` 範例結果： 2022-06-17 03:04:05.000000000

函式	輸出資料類型	描述
from_iso8601_date（string）	date	將 ISO 8601 日期字串剖析為指定日期的 UTC 00：00：00 的內部時間戳記格式。 `SELECT from_iso8601_date('2022-07-17')` 範例結果： 2022-07-17
to_iso8601（時間戳記），to_iso8601（日期）	varchar	傳回 輸入的 ISO8601 格式字串。 `SELECT to_iso8601(from_iso8601_date('2022-06-17'))` 範例結果： 2022-06-17
from_milliseconds（邊界）	timestamp	`SELECT from_milliseconds(1)` 範例結果： 1970-01-01 00:00:00.001000000
from_nanoseconds（聯名）	timestamp	`select from_nanoseconds(300000001)` 範例結果： 1970-01-01 00:00:00.300000001

函式	輸出資料類型	描述
from_unixtime（雙）	timestamp	傳回與提供的 unixtime 對應的時間戳記。 `SELECT from_unixtime(1)` 範例結果： `1970-01-01 00:00:01.000000000`
本地時間	time	傳回 中的目前時間UTC。未使用括號。 `SELECT localtime` 範例結果： `17:58:22.654000000` **Note** 這也是預留的關鍵字。如需預留關鍵字的清單，請參閱 保留的關鍵字。

函式	輸出資料類型	描述
本機時間戳記	timestamp	傳回 中的目前時間戳記UTC。未使用括號。 `SELECT localtimestamp` 範例結果： `2022-07-07 17:59:04.368000000` **Note** 這也是預留的關鍵字。如需預留關鍵字的清單，請參閱 保留的關鍵字。
to_milliseconds （間隔日到秒），to_milliseconds （時間戳記）	bigint	`SELECT to_milliseconds(INTERVAL '2' DAY + INTERVAL '3' HOUR)` 範例結果： `183600000` `SELECT to_milliseconds(TIMESTAMP '2022-06-17 17:44:43.771000000')` 範例結果： `1655487883771`

函式	輸出資料類型	描述
to_nanoseconds（間隔日到秒），to_nanoseconds（時間戳記）	bigint	`SELECT to_nanoseconds(INTERVAL '2' DAY + INTERVAL '3' HOUR)` 範例結果：`183600000000000` `SELECT to_nanoseconds(TIMESTAMP '2022-06-17 17:44:43.771000678')` 範例結果：`1655487883771000678`
to_unixtime（時間戳記）	double	傳回所提供時間戳記的 unixtime。 `SELECT to_unixtime('2022-06-17 17:44:43.771000000')` 範例結果：`1.655487883771E9`

函式	輸出資料類型	描述
date_trunc（單位，時間戳記）	timestamp	傳回截斷為單位的時間戳記，其中單位是 【秒、分鐘、小時、日、週、月、季或年】 之一。 `SELECT date_trunc('minute', TIMESTAMP '2022-06-17 17:44:43.771000000')` 範例結果： `2022-06-17 17:44:00.000000000`

間隔和持續時間

的 Timestream LiveAnalytics 支援日期和時間的下列間隔和持續時間函數。

函式	輸出資料類型	描述
date_add（unit， bigint，date），date_add（unit，bigint， time），date_add（varchar（x），bigint，timestamp）	timestamp	新增單位的重頭，其中單位是 【秒、分鐘、小時、天、週、月、季度或年】 之一。 `SELECT date_add('hour', 9, TIMESTAMP '2022-06-17 00:00:00')` 範例結果： `2022-06-17 09:00:00.000000000`
date_diff（單位、日期、日期）、date_diff（單位、時間、時間）、date_diff（單位、時間戳記、時間戳記）	bigint	傳回差異，其中單位為 【秒、分鐘、小時、天、週、月、季度或年】 之一。

函式	輸出資料類型	描述
		`SELECT date_diff('day', DATE '2020-03-01', DATE '2020-03-02')` 範例結果：1
parse_duration（字串）	間隔	剖析輸入字串以傳回interval對等的字串。 `SELECT parse_duration('42.8ms')` 範例結果：0 00:00:00.042800000 `SELECT typeof(parse_duration('42.8ms'))` 範例結果：interval day to second

函式	輸出資料類型	描述
bin （時間戳記、間隔）	timestamp	將timestamp 參數的整數值四捨五入到interval參數整數值的最近倍數。 此傳回值的意義可能不明顯。首先使用整數算術計算，方法是將時間戳記整數除以間隔整數，然後將結果乘以間隔整數。 請記住，時間戳記會將UTC時間點指定為自 epoch 以來經過的秒數分數 POSIX （1970 年 1 月 1 日），傳回值很少與行事曆單位一致。例如，如果您指定 30 天的間隔，則自 epoch 以來的所有天數都會分為 30 天增量，並傳回最近 30 天增量的開始，這與日曆月沒有關係。 以下是一些範例： `bin(TIMESTAMP '2022-06-17 10:15:20', 5m) ==> 2022-06-17 10:15:00.000000000` `bin(TIMESTAMP '2022-06-17 10:15:20', 1d) ==> 2022-06-17 00:00:00.000000000` `bin(TIMESTAMP '2022-06-17 10:15:20', 10day) ==> 2022-06-17 00:00:00.000000000` `bin(TIMESTAMP '2022-06-17 10:15:20', 30day)`

函式	輸出資料類型	描述
		==> 2022-05-28 00:00:00.000000000
前 （ 間隔 ）	timestamp	傳回對應於 current_timestamp 的值interval。 `SELECT ago(1d)` 範例結果： 2022-07-06 21:08:53.245000000
間隔常值，例如 1h、1d 和 30m	間隔	間隔常值是 parse_duration (string) 的便利性。例如，1d 與 parse_duration('1d') 相同。這允許使用常值，無論使用間隔的位置為何。例如，ago(1d) 和 bin(*<timestamp>* , 1m)。

某些間隔常值可做為 parse_duration 的短手。例如， parse_duration('1day')、1day、parse_duration('1d')和1d每個傳回1 00:00:00.000000000的類型為 interval day to second。允許使用提供給 的格式。 parse_duration例如， parse_duration('1day')也會傳回 00:00:00.000000000。但1 day不是間隔常值。

與 相關的單位interval day to second為 ns、nanosecond、us、microsecond、ms、毫秒、s、second、m、min、h、hr、d 和 day。

也有 interval year to month。與間隔年到月相關的單位為 y、年和月。例如， SELECT 1year傳回 1-0。 SELECT 12month也會傳回 1-0。 SELECT 8month傳回 0-8。

雖然 的 單位quarter也可用於某些函數，例如 date_trunc和 date_add，quarter但 不能作為間隔常值的一部分使用。

格式化和剖析

Timestream for LiveAnalytics 支援日期和時間的下列格式化和剖析函數。

函式	輸出資料類型	描述
date_format（時間戳記，varchar（x））	varchar	如需此函數使用的格式規格符的詳細資訊，請參閱 https://trino.io/docs/current/functions/datetime.html#mysql-date-functions `SELECT date_format(TIMESTAMP '2019-10-20 10:20:20', '%Y-%m-%d %H:%i:%s')` 範例結果：`2019-10-20 10:20:20`
date_parse（varchar（x）、varchar（y））	timestamp	如需此函數使用的格式規格符的詳細資訊，請參閱 https://trino.io/docs/current/functions/datetime.html#mysql-date-functions `SELECT date_parse('2019-10-20 10:20:20', '%Y-%m-%d %H:%i:%s')` 範例結果：`2019-10-20 10:20:20.000000000`
format_datetime（timestamp，varchar（x））	varchar	如需此函數使用的格式字串的詳細資訊，請參閱 http://joda-time.sourceforge.net/apidocs/org/joda/time/format/DateTimeFormat.html `SELECT format_datetime(parse_datet`

函式	輸出資料類型	描述
		ime('1968-01-13 12', 'yyyy-MM-dd HH'), 'yyyy-MM-dd HH') 範例結果： 1968-01-13 12
parse_datetime (var char (x) 、varchar (y))	timestamp	如需此函數使用的格式字串的詳細資訊，請參閱 http://joda-time.sourceforge.net/apidocs/org/joda/time/format/DateTimeFormat.html SELECT parse_datetime('2019-12-29 10:10 PST', 'uuuu-LL-dd HH:mm z') 範例結果： 2019-12-29 18:10:00.000000000

擷取

的 Timestream LiveAnalytics 支援日期和時間的下列擷取函數。擷取函數是其餘便利函數的基礎。

函式	輸出資料類型	描述
擷取	bigint	從時間戳記擷取欄位，其中欄位為 【YEAR、QUARTER、MONTHWEEK、DAY、DAY_OF_MONTH、DAY_OF_WEEK、DOW、DAY_OF_YEAR、DOY、、YEAR_OF_WEEK、YOW、HOUR、、MINUTE或 SECOND】 之一。

函式	輸出資料類型	描述
		`SELECT extract(YEAR FROM '2019-10-12 23:10:34.000000000')` 範例結果： 2019
day（時間戳記）、day（日期）、day（間隔日到秒）	bigint	`SELECT day('2019-10-12 23:10:34.000000000')` 範例結果： 12
day_of_month（時間戳記）、day_of_month（日期）、day_of_month（間隔日到秒）	bigint	`SELECT day_of_month('2019-10-12 23:10:34.000000000')` 範例結果： 12
day_of_week（時間戳記）、day_of_week（日期）	bigint	`SELECT day_of_week('2019-10-12 23:10:34.000000000')` 範例結果： 6
day_of_year（時間戳記）、day_of_year（日期）	bigint	`SELECT day_of_year('2019-10-12 23:10:34.000000000')` 範例結果： 285
dow（時間戳記）、dw（日期）	bigint	day_of_week 的別名
doy（時間戳記）、doy（日期）	bigint	天_of_year 的別名

函式	輸出資料類型	描述
hour（時間戳記）、hour（時間）、hour（間隔日到秒）	bigint	`SELECT hour('2019-10-12 23:10:34.000000000')` 範例結果：23
毫秒（時間戳記）、毫秒（時間）、毫秒（間隔日到秒）	bigint	`SELECT millisecond('2019-10-12 23:10:34.000000000')` 範例結果：0
minute（時間戳記）、min（時間）、mini（間隔日到秒）	bigint	`SELECT minute('2019-10-12 23:10:34.000000000')` 範例結果：10
month（時間戳記）、month（日期）、month（年間到月間）	bigint	`SELECT month('2019-10-12 23:10:34.000000000')` 範例結果：10
奈秒（時間戳記）、奈秒（時間）、奈秒（間隔日到秒）	bigint	`SELECT nanosecond(current_timestamp)` 範例結果：162000000
quarter（時間戳記）、quarter（日期）	bigint	`SELECT quarter('2019-10-12 23:10:34.000000000')` 範例結果：4

函式	輸出資料類型	描述
second（時間戳記）、second（時間）、second（間隔日到秒）	bigint	`SELECT second('2019-10-12 23:10:34.000000000')` 範例結果： 34
week（時間戳記）、week（日期）	bigint	`SELECT week('2019-10-12 23:10:34.000000000')` 範例結果： 41
week_of_year（時間戳記）、week_of_year（日期）	bigint	一週的別名
year（時間戳記）、year（日期）、year（年與月的間隔）	bigint	`SELECT year('2019-10-12 23:10:34.000000000')` 範例結果： 2019
year_of_week（時間戳記）、year_of_week（日期）	bigint	`SELECT year_of_week('2019-10-12 23:10:34.000000000')` 範例結果： 2019
yow（時間戳記）、yow（日期）	bigint	year_of_week 的別名

彙總函數

的 Timestream LiveAnalytics 支援下列彙總函數。

函式	輸出資料類型	描述
任意（x）	【與輸入相同】	如果存在任意非 Null 值，則傳回 x。 `SELECT arbitrary(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 範例結果：1
array_agg（x）	array<【與輸入相同】	傳回從輸入 x 元素建立的陣列。 `SELECT array_agg(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 範例結果：[1,2,3,4]
avg（x）	double	傳回所有輸入值的平均值（算術平均值）。 `SELECT avg(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 範例結果：2.5
bool_and（布林值） every（布林值）	boolean	TRUE 如果每個輸入值都是，則傳回 TRUE，否則傳回 FALSE。 `SELECT bool_and(t.c) FROM (VALUES true, true, false, true) AS t(c)`

函式	輸出資料類型	描述
		範例結果： `false`
bool_or (布林值)	boolean	TRUE 如果任何輸入值為 ，則傳回 TRUE，否則傳回 FALSE。 `SELECT bool_or(t.c) FROM (VALUES true, true, false, true) AS t(c)` 範例結果： `true`
count (*) count (x)	bigint	count (*) 會傳回輸入列的數量。 count (x) 傳回非空輸入值的數量。 `SELECT count(t.c) FROM (VALUES true, true, false, true) AS t(c)` 範例結果： `4`
count_if (x)	bigint	傳回TRUE輸入值的數量。 `SELECT count_if(t.c) FROM (VALUES true, true, false, true) AS t(c)` 範例結果： `3`

函式	輸出資料類型	描述
geometric_mean (x)	double	傳回所有輸入值的幾何平均值。 `SELECT geometric_mean(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 範例結果： 2.213363839400643
max_by (x， y)	【與 x 相同】	傳回在所有輸入值中與 y 最大值相關聯的 x 值。 `SELECT max_by(t.c1, t.c2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 範例結果： d
max_by (x、y、n)	array<【與 x 相同】 >	以 y 的遞減順序傳回與 y 所有輸入值 n 最大值相關聯的 x n 值。 `SELECT max_by(t.c1, t.c2, 2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 範例結果： [d,c]

函式	輸出資料類型	描述
min_by (x ， y)	【與 x 相同】	傳回在所有輸入值上與 y 最小值相關聯的 x 值。 `SELECT min_by(t.c1, t.c2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 範例結果： a
min_by (x、y、n)	array<【與 x 相同】 >	以 y 的遞增順序傳回與 y 所有輸入值 n 最小相關聯的 x n 值。 `SELECT min_by(t.c1, t.c2, 2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 範例結果： [a,b]
max (x)	【與輸入相同】	傳回所有輸入值的最大值。 `SELECT max(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 範例結果： 4

函式	輸出資料類型	描述
max (x , n)	array<【與 x 相同】 >	傳回 x 所有輸入值的 n 最大值。 `SELECT max(t.c, 2) FROM (VALUES 1, 2, 3, 4) AS t(c)` 範例結果： [4,3]
min (x)	【與輸入相同】	傳回所有輸入值的最小值。 `SELECT min(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 範例結果： 1
min (x , n)	array<【與 x 相同】 >	傳回 x 所有輸入值的 n 個最小值。 `SELECT min(t.c, 2) FROM (VALUES 1, 2, 3, 4) AS t(c)` 範例結果： [1,2]
sum (x)	【與輸入相同】	傳回所有輸入值的總和。 `SELECT sum(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 範例結果： 10

函式	輸出資料類型	描述
bitwise_and_agg (x)	bigint	傳回 2 的補數表示法中所有輸入值AND的位元。 `SELECT bitwise_and_agg(t.c) FROM (VALUES 1, -3) AS t(c)` 範例結果： 1
bitwise_or_agg (x)	bigint	傳回 2 的補數表示法中所有輸入值的位元 OR。 `SELECT bitwise_or_agg(t.c) FROM (VALUES 1, -3) AS t(c)` 範例結果： -3
approx_distinct (x)	bigint	傳回相異輸入值的大約數量。此函數提供 count (DISTINCT x) 的近似值。如果所有輸入值都是 null，則會傳回零。此函數應該會產生 2.3% 的標準錯誤，這是所有可能集合中（大約正常）錯誤分佈的標準差。它不保證任何特定輸入集的錯誤上限。 `SELECT approx_distinct(t.c) FROM (VALUES 1, 2, 3, 4, 8) AS t(c)` 範例結果： 5

函式	輸出資料類型	描述
approx_distinct（x， e）	bigint	傳回相異輸入值的大約數量。此函數提供 count（DISTINCT x）的近似值。如果所有輸入值都是 null，則會傳回零。此函數應該會產生不超過 e 的標準錯誤，這是所有可能集合中（大約正常）錯誤分佈的標準差。它不保證任何特定輸入集的錯誤上限。此函數目前的實作需要 e 位於 【0.0040625，0.26000】 的範圍內。 `SELECT approx_distinct(t.c, 0.2) FROM (VALUES 1, 2, 3, 4, 8) AS t(c)` 範例結果： 5
approx_percentile（x， 百分比）	【與 x 相同】	傳回指定百分比的所有 x 輸入值的近似百分位數。百分比的值必須介於零和一之間，且所有輸入列都必須為常數。 `SELECT approx_percentile(t.c, 0.4) FROM (VALUES 1, 2, 3, 4) AS t(c)` 範例結果： 2

函式	輸出資料類型	描述
approx_percentile（x， 百分比）	array<【與 x 相同】 >	傳回每個指定百分比的所有 x 輸入值的近似百分位數。百分比陣列的每個元素必須介於零和一之間，且所有輸入列的陣列都必須為常數。 `SELECT approx_percentile(t.c, ARRAY[0.1, 0.8, 0.8]) FROM (VALUES 1, 2, 3, 4) AS t(c)` 範例結果： `[ 1,4,4 ]`
approx_percentile（x、w、百分比）	【與 x 相同】	以 p 百分比傳回使用每個項目權重 w 的所有 x 輸入值的近似權重百分位數。權重必須至少為一個整數值。它實際上是百分位數集中值 x 的複寫計數。p 的值必須介於零和一之間，且所有輸入列都必須為常數。 `SELECT approx_percentile(t.c, 1, 0.1) FROM (VALUES 1, 2, 3, 4) AS t(c)` 範例結果： 1

函式	輸出資料類型	描述
approx_percentile (x、w、百分比)	array<【與 x 相同】 >	針對陣列中指定的每個百分比，使用每個項目的權重 w，傳回 x 所有輸入值的大約權重百分位數。權重必須至少為一個整數值。它實際上是百分位數集中值 x 的複寫計數。陣列的每個元素必須介於零和一之間，且所有輸入列的陣列都必須為常數。 `SELECT approx_percentile(t.c, 1, ARRAY[0.1, 0.8, 0.8]) FROM (VALUES 1, 2, 3, 4) AS t(c)` 範例結果：`[ 1,4,4 ]`
approx_percentile (x、w、%、準確性)	【與 x 相同】	使用每個項目的權重 w，以 p 百分比傳回 x 所有輸入值的近似權重百分位數，且具有準確度的最大排名錯誤。權重必須至少為一個整數值。它實際上是百分位數集中值 x 的複寫計數。p 的值必須介於零和一之間，且所有輸入列都必須為常數。準確度必須是大於零且小於一的值，且所有輸入列的值必須保持不變。 `SELECT approx_percentile(t.c, 1, 0.1, 0.5) FROM (VALUES 1, 2, 3, 4) AS t(c)` 範例結果：`1`

函式	輸出資料類型	描述
corr (y ， x)	double	傳回輸入值的關聯係數。 `SELECT corr(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 範例結果： `1.0`
covar_pop (y ， x)	double	傳回輸入值的母群體共變數。 `SELECT covar_pop(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 範例結果： `1.25`
covar_samp (y ， x)	double	傳回輸入值的範例共變數。 `SELECT covar_samp(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 範例結果： `1.6666666666666667`

函式	輸出資料類型	描述
regr_intercept (y ， x)	double	傳回輸入值的線性迴歸截距。y 是相依值。x 是獨立值。 `SELECT regr_intercept(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 範例結果： `0.0`
regr_slope (y ， x)	double	傳回輸入值的線性迴歸斜率。y 是相依值。x 是獨立值。 `SELECT regr_slope(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 範例結果： `1.0`
偏斜 （ x ）	double	傳回所有輸入值的偏斜。 `SELECT skewness(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 範例結果： `0.8978957037987335`

函式	輸出資料類型	描述
stddev_pop (x)	double	傳回所有輸入值的母群體標準差。 `SELECT stddev_pop(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 範例結果：2.4166091947189146
stddev_samp (x) stddev (x)	double	傳回所有輸入值的範例標準差。 `SELECT stddev_samp(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 範例結果：2.7018512172212592
var_pop (x)	double	傳回所有輸入值的母群體差異。 `SELECT var_pop(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 範例結果：5.840000000000001

函式	輸出資料類型	描述
var_samp (x) 變數 （ x ）	double	傳回所有輸入值的範例差異。 `SELECT var_samp(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 範例結果： 7.3000000 00000001

範圍函數

視窗函數會跨查詢結果的資料列執行計算。它們在HAVING子句之後但在 ORDER BY 子句之前執行。叫用視窗函數需要使用 OVER子句來指定視窗的特殊語法。視窗有三個元件：

- 分割區規格，將輸入列分隔為不同的分割區。這類似於 GROUP BY 子句如何將資料列分成不同的群組，以便彙總函數。
- 排序規格，其決定視窗函數處理輸入列的順序。
- 視窗框架，指定要由指定資料列的函數處理的資料列滑動視窗。如果未指定影格，則預設為 RANGE UNBOUNDED PRECEDING，這與 RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT 相同ROW。此影格包含從分割區開始到目前資料列最後一個對等的所有資料列。

所有彙總函數都可以透過新增子OVER句來用作視窗函數。彙總函數會針對目前資料列的視窗框架內資料列上的每一列進行計算。除了彙總函數之外， 的 Timestream LiveAnalytics 還支援下列排名和值函數。

函式	輸出資料類型	描述
cume_dist （ ）	bigint	傳回一組值中值的累積分佈。結果是視窗分割區視窗順序中的列前面的資料列數或對等數，除以視窗分割區中的資料列總數。因此，排序中的任何綁定值都會評估為相同的分佈值。

函式	輸出資料類型	描述
dense_rank ( )	bigint	傳回一組值中值的等級。這類似於 rank ( ) ，但綁定值不會在序列中產生間隙。
ntile (n)	bigint	將每個視窗分割區的資料列分割為 n 個儲存貯體，範圍從 1 到最多 n 個。儲存貯體值最多會與 1 不同。如果分割區中的資料列數目未平均劃分為儲存貯體數目，則剩餘值會從第一個儲存貯體開始，每個儲存貯體分配一個。
percent_rank ( )	double	傳回值群組中值的百分比排名。結果為 （r - 1） / （n - 1），其中 r 是資料列的 rank ( ) ，n 是視窗分割區中的資料列總數。
rank ( )	bigint	傳回一組值中值的等級。排名是一加列前面的列數，這些列與列不對等。因此，排序中的綁定值會在序列中產生間隙。會為每個視窗分割區執行排名。
row_number ( )	bigint	根據視窗分割區內的資料列順序，傳回每列的唯一序號，從一個開始。
first_value (x)	【與輸入相同】	傳回視窗的第一個值。此函數範圍為視窗框架。函數會將表達式或目標作為其參數。

函式	輸出資料類型	描述
last_value (x)	【與輸入相同】	傳回視窗的最後一個值。此函數範圍為視窗框架。函數會將表達式或目標作為其參數。
nth_value (x， 偏移)	【與輸入相同】	傳回從視窗開始的指定偏移值。偏移從 1 開始。偏移可以是任何純量表達式。如果偏移值為 null 或大於視窗中的值數目，則會傳回 null。偏移為零或負值是錯誤。函數會將表達式或目標作為其第一個參數。
lead (x【， offset【， default_value】】)	【與輸入相同】	傳回視窗中目前資料列之後偏移資料列的值。偏移從 0 開始，這是目前的資料列。偏移可以是任何純量表達式。預設偏移為 1。如果偏移值為 Null 或大於視窗，則會傳回 default_value，或未指定 Null。函數會將表達式或目標作為其第一個參數。
lag (x【， offset【， default_value】】)	【與輸入相同】	傳回在視窗中目前資料列之前偏移資料列的值 偏移從 0 開始，這是目前的資料列。偏移可以是任何純量表達式。預設偏移為 1。如果偏移值為 Null 或大於視窗，則會傳回 default_value，或未指定 Null。函數會將表達式或目標作為其第一個參數。

範例查詢

本節包含 LiveAnalytics查詢語言的 Timestream 範例使用案例。

主題

簡單查詢

以下內容會取得資料表最近新增的 10 個資料點。

```
SELECT * FROM <database_name>.<table_name>
ORDER BY time DESC
LIMIT 10
```

以下內容會取得特定量值的 5 個最舊資料點。

```
SELECT * FROM <database_name>.<table_name>
WHERE measure_name = '<measure_name>'
ORDER BY time ASC
LIMIT 5
```

下列 適用於奈秒精細度時間戳記。

```
SELECT now() AS time_now
, now() - (INTERVAL '12' HOUR) AS twelve_hour_earlier -- Compatibility with ANSI SQL
, now() - 12h AS also_twelve_hour_earlier -- Convenient time interval literals
, ago(12h) AS twelve_hours_ago -- More convenience with time functionality
, bin(now(), 10m) AS time_binned -- Convenient time binning support
, ago(50ns) AS fifty_ns_ago -- Nanosecond support
, now() + (1h + 50ns) AS hour_fifty_ns_future
```

多重測量記錄的測量值會依資料欄名稱識別。單一測量記錄的測量值由 識別measure_value::*<data_type>*，其中 *<data_type>*是 double、boolean、 bigint或 之一varchar，如 中所述支援的資料類型。如需量值建模方式的詳細資訊，請參閱單一資料表與多個資料表 。

下列會從具有 之 的speed多量值記錄擷取名為 measure_name的量值值IoTMulti-stats。

```
SELECT speed FROM <database_name>.<table_name> where measure_name = 'IoTMulti-stats'
```

下列會從具有 measure_name的單一測量記錄擷取double值load。

```
SELECT measure_value::double FROM <database_name>.<table_name> WHERE measure_name =
 'load'
```

具有時間序列函數的查詢

主題

- 資料集和查詢範例

資料集和查詢範例

您可以使用 Timestream for LiveAnalytics 來了解和改善服務和應用程式的效能和可用性。以下是在該資料表上執行的範例資料表和範例查詢。

資料表ec2_metrics存放遙測資料，例如CPU使用率和其他EC2執行個體的指標。您可以檢視下表。

時間	region	az	Hostname (主機名稱)	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19：00：00.000000000	us-east-1	us-east-1a	前端01	cpu_utilization	35.1	null
2019-12-04 19：00：00.000000000	us-east-1	us-east-1a	前端01	memory_utilization	55.3	null
2019-12-04 19：00：00.000000000	us-east-1	us-east-1a	前端01	network_bytes_in	null	1,500
2019-12-04 19：	us-east-1	us-east-1a	前端01	network_bytes_out	null	6,700

時間	region	az	Hostname (主機名稱)	measure_name	measure_value::double	measure_value::bigint
00：00.000000000						
2019-12-04 19：00：00.000000000	us-east-1	us-east-1b	前端02	cpu_utilization	38.5	null
2019-12-04 19：00：00.000000000	us-east-1	us-east-1b	前端02	memory_utilization	58.4	null
2019-12-04 19：00：00.000000000	us-east-1	us-east-1b	前端02	network_bytes_in	null	23，000
2019-12-04 19：00：00.000000000	us-east-1	us-east-1b	前端02	network_bytes_out	null	12,000
2019-12-04 19：00：00.000000000	us-east-1	us-east-1c	前端03	cpu_utilization	45.0	null
2019-12-04 19：00：00.000000000	us-east-1	us-east-1c	前端03	memory_utilization	65.8	null

時間	region	az	Hostname (主機名稱)	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19：00：00.000000000	us-east-1	us-east-1c	前端03	network_bytes_in	null	15,000
2019-12-04 19：00：00.000000000	us-east-1	us-east-1c	前端03	network_bytes_out	null	836，000
2019-12-04 19：00：05.000000000	us-east-1	us-east-1a	前端01	cpu_utilization	55.2	null
2019-12-04 19：00：05.000000000	us-east-1	us-east-1a	前端01	memory_utilization	75.0	null
2019-12-04 19：00：05.000000000	us-east-1	us-east-1a	前端01	network_bytes_in	null	1，245
2019-12-04 19：00：05.000000000	us-east-1	us-east-1a	前端01	network_bytes_out	null	68，432
2019-12-04 19：00：08.000000000	us-east-1	us-east-1b	前端02	cpu_utilization	65.6	null

時間	region	az	Hostname (主機名稱)	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19：00：08.000000000	us-east-1	us-east-1b	前端02	memory_utilization	85.3	null
2019-12-04 19：00：08.000000000	us-east-1	us-east-1b	前端02	network_bytes_in	null	1，245
2019-12-04 19：00：08.000000000	us-east-1	us-east-1b	前端02	network_bytes_out	null	68，432
2019-12-04 19：00：20.000000000	us-east-1	us-east-1c	前端03	cpu_utilization	12.1	null
2019-12-04 19：00：20.000000000	us-east-1	us-east-1c	前端03	memory_utilization	32.0	null
2019-12-04 19：00：20.000000000	us-east-1	us-east-1c	前端03	network_bytes_in	null	1，400
2019-12-04 19：00：20.000000000	us-east-1	us-east-1c	前端03	network_bytes_out	null	345

時間	region	az	Hostname (主機名稱)	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19：00：10.000000000	us-east-1	us-east-1a	前端01	cpu_utilization	15.3	null
2019-12-04 19：00：10.000000000	us-east-1	us-east-1a	前端01	memory_utilization	35.4	null
2019-12-04 19：00：10.000000000	us-east-1	us-east-1a	前端01	network_bytes_in	null	23
2019-12-04 19：00：10.000000000	us-east-1	us-east-1a	前端01	network_bytes_out	null	0
2019-12-04 19：00：16.000000000	us-east-1	us-east-1b	前端02	cpu_utilization	44.0	null
2019-12-04 19：00：16.000000000	us-east-1	us-east-1b	前端02	memory_utilization	64.2	null
2019-12-04 19：00：16.000000000	us-east-1	us-east-1b	前端02	network_bytes_in	null	1，450

時間	region	az	Hostname (主機名稱)	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19：00：16.000000000	us-east-1	us-east-1b	前端02	network_bytes_out	null	200
2019-12-04 19：00：40.000000000	us-east-1	us-east-1c	前端03	cpu_utilization	66.4	null
2019-12-04 19：00：40.000000000	us-east-1	us-east-1c	前端03	memory_utilization	86.3	null
2019-12-04 19：00：40.000000000	us-east-1	us-east-1c	前端03	network_bytes_in	null	300
2019-12-04 19：00：40.000000000	us-east-1	us-east-1c	前端03	network_bytes_out	null	423

尋找過去 2 小時內特定EC2主機的平均值、p90、p95 和 p99 CPU使用率：

```
SELECT region, az, hostname, BIN(time, 15s) AS binned_timestamp,
    ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization,
    ROUND(APPROX_PERCENTILE(measure_value::double, 0.9), 2) AS p90_cpu_utilization,
    ROUND(APPROX_PERCENTILE(measure_value::double, 0.95), 2) AS p95_cpu_utilization,
    ROUND(APPROX_PERCENTILE(measure_value::double, 0.99), 2) AS p99_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
```

```
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
GROUP BY region, hostname, az, BIN(time, 15s)
ORDER BY binned_timestamp ASC
```

識別CPU使用率比過去 2 小時內整個機群的平均CPU使用率高出 10% 或更多的EC2主機：

```
WITH avg_fleet_utilization AS (
    SELECT COUNT(DISTINCT hostname) AS total_host_count, AVG(measure_value::double) AS
 fleet_avg_cpu_utilization
    FROM "sampleDB".DevOps
    WHERE measure_name = 'cpu_utilization'
        AND time > ago(2h)
), avg_per_host_cpu AS (
    SELECT region, az, hostname, AVG(measure_value::double) AS avg_cpu_utilization
    FROM "sampleDB".DevOps
    WHERE measure_name = 'cpu_utilization'
        AND time > ago(2h)
    GROUP BY region, az, hostname
)
SELECT region, az, hostname, avg_cpu_utilization, fleet_avg_cpu_utilization
FROM avg_fleet_utilization, avg_per_host_cpu
WHERE avg_cpu_utilization > 1.1 * fleet_avg_cpu_utilization
ORDER BY avg_cpu_utilization DESC
```

尋找過去 2 小時內，特定EC2主機以 30 秒為間隔的平均CPU使用率：

```
SELECT BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double), 2) AS
 avg_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
GROUP BY hostname, BIN(time, 30s)
ORDER BY binned_timestamp ASC
```

找出過去 2 小時內特定EC2主機以 30 秒間隔固定的平均CPU使用率，並使用線性插補填入缺少的值：

```
WITH binned_timeseries AS (
    SELECT hostname, BIN(time, 30s) AS binned_timestamp,
 ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization
    FROM "sampleDB".DevOps
```

```
    WHERE measure_name = 'cpu_utilization'
        AND hostname = 'host-Hovjv'
        AND time > ago(2h)
    GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
    SELECT hostname,
        INTERPOLATE_LINEAR(
            CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
                SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
    FROM binned_timeseries
    GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

找出過去 2 小時內特定EC2主機以 30 秒間隔固定的平均CPU使用率，根據上次轉送的觀察，使用插補來填入遺失值：

```
WITH binned_timeseries AS (
    SELECT hostname, BIN(time, 30s) AS binned_timestamp,
 ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization
    FROM "sampleDB".DevOps
    WHERE measure_name = 'cpu_utilization'
        AND hostname = 'host-Hovjv'
        AND time > ago(2h)
    GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
    SELECT hostname,
        INTERPOLATE_LOCF(
            CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
                SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
    FROM binned_timeseries
    GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

具有彙總函數的查詢

以下是 IoT 案例資料集範例，以說明具有彙總函數的查詢。

主題

範例資料

Timestream 可讓您存放和分析 IoT 感應器資料，例如一或多個卡車機群的位置、油耗、速度和負載容量，以實現有效的機群管理。以下是資料表 iot_trucks 的結構描述和部分資料，該資料表存放遙測，例如卡車的位置、油耗、速度和負載容量。

時間	truck_id	Make	模型	機群	fuel_capacity	load_capacity	measure_name	measure_value::double	measure_value::varchar
2019-12-4 19:00:00.0000000	1234567	GMC	Astro	Alpha	100	500	fuel_reading	65.2	null
2019-12-4 19:00:00.0000000	1234567	GMC	Astro	Alpha	100	500	載入	400.0	null
2019-12-4 19:00:00.0000000	1234567	GMC	Astro	Alpha	100	500	speed	90.2	null

時間	truck_id	Make	模型	機群	fuel_capacity	load_capacity	measure_name	measure_value::double	measure_value::varchar
2019-12-4 19：00：00.0000000	1234567	GMC	Astro	Alpha	100	500	location	null	47.6062 N，122.3321 W
2019-12-4 19：00：00.0000000	1234567	肯沃思	W900	Alpha	150	1000	fuel_reading	10.1	null
2019-12-4 19：00：00.0000000	1234567	肯沃思	W900	Alpha	150	1000	載入	950.3	null
2019-12-4 19：00：00.0000000	1234567	肯沃思	W900	Alpha	150	1000	speed	50.8	null
2019-12-4 19：00：00.0000000	1234567	肯沃思	W900	Alpha	150	1000	location	null	北緯 40.7128 度、西緯 74.0060 度

查詢範例

取得機群中每輛卡車監控的所有感應器屬性和值清單。

```
SELECT
    truck_id,
    fleet,
    fuel_capacity,
    model,
    load_capacity,
    make,
    measure_name
FROM "sampleDB".IoT
GROUP BY truck_id, fleet, fuel_capacity, model, load_capacity, make, measure_name
```

取得過去 24 小時內機群中每輛卡車的最新燃油讀數。

```
WITH latest_recorded_time AS (
    SELECT
        truck_id,
        max(time) as latest_time
    FROM "sampleDB".IoT
    WHERE measure_name = 'fuel-reading'
    AND time >= ago(24h)
    GROUP BY truck_id
)
SELECT
    b.truck_id,
    b.fleet,
    b.make,
    b.model,
    b.time,
    b.measure_value::double as last_reported_fuel_reading
FROM
latest_recorded_time a INNER JOIN "sampleDB".IoT b
ON a.truck_id = b.truck_id AND b.time = a.latest_time
WHERE b.measure_name = 'fuel-reading'
AND b.time > ago(24h)
ORDER BY b.truck_id
```

識別過去 48 小時內以低燃料 （低於 10 % ） 運作的卡車：

```
WITH low_fuel_trucks AS (
```

```
    SELECT time, truck_id, fleet, make, model, (measure_value::double/
cast(fuel_capacity as double)*100) AS fuel_pct
    FROM "sampleDB".IoT
    WHERE time >= ago(48h)
    AND (measure_value::double/cast(fuel_capacity as double)*100) < 10
    AND measure_name = 'fuel-reading'
),
other_trucks AS (
SELECT time, truck_id, (measure_value::double/cast(fuel_capacity as double)*100) as
 remaining_fuel
    FROM "sampleDB".IoT
    WHERE time >= ago(48h)
    AND truck_id IN (SELECT truck_id FROM low_fuel_trucks)
    AND (measure_value::double/cast(fuel_capacity as double)*100) >= 10
    AND measure_name = 'fuel-reading'
),
trucks_that_refuelled AS (
    SELECT a.truck_id
    FROM low_fuel_trucks a JOIN other_trucks b
    ON a.truck_id = b.truck_id AND b.time >= a.time
)
SELECT DISTINCT truck_id, fleet, make, model, fuel_pct
FROM low_fuel_trucks
WHERE truck_id NOT IN (
    SELECT truck_id FROM trucks_that_refuelled
)
```

尋找過去一週每輛卡車的平均負載和最大速度：

```
SELECT
    bin(time, 1d) as binned_time,
    fleet,
    truck_id,
    make,
    model,
    AVG(
        CASE WHEN measure_name = 'load' THEN measure_value::double ELSE NULL END
    ) AS avg_load_tons,
    MAX(
        CASE WHEN measure_name = 'speed' THEN measure_value::double ELSE NULL END
    ) AS max_speed_mph
FROM "sampleDB".IoT
WHERE time >= ago(7d)
```

```
AND measure_name IN ('load', 'speed')
GROUP BY fleet, truck_id, make, model, bin(time, 1d)
ORDER BY truck_id
```

取得過去一週每輛卡車的負載效率：

```
WITH average_load_per_truck AS (
    SELECT
        truck_id,
        avg(measure_value::double)  AS avg_load
    FROM "sampleDB".IoT
    WHERE measure_name = 'load'
    AND time >= ago(7d)
    GROUP BY truck_id, fleet, load_capacity, make, model
),
truck_load_efficiency AS (
    SELECT
        a.truck_id,
        fleet,
        load_capacity,
        make,
        model,
        avg_load,
        measure_value::double,
        time,
        (measure_value::double*100)/avg_load as load_efficiency -- ,
 approx_percentile(avg_load_pct, DOUBLE '0.9')
    FROM "sampleDB".IoT a JOIN average_load_per_truck b
    ON a.truck_id = b.truck_id
    WHERE a.measure_name = 'load'
)
SELECT
    truck_id,
    time,
    load_efficiency
FROM truck_load_efficiency
ORDER BY truck_id, time
```

API 參考

本節包含 Amazon Timestream 的API參考文件。

Timestream 有兩個 APIs：查詢和寫入。

- 寫入API可讓您執行操作，例如建立資料表、標記資源，以及將記錄寫入 Timestream。
- 查詢API可讓您執行查詢操作。

> **Note**
>
> 兩者都APIs包含 DescribeEndpoints 動作。對於查詢和寫入， DescribeEndpoints 動作相同。

您可以閱讀API下列各項的詳細資訊，以及資料類型、常見錯誤和參數。

> **Note**
>
> 如需所有 AWS 服務常見的錯誤碼，請參閱AWS 支援區段 。

主題

動作

Amazon Timestream Write 支援下列動作：

- CreateBatchLoadTask
- CreateDatabase
- CreateTable
- DeleteDatabase
- DeleteTable
- DescribeBatchLoadTask
- DescribeDatabase

- DescribeEndpoints
- DescribeTable
- ListBatchLoadTasks
- ListDatabases
- ListTables
- ListTagsForResource
- ResumeBatchLoadTask
- TagResource
- UntagResource
- UpdateDatabase
- UpdateTable
- WriteRecords

Amazon Timestream Query 支援下列動作：

- CancelQuery
- CreateScheduledQuery
- DeleteScheduledQuery
- DescribeAccountSettings
- DescribeEndpoints
- DescribeScheduledQuery
- ExecuteScheduledQuery
- ListScheduledQueries
- ListTagsForResource
- PrepareQuery
- Query
- TagResource
- UntagResource
- UpdateAccountSettings
- UpdateScheduledQuery

Amazon Timestream 寫入

Amazon Timestream Write 支援下列動作：

- CreateBatchLoadTask
- CreateDatabase
- CreateTable
- DeleteDatabase
- DeleteTable
- DescribeBatchLoadTask
- DescribeDatabase
- DescribeEndpoints
- DescribeTable
- ListBatchLoadTasks
- ListDatabases
- ListTables
- ListTagsForResource
- ResumeBatchLoadTask
- TagResource
- UntagResource
- UpdateDatabase
- UpdateTable
- WriteRecords

CreateBatchLoadTask

服務：Amazon Timestream Write

建立新的 Timestream 批次載入任務。批次載入任務會處理 S3 位置中CSV來源的資料，並寫入 Timestream 資料表。從來源到目標的映射會在批次載入任務中定義。錯誤和事件會寫入 S3 位置的報告。對於報告，如果未指定 AWS KMS 金鑰，則當 SSE_S3是 選項時，報告將使用 S3 受管金鑰加密。否則會擲回錯誤。如需詳細資訊，請參閱 AWS 受管金鑰。服務配額適用。如需詳細資訊，請參閱程式碼範例 。

請求語法

```
{
   "ClientToken": "string",
   "DataModelConfiguration": {
      "DataModel": {
         "DimensionMappings": [
            {
               "DestinationColumn": "string",
               "SourceColumn": "string"
            }
         ],
         "MeasureNameColumn": "string",
         "MixedMeasureMappings": [
            {
               "MeasureName": "string",
               "MeasureValueType": "string",
               "MultiMeasureAttributeMappings": [
                  {
                     "MeasureValueType": "string",
                     "SourceColumn": "string",
                     "TargetMultiMeasureAttributeName": "string"
                  }
               ],
               "SourceColumn": "string",
               "TargetMeasureName": "string"
            }
         ],
         "MultiMeasureMappings": {
            "MultiMeasureAttributeMappings": [
               {
                  "MeasureValueType": "string",
                  "SourceColumn": "string",
                  "TargetMultiMeasureAttributeName": "string"
```

```
                }
            ],
            "TargetMultiMeasureName": "string"
         },
         "TimeColumn": "string",
         "TimeUnit": "string"
      },
      "DataModelS3Configuration": {
         "BucketName": "string",
         "ObjectKey": "string"
      }
   },
   "DataSourceConfiguration": {
      "CsvConfiguration": {
         "ColumnSeparator": "string",
         "EscapeChar": "string",
         "NullValue": "string",
         "QuoteChar": "string",
         "TrimWhiteSpace": boolean
      },
      "DataFormat": "string",
      "DataSourceS3Configuration": {
         "BucketName": "string",
         "ObjectKeyPrefix": "string"
      }
   },
   "RecordVersion": number,
   "ReportConfiguration": {
      "ReportS3Configuration": {
         "BucketName": "string",
         "EncryptionOption": "string",
         "KmsKeyId": "string",
         "ObjectKeyPrefix": "string"
      }
   },
   "TargetDatabaseName": "string",
   "TargetTableName": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

ClientToken

類型：字串

長度限制：長度下限為 1。長度上限為 64。

必要：否

DataModelConfiguration

類型：DataModelConfiguration 物件

必要：否

DataSourceConfiguration

定義批次載入任務之資料來源的組態詳細資訊。

類型：DataSourceConfiguration 物件

必要：是

RecordVersion

類型：Long

必要：否

ReportConfiguration

批次載入任務的報告組態。這包含錯誤報告存放位置的詳細資訊。

類型：ReportConfiguration 物件

必要：是

TargetDatabaseName

批次載入任務的目標 Timestream 資料庫。

類型：字串

模式：`[a-zA-Z0-9_.-]+`

必要：是

TargetTableName

批次載入任務的目標時間串流表。

類型：字串

模式：[a-zA-Z0-9_.-]+

必要：是

回應語法

```
{
   "TaskId": "string"
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

TaskId

批次載入任務的 ID。

類型：字串

長度限制：長度下限為 3。長度上限為 32。

模式：[A-Z0-9]+

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

ConflictException

Timestream 無法處理此請求，因為它包含已存在的資源。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ServiceQuotaExceededException

此帳戶的 資源執行個體配額超過上限。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面

- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

CreateDatabase

服務：Amazon Timestream Write

建立新的 Timestream 資料庫。如果未指定 AWS KMS 金鑰，則會使用您帳戶中的 Timestream 受管 AWS KMS 金鑰來加密資料庫。如需詳細資訊，請參閱 AWS 受管金鑰。服務配額適用。如需詳細資訊，請參閱程式碼範例 。

請求語法

```
{
   "DatabaseName": "string",
   "KmsKeyId": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

DatabaseName

Timestream 資料庫的名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

模式：[a-zA-Z0-9_.-]+

必要：是

KmsKeyId

資料庫的 AWS KMS 金鑰。如果未指定 AWS KMS 金鑰，則會使用您帳戶中的 Timestream 受管 AWS KMS 金鑰來加密資料庫。如需詳細資訊，請參閱 AWS 受管金鑰。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：否

Tags

標記資料表的鍵值對清單。

類型：Tag 物件陣列

陣列成員：項目數下限為 0。項目數上限為 200。

必要：否

回應語法

```
{
   "Database": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "KmsKeyId": "string",
      "LastUpdatedTime": number,
      "TableCount": number
   }
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

Database

新建立的 Timestream 資料庫。

類型：Database 物件

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

ConflictException

Timestream 無法處理此請求，因為它包含已存在的資源。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ServiceQuotaExceededException

此帳戶的 資源執行個體配額超過上限。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

CreateTable

服務：Amazon Timestream Write

將新資料表新增至您帳戶中的現有資料庫。在 中 AWS 帳戶，如果資料表名稱位於相同的資料庫中，則每個區域中的資料表名稱至少必須是唯一的。如果資料表位於不同的資料庫中，您可能在同一區域中具有相同的資料表名稱。建立資料表時，您必須指定資料表名稱、資料庫名稱，以及保留屬性。服務配額適用。有關詳細資訊，請參閱程式碼範例。

請求語法

```
{
   "DatabaseName": "string",
   "MagneticStoreWriteProperties": {
      "EnableMagneticStoreWrites": boolean,
      "MagneticStoreRejectedDataLocation": {
         "S3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "KmsKeyId": "string",
            "ObjectKeyPrefix": "string"
         }
      }
   },
   "RetentionProperties": {
      "MagneticStoreRetentionPeriodInDays": number,
      "MemoryStoreRetentionPeriodInHours": number
   },
   "Schema": {
      "CompositePartitionKey": [
         {
            "EnforcementInRecord": "string",
            "Name": "string",
            "Type": "string"
         }
      ]
   },
   "TableName": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
```

```
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

DatabaseName

Timestream 資料庫的名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

模式：`[a-zA-Z0-9_.-]+`

必要：是

MagneticStoreWriteProperties

包含啟用磁性存放區寫入時要在表格上設定的屬性。

類型：MagneticStoreWriteProperties 物件

必要：否

RetentionProperties

時間序列資料必須存放在記憶體存放區和磁性存放區的持續時間。

類型：RetentionProperties 物件

必要：否

Schema

資料表的結構描述。

類型：Schema 物件

必要：否

TableName

Timestream 資料表的名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

模式：[a-zA-Z0-9_.-]+

必要：是

Tags

標記資料表的鍵值對清單。

類型：Tag 物件陣列

陣列成員：項目數下限為 0。項目數上限為 200。

必要：否

回應語法

```
{
   "Table": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "LastUpdatedTime": number,
      "MagneticStoreWriteProperties": {
         "EnableMagneticStoreWrites": boolean,
         "MagneticStoreRejectedDataLocation": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "KmsKeyId": "string",
               "ObjectKeyPrefix": "string"
            }
         }
      },
      "RetentionProperties": {
         "MagneticStoreRetentionPeriodInDays": number,
         "MemoryStoreRetentionPeriodInHours": number
      },
      "Schema": {
         "CompositePartitionKey": [
            {
```

```
                "EnforcementInRecord": "string",
                "Name": "string",
                "Type": "string"
            }
         ]
      },
      "TableName": "string",
      "TableStatus": "string"
   }
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

Table

新建立的 Timestream 資料表。

類型：Table 物件

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

ConflictException

Timestream 無法處理此請求，因為它包含已存在的資源。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ServiceQuotaExceededException

此帳戶的 資源執行個體配額超過上限。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2

- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

DeleteDatabase
服務：Amazon Timestream Write

刪除指定的 Timestream 資料庫。這是不可逆的操作。刪除資料庫後，無法復原資料表中的時間序列資料。

> **Note**
>
> 必須先刪除資料庫中的所有資料表，否則會擲回 ValidationException 錯誤。
> 由於分散式重試的性質，操作可能會傳回成功或 ResourceNotFoundException。客戶應該將他們視為同等客戶。

有關詳細資訊，請參閱程式碼範例。

請求語法

```
{
   "DatabaseName": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

DatabaseName

要刪除的 Timestream 資料庫名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：是

回應元素

如果動作成功，服務會傳回 200 HTTP 回應，其中內HTTP文為空白。

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET

- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

DeleteTable
服務：Amazon Timestream Write

刪除指定的時間串流資料表。這是不可逆的操作。刪除時間串流資料庫資料表後，儲存在資料表中的時間序列資料將無法復原。

> **Note**
>
> 由於分散式重試的性質，操作可能會傳回成功或 ResourceNotFoundException。客戶應該將他們視為同等客戶。

有關詳細資訊，請參閱程式碼範例。

請求語法

```
{
   "DatabaseName": "string",
   "TableName": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

DatabaseName

要刪除 Timestream 資料庫的資料庫名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：是

TableName

要刪除的 Timestream 資料表名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：是

回應元素

如果動作成功，服務會傳回 200 HTTP 回應，其中內HTTP文為空白。

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

DescribeBatchLoadTask
服務：Amazon Timestream Write

傳回批次載入任務的相關資訊，包括組態、映射、進度和其他詳細資訊。服務配額適用。有關詳細資訊，請參閱程式碼範例。

請求語法

```
{
   "TaskId": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

TaskId

批次載入任務的 ID。

類型：字串

長度限制：長度下限為 3。長度上限為 32。

模式：[A-Z0-9]+

必要：是

回應語法

```
{
   "BatchLoadTaskDescription": {
      "CreationTime": number,
      "DataModelConfiguration": {
         "DataModel": {
            "DimensionMappings": [
               {
                  "DestinationColumn": "string",
                  "SourceColumn": "string"
               }
            ],
```

```
            "MeasureNameColumn": "string",
            "MixedMeasureMappings": [
               {
                  "MeasureName": "string",
                  "MeasureValueType": "string",
                  "MultiMeasureAttributeMappings": [
                     {
                        "MeasureValueType": "string",
                        "SourceColumn": "string",
                        "TargetMultiMeasureAttributeName": "string"
                     }
                  ],
                  "SourceColumn": "string",
                  "TargetMeasureName": "string"
               }
            ],
            "MultiMeasureMappings": {
               "MultiMeasureAttributeMappings": [
                  {
                     "MeasureValueType": "string",
                     "SourceColumn": "string",
                     "TargetMultiMeasureAttributeName": "string"
                  }
               ],
               "TargetMultiMeasureName": "string"
            },
            "TimeColumn": "string",
            "TimeUnit": "string"
         },
         "DataModelS3Configuration": {
            "BucketName": "string",
            "ObjectKey": "string"
         }
      },
      "DataSourceConfiguration": {
         "CsvConfiguration": {
            "ColumnSeparator": "string",
            "EscapeChar": "string",
            "NullValue": "string",
            "QuoteChar": "string",
            "TrimWhiteSpace": boolean
         },
         "DataFormat": "string",
         "DataSourceS3Configuration": {
```

```
            "BucketName": "string",
            "ObjectKeyPrefix": "string"
         }
      },
      "ErrorMessage": "string",
      "LastUpdatedTime": number,
      "ProgressReport": {
         "BytesMetered": number,
         "FileFailures": number,
         "ParseFailures": number,
         "RecordIngestionFailures": number,
         "RecordsIngested": number,
         "RecordsProcessed": number
      },
      "RecordVersion": number,
      "ReportConfiguration": {
         "ReportS3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "KmsKeyId": "string",
            "ObjectKeyPrefix": "string"
         }
      },
      "ResumableUntil": number,
      "TargetDatabaseName": "string",
      "TargetTableName": "string",
      "TaskId": "string",
      "TaskStatus": "string"
   }
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

BatchLoadTaskDescription

批次載入任務的說明。

類型：BatchLoadTaskDescription 物件

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2

- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

DescribeDatabase
服務：Amazon Timestream Write

傳回資料庫的相關資訊，包括資料庫名稱、建立資料庫的時間，以及在資料庫中找到的資料表總數。服務配額適用。有關詳細資訊，請參閱程式碼範例。

請求語法

```
{
   "DatabaseName": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

DatabaseName

Timestream 資料庫的名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：是

回應語法

```
{
   "Database": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "KmsKeyId": "string",
      "LastUpdatedTime": number,
      "TableCount": number
   }
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

Database

Timestream 資料表的名稱。

類型：Database 物件

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

DescribeEndpoints
服務：Amazon Timestream Write

傳回可用端點的清單，以針對 進行 Timestream API呼叫。API 此操作可透過寫入和查詢 進行APIs。

由於 Timestream SDKs 旨在透明地使用服務的架構，包括服務端點的管理和映射，因此我們不建議您使用此操作，API除非 ：

- 您正在VPC搭配 Timestream 使用端點 （ AWS PrivateLink ）
- 您的應用程式使用尚未SDK支援的程式設計語言
- 您需要更好地控制用戶端實作

如需如何使用和何時實作 的詳細資訊 DescribeEndpoints，請參閱端點探索模式 。

回應語法

```
{
   "Endpoints": [
      {
         "Address": "string",
         "CachePeriodInMinutes": number
      }
   ]
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

Endpoints

發出DescribeEndpoints請求時，會傳回Endpoints物件。

類型：Endpoint 物件陣列

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

DescribeTable
服務：Amazon Timestream Write

傳回資料表的相關資訊，包括資料表名稱、資料庫名稱、記憶體存放區和磁性存放區的保留期間。服務配額適用。有關詳細資訊，請參閱程式碼範例。

請求語法

```
{
   "DatabaseName": "string",
   "TableName": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

DatabaseName

Timestream 資料庫的名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：是

TableName

Timestream 資料表的名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：是

回應語法

```
{
   "Table": {
```

```
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "LastUpdatedTime": number,
      "MagneticStoreWriteProperties": {
         "EnableMagneticStoreWrites": boolean,
         "MagneticStoreRejectedDataLocation": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "KmsKeyId": "string",
               "ObjectKeyPrefix": "string"
            }
         }
      },
      "RetentionProperties": {
         "MagneticStoreRetentionPeriodInDays": number,
         "MemoryStoreRetentionPeriodInHours": number
      },
      "Schema": {
         "CompositePartitionKey": [
            {
               "EnforcementInRecord": "string",
               "Name": "string",
               "Type": "string"
            }
         ]
      },
      "TableName": "string",
      "TableStatus": "string"
   }
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

Table

Timestream 資料表。

類型：Table 物件

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET

- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

ListBatchLoadTasks
服務：Amazon Timestream Write

提供批次載入任務的清單，以及任務可繼續直到 時的名稱、狀態，以及其他詳細資訊。有關詳細資訊，請參閱程式碼範例。

請求語法

```
{
   "MaxResults": number,
   "NextToken": "string",
   "TaskStatus": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

MaxResults

輸出中要傳回的項目總數。如果可用的項目總數超過指定的值， NextToken 則會在輸出中提供 。若要繼續分頁，請提供 NextToken 值作為後續API調用的引數。

類型：整數

有效範圍：最小值為 1。最大值為 100。

必要：否

NextToken

用以指定分頁開始位置的字符。這是先前截斷的回應 NextToken 中的 。

類型：字串

必要：否

TaskStatus

批次載入任務的狀態。

類型：字串

有效值:CREATED | IN_PROGRESS | FAILED | SUCCEEDED | PROGRESS_STOPPED | PENDING_RESUME

必要：否

回應語法

```
{
   "BatchLoadTasks": [
      {
         "CreationTime": number,
         "DatabaseName": "string",
         "LastUpdatedTime": number,
         "ResumableUntil": number,
         "TableName": "string",
         "TaskId": "string",
         "TaskStatus": "string"
      }
   ],
   "NextToken": "string"
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

BatchLoadTasks

批次載入任務詳細資訊的清單。

類型：BatchLoadTask 物件陣列

NextToken

用以指定分頁開始位置的字符。提供下一個 ListBatchLoadTasksRequest。

類型：字串

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python

- AWS SDK 適用於 Ruby V3

ListDatabases
服務：Amazon Timestream Write

傳回 Timestream 資料庫的清單。服務配額適用。有關詳細資訊，請參閱程式碼範例。

請求語法

```
{
   "MaxResults": number,
   "NextToken": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

MaxResults

在輸出中傳回的項目總數。如果可用的項目總數超過指定的值， NextToken 則會在輸出中提供 。若要繼續分頁，請提供 NextToken 值作為後續API調用的引數。

類型：整數

有效範圍：最小值為 1。最大值為 20。

必要：否

NextToken

分頁權杖。若要恢復分頁，請提供 NextToken 值作為後續API調用的引數。

類型：字串

必要：否

回應語法

```
{
   "Databases": [
      {
```

```
            "Arn": "string",
            "CreationTime": number,
            "DatabaseName": "string",
            "KmsKeyId": "string",
            "LastUpdatedTime": number,
            "TableCount": number
        }
    ],
    "NextToken": "string"
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

Databases

資料庫名稱清單。

類型：Database 物件陣列

NextToken

分頁權杖。當回應被截斷時，會傳回此參數。

類型：字串

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

ListTables
服務：Amazon Timestream Write

提供資料表清單，以及每個資料表的名稱、狀態和保留屬性。有關詳細資訊，請參閱程式碼範例。

請求語法

```
{
   "DatabaseName": "string",
   "MaxResults": number,
   "NextToken": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

DatabaseName

Timestream 資料庫的名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：否

MaxResults

在輸出中傳回的項目總數。如果可用的項目總數超過指定的值， NextToken 則會在輸出中提供 。若要繼續分頁，請提供 NextToken 值作為後續API調用的引數。

類型：整數

有效範圍：最小值為 1。最大值為 20。

必要：否

NextToken

分頁權杖。若要恢復分頁，請提供 NextToken 值作為後續API調用的引數。

類型：字串

必要：否

回應語法

```
{
   "NextToken": "string",
   "Tables": [
      {
         "Arn": "string",
         "CreationTime": number,
         "DatabaseName": "string",
         "LastUpdatedTime": number,
         "MagneticStoreWriteProperties": {
            "EnableMagneticStoreWrites": boolean,
            "MagneticStoreRejectedDataLocation": {
               "S3Configuration": {
                  "BucketName": "string",
                  "EncryptionOption": "string",
                  "KmsKeyId": "string",
                  "ObjectKeyPrefix": "string"
               }
            }
         },
         "RetentionProperties": {
            "MagneticStoreRetentionPeriodInDays": number,
            "MemoryStoreRetentionPeriodInHours": number
         },
         "Schema": {
            "CompositePartitionKey": [
               {
                  "EnforcementInRecord": "string",
                  "Name": "string",
                  "Type": "string"
               }
            ]
         },
         "TableName": "string",
         "TableStatus": "string"
      }
   ]
```

```
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

NextToken

用以指定分頁開始位置的字符。這是先前截斷的回應 NextToken 中的 。

類型：字串

Tables

資料表清單。

類型：Table 物件陣列

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

ListTagsForResource
服務：Amazon Timestream Write

列出 Timestream 資源上的所有標籤。

請求語法

```
{
   "ResourceARN": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

ResourceARN

要列出標籤的 Timestream 資源。此值是 Amazon Resource Name （ ARN ）。

類型：字串

長度限制：長度下限為 1。長度上限為 1011。

必要：是

回應語法

```
{
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

Tags

目前與 Timestream 資源相關聯的標籤。

類型：Tag 物件陣列

陣列成員：項目數下限為 0。項目數上限為 200。

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET

- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

ResumeBatchLoadTask
服務：Amazon Timestream Write

請求語法

```
{
   "TaskId": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

TaskId

要繼續的批次載入任務 ID。

類型：字串

長度限制：長度下限為 3。長度上限為 32。

模式：[A-Z0-9]+

必要：是

回應元素

如果動作成功，服務會傳回 200 HTTP 回應，其中內HTTP文為空白。

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python

- AWS SDK 適用於 Ruby V3

TagResource
服務：Amazon Timestream Write

將一組標籤與 Timestream 資源建立關聯。然後，您可以啟用這些使用者定義的標籤，使其出現在 Billing and Cost Management 主控台上，以進行成本分配追蹤。

請求語法

```
{
   "ResourceARN": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

ResourceARN

識別應新增標籤的 Timestream 資源。此值是 Amazon Resource Name （ ARN ）。

類型：字串

長度限制：長度下限為 1。長度上限為 1011。

必要：是

Tags

要指派給 Timestream 資源的標籤。

類型：Tag 物件陣列

陣列成員：項目數下限為 0。項目數上限為 200。

必要：是

回應元素

如果動作成功，服務會傳回 200 HTTP 回應，其中內HTTP文為空白。

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ServiceQuotaExceededException

此帳戶的 資源執行個體配額超過上限。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++

- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

UntagResource
服務：Amazon Timestream Write

從 Timestream 資源中移除標籤的關聯。

請求語法

```
{
   "ResourceARN": "string",
   "TagKeys": [ "string" ]
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

ResourceARN

標籤將從中移除的 Timestream 資源。此值是 Amazon Resource Name （ ARN ）。

類型：字串

長度限制：長度下限為 1。長度上限為 1011。

必要：是

TagKeys

標籤金鑰清單。資源的現有標籤，其金鑰是此清單的成員，將會從 Timestream 資源中移除。

類型：字串陣列

陣列成員：項目數下限為 0。項目數上限為 200。

長度限制：長度下限為 1。長度上限為 128。

必要：是

回應元素

如果動作成功，服務會傳回 200 HTTP 回應，其中內HTTP文為空白。

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ServiceQuotaExceededException

此帳戶的 資源執行個體配額超過上限。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2

- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

UpdateDatabase
服務：Amazon Timestream Write

修改現有資料庫的 AWS KMS 金鑰。更新資料庫時，您必須指定要使用的新 AWS KMS 金鑰的資料庫名稱和識別碼（KmsKeyId）。如果有任何並行UpdateDatabase請求，第一個寫入器會獲勝。

有關詳細資訊，請參閱程式碼範例。

請求語法

```
{
   "DatabaseName": "string",
   "KmsKeyId": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

DatabaseName

資料庫的名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：是

KmsKeyId

用來加密資料庫中存放資料之新 AWS KMS 金鑰（KmsKeyId）的識別碼。如果KmsKeyId目前向資料庫註冊的 與請求KmsKeyId中的 相同，則不會有任何更新。

您可以使用下列KmsKeyId任一項來指定：

- 金鑰 ID：1234abcd-12ab-34cd-56ef-1234567890ab
- 金鑰ARN： arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
- 別名名稱：alias/ExampleAlias

- 別名ARN： `arn:aws:kms:us-east-1:111122223333:alias/ExampleAlias`

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：是

回應語法

```
{
   "Database": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "KmsKeyId": "string",
      "LastUpdatedTime": number,
      "TableCount": number
   }
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

Database

資料表的頂層容器。資料庫和資料表是 Amazon Timestream 中的基本管理概念。資料庫中的所有資料表都會使用相同的 AWS KMS 金鑰加密。

類型：Database 物件

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ServiceQuotaExceededException

此帳戶的 資源執行個體配額超過上限。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++

- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

UpdateTable

服務：Amazon Timestream Write

修改 Timestream 資料表的記憶體存放區和磁性存放區的保留期間。請注意，保留期間的變更會立即生效。例如，如果記憶體存放區的保留期最初設定為 2 小時，然後變更為 24 小時，則記憶體存放區將能夠保存 24 小時的資料，但在進行此變更的 22 小時後，將填入 24 小時的資料。Timestream 不會從磁性存放區擷取資料以填入記憶體存放區。

有關詳細資訊，請參閱程式碼範例。

請求語法

```
{
   "DatabaseName": "string",
   "MagneticStoreWriteProperties": {
      "EnableMagneticStoreWrites": boolean,
      "MagneticStoreRejectedDataLocation": {
         "S3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "KmsKeyId": "string",
            "ObjectKeyPrefix": "string"
         }
      }
   },
   "RetentionProperties": {
      "MagneticStoreRetentionPeriodInDays": number,
      "MemoryStoreRetentionPeriodInHours": number
   },
   "Schema": {
      "CompositePartitionKey": [
         {
            "EnforcementInRecord": "string",
            "Name": "string",
            "Type": "string"
         }
      ]
   },
   "TableName": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

DatabaseName

Timestream 資料庫的名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：是

MagneticStoreWriteProperties

包含啟用磁性存放區寫入時要在表格上設定的屬性。

類型：MagneticStoreWriteProperties 物件

必要：否

RetentionProperties

記憶體存放區和磁性存放區的保留期間。

類型：RetentionProperties 物件

必要：否

Schema

資料表的結構描述。

類型：Schema 物件

必要：否

TableName

Timestream 資料表的名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：是

回應語法

```
{
   "Table": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "LastUpdatedTime": number,
      "MagneticStoreWriteProperties": {
         "EnableMagneticStoreWrites": boolean,
         "MagneticStoreRejectedDataLocation": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "KmsKeyId": "string",
               "ObjectKeyPrefix": "string"
            }
         }
      },
      "RetentionProperties": {
         "MagneticStoreRetentionPeriodInDays": number,
         "MemoryStoreRetentionPeriodInHours": number
      },
      "Schema": {
         "CompositePartitionKey": [
            {
               "EnforcementInRecord": "string",
               "Name": "string",
               "Type": "string"
            }
         ]
      },
      "TableName": "string",
      "TableStatus": "string"
   }
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

Table

更新的 Timestream 資料表。

類型：Table 物件

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

WriteRecords
服務：Amazon Timestream Write

可讓您將時間序列資料寫入 Timestream。您可以指定要插入系統的單一資料點或一批資料點。Timestream 為您提供彈性結構描述，可根據您叫用寫入資料庫時所指定的資料點的維度名稱和資料類型，自動偵測 Timestream 資料表的資料欄名稱和資料類型。

Timestream 支援最終一致性僅供讀取語義。這表示當您在將一批資料寫入 Timestream 後立即查詢資料時，查詢結果可能不會反映最近完成的寫入操作的結果。結果也可能包含一些過時的資料。如果您在短時間內重複查詢請求，結果應傳回最新的資料。服務配額適用。

有關詳細資訊，請參閱程式碼範例。

Upserts

您可以在WriteRecords請求中使用 Version 參數來更新資料點。Timestream 會追蹤每個記錄的版本編號。 未針對請求中的記錄指定1時， Version預設為 。Timestream 會在收到寫入請求Version時，更新現有記錄的測量值及其 ，該記錄的寫入請求具有較高的Version數字。當它收到更新請求，其中測量值與現有記錄的值相同時，如果大於現有值 Version，則 Timestream 仍會更新 Version。只要 的值Version持續增加，您就可以視需要更新資料點多次。

例如，假設您在請求Version中沒有指示的情況下撰寫新記錄。Timestream 會儲存此記錄，並Version設定為 1。現在，假設您嘗試使用具有不同測量值的相同記錄WriteRecords請求來更新此記錄，但和之前一樣，不提供 Version。在此情況下，Timestream 將使用 拒絕此更新，RejectedRecordsException因為更新的記錄版本不大於 版本的現有值。

不過，如果您要重新傳送Version設定為 的更新請求2，Timestream 會成功更新記錄的值，而 Version會設定為 2。接下來，假設您已傳送具有相同記錄和相同量值的WriteRecords請求，但將 Version 設為 3。在此情況下，Timestream 只會更新Version為 3。任何進一步的更新都需要傳送大於 的版本號碼3，否則更新請求會收到 RejectedRecordsException。

請求語法

```
{
   "CommonAttributes": {
      "Dimensions": [
         {
            "DimensionValueType": "string",
            "Name": "string",
            "Value": "string"
         }
      ],
```

```
      "MeasureName": "string",
      "MeasureValue": "string",
      "MeasureValues": [
         {
            "Name": "string",
            "Type": "string",
            "Value": "string"
         }
      ],
      "MeasureValueType": "string",
      "Time": "string",
      "TimeUnit": "string",
      "Version": number
   },
   "DatabaseName": "string",
   "Records": [
      {
         "Dimensions": [
            {
               "DimensionValueType": "string",
               "Name": "string",
               "Value": "string"
            }
         ],
         "MeasureName": "string",
         "MeasureValue": "string",
         "MeasureValues": [
            {
               "Name": "string",
               "Type": "string",
               "Value": "string"
            }
         ],
         "MeasureValueType": "string",
         "Time": "string",
         "TimeUnit": "string",
         "Version": number
      }
   ],
   "TableName": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

CommonAttributes

包含請求中所有記錄共用的常見量值、維度、時間和版本屬性的記錄。將資料寫入 Timestream 時，指定的量值和維度屬性會與記錄物件中的量值和維度屬性合併。維度可能不會重疊，或者ValidationException會擲回 。換句話說，記錄必須包含具有唯一名稱的維度。

類型：Record 物件

必要：否

DatabaseName

Timestream 資料庫的名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：是

Records

包含每個時間序列資料點之唯一量值、維度、時間和版本屬性的記錄陣列。

類型：Record 物件陣列

陣列成員：項目數下限為 1。項目數上限為 100。

必要：是

TableName

Timestream 資料表的名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：是

回應語法

```
{
   "RecordsIngested": {
      "MagneticStore": number,
      "MemoryStore": number,
      "Total": number
   }
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

RecordsIngested

此請求擷取的記錄相關資訊。

類型：RecordsIngested 物件

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，Timestream 無法完整處理此請求。

HTTP 狀態碼：500

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

RejectedRecordsException

WriteRecords 在下列情況下， 會捨棄此例外狀況：

- 具有重複資料的記錄，其中有多個具有相同維度、時間戳記和量值名稱的記錄，但：
  - 測量值不同
  - 請求中沒有版本，或新記錄中的版本值等於或低於現有值

  在此情況下，如果 Timestream 拒絕資料，`RejectedRecords`回應中的`ExistingVersion`欄位會顯示目前記錄的版本。若要強制更新，您可以將記錄集的 版本重新傳送請求至大於 的值`ExistingVersion`。
- 時間戳記超出記憶體存放區保留期間的記錄。
- 維度或量值超過 Timestream 定義限制的記錄。

如需詳細資訊，請參閱 Amazon Timestream 開發人員指南中的配額。

HTTP 狀態碼：400

ResourceNotFoundException

操作嘗試存取不存在的資源。資源可能未正確指定，或其狀態可能不是 ACTIVE。

HTTP 狀態碼：400

ThrottlingException

使用者提出的請求過多，且超過服務配額。請求受到調節。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++

- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

Amazon Timestream 查詢

Amazon Timestream Query 支援下列動作：

- CancelQuery
- CreateScheduledQuery
- DeleteScheduledQuery
- DescribeAccountSettings
- DescribeEndpoints
- DescribeScheduledQuery
- ExecuteScheduledQuery
- ListScheduledQueries
- ListTagsForResource
- PrepareQuery
- Query
- TagResource
- UntagResource
- UpdateAccountSettings
- UpdateScheduledQuery

CancelQuery
服務：Amazon Timestream Query

取消已發出的查詢。只有在發出取消請求之前，查詢尚未完成執行時，才會提供取消。因為取消是無能操作，所以後續的取消請求會傳回 CancellationMessage，表示查詢已取消。有關詳細資訊，請參閱程式碼範例。

請求語法

```
{
   "QueryId": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

QueryId

需要取消的查詢 ID。 QueryID 會傳回為查詢結果的一部分。

類型：字串

長度限制：長度下限為 1。長度上限為 64。

模式：[a-zA-Z0-9]+

必要：是

回應語法

```
{
   "CancellationMessage": "string"
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

CancellationMessage

當 QueryId 指定的查詢CancelQuery請求已發出時， CancellationMessage 便會傳回。

類型：字串

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，服務無法完整處理此請求。

HTTP 狀態碼：400

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面

- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

CreateScheduledQuery

服務：Amazon Timestream Query

建立將按照設定的排程代表您執行的排程查詢。Timestream 假定提供的執行角色作為用於執行查詢的部分 ScheduledQueryExecutionRoleArn 參數。您可以使用 NotificationConfiguration 參數為排程查詢操作設定通知。

請求語法

```
{
   "ClientToken": "string",
   "ErrorReportConfiguration": {
      "S3Configuration": {
         "BucketName": "string",
         "EncryptionOption": "string",
         "ObjectKeyPrefix": "string"
      }
   },
   "KmsKeyId": "string",
   "Name": "string",
   "NotificationConfiguration": {
      "SnsConfiguration": {
         "TopicArn": "string"
      }
   },
   "QueryString": "string",
   "ScheduleConfiguration": {
      "ScheduleExpression": "string"
   },
   "ScheduledQueryExecutionRoleArn": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ],
   "TargetConfiguration": {
      "TimestreamConfiguration": {
         "DatabaseName": "string",
         "DimensionMappings": [
            {
               "DimensionValueType": "string",
               "Name": "string"
```

```
            }
         ],
         "MeasureNameColumn": "string",
         "MixedMeasureMappings": [
            {
               "MeasureName": "string",
               "MeasureValueType": "string",
               "MultiMeasureAttributeMappings": [
                  {
                     "MeasureValueType": "string",
                     "SourceColumn": "string",
                     "TargetMultiMeasureAttributeName": "string"
                  }
               ],
               "SourceColumn": "string",
               "TargetMeasureName": "string"
            }
         ],
         "MultiMeasureMappings": {
            "MultiMeasureAttributeMappings": [
               {
                  "MeasureValueType": "string",
                  "SourceColumn": "string",
                  "TargetMultiMeasureAttributeName": "string"
               }
            ],
            "TargetMultiMeasureName": "string"
         },
         "TableName": "string",
         "TimeColumn": "string"
      }
   }
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

ClientToken

使用 ClientToken 呼叫錯 CreateScheduledQuery 位，換句話說，重複提出相同的請求會產生相同的結果。提出多個相同的 CreateScheduledQuery 請求與提出單一請求具有相同的效果。

- 如果 CreateScheduledQuery 呼叫時沒有 `ClientToken`，則查詢會`ClientToken`代表您SDK產生 。
- 8 小時後，任何具有相同 `ClientToken` 的請求會視為新的請求。

類型：字串

長度限制：長度下限為 32。長度上限為 128。

必要：否

ErrorReportConfiguration

錯誤報告的組態。寫入查詢結果時遇到問題時，會產生錯誤報告。

類型：ErrorReportConfiguration 物件

必要：是

KmsKeyId

用來加密排程查詢資源的 Amazon KMS金鑰，靜態。如果未指定 Amazon KMS金鑰，排程查詢資源將使用 Timestream 擁有的 Amazon KMS金鑰加密。若要指定KMS金鑰，請使用金鑰 ID、金鑰 ARN、別名名稱或別名 ARN。使用別名時，請在名稱加上 alias/ 字首。

如果 ErrorReportConfiguration 使用 `SSE_KMS`作為加密類型，則相同 KmsKeyId 用於加密靜態錯誤報告。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：否

Name

排程查詢的名稱。

類型：字串

長度限制：長度下限為 1。長度上限為 64。

模式：`[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必要：是

NotificationConfiguration

排程查詢的通知組態。Timestream 會在查詢執行完成時、狀態更新或刪除查詢時傳送通知。

類型：NotificationConfiguration 物件

必要：是

QueryString

要執行的查詢字串。參數名稱可以在後跟一個識別碼的查詢字串 @ 字元中指定。命名參數 `@scheduled_runtime` 是保留的，並且可以在查詢中使用，以取得排程執行查詢的時間。

根據 ScheduleConfiguration 參數計算的時間戳記將是每次查詢執行的`@scheduled_runtime`參數值。例如，假設某個排程查詢執行個體是在 2021-12-01 00:00:00 執行。在此執行個體中，在叫用查詢時，`@scheduled_runtime` 參數會初始化為時間戳記 2021-12-01 00:00:00。

類型：字串

長度限制：長度下限為 1。長度上限為 262144。

必要：是

ScheduleConfiguration

查詢的排程組態。

類型：ScheduleConfiguration 物件

必要：是

ScheduledQueryExecutionRoleArn

執行排程查詢時，Timestream 將擔任ARNIAM的角色的 。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：是

Tags

用來標示排程查詢的鍵/值對清單。

類型：Tag 物件陣列

陣列成員：項目數下限為 0。項目數上限為 200。

必要：否

TargetConfiguration

用於寫入查詢結果的組態。

類型：TargetConfiguration 物件

必要：否

回應語法

```
{
   "Arn": "string"
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

Arn

ARN 建立的排程查詢。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

ConflictException

無法輪詢已取消查詢的結果。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，服務無法完整處理此請求。

HTTP 狀態碼：400

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ServiceQuotaExceededException

您已超過服務配額。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2

- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

DeleteScheduledQuery
服務：Amazon Timestream Query

刪除指定的排程查詢。這是不可逆的操作。

請求語法

```
{
   "ScheduledQueryArn": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

ScheduledQueryArn

排程查詢的 ARN。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：是

回應元素

如果動作成功，服務會傳回 200 HTTP 回應，其中內HTTP文為空白。

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，服務無法完整處理此請求。

HTTP 狀態碼：400

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

找不到請求的資源。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

DescribeAccountSettings
服務：Amazon Timestream Query

描述您帳戶的設定，其中包含查詢定價模型，以及服務可用於查詢工作負載TCUs的已設定上限。

您只需支付工作負載所使用的運算單位持續時間的費用。

回應語法

```
{
   "MaxQueryTCU": number,
   "QueryPricingModel": "string"
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

MaxQueryTCU

服務在任何時間點用來服務查詢的 Timestream 運算單位 （TCUs） 數量上限。

類型：整數

QueryPricingModel

您帳戶中查詢的定價模型。

類型：字串

有效值:BYTES_SCANNED | COMPUTE_UNITS

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，服務無法完整處理此請求。

HTTP 狀態碼：400

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

DescribeEndpoints
服務：Amazon Timestream Query

DescribeEndpoints 傳回可用端點的清單，以針對 進行 Timestream API呼叫。這可透過寫入和查詢API取得。

由於 Timestream SDKs 旨在透明地使用服務的架構，包括服務端點的管理和映射，因此不建議您使用此功能，API除非 ：

- 您正在VPC搭配 Timestream 使用端點 （ AWS PrivateLink ）
- 您的應用程式使用尚未SDK支援的程式設計語言
- 您需要更好地控制用戶端實作

如需如何使用和何時實作 的詳細資訊 DescribeEndpoints，請參閱端點探索模式 。

回應語法

```
{
   "Endpoints": [
      {
         "Address": "string",
         "CachePeriodInMinutes": number
      }
   ]
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

Endpoints

發出DescribeEndpoints請求時，會傳回Endpoints物件。

類型：Endpoint 物件陣列

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

InternalServerException

由於內部伺服器錯誤，服務無法完整處理此請求。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

DescribeScheduledQuery
服務：Amazon Timestream Query

提供有關排程查詢的詳細資訊。

請求語法

```
{
   "ScheduledQueryArn": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

ScheduledQueryArn

排程查詢的 ARN。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：是

回應語法

```
{
   "ScheduledQuery": {
      "Arn": "string",
      "CreationTime": number,
      "ErrorReportConfiguration": {
         "S3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "ObjectKeyPrefix": "string"
         }
      },
      "KmsKeyId": "string",
      "LastRunSummary": {
         "ErrorReportLocation": {
```

```
            "S3ReportLocation": {
               "BucketName": "string",
               "ObjectKey": "string"
            }
         },
         "ExecutionStats": {
            "BytesMetered": number,
            "CumulativeBytesScanned": number,
            "DataWrites": number,
            "ExecutionTimeInMillis": number,
            "QueryResultRows": number,
            "RecordsIngested": number
         },
         "FailureReason": "string",
         "InvocationTime": number,
         "QueryInsightsResponse": {
            "OutputBytes": number,
            "OutputRows": number,
            "QuerySpatialCoverage": {
               "Max": {
                  "PartitionKey": [ "string" ],
                  "TableArn": "string",
                  "Value": number
               }
            },
            "QueryTableCount": number,
            "QueryTemporalRange": {
               "Max": {
                  "TableArn": "string",
                  "Value": number
               }
            }
         },
         "RunStatus": "string",
         "TriggerTime": number
      },
      "Name": "string",
      "NextInvocationTime": number,
      "NotificationConfiguration": {
         "SnsConfiguration": {
            "TopicArn": "string"
         }
      },
      "PreviousInvocationTime": number,
```

```
      "QueryString": "string",
      "RecentlyFailedRuns": [
         {
            "ErrorReportLocation": {
               "S3ReportLocation": {
                  "BucketName": "string",
                  "ObjectKey": "string"
               }
            },
            "ExecutionStats": {
               "BytesMetered": number,
               "CumulativeBytesScanned": number,
               "DataWrites": number,
               "ExecutionTimeInMillis": number,
               "QueryResultRows": number,
               "RecordsIngested": number
            },
            "FailureReason": "string",
            "InvocationTime": number,
            "QueryInsightsResponse": {
               "OutputBytes": number,
               "OutputRows": number,
               "QuerySpatialCoverage": {
                  "Max": {
                     "PartitionKey": [ "string" ],
                     "TableArn": "string",
                     "Value": number
                  }
               },
               "QueryTableCount": number,
               "QueryTemporalRange": {
                  "Max": {
                     "TableArn": "string",
                     "Value": number
                  }
               }
            },
            "RunStatus": "string",
            "TriggerTime": number
         }
      ],
      "ScheduleConfiguration": {
         "ScheduleExpression": "string"
      },
```

```
      "ScheduledQueryExecutionRoleArn": "string",
      "State": "string",
      "TargetConfiguration": {
         "TimestreamConfiguration": {
            "DatabaseName": "string",
            "DimensionMappings": [
               {
                  "DimensionValueType": "string",
                  "Name": "string"
               }
            ],
            "MeasureNameColumn": "string",
            "MixedMeasureMappings": [
               {
                  "MeasureName": "string",
                  "MeasureValueType": "string",
                  "MultiMeasureAttributeMappings": [
                     {
                        "MeasureValueType": "string",
                        "SourceColumn": "string",
                        "TargetMultiMeasureAttributeName": "string"
                     }
                  ],
                  "SourceColumn": "string",
                  "TargetMeasureName": "string"
               }
            ],
            "MultiMeasureMappings": {
               "MultiMeasureAttributeMappings": [
                  {
                     "MeasureValueType": "string",
                     "SourceColumn": "string",
                     "TargetMultiMeasureAttributeName": "string"
                  }
               ],
               "TargetMultiMeasureName": "string"
            },
            "TableName": "string",
            "TimeColumn": "string"
         }
      }
   }
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

ScheduledQuery

排程查詢。

類型：ScheduledQueryDescription 物件

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，服務無法完整處理此請求。

HTTP 狀態碼：400

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

找不到請求的資源。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

ExecuteScheduledQuery
服務：Amazon Timestream Query

您可以使用此功能手動API執行排程查詢。

如果您啟用 QueryInsights，這API也會傳回與作為 Amazon SNS通知一部分而執行之查詢相關的洞見和指標。 會QueryInsights協助調整查詢的效能。如需 的詳細資訊QueryInsights，請參閱使用查詢洞察來最佳化 Amazon Timestream 中的查詢。

請求語法

```
{
   "ClientToken": "string",
   "InvocationTime": number,
   "QueryInsights": {
      "Mode": "string"
   },
   "ScheduledQueryArn": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

ClientToken

未使用。

類型：字串

長度限制：長度下限為 32。長度上限為 128。

必要：否

InvocationTime

中的時間戳記UTC。查詢將如同在此時間戳記叫用一樣執行。

類型：Timestamp

必要：是

QueryInsights

封裝啟用 的設定QueryInsights。

啟用 會在您執行查詢的 Amazon SNS通知中QueryInsights傳回洞見和指標。您可以使用QueryInsights來調整查詢效能和成本。

類型：ScheduledQueryInsights 物件

必要：否

ScheduledQueryArn

ARN 排程查詢。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：是

回應元素

如果動作成功，服務會傳回 200 HTTP 回應，其中內HTTP文為空白。

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，服務無法完整處理此請求。

HTTP 狀態碼：400

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

找不到請求的資源。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

範例

ENABLED_WITH_RATE_CONTROL 模式的排程查詢通知訊息

下列範例顯示 `QueryInsights` 參數`ENABLED_WITH_RATE_CONTROL`模式的成功排程查詢通知訊息。

```
"SuccessNotificationMessage": {
    "type": "MANUAL_TRIGGER_SUCCESS",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-49c6ed55-
c2e7-4cc2-9956-4a0ecea13420-80e05b035236a4c3",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1723710546,
        "triggerTimeMillis": 1723710547490,
        "runStatus": "MANUAL_TRIGGER_SUCCESS",
        "executionStats": {
            "executionTimeInMillis": 17343,
            "dataWrites": 1024,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 600,
            "recordsIngested": 1,
            "queryResultRows": 1
        },
        "queryInsightsResponse": {
            "querySpatialCoverage": {
                "max": {
                    "value": 1.0,
```

```
                    "tableArn": "arn:aws:timestream:<Region>:<Account>:database/BaseDb/
table/BaseTable",
                    "partitionKey": [
                        "measure_name"
                    ]
                }
            },
            "queryTemporalRange": {
                "max": {
                    "value": 2399999999999,
                    "tableArn": "arn:aws:timestream:<Region>:<Account>:database/BaseDb/
table/BaseTable"
                }
            },
            "queryTableCount": 1,
            "outputRows": 1,
            "outputBytes": 59
        }
    }
}
```

DISABLED 模式的排程查詢通知訊息

下列範例顯示 QueryInsights 參數DISABLED模式的成功排程查詢通知訊息。

```
"SuccessNotificationMessage": {
    "type": "MANUAL_TRIGGER_SUCCESS",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-
fa109d9e-6528-4a0d-ac40-482fa05e657f-140faaeecdc5b2a7",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1723711401,
        "triggerTimeMillis": 1723711402144,
        "runStatus": "MANUAL_TRIGGER_SUCCESS",
        "executionStats": {
            "executionTimeInMillis": 17992,
            "dataWrites": 1024,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 600,
            "recordsIngested": 1,
            "queryResultRows": 1
        }
    }
}
```

ENABLED_WITH_RATE_CONTROL 模式的失敗通知訊息

下列範例顯示 QueryInsights 參數ENABLED_WITH_RATE_CONTROL模式的失敗排程查詢通知訊息。

```
"FailureNotificationMessage": {
    "type": "MANUAL_TRIGGER_FAILURE",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-
b261670d-790c-4116-9db5-0798071b18b1-b7e27a1d79be226d",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1727915513,
        "triggerTimeMillis": 1727915513894,
        "runStatus": "MANUAL_TRIGGER_FAILURE",
        "executionStats": {
            "executionTimeInMillis": 10777,
            "dataWrites": 0,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 0,
            "recordsIngested": 0,
            "queryResultRows": 4
        },
        "errorReportLocation": {
            "s3ReportLocation": {
                "bucketName": "my-amzn-s3-demo-bucket",
                "objectKey": "4my-organization-f7a3c5d065a1a95e/1727915513/
MANUAL/1727915513894/5e14b3df-b147-49f4-9331-784f749b68ae"
            }
        },
        "failureReason": "Schedule encountered some errors and is incomplete. Please
 take a look at error report for further details"
    }
}
```

DISABLED 模式的失敗通知訊息

下列範例顯示 QueryInsights 參數DISABLED模式的失敗排程查詢通知訊息。

```
"FailureNotificationMessage": {
    "type": "MANUAL_TRIGGER_FAILURE",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-
b261670d-790c-4116-9db5-0798071b18b1-b7e27a1d79be226d",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1727915194,
```

```
        "triggerTimeMillis": 1727915195119,
        "runStatus": "MANUAL_TRIGGER_FAILURE",
        "executionStats": {
            "executionTimeInMillis": 10777,
            "dataWrites": 0,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 0,
            "recordsIngested": 0,
            "queryResultRows": 4
        },
        "errorReportLocation": {
            "s3ReportLocation": {
                "bucketName": "my-amzn-s3-demo-bucket",
                "objectKey": "4my-organization-b7e27a1d79be226d/1727915194/
MANUAL/1727915195119/08dea9f5-9a0a-4e63-a5f7-ded23247bb98"
            }
        },
        "failureReason": "Schedule encountered some errors and is incomplete. Please
 take a look at error report for further details"
    }
}
```

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

ListScheduledQueries
服務：Amazon Timestream Query

取得呼叫者 Amazon 帳戶和區域中所有排程查詢的清單。 `ListScheduledQueries` 最終一致。

請求語法

```
{
   "MaxResults": number,
   "NextToken": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

MaxResults

在輸出中傳回的項目數量上限。如果可用的項目總數超過指定的值，NextToken則會在輸出中提供 。若要繼續分頁，請提供 NextToken值作為對 的後續呼叫的引數`ListScheduledQueriesRequest`。

類型：整數

有效範圍：最小值為 1。最大值為 1000。

必要：否

NextToken

用於恢復分頁的分頁權杖。

類型：字串

必要：否

回應語法

```
{
```

```
   "NextToken": "string",
   "ScheduledQueries": [
      {
         "Arn": "string",
         "CreationTime": number,
         "ErrorReportConfiguration": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "ObjectKeyPrefix": "string"
            }
         },
         "LastRunStatus": "string",
         "Name": "string",
         "NextInvocationTime": number,
         "PreviousInvocationTime": number,
         "State": "string",
         "TargetDestination": {
            "TimestreamDestination": {
               "DatabaseName": "string",
               "TableName": "string"
            }
         }
      }
   ]
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

NextToken

用以指定分頁開始位置的字符。這是先前截斷的回應 NextToken 中的 。

類型：字串

ScheduledQueries

排程查詢的清單。

類型：ScheduledQuery 物件陣列

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，服務無法完整處理此請求。

HTTP 狀態碼：400

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2

- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

ListTagsForResource
服務：Amazon Timestream Query

列出 Timestream 查詢資源上的所有標籤。

請求語法

```
{
   "MaxResults": number,
   "NextToken": "string",
   "ResourceARN": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

MaxResults

要傳回的標籤數目上限。

類型：整數

有效範圍：最小值為 1。最大值為 200。

必要：否

NextToken

用於恢復分頁的分頁權杖。

類型：字串

必要：否

ResourceARN

要列出標籤的 Timestream 資源。此值是 Amazon Resource Name （ ARN ）。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：是

回應語法

```
{
   "NextToken": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

NextToken

分頁權杖，用於在後續呼叫 時繼續分頁ListTagsForResourceResponse。

類型：字串

Tags

目前與 Timestream 資源相關聯的標籤。

類型：Tag 物件陣列

陣列成員：項目數下限為 0。項目數上限為 200。

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

找不到請求的資源。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

PrepareQuery
服務：Amazon Timestream Query

同步操作，可讓您提交查詢，其中包含由 Timestream 存放供稍後執行的參數。Timestream 僅支援將 `ValidateOnly` 設定為 的此操作`true`。

請求語法

```
{
   "QueryString": "string",
   "ValidateOnly": boolean
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

QueryString

您要用作準備陳述式的 Timestream 查詢字串。參數名稱可以在後跟一個識別碼的查詢字串 @ 字元中指定。

類型：字串

長度限制：長度下限為 1。長度上限為 262144。

必要：是

ValidateOnly

透過將此值設定為 `true`，Timestream 只會驗證查詢字串是否為有效的 Timestream 查詢，而不會儲存準備好的查詢以供日後使用。

類型：布林值

必要：否

回應語法

```
{
   "Columns": [
      {
```

```
         "Aliased": boolean,
         "DatabaseName": "string",
         "Name": "string",
         "TableName": "string",
         "Type": {
            "ArrayColumnInfo": {
               "Name": "string",
               "Type": "Type"
            },
            "RowColumnInfo": [
               {
                  "Name": "string",
                  "Type": "Type"
               }
            ],
            "ScalarType": "string",
            "TimeSeriesMeasureValueColumnInfo": {
               "Name": "string",
               "Type": "Type"
            }
         }
      }
   ],
   "Parameters": [
      {
         "Name": "string",
         "Type": {
            "ArrayColumnInfo": {
               "Name": "string",
               "Type": "Type"
            },
            "RowColumnInfo": [
               {
                  "Name": "string",
                  "Type": "Type"
               }
            ],
            "ScalarType": "string",
            "TimeSeriesMeasureValueColumnInfo": {
               "Name": "string",
               "Type": "Type"
            }
         }
      }
```

```
    ],
    "QueryString": "string"
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

Columns

已提交查詢字串的SELECT子句資料欄清單。

類型：SelectColumn 物件陣列

Parameters

已提交查詢字串中使用的參數清單。

類型：ParameterMapping 物件陣列

QueryString

您要準備的查詢字串。

類型：字串

長度限制：長度下限為 1。長度上限為 262144。

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，服務無法完整處理此請求。

HTTP 狀態碼：400

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

Query
服務：Amazon Timestream Query

Query 是一項同步操作，可讓您針對 Amazon Timestream 資料執行查詢。

如果您啟用 QueryInsights，這API也會傳回與您執行的查詢相關的洞察和指標。 QueryInsights 可協助您調校查詢的效能。如需 的詳細資訊QueryInsights，請參閱使用查詢洞察來最佳化 Amazon Timestream 中的查詢。

> **Note**
>
> 您允許在QueryInsights啟用的情況下執行的QueryAPI請求數目上限為每秒 1 個查詢（QPS）。如果您超過此查詢速率，可能會導致限流。

Query 會在 60 秒後逾時。您必須更新 中的預設逾時SDK，才能支援 60 秒的逾時。如需詳細資訊，請參閱程式碼範例。

在下列情況下，您的查詢請求將會失敗：

- 如果您在 5 分鐘不透明度時段之外使用相同的用戶端字符提交Query請求。
- 如果您使用相同的用戶端權杖提交Query請求，但變更其他參數，請在 5 分鐘的無效時段內提交。
- 如果資料列的大小（包括查詢中繼資料）超過 1 MB，則查詢會失敗，並顯示下列錯誤訊息：

  ```
  Query aborted as max page response size has been exceeded by the output result row
  ```

- 如果查詢啟動器和結果讀取器的IAM主體不同和/或查詢啟動器，且結果讀取器在查詢請求中沒有相同的查詢字串，則查詢會失敗並出現Invalid pagination token錯誤。

請求語法

```
{
   "ClientToken": "string",
   "MaxRows": number,
   "NextToken": "string",
   "QueryInsights": {
      "Mode": "string"
   },
   "QueryString": "string"
```

```
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

ClientToken

發出Query請求時，最多指定 64 ASCII 個字元的唯一區分大小寫字串。提供 會ClientToken呼叫 Query 等電位 。這表示重複執行相同的查詢會產生相同的結果。換句話說，發出多個相同的Query請求與發出單一請求具有相同的效果。在查詢ClientToken中使用 時，請注意下列事項：

- 如果查詢API是在沒有 的情況下現化ClientToken，則查詢ClientToken會代表您SDK產生 。
- 如果Query調用僅包含 ClientToken但不包含 NextToken，則 的調用Query會被視為新的查詢執行。
- 如果調用包含 NextToken，該特定調用會假設是先前呼叫查詢 的後續調用API，並傳回結果集。
- 4 小時後，任何具有相同 的請求ClientToken都會視為新的請求。

類型：字串

長度限制：長度下限為 32。長度上限為 128。

必要：否

MaxRows

Query 輸出中要傳回的資料列總數。Query 具有指定MaxRows值的 初始執行會在兩個情況下傳回查詢的結果集：

- 結果的大小小於 1MB。
- 結果集中的資料列數小於 的值maxRows。

否則， 的初始調用Query只會傳回 NextToken，然後可用於後續呼叫，以擷取結果集。若要繼續分頁，請在後續命令中提供 NextToken值。

如果資料列大小較大（例如資料列包含許多資料欄），Timestream 可能會傳回較少的資料列，以避免回應大小超過 1 MB 的限制。如果MaxRows未提供，Timestream 將傳送必要的資料列數目，以符合 1 MB 的限制。

類型：整數

有效範圍：最小值為 1。最大值為 1000。

必要：否

NextToken

用於傳回一組結果的分頁權杖。使用 QueryAPI叫用 時NextToken，會假設該特定叫用是先前呼叫 的後續叫用Query，並傳回結果集。不過，如果Query調用僅包含 ClientToken，則 的調用Query會被視為新的查詢執行。

在查詢 NextToken 中使用 時，請注意下列事項：

- 分頁權杖最多可以用於五個Query調用，或長達 1 小時的持續時間，以先發生者為準。
- 使用相同的 NextToken會傳回相同的記錄集。若要繼續瀏覽結果集，您必須使用最新的 nextToken。
- 假設Query調用會傳回兩個NextToken值，TokenA且 TokenB。如果 TokenB 用於後續Query調用，則 TokenA 會失效，且無法重複使用。
- 若要在分頁開始後從查詢請求上一個結果集，您必須重新叫用查詢 API。
- 最新的 NextToken應該用來分頁，直到傳回null為止，此時NextToken應使用新的 。
- 如果查詢啟動器和結果讀取器的IAM主體不同和/或查詢啟動器，且結果讀取器在查詢請求中沒有相同的查詢字串，則查詢會失敗並出現Invalid pagination token錯誤。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：否

QueryInsights

封裝啟用 的設定QueryInsights。

啟用除了您執行查詢的查詢結果之外，還QueryInsights傳回洞見和指標。您可以使用 QueryInsights來調整查詢效能。

類型：QueryInsights 物件

必要：否

QueryString

Timestream 要執行的查詢。

類型：字串

長度限制：長度下限為 1。長度上限為 262144。

必要：是

回應語法

```
{
   "ColumnInfo": [
      {
         "Name": "string",
         "Type": {
            "ArrayColumnInfo": "ColumnInfo",
            "RowColumnInfo": [
               "ColumnInfo"
            ],
            "ScalarType": "string",
            "TimeSeriesMeasureValueColumnInfo": "ColumnInfo"
         }
      }
   ],
   "NextToken": "string",
   "QueryId": "string",
   "QueryInsightsResponse": {
      "OutputBytes": number,
      "OutputRows": number,
      "QuerySpatialCoverage": {
         "Max": {
            "PartitionKey": [ "string" ],
            "TableArn": "string",
            "Value": number
         }
      },
      "QueryTableCount": number,
      "QueryTemporalRange": {
         "Max": {
            "TableArn": "string",
            "Value": number
         }
      },
      "UnloadPartitionCount": number,
      "UnloadWrittenBytes": number,
```

```
      "UnloadWrittenRows": number
   },
   "QueryStatus": {
      "CumulativeBytesMetered": number,
      "CumulativeBytesScanned": number,
      "ProgressPercentage": number
   },
   "Rows": [
      {
         "Data": [
            {
               "ArrayValue": [
                  "Datum"
               ],
               "NullValue": boolean,
               "RowValue": "Row",
               "ScalarValue": "string",
               "TimeSeriesValue": [
                  {
                     "Time": "string",
                     "Value": "Datum"
                  }
               ]
            }
         ]
      }
   ]
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

ColumnInfo

傳回結果集的資料欄資料類型。

類型：ColumnInfo 物件陣列

NextToken

分頁權杖，可在Query通話中再次使用，以取得下一組結果。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

QueryId

指定查詢的唯一 ID。

類型：字串

長度限制：長度下限為 1。長度上限為 64。

模式：[a-zA-Z0-9]+

QueryInsightsResponse

封裝QueryInsights包含與您執行之查詢相關的洞察和指標。

類型：QueryInsightsResponse 物件

QueryStatus

查詢狀態的相關資訊，包括進度和掃描的位元組。

類型：QueryStatus 物件

Rows

查詢傳回的結果集資料列。

類型：Row 物件陣列

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

ConflictException

無法輪詢已取消查詢的結果。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，服務無法完整處理此請求。

HTTP 狀態碼：400

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

QueryExecutionException

Timestream 無法成功執行查詢。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python

- AWS SDK 適用於 Ruby V3

TagResource
服務：Amazon Timestream Query

將一組標籤與 Timestream 資源建立關聯。然後，您可以啟用這些使用者定義的標籤，使其出現在 Billing and Cost Management 主控台上，以進行成本分配追蹤。

請求語法

```
{
   "ResourceARN": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

ResourceARN

識別應新增標籤的 Timestream 資源。此值是 Amazon Resource Name （ ARN ）。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：是

Tags

要指派給 Timestream 資源的標籤。

類型：Tag 物件陣列

陣列成員：項目數下限為 0。項目數上限為 200。

必要：是

回應元素

如果動作成功，服務會傳回 200 HTTP 回應，其中內HTTP文為空白。

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

找不到請求的資源。

HTTP 狀態碼：400

ServiceQuotaExceededException

您已超過服務配額。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++

- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

UntagResource
服務：Amazon Timestream Query

從 Timestream 查詢資源中移除標籤的關聯。

請求語法

```
{
   "ResourceARN": "string",
   "TagKeys": [ "string" ]
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

ResourceARN

標籤將從中移除的 Timestream 資源。此值是 Amazon Resource Name （ ARN ）。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：是

TagKeys

標籤索引鍵清單。資源的現有標籤，其金鑰是此清單的成員，將會從 Timestream 資源中移除。

類型：字串陣列

陣列成員：項目數下限為 0。項目數上限為 200。

長度限制：長度下限為 1。長度上限為 128。

必要：是

回應元素

如果動作成功，服務會傳回 200 HTTP 回應，其中內HTTP文為空白。

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

找不到請求的資源。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

UpdateAccountSettings
服務：Amazon Timestream Query

轉換您的帳戶以TCUs用於查詢定價，並修改您設定的最大查詢運算單位。如果您將 的值減少MaxQueryTCU為所需的組態，則新值可能需要最多 24 小時才能生效。

> **Note**
>
> 將帳戶轉換為TCUs用於查詢定價之後，您無法轉換為使用掃描查詢定價的位元組。

請求語法

```
{
   "MaxQueryTCU": number,
   "QueryPricingModel": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

MaxQueryTCU

服務在任何時間點用來服務查詢的運算單位數量上限。若要執行查詢，您必須設定最小容量為 4 TCU。您可以設定 4 的TCU倍數上限，例如 4、8、16、32 等。

支援的最大值MaxQueryTCU為 1000。若要請求提高此軟限制，請聯絡 AWS 支援。如需 的預設配額的相關資訊maxQueryTCU，請參閱預設配額 。

類型：整數

必要：否

QueryPricingModel

帳戶中查詢的定價模型。

> **Note**
>
> QueryPricingModel 參數由數個 Timestream 操作使用；但UpdateAccountSettingsAPI操作無法識別 以外的任何值COMPUTE_UNITS。

類型：字串

有效值:BYTES_SCANNED | COMPUTE_UNITS

必要：否

回應語法

```
{
   "MaxQueryTCU": number,
   "QueryPricingModel": "string"
}
```

回應元素

如果動作成功，服務會傳回 200 HTTP 個回應。

服務會以 JSON 格式傳回下列資料。

MaxQueryTCU

服務在任何時間點用來服務查詢的已設定運算單位數量上限。

類型：整數

QueryPricingModel

帳戶的定價模型。

類型：字串

有效值:BYTES_SCANNED | COMPUTE_UNITS

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，服務無法完整處理此請求。

HTTP 狀態碼：400

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2
- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python

- AWS SDK 適用於 Ruby V3

UpdateScheduledQuery
服務：Amazon Timestream Query

更新排程查詢。

請求語法

```
{
   "ScheduledQueryArn": "string",
   "State": "string"
}
```

請求參數

如需所有動作的一般參數資訊，請參閱《Common Parameters》。

請求接受下列JSON格式的資料。

ScheduledQueryArn

ARN 排程查詢。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：是

State

排程查詢的狀態。

類型：字串

有效值:ENABLED | DISABLED

必要：是

回應元素

如果動作成功，服務會傳回 200 HTTP 回應，其中內HTTP文為空白。

錯誤

如需所有動作常見錯誤的資訊，請參閱常見錯誤。

AccessDeniedException

您無權執行此動作。

HTTP 狀態碼：400

InternalServerException

由於內部伺服器錯誤，服務無法完整處理此請求。

HTTP 狀態碼：400

InvalidEndpointException

請求的端點無效。

HTTP 狀態碼：400

ResourceNotFoundException

找不到請求的資源。

HTTP 狀態碼：400

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationException

無效或格式不正確的請求。

HTTP 狀態碼：400

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS 命令列介面
- AWS SDK 適用於 。NET
- AWS SDK 適用於 C++
- AWS SDK for Go v2

- AWS SDK 適用於 Java V2
- AWS SDK for JavaScript V3
- AWS SDK 適用於 PHP V3
- AWS SDK 適用於 Python
- AWS SDK 適用於 Ruby V3

資料類型

Amazon Timestream Write 支援下列資料類型：

- BatchLoadProgressReport
- BatchLoadTask
- BatchLoadTaskDescription
- CsvConfiguration
- Database
- DataModel
- DataModelConfiguration
- DataModelS3Configuration
- DataSourceConfiguration
- DataSourceS3Configuration
- Dimension
- DimensionMapping
- Endpoint
- MagneticStoreRejectedDataLocation
- MagneticStoreWriteProperties
- MeasureValue
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- PartitionKey
- Record
- RecordsIngested

- RejectedRecord
- ReportConfiguration
- ReportS3Configuration
- RetentionProperties
- S3Configuration
- Schema
- Table
- Tag

Amazon Timestream Query 支援下列資料類型：

- ColumnInfo
- Datum
- DimensionMapping
- Endpoint
- ErrorReportConfiguration
- ErrorReportLocation
- ExecutionStats
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- NotificationConfiguration
- ParameterMapping
- QueryInsights
- QueryInsightsResponse
- QuerySpatialCoverage
- QuerySpatialCoverageMax
- QueryStatus
- QueryTemporalRange
- QueryTemporalRangeMax
- Row

- S3Configuration
- S3ReportLocation
- ScheduleConfiguration
- ScheduledQuery
- ScheduledQueryDescription
- ScheduledQueryInsights
- ScheduledQueryInsightsResponse
- ScheduledQueryRunSummary
- SelectColumn
- SnsConfiguration
- Tag
- TargetConfiguration
- TargetDestination
- TimeSeriesDataPoint
- TimestreamConfiguration
- TimestreamDestination
- Type

Amazon Timestream 寫入

Amazon Timestream Write 支援下列資料類型：

- BatchLoadProgressReport
- BatchLoadTask
- BatchLoadTaskDescription
- CsvConfiguration
- Database
- DataModel
- DataModelConfiguration
- DataModelS3Configuration
- DataSourceConfiguration
- DataSourceS3Configuration

- Dimension
- DimensionMapping
- Endpoint
- MagneticStoreRejectedDataLocation
- MagneticStoreWriteProperties
- MeasureValue
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- PartitionKey
- Record
- RecordsIngested
- RejectedRecord
- ReportConfiguration
- ReportS3Configuration
- RetentionProperties
- S3Configuration
- Schema
- Table
- Tag

BatchLoadProgressReport
服務：Amazon Timestream Write

批次載入任務進度的詳細資訊。

目錄

BytesMetered

類型：Long

必要：否

FileFailures

類型：Long

必要：否

ParseFailures

類型：Long

必要：否

RecordIngestionFailures

類型：Long

必要：否

RecordsIngested

類型：Long

必要：否

RecordsProcessed

類型：Long

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

BatchLoadTask
服務：Amazon Timestream Write

批次載入任務的詳細資訊。

目錄

CreationTime

建立 Timestream 批次載入任務的時間。

類型：Timestamp

必要：否

DatabaseName

批次載入任務載入資料的資料庫名稱。

類型：字串

必要：否

LastUpdatedTime

Timestream 批次載入任務上次更新的時間。

類型：Timestamp

必要：否

ResumableUntil

類型：Timestamp

必要：否

TableName

批次載入任務載入資料的資料表名稱。

類型：字串

必要：否

TaskId

批次載入任務的 ID。

類型：字串

長度限制：長度下限為 3。長度上限為 32。

模式：[A-Z0-9]+

必要：否

TaskStatus

批次載入任務的狀態。

類型：字串

有效值:CREATED | IN_PROGRESS | FAILED | SUCCEEDED | PROGRESS_STOPPED | PENDING_RESUME

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

BatchLoadTaskDescription
服務：Amazon Timestream Write

批次載入任務的詳細資訊。

目錄

CreationTime

建立 Timestream 批次載入任務的時間。

類型：Timestamp

必要：否

DataModelConfiguration

批次載入任務的資料模型組態。這包含批次載入任務資料模型存放位置的詳細資訊。

類型：DataModelConfiguration 物件

必要：否

DataSourceConfiguration

批次載入任務的資料來源組態詳細資訊。

類型：DataSourceConfiguration 物件

必要：否

ErrorMessage

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：否

LastUpdatedTime

Timestream 批次載入任務上次更新的時間。

類型：Timestamp

必要：否

ProgressReport

類型：BatchLoadProgressReport 物件

必要：否

RecordVersion

類型：Long

必要：否

ReportConfiguration

批次載入任務的報告組態。這包含錯誤報告存放位置的詳細資訊。

類型：ReportConfiguration 物件

必要：否

ResumableUntil

類型：Timestamp

必要：否

TargetDatabaseName

類型：字串

必要：否

TargetTableName

類型：字串

必要：否

TaskId

批次載入任務的 ID。

類型：字串

長度限制：長度下限為 3。長度上限為 32。

模式：[A-Z0-9]+

必要：否

TaskStatus

批次載入任務的狀態。

類型：字串

有效值:CREATED | IN_PROGRESS | FAILED | SUCCEEDED | PROGRESS_STOPPED | PENDING_RESUME

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

CsvConfiguration
服務：Amazon Timestream Write

分隔的資料格式，其中資料欄分隔符號可以是逗號，而記錄分隔符號是新的行字元。

目錄

ColumnSeparator

資料欄分隔符號可以是逗號（' , '）、管道（'|）、分號（' ; '）、 tab（'/t'）或空格（' '）。

類型：字串

長度限制條件：固定長度為 1。

必要：否

EscapeChar

逸出字元可以是其中之一

類型：字串

長度限制條件：固定長度為 1。

必要：否

NullValue

可以是空格（' '）。

類型：字串

長度限制：長度下限為 1。長度上限為 256。

必要：否

QuoteChar

可以是單引號（'）或雙引號（"）。

類型：字串

長度限制條件：固定長度為 1。

必要：否

TrimWhiteSpace

指定修剪前後的空格。

類型：布林值

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

Database
服務：Amazon Timestream Write

資料表的頂層容器。資料庫和資料表是 Amazon Timestream 中的基本管理概念。資料庫中的所有資料表都會使用相同的 AWS KMS 金鑰加密。

目錄

Arn

唯一識別此資料庫的 Amazon Resource Name。

類型：字串

必要：否

CreationTime

建立資料庫的時間，從 Unix epoch 時間計算。

類型：Timestamp

必要：否

DatabaseName

Timestream 資料庫的名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：否

KmsKeyId

用於加密資料庫中儲存之資料的 AWS KMS 金鑰識別碼。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：否

LastUpdatedTime

此資料庫上次更新的時間。

類型：Timestamp

必要：否

TableCount

Timestream 資料庫中找到的資料表總數。

類型：Long

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

DataModel

服務：Amazon Timestream Write

批次載入任務的資料模型。

目錄

DimensionMappings

維度的來源至目標映射。

類型：DimensionMapping 物件陣列

陣列成員：項目數下限為 1。

必要：是

MeasureNameColumn

類型：字串

長度限制：長度下限為 1。長度上限為 256。

必要：否

MixedMeasureMappings

量值的來源至目標映射。

類型：MixedMeasureMapping 物件陣列

陣列成員：項目數下限為 1。

必要：否

MultiMeasureMappings

多測量記錄的來源至目標映射。

類型：MultiMeasureMappings 物件

必要：否

TimeColumn

要映射到時間的來源欄。

類型：字串

長度限制：長度下限為 1。長度上限為 256。

必要：否

TimeUnit

時間戳記單位的精細度。它指出時間值是秒、毫秒、奈秒或其他支援的值。預設值為 `MILLISECONDS`。

類型：字串

有效值:`MILLISECONDS | SECONDS | MICROSECONDS | NANOSECONDS`

必要：否

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

DataModelConfiguration
服務：Amazon Timestream Write

目錄

DataModel

類型：DataModel 物件

必要：否

DataModelS3Configuration

類型：DataModelS3Configuration 物件

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

DataModelS3Configuration
服務：Amazon Timestream Write

目錄

BucketName

類型：字串

長度限制：長度下限為 3。長度上限為 63。

模式：[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]

必要：否

ObjectKey

類型：字串

長度限制：長度下限為 1。長度上限為 1024。

模式：[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

DataSourceConfiguration
服務：Amazon Timestream Write

定義資料來源的組態詳細資訊。

目錄

DataFormat

這是目前 CSV。

類型：字串

有效值:CSV

必要：是

DataSourceS3Configuration

包含要載入資料之檔案的 S3 位置組態。

類型：DataSourceS3Configuration 物件

必要：是

CsvConfiguration

分隔的資料格式，其中資料欄分隔符號可以是逗號，而記錄分隔符號是新行字元。

類型：CsvConfiguration 物件

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

DataSourceS3Configuration
服務：Amazon Timestream Write

目錄

BucketName

客戶 S3 儲存貯體的 儲存貯體名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 63。

模式：`[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

必要：是

ObjectKeyPrefix

類型：字串

長度限制：長度下限為 1。長度上限為 1024。

模式：`[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

Dimension
服務：Amazon Timestream Write

代表時間序列的中繼資料屬性。例如，EC2執行個體的名稱和可用區域或風力發電機製造商的名稱是維度。

目錄

Name

Dimension 代表時間序列的中繼資料屬性。例如，EC2執行個體的名稱和可用區域或風力發電機製造商的名稱是維度。

如需維度名稱的限制，請參閱命名限制 。

類型：字串

長度限制：長度下限為 1。長度上限為 60。

必要：是

Value

維度值。

類型：字串

必要：是

DimensionValueType

時間序列資料點維度的資料類型。

類型：字串

有效值:VARCHAR

必要：否

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++

- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

DimensionMapping
服務：Amazon Timestream Write

目錄

DestinationColumn

類型：字串

長度限制：長度下限為 1。

必要：否

SourceColumn

類型：字串

長度限制：長度下限為 1。

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

Endpoint
服務：Amazon Timestream Write

表示要為其進行API呼叫的可用端點，以及該端點TTL的 。

目錄

Address

端點地址。

類型：字串

必要：是

CachePeriodInMinutes

端點TTL的 ，以分鐘為單位。

類型：Long

必要：是

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

MagneticStoreRejectedDataLocation
服務：Amazon Timestream Write

在磁性存放區寫入期間，以非同步方式寫入遭拒絕記錄之錯誤報告的位置。

目錄

S3Configuration

在磁性存放區寫入期間，以非同步方式寫入遭拒絕記錄之錯誤報告的 S3 位置組態。

類型：S3Configuration 物件

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

MagneticStoreWriteProperties
服務：Amazon Timestream Write

用於設定磁性存放區寫入的資料表上的屬性集。

目錄

EnableMagneticStoreWrites

勇於啟用磁性存放區寫入的標記。

類型：布林值

必要：是

MagneticStoreRejectedDataLocation

在磁性存放區寫入期間，以非同步方式寫入遭拒絕記錄之錯誤報告的位置。

類型：MagneticStoreRejectedDataLocation 物件

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

MeasureValue
服務：Amazon Timestream Write

代表時間序列的資料屬性。例如，EC2執行個體或風力發電機RPM的 CPU 使用率是量值。MeasureValue 具有名稱和值。

MeasureValue 僅適用於類型 MULTI。使用 MULTI 類型，您可以在單一記錄中傳遞與相同時間序列相關聯的多個資料屬性

目錄

Name

的名稱 MeasureValue。

如需 MeasureValue 名稱的限制，請參閱 Amazon Timestream 開發人員指南中的命名限制。

類型：字串

長度限制：長度下限為 1。

必要：是

Type

包含時間序列資料點 MeasureValue 的 資料類型。

類型：字串

有效值:DOUBLE | BIGINT | VARCHAR | BOOLEAN | TIMESTAMP | MULTI

必要：是

Value

的值 MeasureValue。如需詳細資訊，請參閱資料類型 。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：是

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

MixedMeasureMapping
服務：Amazon Timestream Write

目錄

MeasureValueType

類型：字串

有效值:DOUBLE | BIGINT | VARCHAR | BOOLEAN | TIMESTAMP | MULTI

必要：是

MeasureName

類型：字串

長度限制：長度下限為 1。

必要：否

MultiMeasureAttributeMappings

類型：MultiMeasureAttributeMapping 物件陣列

陣列成員：項目數下限為 1。

必要：否

SourceColumn

類型：字串

長度限制：長度下限為 1。

必要：否

TargetMeasureName

類型：字串

長度限制：長度下限為 1。

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

MultiMeasureAttributeMapping
服務：Amazon Timestream Write

目錄

SourceColumn

類型：字串

長度限制：長度下限為 1。

必要：是

MeasureValueType

類型：字串

有效值:DOUBLE | BIGINT | BOOLEAN | VARCHAR | TIMESTAMP

必要：否

TargetMultiMeasureAttributeName

類型：字串

長度限制：長度下限為 1。

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

MultiMeasureMappings
服務：Amazon Timestream Write

目錄

MultiMeasureAttributeMappings

類型：MultiMeasureAttributeMapping 物件陣列

陣列成員：項目數下限為 1。

必要：是

TargetMultiMeasureName

類型：字串

長度限制：長度下限為 1。

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

PartitionKey
服務：Amazon Timestream Write

用於分割資料表中資料的屬性。維度索引鍵使用維度名稱指定為分割區索引鍵的維度值來分割資料，而使用量值名稱（'measure_name' 欄的值）來測量索引鍵分割區資料。

目錄

Type

分割區金鑰的類型。選項為 DIMENSION（尺寸索引鍵）和 MEASURE（測量索引鍵）。

類型：字串

有效值:`DIMENSION | MEASURE`

必要：是

EnforcementInRecord

擷取記錄中維度索引鍵規格的強制執行層級。選項為 REQUIRED（必須指定尺寸索引鍵）和 OPTIONAL（不需要指定尺寸索引鍵）。

類型：字串

有效值:`REQUIRED | OPTIONAL`

必要：否

Name

用於維度索引鍵的屬性名稱。

類型：字串

長度限制：長度下限為 1。

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++

- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

Record
服務：Amazon Timestream Write

代表寫入 Timestream 的時間序列資料點。每個記錄都包含維度陣列。維度代表時間序列資料點的中繼資料屬性，例如執行個體名稱或EC2執行個體的可用區域。記錄也包含量值名稱，也就是收集的量值名稱（例如，EC2執行個體的使用CPU率）。此外，記錄包含測量值和值類型，這是測量值的資料類型。此外，記錄包含收集量值時的時間戳記，以及時間戳記單位，其代表時間戳記的精細度。

記錄有一個`Version`欄位，這是 64 位元`long`，可用來更新資料點。只有在寫入請求中記錄的`Version`屬性高於現有記錄時，才會寫入具有相同維度、時間戳記和量值名稱但不同量值的重複記錄。對於沒有 `Version` 欄位的記錄，時間串流預設為 `Version` 1的 。

目錄

Dimensions

包含時間序列資料點的維度清單。

類型：Dimension 物件陣列

陣列成員：最多 128 個項目。

必要：否

MeasureName

Measure 代表時間序列的資料屬性。例如，EC2執行個體的CPU利用率或風力發電機RPM的 是指標。

類型：字串

長度限制：長度下限為 1。長度上限為 256。

必要：否

MeasureValue

包含時間序列資料點的測量值。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：否

MeasureValues

包含時間序列資料點 MeasureValue 的 清單。

這僅適用於類型 MULTI。對於純量值，請直接使用記錄的MeasureValue屬性。

類型：MeasureValue 物件陣列

必要：否

MeasureValueType

包含時間序列資料點的測量值資料類型。預設類型為 DOUBLE。如需詳細資訊，請參閱資料類型 。

類型：字串

有效值:DOUBLE | BIGINT | VARCHAR | BOOLEAN | TIMESTAMP | MULTI

必要：否

Time

包含收集資料點的測量值的時間。時間值加上 單位提供自 epoch 以來經過的時間。例如，如果時間值為 12345且單位為 ms，則自 epoch 以來12345 ms已經過。

類型：字串

長度限制：長度下限為 1。長度上限為 256。

必要：否

TimeUnit

時間戳記單位的精細度。它指出時間值是秒、毫秒、奈秒或其他支援的值。預設值為 MILLISECONDS。

類型：字串

有效值:MILLISECONDS | SECONDS | MICROSECONDS | NANOSECONDS

必要：否

Version

用於記錄更新的 64 位元屬性。寫入版本編號較高的重複資料請求，將會更新現有的量值和版本。如果測量值相同， Version 仍會更新。預設值為 1。

> **Note**
>
> `Version` 必須1大於 ，否則您會收到`ValidationException`錯誤。

類型：Long

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

RecordsIngested
服務：Amazon Timestream Write

此請求擷取的記錄資訊。

目錄

MagneticStore

擷取至磁性存放區的記錄計數。

類型：整數

必要：否

MemoryStore

擷取至記憶體存放區的記錄計數。

類型：整數

必要：否

Total

成功擷取的記錄總數。

類型：整數

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

RejectedRecord
服務：Amazon Timestream Write

表示因資料驗證問題而未成功插入 Timestream 的記錄，這些問題必須在重新將時間序列資料插入系統之前解決。

目錄

ExistingVersion

記錄的現有版本。此值會在相同記錄存在且版本高於寫入請求中的版本的情況下填入。

類型：Long

必要：否

Reason

記錄未成功插入 Timestream 的原因。失敗的可能原因包括：

- 具有重複資料的記錄，其中有多個具有相同維度、時間戳記和量值名稱的記錄，但：
  - 測量值不同
  - 請求中沒有版本，或新記錄中的版本值等於或低於現有值

  如果 Timestream 拒絕此案例的資料，`RejectedRecords`回應中的`ExistingVersion`欄位會顯示目前記錄的版本。若要強制更新，您可以將記錄集的 版本重新傳送請求至大於 的值`ExistingVersion`。
- 時間戳記超出記憶體存放區保留期間的記錄。

  > **Note**
  >
  > 更新保留時段時，如果您立即嘗試在新視窗中擷取資料，則會收到`RejectedRecords`例外狀況。若要避免`RejectedRecords`例外狀況，請等待新視窗的持續時間擷取新資料。如需詳細資訊，請參閱設定 Timestream 的最佳實務，以及 Timestream 中儲存運作方式的說明。
- 維度或量值超過 Timestream 定義限制的記錄。

如需詳細資訊，請參閱《Timestream 開發人員指南》中的存取管理。

類型：字串

必要：否

RecordIndex

輸入請求中的記錄索引 WriteRecords。索引以 0 開頭。

類型：整數

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

ReportConfiguration
服務：Amazon Timestream Write

批次載入任務的報告組態。這包含錯誤報告存放位置的詳細資訊。

目錄

ReportS3Configuration

設定 S3 位置，以寫入批次載入的錯誤報告和事件。

類型：ReportS3Configuration 物件

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

ReportS3Configuration
服務：Amazon Timestream Write

目錄

BucketName

類型：字串

長度限制：長度下限為 3。長度上限為 63。

模式：[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]

必要：是

EncryptionOption

類型：字串

有效值:SSE_S3 | SSE_KMS

必要：否

KmsKeyId

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：否

ObjectKeyPrefix

類型：字串

長度限制：長度下限為 1。長度上限為 928。

模式：[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

RetentionProperties
服務：Amazon Timestream Write

保留屬性包含時間序列資料必須儲存在磁性存放區和記憶體存放區中的持續時間。

目錄

MagneticStoreRetentionPeriodInDays

資料必須儲存在磁性存放區中的持續時間。

類型：Long

有效範圍：最小值為 1。最大值為 73000。

必要：是

MemoryStoreRetentionPeriodInHours

資料必須儲存在記憶體存放區中的持續時間。

類型：Long

有效範圍：最小值為 1。最大值為 8766。

必要：是

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

S3Configuration
服務：Amazon Timestream Write

指定 S3 位置的組態。

目錄

BucketName

客戶 S3 儲存貯體的 儲存貯體名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 63。

模式：[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]

必要：否

EncryptionOption

客戶 S3 位置的加密選項。選項是使用 S3 受管金鑰或 AWS 受管金鑰進行 S3 伺服器端加密。

類型：字串

有效值:SSE_S3 | SSE_KMS

必要：否

KmsKeyId

使用 AWS 受管 AWS KMS 金鑰加密時，客戶 S3 位置的金鑰 ID。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：否

ObjectKeyPrefix

客戶 S3 位置的物件金鑰預覽。

類型：字串

長度限制：長度下限為 1。長度上限為 928。

模式：`[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

Schema

服務：Amazon Timestream Write

結構描述指定資料表的預期資料模型。

目錄

CompositePartitionKey

非空白的分割區索引鍵清單，定義用於分割資料表資料的屬性。清單的順序決定分割區階層。建立資料表後，無法變更每個分割區金鑰的名稱和類型，以及分割區金鑰順序。不過，可以變更每個分割區金鑰的強制執行層級。

類型：PartitionKey 物件陣列

陣列成員：項目數下限為 1。

必要：否

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

Table
服務：Amazon Timestream Write

代表 Timestream 中的資料庫資料表。資料表包含一或多個相關的時間序列。您可以修改資料表的記憶體存放區和磁性存放區的保留期間。

目錄

Arn

唯一識別此資料表的 Amazon Resource Name。

類型：字串

必要：否

CreationTime

建立 Timestream 資料表的時間。

類型：Timestamp

必要：否

DatabaseName

包含此資料表的 Timestream 資料庫名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：否

LastUpdatedTime

Timestream 資料表上次更新的時間。

類型：Timestamp

必要：否

MagneticStoreWriteProperties

包含啟用磁性存放區寫入時要在表格上設定的屬性。

類型：MagneticStoreWriteProperties 物件

必要：否

RetentionProperties

記憶體存放區和磁帶存放區的保留期間。

類型：RetentionProperties 物件

必要：否

Schema

資料表的結構描述。

類型：Schema 物件

必要：否

TableName

Timestream 資料表的名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 256。

必要：否

TableStatus

資料表的目前狀態：

- DELETING - 正在刪除資料表。
- ACTIVE - 資料表已就緒可供使用。

類型：字串

有效值:ACTIVE | DELETING | RESTORING

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

Tag
服務：Amazon Timestream Write

標籤是您指派給 Timestream 資料庫and/or table. Each tag consists of a key and an optional value, both of which you define. With tags, you can categorize databases and/or資料表的標籤，例如依用途、擁有者或環境。

目錄

Key

標籤鍵。標籤鍵會區分大小寫。

類型：字串

長度限制：長度下限為 1。長度上限為 128。

必要：是

Value

標籤的值。標籤值區分大小寫，可以是 null。

類型：字串

長度限制：長度下限為 0。長度上限為 256。

必要：是

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

Amazon Timestream 查詢

Amazon Timestream Query 支援下列資料類型：

- ColumnInfo

- Datum
- DimensionMapping
- Endpoint
- ErrorReportConfiguration
- ErrorReportLocation
- ExecutionStats
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- NotificationConfiguration
- ParameterMapping
- QueryInsights
- QueryInsightsResponse
- QuerySpatialCoverage
- QuerySpatialCoverageMax
- QueryStatus
- QueryTemporalRange
- QueryTemporalRangeMax
- Row
- S3Configuration
- S3ReportLocation
- ScheduleConfiguration
- ScheduledQuery
- ScheduledQueryDescription
- ScheduledQueryInsights
- ScheduledQueryInsightsResponse
- ScheduledQueryRunSummary
- SelectColumn
- SnsConfiguration
- Tag

- TargetConfiguration
- TargetDestination
- TimeSeriesDataPoint
- TimestreamConfiguration
- TimestreamDestination
- Type

ColumnInfo

服務：Amazon Timestream Query

包含查詢結果的中繼資料，例如資料欄名稱、資料類型和其他屬性。

目錄

Type

結果集資料欄的資料類型。資料類型可以是純量或複雜值。純量資料類型是整數、字串、雙數、布林值和其他。複雜的資料類型是陣列、資料列等類型。

類型：Type 物件

必要：是

Name

結果集資料欄的名稱。結果集的名稱適用於所有資料類型的資料欄，陣列除外。

類型：字串

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

Datum

服務：Amazon Timestream Query

Datum 代表查詢結果中的單一資料點。

目錄

ArrayValue

指示資料點是否為陣列。

類型：Datum 物件陣列

必要：否

NullValue

指示資料點是否為 null。

類型：布林值

必要：否

RowValue

指示資料點是否為資料列。

類型：Row 物件

必要：否

ScalarValue

指示資料點是否為純量值，例如整數、字串、雙值或布林值。

類型：字串

必要：否

TimeSeriesValue

指示資料點是否為 Timeeries 資料類型。

類型：TimeSeriesDataPoint 物件陣列

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

DimensionMapping
服務：Amazon Timestream Query

這種類型用於將查詢結果中的資料行映射至目的地資料表中的維度。

目錄

DimensionValueType

維度的類型。

類型：字串

有效值:VARCHAR

必要：是

Name

查詢結果中資料行的名稱。

類型：字串

必要：是

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

Endpoint
服務：Amazon Timestream Query

表示要為其進行API呼叫的可用端點，以及該端點TTL的 。

目錄

Address

端點地址。

類型：字串

必要：是

CachePeriodInMinutes

端點TTL的 ，以分鐘為單位。

類型：Long

必要：是

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

ErrorReportConfiguration
服務：Amazon Timestream Query

錯誤報告所需的組態。

目錄

S3Configuration

錯誤報告的 S3 組態。

類型：S3Configuration 物件

必要：是

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

ErrorReportLocation
服務：Amazon Timestream Query

這包含單一排程查詢呼叫的錯誤報告位置。

目錄

S3ReportLocation

寫入錯誤報告的 S3 位置。

類型：S3ReportLocation 物件

必要：否

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

ExecutionStats

服務：Amazon Timestream Query

單一排程查詢執行的統計資料。

目錄

BytesMetered

為單一排程查詢執行計量的位元組。

類型：Long

必要：否

CumulativeBytesScanned

掃描單一排程查詢執行的位元組。

類型：Long

必要：否

DataWrites

資料會針對單一排程查詢執行中擷取的記錄進行計量。

類型：Long

必要：否

ExecutionTimeInMillis

排程查詢執行完成所需的總時間，以毫秒為單位。

類型：Long

必要：否

QueryResultRows

在擷取目的地資料來源之前，來自執行查詢的輸出中存在的資料列數。

類型：Long

必要：否

RecordsIngested

針對單一排程查詢執行擷取的記錄數目。

類型：Long

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

MixedMeasureMapping
服務：Amazon Timestream Query

MixedMeasureMappings 是可用於將資料擷取到衍生資料表中狹窄和多重量值混合的映射。

目錄

MeasureValueType

要從 讀取的值類型sourceColumn。如果映射適用於 MULTI，請使用 MeasureValueTypeMULTI。

類型：字串

有效值:`BIGINT | BOOLEAN | DOUBLE | VARCHAR | MULTI`

必要：是

MeasureName

請參考結果資料列中的 measure_name 值。如果提供 ，則此欄位 MeasureNameColumn 為必填欄位。

類型：字串

必要：否

MultiMeasureAttributeMappings

當 measureValueType 為 時為必要MULTI。MULTI 值量值的屬性映射。

類型：MultiMeasureAttributeMapping 物件陣列

陣列成員：項目數下限為 1。

必要：否

SourceColumn

此欄位是指要從中讀取度量值以進行結果實體化的來源資料行。

類型：字串

必要：否

TargetMeasureName

要使用的目標度量名稱。如果未提供，預設的目標量值名稱會是 measure-name，否則會 sourceColumn 是提供。

類型：字串

必要：否

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

MultiMeasureAttributeMapping
服務：Amazon Timestream Query

MULTI 值量值的屬性映射。

目錄

MeasureValueType

從來源資料行讀取的屬性類型。

類型：字串

有效值:BIGINT | BOOLEAN | DOUBLE | VARCHAR | TIMESTAMP

必要：是

SourceColumn

要從其中讀取屬性值的來源資料行。

類型：字串

必要：是

TargetMultiMeasureAttributeName

用於衍生資料表中屬性名稱的自訂名稱。如果未提供，則會使用來源資料行名稱。

類型：字串

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

MultiMeasureMappings
服務：Amazon Timestream Query

僅 MultiMeasureMappings 提供 MixedMeasureMappings 或 之一。 MultiMeasureMappings 可用來在衍生資料表中將資料擷取為多重量值。

目錄

MultiMeasureAttributeMappings

必要。用於將查詢結果映射至擷取多重度量屬性資料的屬性映射。

類型：MultiMeasureAttributeMapping 物件陣列

陣列成員：項目數下限為 1。

必要：是

TargetMultiMeasureName

衍生資料表中目標多重度量名稱的名稱。 measureNameColumn 未提供時，需要此輸入。MeasureNameColumn 如果提供 ，則該資料欄的值將用作多量值名稱。

類型：字串

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

NotificationConfiguration
服務：Amazon Timestream Query

排程查詢的通知組態。Timestream 會在建立排程查詢、更新其狀態或將其刪除時傳送通知。

目錄

SnsConfiguration

SNS 組態的詳細資訊。

類型：SnsConfiguration 物件

必要：是

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

ParameterMapping
服務：Amazon Timestream Query

具名參數的映射。

目錄

Name

參數名稱。

類型：字串

必要：是

Type

包含查詢結果集中資料欄的資料類型。資料類型可以是純量或複雜。支援的純量資料類型為整數、布林值、字串、雙、時間戳記、日期、時間和間隔。支援的複雜資料類型為陣列、資料列和時間記錄。

類型：Type 物件

必要：是

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

QueryInsights
服務：Amazon Timestream Query

QueryInsights 是一項效能調校功能，可協助您最佳化查詢、降低成本並改善效能。透過 QueryInsights，您可以評估查詢的截斷效率，並識別需要改進的區域，以增強查詢效能。使用 QueryInsights，您也可以在時間與空間修剪方面分析查詢的有效性，並找出改善效能的機會。具體而言，您可以評估查詢使用時間型和分割區金鑰型索引策略最佳化資料擷取的能力。若要最佳化查詢效能，您必須微調管理查詢執行的時間和空間參數。

提供的關鍵指標QueryInsights是 QuerySpatialCoverage和 QueryTemporalRange。QuerySpatialCoverage指出查詢掃描的空間軸有多少，而較低的值更有效率。QueryTemporalRange顯示掃描的時間範圍，而較窄的範圍則表現較佳。

的益處 QueryInsights

以下是使用 的主要優點QueryInsights：

- 識別低效率的查詢 – QueryInsights提供查詢存取資料表的時間型和屬性型剪除資訊。此資訊可協助您識別次佳存取的資料表。
- 最佳化資料模型和分割 – 您可以使用QueryInsights資訊來存取和微調資料模型和分割策略。
- 調整查詢 – QueryInsights強調更有效地使用索引的機會。

> **Note**
>
> 您允許在QueryInsights啟用的情況下執行的QueryAPI請求數目上限為每秒 1 個查詢（QPS）。如果您超過此查詢速率，可能會導致限流。

目錄

Mode

提供下列模式以啟用 QueryInsights：

- ENABLED_WITH_RATE_CONTROL – QueryInsights啟用正在處理的查詢。此模式也包含速率控制機制，將QueryInsights功能限制為每秒 1 個查詢 （QPS）。
- DISABLED – 停用 QueryInsights。

類型：字串

有效值:ENABLED_WITH_RATE_CONTROL | DISABLED

必要：是

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

QueryInsightsResponse
服務：Amazon Timestream Query

提供與您執行之查詢相關的各種洞察和指標。

目錄

OutputBytes

指示以位元組為單位設定的查詢結果大小。您可以使用此資料來驗證結果集是否已作為查詢調校練習的一部分變更。

類型：Long

必要：否

OutputRows

指示作為查詢結果集的一部分傳回的資料列總數。您可以使用此資料來驗證結果集中的資料列數是否已變更，作為查詢調校練習的一部分。

類型：Long

必要：否

QuerySpatialCoverage

提供查詢空間涵蓋範圍的洞察，包括具有次佳（最大）空間修剪的資料表。此資訊可協助您識別分割策略中需要改進的區域，以增強空間刪除。

類型：QuerySpatialCoverage 物件

必要：否

QueryTableCount

指示查詢中的資料表數目。

類型：Long

必要：否

QueryTemporalRange

提供查詢時間範圍的洞察，包括具有最大（最大）時間範圍的資料表。以下是最佳化以時間為基礎的剪除的一些潛在選項：

- 新增缺少的時間述詞。
- 移除時間述詞周圍的函數。
- 將時間述詞新增至所有子查詢。

類型：QueryTemporalRange 物件

必要：否

UnloadPartitionCount

指示 Unload操作建立的分割區。

類型：Long

必要：否

UnloadWrittenBytes

指示 Unload操作所寫入的大小，以位元組為單位。

類型：Long

必要：否

UnloadWrittenRows

指示Unload查詢寫入的資料列。

類型：Long

必要：否

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

QuerySpatialCoverage
服務：Amazon Timestream Query

提供查詢空間涵蓋範圍的洞察，包括具有次佳（最大）空間修剪的資料表。此資訊可協助您識別分割策略中需要改進的領域，以增強空間修剪

例如，您可以使用QuerySpatialCoverage資訊執行下列動作：

- 新增 measure_name 或使用客戶定義的分割區金鑰（CDPK）述詞。
- 如果您已執行上述動作，請移除它們周圍的函數或子句，例如 LIKE。

目錄

Max

提供對已執行查詢的空間涵蓋範圍，以及具有最無效空間修剪的資料表的深入見解。

- Value – 空間涵蓋範圍的最大比率。
- TableArn – 具有次佳空間修剪之資料表的 Amazon Resource Name（ARN）。
- PartitionKey – 用於分割的分割區金鑰，可以是預設值measure_name或 CDPK。

類型：QuerySpatialCoverageMax 物件

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

QuerySpatialCoverageMax
服務：Amazon Timestream Query

提供資料表的深入見解，其中包含查詢掃描的最次理想空間範圍。

目錄

PartitionKey

用於分割的分割區金鑰，可以是預設measure_name或客戶定義的分割區金鑰 。

類型：字串陣列

必要：否

TableArn

資料表的 Amazon Resource Name （ ARN ），具有最次佳的空間修剪。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：否

Value

空間涵蓋範圍的最大比率。

類型：Double

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

QueryStatus

服務：Amazon Timestream Query

查詢狀態的相關資訊，包括進度和掃描的位元組。

目錄

CumulativeBytesMetered

查詢掃描的資料量，以您需付費的位元組為單位。這是累積總和，代表自查詢開始後您將要支付的資料總量。費用只會套用一次，並在查詢完成執行或查詢取消時套用。

類型：Long

必要：否

CumulativeBytesScanned

查詢掃描的資料量，以位元組為單位。這是累積總和，代表查詢開始後掃描的位元組總數。

類型：Long

必要：否

ProgressPercentage

查詢的進度，以百分比表示。

類型：Double

必要：否

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

QueryTemporalRange
服務：Amazon Timestream Query

提供查詢時間範圍的洞察，包括具有最大（最大）時間範圍的資料表。

目錄

Max

封裝下列屬性，這些屬性可提供對時間軸上最次佳執行資料表的洞察：

- `Value` – 查詢開始和結束之間的最長持續時間，以奈秒為單位。
- `TableArn` – 查詢最大時間範圍之資料表的 Amazon Resource Name（ARN）。

類型：QueryTemporalRangeMax 物件

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

QueryTemporalRangeMax
服務：Amazon Timestream Query

提供資料表的深入見解，其中包含查詢掃描到的最次佳時間修剪。

目錄

TableArn

查詢最大時間範圍之資料表的 Amazon Resource Name （ARN）。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：否

Value

查詢開始和結束之間的最長持續時間，以奈秒為單位。

類型：Long

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

Row
服務：Amazon Timestream Query

代表查詢結果中的單一資料列。

目錄

Data

結果集單一資料列中的資料點清單。

類型：Datum 物件陣列

必要：是

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

S3Configuration

服務：Amazon Timestream Query

執行查詢所產生之錯誤報告的 S3 位置詳細資訊。

目錄

BucketName

將在其下建立錯誤報告的 S3 儲存貯體名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 63。

模式：`[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

必要：是

EncryptionOption

錯誤報告的靜態加密選項。如果未指定加密選項，Timestream 將選擇 SSE_S3 作為預設值。

類型：字串

有效值:`SSE_S3 | SSE_KMS`

必要：否

ObjectKeyPrefix

錯誤報告索引鍵的字首。在預設情況下，Timestream 會將以下字首新增到錯誤報告路徑。

類型：字串

長度限制：長度下限為 1。長度上限為 896。

模式：`[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

S3ReportLocation
服務：Amazon Timestream Query

排程查詢執行的 S3 報告位置。

目錄

BucketName

S3 儲存貯體名稱。

類型：字串

長度限制：長度下限為 3。長度上限為 63。

模式：`[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

必要：否

ObjectKey

S3 金鑰。

類型：字串

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

ScheduleConfiguration
服務：Amazon Timestream Query

查詢排程的組態。

目錄

ScheduleExpression

表示何時觸發排程查詢執行的運算式。這可以是 Cron 運算式或 Rate 運算式。

類型：字串

長度限制：長度下限為 1。長度上限為 256。

必要：是

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

ScheduledQuery
服務：Amazon Timestream Query

排程查詢

目錄

Arn

Amazon Resource Name。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：是

Name

排程查詢的名稱。

類型：字串

長度限制：長度下限為 1。長度上限為 64。

模式：[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+

必要：是

State

排程查詢的狀態。

類型：字串

有效值:ENABLED | DISABLED

必要：是

CreationTime

排程查詢的建立時間。

類型：Timestamp

必要：否

ErrorReportConfiguration

排程查詢錯誤報告的組態。

類型：ErrorReportConfiguration 物件

必要：否

LastRunStatus

上次排程查詢執行的狀態。

類型：字串

有效值:AUTO_TRIGGER_SUCCESS | AUTO_TRIGGER_FAILURE | MANUAL_TRIGGER_SUCCESS | MANUAL_TRIGGER_FAILURE

必要：否

NextInvocationTime

下次執行排程查詢時。

類型：Timestamp

必要：否

PreviousInvocationTime

上次執行排程查詢的時間。

類型：Timestamp

必要：否

TargetDestination

將寫入最終排程查詢結果的目標資料來源。

類型：TargetDestination 物件

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

ScheduledQueryDescription
服務：Amazon Timestream Query

描述排程查詢的結構。

目錄

Arn

排程查詢 ARN。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：是

Name

排程查詢的名稱。

類型：字串

長度限制：長度下限為 1。長度上限為 64。

模式：`[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必要：是

NotificationConfiguration

通知組態。

類型：NotificationConfiguration 物件

必要：是

QueryString

要執行的查詢。

類型：字串

長度限制：長度下限為 1。長度上限為 262144。

必要：是

ScheduleConfiguration

排程組態。

類型：ScheduleConfiguration 物件

必要：是

State

排程查詢的狀態。

類型：字串

有效值:ENABLED | DISABLED

必要：是

CreationTime

排程查詢的建立時間。

類型：Timestamp

必要：否

ErrorReportConfiguration

排程查詢的錯誤報告組態。

類型：ErrorReportConfiguration 物件

必要：否

KmsKeyId

用於加密排程查詢資源的客戶提供的KMS金鑰。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：否

LastRunSummary

上次排程查詢執行的執行期摘要。

類型：ScheduledQueryRunSummary 物件

必要：否

NextInvocationTime

下次排程查詢排程執行時。

類型：Timestamp

必要：否

PreviousInvocationTime

上次執行查詢的時間。

類型：Timestamp

必要：否

RecentlyFailedRuns

最後五次失敗排程查詢執行的執行期摘要。

類型：ScheduledQueryRunSummary 物件陣列

必要：否

ScheduledQueryExecutionRoleArn

IAM Timestream 用來執行排程查詢的角色。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：否

TargetConfiguration

排程查詢目標存放區組態。

類型：TargetConfiguration 物件

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

ScheduledQueryInsights
服務：Amazon Timestream Query

封裝在 QueryInsights上啟用 的設定ExecuteScheduledQueryRequest。

目錄

Mode

提供下列模式以啟用 ScheduledQueryInsights：

- ENABLED_WITH_RATE_CONTROL – ScheduledQueryInsights啟用正在處理的查詢。此模式也包含速率控制機制，將QueryInsights功能限制為每秒 1 個查詢 （QPS）。
- DISABLED – 停用 ScheduledQueryInsights。

類型：字串

有效值:ENABLED_WITH_RATE_CONTROL | DISABLED

必要：是

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

ScheduledQueryInsightsResponse
服務：Amazon Timestream Query

提供與已執行 `ExecuteScheduledQueryRequest` 相關的各種洞察和指標。

目錄

OutputBytes

指示以位元組為單位設定的查詢結果大小。您可以使用此資料來驗證結果集是否已作為查詢調校練習的一部分變更。

類型：Long

必要：否

OutputRows

指示作為查詢結果集的一部分傳回的資料列總數。您可以使用此資料來驗證結果集中的資料列數是否已變更，作為查詢調校練習的一部分。

類型：Long

必要：否

QuerySpatialCoverage

提供查詢空間涵蓋範圍的洞察，包括具有次佳（最大）空間修剪的資料表。此資訊可協助您識別分割策略中需要改進的區域，以增強空間刪除。

類型：QuerySpatialCoverage 物件

必要：否

QueryTableCount

指示查詢中的資料表數目。

類型：Long

必要：否

QueryTemporalRange

提供查詢時間範圍的洞察，包括具有最大（最大）時間範圍的資料表。以下是最佳化時間型剪除的一些潛在選項：

- 新增缺少的時間述詞。
- 移除時間述詞周圍的函數。
- 將時間述詞新增至所有子查詢。

類型：QueryTemporalRange 物件

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

ScheduledQueryRunSummary
服務：Amazon Timestream Query

執行排程查詢的摘要

目錄

ErrorReportLocation

錯誤報告的 S3 位置。

類型：ErrorReportLocation 物件

必要：否

ExecutionStats

排程執行的執行期統計資料。

類型：ExecutionStats 物件

必要：否

FailureReason

故障時排程查詢的錯誤訊息。您可能需要查看錯誤報告，以取得更詳細的錯誤原因。

類型：字串

必要：否

InvocationTime

InvocationTime 用於此執行。這是排定查詢執行的時間。參數`@scheduled_runtime`可用於查詢以取得 值。

類型：Timestamp

必要：否

QueryInsightsResponse

提供與排程查詢的執行摘要相關的各種洞察和指標。

類型：ScheduledQueryInsightsResponse 物件

必要：否

RunStatus

排程查詢執行的狀態。

類型：字串

有效值:AUTO_TRIGGER_SUCCESS | AUTO_TRIGGER_FAILURE | MANUAL_TRIGGER_SUCCESS | MANUAL_TRIGGER_FAILURE

必要：否

TriggerTime

查詢執行的實際時間。

類型：Timestamp

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

SelectColumn
服務：Amazon Timestream Query

查詢傳回的資料欄詳細資訊。

目錄

Aliased

正確，如果資料欄名稱由查詢別名。否則為錯。

類型：布林值

必要：否

DatabaseName

具有此資料欄的資料庫。

類型：字串

必要：否

Name

欄的名稱。

類型：字串

必要：否

TableName

具有此資料欄的資料庫中的資料表。

類型：字串

必要：否

Type

包含查詢結果集中資料欄的資料類型。資料類型可以是純量或複雜。支援的純量資料類型為整數、布林值、字串、雙、時間戳記、日期、時間和間隔。支援的複雜資料類型為陣列、資料列和時間記錄。

類型：Type 物件

必要：否

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

SnsConfiguration
服務：Amazon Timestream Query

傳送通知SNS所需的詳細資訊。

目錄

TopicArn

SNS 將傳送排程查詢狀態通知ARN的主題。

類型：字串

長度限制：長度下限為 1。長度上限為 2048。

必要：是

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

Tag
服務：Amazon Timestream Query

標籤是您指派給 Timestream 資料庫and/or table. Each tag consists of a key and an optional value, both of which you define. Tags enable you to categorize databases and/or資料表的標籤，例如依用途、擁有者或環境。

目錄

Key

標籤鍵。標籤鍵會區分大小寫。

類型：字串

長度限制：長度下限為 1。長度上限為 128。

必要：是

Value

標籤的值。標籤值區分大小寫，可以是 null。

類型：字串

長度限制：長度下限為 0。長度上限為 256。

必要：是

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

TargetConfiguration
服務：Amazon Timestream Query

用於寫入查詢輸出的組態。

目錄

TimestreamConfiguration

將資料寫入 Timestream 資料庫和資料表所需的組態。

類型：TimestreamConfiguration 物件

必要：是

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

TargetDestination
服務：Amazon Timestream Query

寫入目標資料來源資料的目標詳細資訊。目前支援的資料來源為 Timestream。

目錄

TimestreamDestination

查詢 Timestream 資料來源的結果目的地詳細資訊。

類型：TimestreamDestination 物件

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

TimeSeriesDataPoint
服務：Amazon Timestream Query

時間記錄資料類型代表一段時間內量值的值。時間序列是時間戳記和測量值的資料列陣列，資料列會以時間的遞增順序排序。 TimeSeriesDataPoint 是時間序列中的單一資料點。它代表時間序列中（時間、測量值）的組合。

目錄

Time

收集測量值時的時間戳記。

類型：字串

必要：是

Value

資料點的測量值。

類型：Datum 物件

必要：是

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

TimestreamConfiguration
服務：Amazon Timestream Query

要將資料寫入 Timestream 資料庫和資料表的組態。此組態可讓使用者將查詢結果選取資料行映射至目的地資料表資料行。

目錄

DatabaseName

要將查詢結果寫入其中的 Timestream 資料庫名稱。

類型：字串

必要：是

DimensionMappings

這是為了允許將查詢結果中的資料行映射至目的地資料表中的維度。

類型：DimensionMapping 物件陣列

必要：是

TableName

要將查詢結果寫入其中的 Timestream 資料表名稱。該資料表應該在 Timestream 組態提供的相同資料庫中。

類型：字串

必要：是

TimeColumn

來自查詢結果的資料行，應該用作目的地資料表中的時間資料行。此 的資料欄類型應為 TIMESTAMP。

類型：字串

必要：是

MeasureNameColumn

度量資料行的名稱。

類型：字串

必要：否

MixedMeasureMappings

指定如何將度量映射至多重度量記錄。

類型：MixedMeasureMapping 物件陣列

陣列成員：項目數下限為 1。

必要：否

MultiMeasureMappings

多重度量映射。

類型：MultiMeasureMappings 物件

必要：否

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

TimestreamDestination
服務：Amazon Timestream Query

排程查詢的目的地。

目錄

DatabaseName

Timestream 資料庫名稱。

類型：字串

必要：否

TableName

時間串流資料表名稱。

類型：字串

必要：否

另請參閱

如需在其中一種語言特定 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

Type
服務：Amazon Timestream Query

包含查詢結果集中資料欄的資料類型。資料類型可以是純量或複雜。支援的純量資料類型為整數、布林值、字串、雙、時間戳記、日期、時間和間隔。支援的複雜資料類型為陣列、資料列和時間記錄。

目錄

ArrayColumnInfo

指示資料欄是否為陣列。

類型：ColumnInfo 物件

必要：否

RowColumnInfo

指示資料欄是否為資料列。

類型：ColumnInfo 物件陣列

必要：否

ScalarType

指示資料欄是否為類型字串、整數、布林值、雙、時間戳記、日期、時間。如需詳細資訊，請參閱支援的資料類型 。

類型：字串

有效值:VARCHAR | BOOLEAN | BIGINT | DOUBLE | TIMESTAMP | DATE | TIME | INTERVAL_DAY_TO_SECOND | INTERVAL_YEAR_TO_MONTH | UNKNOWN | INTEGER

必要：否

TimeSeriesMeasureValueColumnInfo

指示資料欄是否為 Timeeries 資料類型。

類型：ColumnInfo 物件

必要：否

另請參閱

如需在其中一種語言特定的 API中使用此功能的詳細資訊 AWS SDKs，請參閱下列內容：

- AWS SDK 適用於 C++
- AWS SDK 適用於 Java V2
- AWS SDK 適用於 Ruby V3

常見錯誤

本節列出所有 AWS 服務API動作常見的錯誤。如需此服務API動作的特定錯誤，請參閱該API動作的主題。

AccessDeniedException

您沒有足夠存取權可執行此動作。

HTTP 狀態碼：400

IncompleteSignature

請求簽章不符合 AWS 標準。

HTTP 狀態碼：400

InternalFailure

由於不明的錯誤、例外狀況或故障，處理請求失敗。

HTTP 狀態碼：500

InvalidAction

請求的動作或操作無效。確認已正確輸入動作。

HTTP 狀態碼：400

InvalidClientTokenId

提供的 X.509 憑證或 AWS 存取金鑰 ID 不存在於我們的記錄中。

HTTP 狀態碼：403

NotAuthorized

您沒有執行此動作的許可。

HTTP 狀態碼：400

OptInRequired

AWS 存取金鑰 ID 需要 服務的訂閱。

HTTP 狀態碼：403

RequestExpired

請求在請求上的日期戳記後超過 15 分鐘或請求過期後超過 15 分鐘 （例如，針對預先簽署的 URLs） 到達服務，或請求上的日期戳記在未來超過 15 分鐘。

HTTP 狀態碼：400

ServiceUnavailable

由於伺服器暫時故障，請求失敗。

HTTP 狀態碼：503

ThrottlingException

由於請求調節，因此請求遭到拒絕。

HTTP 狀態碼：400

ValidationError

輸入無法滿足 AWS 服務指定的限制條件。

HTTP 狀態碼：400

常見參數

以下清單內含所有動作用來簽署 Signature 第 4 版請求的參數以及查詢字串。任何專屬於特定動作的參數則列於該動作的主題中。如需 Signature 第 4 版的詳細資訊，請參閱 IAM 使用者指南 中的簽署 AWS API請求。

Action

要執行的動作。

類型：字串

必要：是

Version

寫入請求的API版本，以 格式表示 YYYY-MM-DD。

類型：字串

必要：是

X-Amz-Algorithm

建立請求簽章時所使用的雜湊演算法。

條件：當您在查詢字串中包含身分驗證資訊，而不是在HTTP授權標頭中時，請指定此參數。

類型：字串

有效值:`AWS4-HMAC-SHA256`

必要：有條件

X-Amz-Credential

憑證範圍值，此為一個字串，其中包含您的存取金鑰、日期、您的目標區域、您請求的服務，以及終止字串 (“aws4_request”)。此值會以下列格式表示：access_key /YYYYMMDD/region /service / aws4_request。

如需詳細資訊，請參閱 IAM 使用者指南 中的建立簽署 AWS API的請求。

條件：當您在查詢字串中包含身分驗證資訊，而不是在HTTP授權標頭中時，請指定此參數。

類型：字串

必要：有條件

X-Amz-Date

用來建立簽署的日期。格式必須為 ISO 8601 基本格式 （YYYYMMDD'T'HHMMSS'Z'）。例如，下列日期時間是有效的 X-Amz-Date值：`20120325T120000Z`。

條件： X-Amz-Date對所有請求都是選用的；它可以用來覆寫用於簽署請求的日期。如果以 8601 ISO 基本格式指定日期標頭， X-Amz-Date則不需要 。使用 時 X-Amz-Date，一律會覆寫日期標頭的值。如需詳細資訊，請參閱 IAM 使用者指南 中的請求簽章元素 AWS API。

類型：字串

必要：有條件

X-Amz-Security-Token

透過呼叫 AWS Security Token Service （ ） 取得的臨時安全權杖AWS STS。如需支援 臨時安全憑證的服務清單 AWS STS，請參閱 IAM 使用者指南 中的 AWS 服務 與 搭配使用IAM。

條件：如果您使用來自 的臨時安全憑證 AWS STS，則必須包含安全權杖。

類型：字串

必要：有條件

X-Amz-Signature

指定從要簽署的字串和衍生的簽署金鑰中計算出的十六進位編碼簽章。

條件：當您在查詢字串中包含身分驗證資訊，而不是在HTTP授權標頭中時，請指定此參數。

類型：字串

必要：有條件

X-Amz-SignedHeaders

指定包含在標準請求中的所有HTTP標頭。如需指定已簽署標頭的詳細資訊，請參閱 IAM 使用者指南 中的建立已簽署 AWS API的請求。

條件：當您在查詢字串中包含身分驗證資訊，而不是在HTTP授權標頭中時，請指定此參數。

類型：字串

必要：有條件

文件歷史記錄

變更	描述	日期
僅文件更新	更新配額主題以隔離預設配額和系統限制。	2024 年 10 月 22 日
Amazon Timestream 現在支援查詢洞察	Timestream 現在包含對查詢洞察功能的支援，可協助您最佳化查詢、改善其效能並降低成本。	2024 年 10 月 22 日

Amazon Timestream for InfluxDB 更新至現有政策。	Amazon Timestream for InfluxDB 已將 `ec2:DescribeRouteTables` 動作新增至現有`AmazonTimestreamInfluxDBFullAccess` 受管政策，以描述您的路由表	2024 年 10 月 8 日
`AmazonTimestreamReadOnlyAccess` – 更新現有政策	的 Timestream LiveAnalytics 已將`DescribeAccountSettings` 許可新增至 `AmazonTimestreamReadOnlyAccess` 受管政策，以描述 AWS 帳戶 設定。	2024 年 6 月 3 日
的 Amazon Timestream LiveAnalytics 現在支援 Timestream Compute Units （TCUs）	適用於 的 Amazon Timestream LiveAnalytics 現在包含對 Timestream Compute Units （TCUs） 的支援，以測量為您的查詢需求配置的運算容量。	2024 年 4 月 29 日
已新增政策	Amazon Timestream for InfluxDB 新增了兩個新政策：一個政策，允許服務管理您帳戶中的網路介面和安全群組。如需詳細資訊，請參閱 AmazonTimestreamInfluxDBServiceRolePolicy。另一個 提供完整的管理存取權，以建立、更新、刪除和列出 Amazon Timestream InfluxDB 執行個體，以及建立和列出參數群組。如需詳細資訊，請參閱 AmazonTimestreamInfluxDBFullAccess。	2024 年 3 月 14 日

Amazon Timestream for InfluxDB 現在已全面推出。	本文件涵蓋 Amazon Timestream for InfluxDB 的初始版本。	2024 年 3 月 14 日
Amazon Timestream for LiveAnalytics Query 事件可在中使用 AWS CloudTrail	的 Amazon Timestream LiveAnalytics 現在會將查詢API資料事件發佈至 AWS CloudTrail。客戶可以稽核其 AWS 帳戶中提出的所有查詢API請求，並查看相關資訊，例如IAM使用者/角色提出請求的對象、提出請求的時間、查詢的資料庫和資料表，以及請求的查詢 ID。	2023 年 9 月 12 日
的 Amazon Timestream LiveAnalytics UNLOAD	適用於 的 Amazon Timestream LiveAnalytics 現在支援UNLOAD將查詢結果匯出至S3。	2023 年 5 月 12 日
Amazon Timestream 以 LiveAnalytics 更新現有政策。	批次載入許可已新增至受管政策。	2023 年 2 月 24 日
適用於 LiveAnalytics 批次載入的 Amazon Timestream。	適用於 的 Amazon Timestream LiveAnalytics 現在支援批次載入功能。	2023 年 2 月 24 日
適用於 的 Amazon Timestream LiveAnalytics 現在支援 AWS Backup。	適用於 的 Amazon Timestream LiveAnalytics 現在支援 AWS Backup。	2022 年 12 月 14 日
Amazon Timestream 可 LiveAnalytics 更新 AWS 受管政策	有關 AWS 受管政策和 Amazon Timestream 的新資訊 LiveAnalytics，包括現有受管政策的更新。	2021 年 11 月 29 日
的 Amazon Timestream LiveAnalytics 支援排程查詢	適用於 的 Amazon Timestream LiveAnalytics 現在支援根據排程代表您執行查詢。	2021 年 11 月 29 日

的 Amazon Timestream LiveAnalytics 支援磁性存放區。	適用於 的 Amazon Timestream LiveAnalytics 現在支援使用磁性儲存進行資料表寫入。	2021 年 11 月 29 日
LiveAnalytics 適用於多測量記錄的 Amazon Timestream。	適用於 的 Amazon Timestream LiveAnalytics 現在支援更精簡的格式來儲存您的時間序列資料。	2021 年 11 月 29 日
Amazon Timestream 可 LiveAnalytics 更新AWS受管政策	有關 AWS 受管政策和 Amazon Timestream 的新資訊 LiveAnalytics，包括現有受管政策的更新。	2021 年 5 月 24 日
適用於 的 Amazon Timestream LiveAnalytics 現在可在歐洲（法蘭克福）區域使用。	適用於 的 Amazon Timestream LiveAnalytics 現在已全面在歐洲（法蘭克福）區域（ ）推出`eu-central-1` 。	2021 年 4 月 23 日
的 Amazon Timestream LiveAnalytics 現在支援VPC端點（AWS PrivateLink）。	適用於 的 Amazon Timestream LiveAnalytics 現在支援使用VPC端點（AWS PrivateLink）。	2021 年 3 月 23 日
Amazon Timestream 現在支援跨資料表查詢。	您可以使用 的 Amazon Timestream LiveAnalytics 來執行跨資料表查詢。	2021 年 2 月 10 日
適用於 的 Amazon Timestream LiveAnalytics 現在支援增強型查詢執行統計資料。	適用於 的 Amazon Timestream LiveAnalytics 現在支援增強型查詢執行統計資料，例如掃描的資料量。	2021 年 2 月 10 日
適用於 的 Amazon Timestream LiveAnalytics 現在支援進階時間序列函數。	您可以使用 的 Amazon Timestream LiveAnalytics 來執行具有進階時間序列函數的SQL查詢，例如導數、積分和關聯。	2021 年 2 月 10 日

的 Amazon Timestream 現在 LiveAnalytics 已PCI符合 HIPAA、 ISO和 的要求。	您現在可以將 Amazon Timestream LiveAnalytics 用於需要 HIPAA、 ISO和 PCI合規基礎設施的工作負載。	2021 年 1 月 27 日
適用於 的 Amazon Timestream LiveAnalytics 現在支援開放原始碼 Telegraf 和 grafana。	您現在可以使用開放原始碼外掛程式驅動的伺服器代理程式 Telegraf 來收集和報告指標，以及使用資料庫的開放原始碼分析和監控平台 Grafana 搭配適用於 的 Amazon Timestream LiveAnalytics。	2020 年 11 月 25 日
適用於 的 Amazon Timestream LiveAnalytics 現在已全面推出。	本文件涵蓋 Amazon Timestream 的初始版本 LiveAnalytics。	2020 年 9 月 30 日

什麼是 InfluxDB 的 Timestream？

Amazon Timestream for InfluxDB 是受管時間序列資料庫引擎，可讓應用程式開發人員和 DevOps 團隊使用開放原始碼 輕鬆 AWS 在 上執行即時時間序列應用程式的 InfluxDB 資料庫APIs。使用 Amazon Timestream for InfluxDB ，輕鬆設定、操作和擴展時間序列工作負載，這些工作負載可以用單位數毫秒查詢回應時間來回應查詢。

Amazon Timestream for InfluxDB 可讓您存取其 2.x 分支上熟悉的 InfluxDB 開放原始碼版本的功能。這表示您今天已與現有 InfluxDB 開放原始碼資料庫搭配使用的程式碼、應用程式和工具，應與 Amazon Timestream for InfluxDB 無縫搭配使用。Amazon Timestream for InfluxDB 可以自動備份資料庫，並讓您的資料庫軟體與最新版本保持最新狀態。此外，Amazon Timestream for InfluxDB 可讓您輕鬆地使用複寫來增強資料庫可用性，並改善資料耐久性。與所有 AWS 服務一樣，不需要預先投資，您只需為所使用的資源付費。

資料庫執行個體

資料庫執行個體是在雲端執行的隔離資料庫環境。這是 Amazon Timestream for InfluxDB 的基本建置區塊。資料庫執行個體可以包含多個使用者建立的資料庫 （或 InfluxDb 2.x 資料庫案例的組織和儲存貯體），而且可以使用您用來存取獨立自我管理 InfluxDB 執行個體的相同用戶端工具和應用程式來存取。使用 AWS 命令列工具、Amazon Timestream InfluxDB API操作或 ，即可輕鬆建立和修改資料庫執行個體 AWS Management Console。

> **Note**
>
> Amazon Timestream for InfluxDB 支援使用 Influx API操作和 Influx UI 存取資料庫。Amazon Timestream for InfluxDB 不允許直接主機存取。

您最多可以有 40 個 Amazon Timestream for InfluxDB 執行個體。

每個資料庫執行個體都有一個資料庫執行個體名稱。此客戶提供的名稱在與 Amazon Timestream for InfluxDB API和 AWS CLI命令互動時，會唯一識別資料庫執行個體。資料庫執行個體名稱對於 AWS 區域中的客戶必須是唯一的。

資料庫執行個體名稱會形成 Timestream for InfluxDB 配置給您執行個體的部分DNS主機名稱。例如，如果您將 influxdb1 指定為資料庫執行個體名稱，Timestream 會自動為您的執行個體配置DNS端點。範例端點是 `influxdb1-3ksj4dla5nfjhi.us-east-1.timestream-influxdb.amazonaws.com`，其中 `influxdb1`是您的執行個體名稱。

在範例端點 中influxdb1-3ksj4dla5nfjhi.us-east-1.timestream-influxdb.amazonaws.com，字串3ksj4dla5nfjhi是由 產生的唯一帳戶識別碼 AWS。範例中3ksj4dla5nfjhi的識別碼不會變更特定區域中指定帳戶的識別碼。因此，此帳戶建立的所有資料庫執行個體都會共用相同的固定識別碼。請考慮固定識別符的下列功能：

- 目前 InfluxDB 的 Timestream 不支援資料庫執行個體重新命名。
- 如果您刪除並重新建立具有相同資料庫執行個體識別符的資料庫執行個體，則端點會相同。
- 如果您使用相同的帳戶在不同區域中建立資料庫執行個體，則內部產生的識別符會有所不同，因為區域不同，如 influxdb2.4a3j5du5ks7md2.us-west-1.timestream-influxdb.amazonaws.com 中所示。

每個資料庫執行個體僅支援一個 InfluxDB 資料庫引擎的 Timestream。

建立資料庫執行個體時，InfluxDB 需要指定組織名稱。資料庫執行個體可以託管多個組織，以及與每個組織相關聯的多個儲存貯體。

Amazon Timestream for InfluxDB 可讓您在建立程序中為資料庫執行個體建立主要使用者帳戶和密碼。此主要使用者具有建立組織、儲存貯體，以及對資料執行讀取、寫入、刪除和升級操作的許可。您也可以存取 InfluxUI，並在 上擷取運算子權杖。您的第一次登入。您也可以從那裡管理所有存取權杖。您必須在建立資料庫執行個體時設定主要使用者密碼，但您可以隨時使用 Influx API、 Influx CLI或 InfluxUI 進行變更。

資料庫執行個體類別

資料庫執行個體類別決定 Amazon Timestream fi UbfkyxDB 資料庫執行個體的運算和記憶體容量。您需要的資料庫執行個體類別取決於您的處理能力和記憶體需求。

資料庫執行個體類別由資料庫執行個體類別類型和大小組成。例如， db.influx 是一種記憶體最佳化的資料庫執行個體類別類型，適用於與執行中的 InfluxDb 工作負載相關的高效能記憶體需求。在db.influx執行個體類別類型中， db.influx.2xlarge是資料庫執行個體類別。此類別的大小為 2xlarge 。

如需執行個體類別定價的詳細資訊，請參閱 Amazon Timestream for InfluxDB 定價。

資料庫執行個體類別的類型

Amazon Timestream for InfluxDB 支援針對 InfluxDB 使用案例最佳化的下列使用案例的資料庫執行個體類別。

- **db.influx**—這些執行個體類別非常適合在開放原始碼 InfluxDB 資料庫中執行記憶體密集型工作負載

資料庫執行個體類別的硬體規格

下列術語說明資料庫執行個體類別的硬體規格：

- vCPU

  虛擬中央處理單元的數目 （CPUs）。虛擬CPU是容量單位，可用來比較資料庫執行個體類別。

- 記憶體 (GiB)

  RAM以 gibibyte 為單位，配置給資料庫執行個體的 。記憶體與 v 之間通常有一致的比率CPU。以 db.influx 執行個體類別為例，其記憶體對 vCPU 比率類似於 r7g EC2 執行個體類別。

- Influx-Optimized

  資料庫執行個體使用最佳化組態堆疊，並為輸入/輸出提供額外專用容量。此最佳化透過減少輸入/輸出與執行個體的其他流量之間的爭用情況，來提供最佳效能。

- 網路頻寬

  相對於其他資料庫執行個體類別的網路速度。在下表中，您可以找到有關 Amazon Timestream for InfluxDB 執行個體類別的硬體詳細資訊。

執行個體類別	vCPU	記憶體 (GiB)	儲存類型	網路頻寬（Gbps）
db.influx.medium	1	8	已IOPS包含流入	10
db.influx.large	2	16	已IOPS包含流入	10
db.influx.xlarge	4	32	已IOPS包含流入	10
db.influx.2xlarge	8	64	已IOPS包含流入	10
db.influx.4xlarge	16	128	已IOPS包含流入	10
db.influx.8xlarge	32	256	已IOPS包含流入	12

執行個體類別	vCPU	記憶體 (GiB)	儲存類型	網路頻寬（Gbps）
db.influx.12xlarge	48	384	已IOPS包含流入	20
db.influx.16xlarge	64	512	已IOPS包含流入	25

InfluxDB 執行個體儲存

Amazon Timestream for InfluxDB 的資料庫執行個體會使用 Influx IOPS 包含的磁碟區作為資料庫和日誌儲存。

在某些情況下，您的資料庫工作負載可能無法達到IOPS已佈建 的 100%。如需詳細資訊，請參閱影響儲存體效能的因素。如需 Timestream for InfluxDB 儲存體定價的詳細資訊，請參閱 Amazon Timestream 定價。

InfluxDB 儲存類型的 Amazon Timestream

Amazon Timestream for InfluxDB 支援 Influx IOPS Included 的一種儲存類型。您可以為 InfluxDB 執行個體建立 Timestream，儲存容量最高可達 16 TB TiB）。

以下是可用儲存體類型的簡短描述：

- Influx IO 含儲存：儲存效能是每秒 I/O 操作（IOPS）的組合，以及儲存磁碟區執行讀取和寫入的速度（儲存輸送量）。在 Influx IOPS 包含的儲存磁碟區上，Amazon Timestream for InfluxDB 提供 3 個儲存層，這些儲存層預先設定了不同類型的工作負載所需的最佳IOPS和輸送量。

InfluxDB 執行個體大小調整

Timestream for InfluxDB 執行個體的最佳組態取決於許多因素，包括擷取速率、批次大小、時間序列基數、並行查詢和查詢類型。為了提供大小調整建議，我們專注於具有下列特性的示範工作負載：

- 資料是由 Telegraf 代理程式機群收集和寫入，這些代理程式機群會從資料中心收集系統、、CPU記憶體、磁碟、IO 等。

  每個寫入請求包含 5000 行。

- 在系統上執行的查詢類型會分類為「中度複雜性」查詢。此查詢類別具有下列特徵：
  - 具有多個函數和一個或兩個規則運算式
  - 也可能具有依子句分組的群組，或取樣為期數週的時間範圍。
  - 通常需要幾百毫秒到幾千毫秒才能執行。
  - CPU 主要偏好查詢效能。

執行個體類別	儲存類型	寫入（每秒行數）	讀取（每秒查詢數）
db.influx.large	包含 3K 的流入 IO	~50，000	<10
db.influx.2xlarge	包含 3K 的流入 IO	~150，000	<25
db.influx.4xlarge	包含 3K 的流入 IO	~200，000	~25
db.influx.4xlarge	包含 12K 的流入 IO	~250，000	~35
db.influx.8xlarge	包含 12K 的流入 IO	~500，000	~50
db.influx.12xlarge	包含 12K 的流入 IO	<750，000	<100

AWS 區域和可用區域

Amazon 雲端運算資源託管於全球的多個地點。這些位置由 AWS 區域和可用區域組成。每個 AWS 區域都是單獨的地理區域。每個 AWS 區域都有多個獨立的位置，稱為可用區域。

> **Note**
>
> 如需尋找 AWS 區域可用區域的詳細資訊，請參閱 Amazon EC2使用者指南 中的區域和區域。

Amazon Timestream for InfluxDB 可讓您將資源，例如資料庫執行個體和資料放置在多個位置。

Amazon 運作 state-of-the-art、高可用性的資料中心。儘管故障極為少見，但仍可能影響相同位置內資料庫執行個體的可用性。若您將所有資料庫執行個體都託管於某一位置，一旦該位置受故障影響，所有資料庫執行個體都將無法使用。

請務必記住，每個 AWS 區域都是完全獨立的。您啟動的任何 Amazon Timestream for InfluxDB 活動（例如，建立資料庫執行個體或列出可用的資料庫執行個體）只會在您目前的預設 AWS 區域中執行。您可以在主控台中或設定 `AWS_DEFAULT_REGION` 環境變數來變更預設 AWS 區域。或者，它可以透過將 `--region` 參數與 AWS Command Line Interface（ ）搭配使用來覆寫AWS CLI。如需詳細資訊，請參閱設定 AWS Command Line Interface，特別是環境變數和命令列選項的相關區段。

若要在特定 AWS 區域中建立或使用 Amazon Timestream for InfluxDB 資料庫執行個體，請使用對應的區域服務端點。

AWS 區域可用性

如需目前可使用 Amazon Timestream for InfluxDB AWS 的區域以及每個區域的端點的詳細資訊，請參閱 Amazon Timestream 端點和配額 。

AWS 區域設計

每個 AWS 區域的設計是要與其他 AWS 區域隔離。此設計可達到最高的容錯能力與穩定性。

當您檢視資源時，只會看到與您指定的 AWS 區域繫結的資源。這是因為 AWS 區域彼此隔離，而且我們不會自動跨 AWS 區域複寫資源。

AWS 可用區域

當您建立資料庫執行個體時，Amazon Timestream for InfluxDB 會根據子網路組態隨機選擇其中一個執行個體。可用區域由 AWS 區域代碼代表，後面接著字母識別符（例如 `us-east-1a`）。

使用 `describe-availability-zones` Amazon EC2命令，如下所示來描述您帳戶所啟用之指定區域內的可用區域。

```
aws ec2 describe-availability-zones --region region-name
```

例如，若要描述為您的帳戶啟用的美國東部（維吉尼亞北部）區域（us-east-1）內的可用區域，請執行下列命令：

```
aws ec2 describe-availability-zones --region us-east-1
```

您無法為多可用區域資料庫部署 中的主要和次要資料庫執行個體選擇可用區域。Amazon Timesteram for InfluxDB 會隨機為您選擇它們。如需多可用區域部署的詳細資訊，請參閱 設定和管理多可用區部署。

Amazon Timestream for InfluxDB 的資料庫執行個體計費

Amazon Timestream for InfluxDB 執行個體的計費是根據下列元件：

- 資料庫執行個體小時數（每小時）— 根據資料庫執行個體的資料庫執行個體類別，例如 db.influx.large。定價以每小時為單位列出，但帳單已採用秒數為計算單位，並以十進位制顯示時間。InfluxDB 的 Amazon Timestream 用量以 1 秒為單位遞增計費，最少 10 分鐘。如需詳細資訊，請參閱資料庫執行個體類別資料庫執行個體類別。
- 儲存（每月每 GiB）— 您已佈建至資料庫執行個體的儲存容量。如需詳細資訊，請參閱InfluxDB 執行個體儲存。
- 資料傳輸（每 GB）— 資料庫執行個體進出網際網路和其他 AWS 區域的資料傳輸。

如需 Amazon Timestream for InfluxDB 定價資訊，請參閱 Amazon Timestream for InfluxDB 定價頁面。

設定 InfluxDB 的 Amazon Timestream

第一次使用 Amazon Timestream for InfluxDB 之前，請完成下列任務：

如果您已有 AWS 帳戶，請了解您的 Amazon Timestream for InfluxDB 需求，並偏好使用 的預設值 IAM和 InfluxDB 的 VPC Timestream for InfluxDB 入門Amazon Timestream 入門。

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟以建立帳戶。

註冊帳戶 AWS

- 前往AWS 登入頁面。
- 選擇建立新帳戶，然後按照指示操作。

> **Note**
>
> 部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

當您註冊 AWS 帳戶時，會建立 AWS 帳戶根使用者。根使用者可存取 帳戶中的所有 AWS 服務和資源。作為安全最佳實務，將管理存取權指派給管理使用者，並且僅使用根使用者來執行 需要根使用者存取權的任務。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時前往 https://aws.amazon.com/ 並選擇我的帳戶 來檢視目前的帳戶活動和管理您的帳戶。

使用者管理

建立管理使用者

建立管理使用者

註冊 AWS 帳戶後，請建立管理使用者，以免將根使用者用於日常任務。

保護 AWS 您的帳戶根使用者

選擇根使用者並輸入 AWS 您的帳戶電子郵件地址，以帳戶擁有者身分登入 AWS Management Console。在下一頁中，輸入您的密碼。如需使用根使用者登入的協助，請參閱登入使用者指南中的以根使用者身分登入 AWS

為您的根使用者開啟多重要素驗證（MFA）。如需指示，請參閱 IAM 使用者指南 中的為 AWS 您的帳戶根使用者（主控台）啟用虛擬MFA裝置。

授予程式設計存取權

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

若要授予使用者程式設計存取權，請選擇下列其中一個選項：

哪個使用者需要程式設計存取權？	到	By
Workforce identity（IAM在 Identity Center 中管理的使用者）	使用暫時憑證簽署對 AWS CLI AWS SDKs、 或 的程式設計請求 AWS APIs。	遵循您要使用的介面指示。* 如需 AWS CLI，請參閱 設定 AWS CLI以使用 AWS IAM Identity Center 在 中 AWS 命令列介面使用者指南 .* 如需 AWS SDKs、工具和 AWS APIs，請參閱 IAM 身分中心身分驗證 在 中 AWS SDKs 和 工具參考指南 。
IAM	使用暫時憑證簽署對 AWS CLI、 SDKs和 的程式設計請求 APIs。	遵循 IAM 使用者指南 中的使用臨時憑證與 AWS 資源的指示。
IAM	（不建議）使用長期憑證簽署 AWS CLI、 SDKs和 的程式設計請求APIs。	遵循您要使用的介面指示。如需 AWS CLI，請參閱 使用IAM使用者憑證進行驗證 在 中 AWS 命令列介面使用者指南

哪個使用者需要程式設計存取權？	到	By
		。如需 AWS SDKs 和 工具，請參閱 使用長期憑證進行驗證 在 中 AWS SDKs 和 工具參考指南 。如需 AWS APIs，請參閱 管理IAM使用者的存取金鑰 在 中 IAM 使用者指南。

判定需求

Amazon Timestream for Influx 的基本建置區塊是資料庫執行個體。在資料庫執行個體中，您可以建立儲存貯體。資料庫執行個體提供的網路位址稱為端點。您的應用程式使用此端點來連接至您的資料庫執行個體。您也可以從瀏覽器使用此相同的端點來存取 InfluxUI。當您建立資料庫執行個體時，您可以指定儲存體、記憶體、資料庫引擎和版本、網路組態和安全性等詳細資訊。您可以透過安全群組來控制對資料庫執行個體的網路存取。

建立資料庫執行個體和安全群組之前，您必須知道您的資料庫執行個體和網路需求。這裡是一些要考慮的注意事項：

- 資源需求 — 應用程式或服務有哪些記憶體和處理器需求？ 您可以使用這些設定來幫助判定要使用的資料庫執行個體類別。如需資料庫執行個體類別的規格，請參閱資料庫執行個體類別 。
- VPC 和 安全群組 — 資料庫執行個體很可能位於虛擬私有雲端 （VPC）。若要連接至您的資料庫執行個體，您必須設定安全群組規則。這些規則的設定方式取決於VPC您使用的類型和使用方式。例如，您可以使用：預設VPC或使用者定義的 VPC。

  下列清單說明每個VPC選項的規則：

  - 預設 VPC — 如果 AWS 您的帳戶VPC在目前 AWS 區域中具有預設值，則VPC設定為支援資料庫執行個體。如果您在建立資料庫執行個體VPC時指定預設值，請務必建立VPC安全群組，以授權從應用程式或服務連線至 Amazon Timestream for InfluxDB 資料庫執行個體。使用VPC主控台上

的安全群組選項或 AWS CLI來建立VPC安全群組。如需詳細資訊，請參閱步驟 3：建立VPC安全群組 。

- 使用者定義 VPC — 如果您想要在建立資料庫執行個體VPC時指定使用者定義，請注意下列事項：
  - 請務必建立VPC安全群組，以授權從應用程式或服務連線至 Amazon Timestream for InfluxDB 資料庫執行個體。使用VPC主控台上的安全群組選項或 AWS CLI來建立VPC安全群組。如需詳細資訊，請參閱步驟 3：建立VPC安全群組 。
  - VPC 必須符合特定需求，才能託管資料庫執行個體，例如至少有兩個子網路，每個子網路都位於單獨的可用區域中。如需詳細資訊，請參閱 Amazon VPCVPCs和 Amazon Timestream for InfluxDB。
- 高可用性 — 您需要容錯移轉支援嗎？ 在 Amazon Timestream for InfluxDB 上，多可用區域部署會在另一個可用區域中建立主要資料庫執行個體和次要待命資料庫執行個體，以進行容錯移轉支援。建議對生產工作負載使用異地同步備份部署以保有高可用性。針對開發和測試目的，您可以使用異地同步備份以外的部署。如需詳細資訊，請參閱多可用區域資料庫執行個體部署。
- IAM 政策 — AWS 您的帳戶是否有政策，授予執行 Amazon Timestream for InfluxDB 操作所需的許可？ 如果您 AWS 使用IAM憑證連線至 ，IAM您的帳戶必須具有政策IAM，授予執行 Amazon Timestream for InfluxDB 控制平面操作所需的許可。如需詳細資訊，請參閱Amazon Timestream for InfluxDB 的身分和存取管理。
- 開放連接埠 — 資料庫接聽的 TCP/IP 連接埠為何？ 某些公司的防火牆可能會封鎖與您的資料庫引擎預設連接埠的連線。InfluxDB 的 Timestream 預設為 8086。
- AWS 區域 — 您希望資料庫位於哪個 AWS 區域？ 將您的資料庫放置在鄰近您的應用程式或 Web 服務的位置可以減少網路延遲。如需詳細資訊，請參閱AWS 區域和可用區域。
- 資料庫磁碟子系統 — 您的儲存需求是什麼？ Amazon Timestream for InfluxDB 為 Influx IOPS 包含儲存類型提供三種組態：
  - Influx Io 已包含 3k IOPS（SSD）
  - Influx Io 已包含 12k IOPS（SSD）
  - Influx Io 已包含 25k IOPS（SSD）

  如需 Amazon Timestream for InfluxDB 儲存體的詳細資訊，請參閱 Amazon Timestream for InfluxDB 資料庫執行個體儲存體。具備建立安全群組和資料庫執行個體所需的資訊時，請繼續進行下一個步驟。

透過VPC建立安全群組，在 中提供資料庫執行個體的存取權

VPC 安全群組提供 中資料庫執行個體的存取權VPC。其作用就像相關資料庫執行個體的防火牆，從資料庫執行個體層級控制傳入和傳出流量。資料庫執行個體建立時預設提供防火牆和可保護資料庫執行個體的預設安全群組。

您必須先新增規則至可讓您連線的安全群組，才可連線到資料庫執行個體。使用您的網路和組態資訊來建立規則以允許存取您的資料庫執行個體。

例如，假設您的應用程式存取 中資料庫執行個體上的資料庫VPC。在此情況下，您必須新增自訂TCP規則，指定應用程式用來存取資料庫的連接埠範圍和 IP 地址。如果您在 Amazon EC2執行個體上有應用程式，您可以使用為 Amazon EC2執行個體設定的安全群組。

建立安全群組以進行VPC存取

若要建立VPC安全群組，請登入 AWS Management Console 並選擇 VPC。

> **Note**
>
> 請確定您在 VPC 主控台中，而不是 Amazon Timesteam for InfluxDB 主控台中。

- 在 的右上角 AWS Management Console，選擇要建立VPC安全群組和資料庫執行個體AWS的區域。在該 AWS 區域的 Amazon VPC 資源清單中，您應該至少會看到一個VPC和數個子網路。如果沒有，則表示您VPC在該 AWS 區域中沒有預設值。
- 在導覽窗格中，選擇 Security Groups (安全群組)。
- 選擇 Create Security Group (建立安全群組)。
- 在安全群組頁面的基本詳細資訊區段中，輸入安全群組名稱和描述 。針對 VPC，選擇您要VPC在其中建立資料庫執行個體的 。
- 在 Inbound rules (入站規則) 中，選擇 Add rule (新增規則)。
  - 對於類型 ，選擇自訂 TCP。
  - 針對來源 ，選擇安全群組名稱，或輸入您存取資料庫執行個體的 IP 地址範圍 （CIDR 值）。如果您選擇 My IP (我的 IP)，此舉允許透過您的瀏覽器中偵測到的 IP 地址存取資料庫執行個體。

  對於來源，請選擇安全群組名稱，或輸入您存取資料庫執行個體的 IP 地址範圍 （CIDR 值）。如果您選擇 My IP (我的 IP)，此舉允許透過您的瀏覽器中偵測到的 IP 地址存取資料庫執行個體。
- (選用) 在 Outbound Rules (輸出規則) 中，新增輸出流量的規則。預設會允許所有傳出流量。
- 選擇建立安全群組。

您可以在建立資料庫執行個體時使用此VPC安全群組作為安全群組。

> **Note**
>
> 如果您使用預設 VPC，則會為您建立跨越所有 子網路的預設VPC子網路群組。建立資料庫執行個體時，您可以選擇預設 eiifccntfVPC，然後選擇資料庫子網路群組的預設。

在完成了設定需求之後，您可以使用您的需求和安全群組建立資料庫執行個體。若要執行此操作，請遵循 建立資料庫執行個體 中的指示。

Timestream for InfluxDB 入門

在下列範例中，您可以了解如何使用 Amazon Timestream for InfluxDB Service 建立和連線至資料庫執行個體。

> **Note**
>
> 您必須完成設定 InfluxDB 的 Amazon Timestream 中的任務，才能建立或連線至資料庫執行個體。

主題

- 建立並連線至 Timestream for InfluxDB 執行個體
- 為您的 InfluxDB 執行個體建立新的運算子權杖

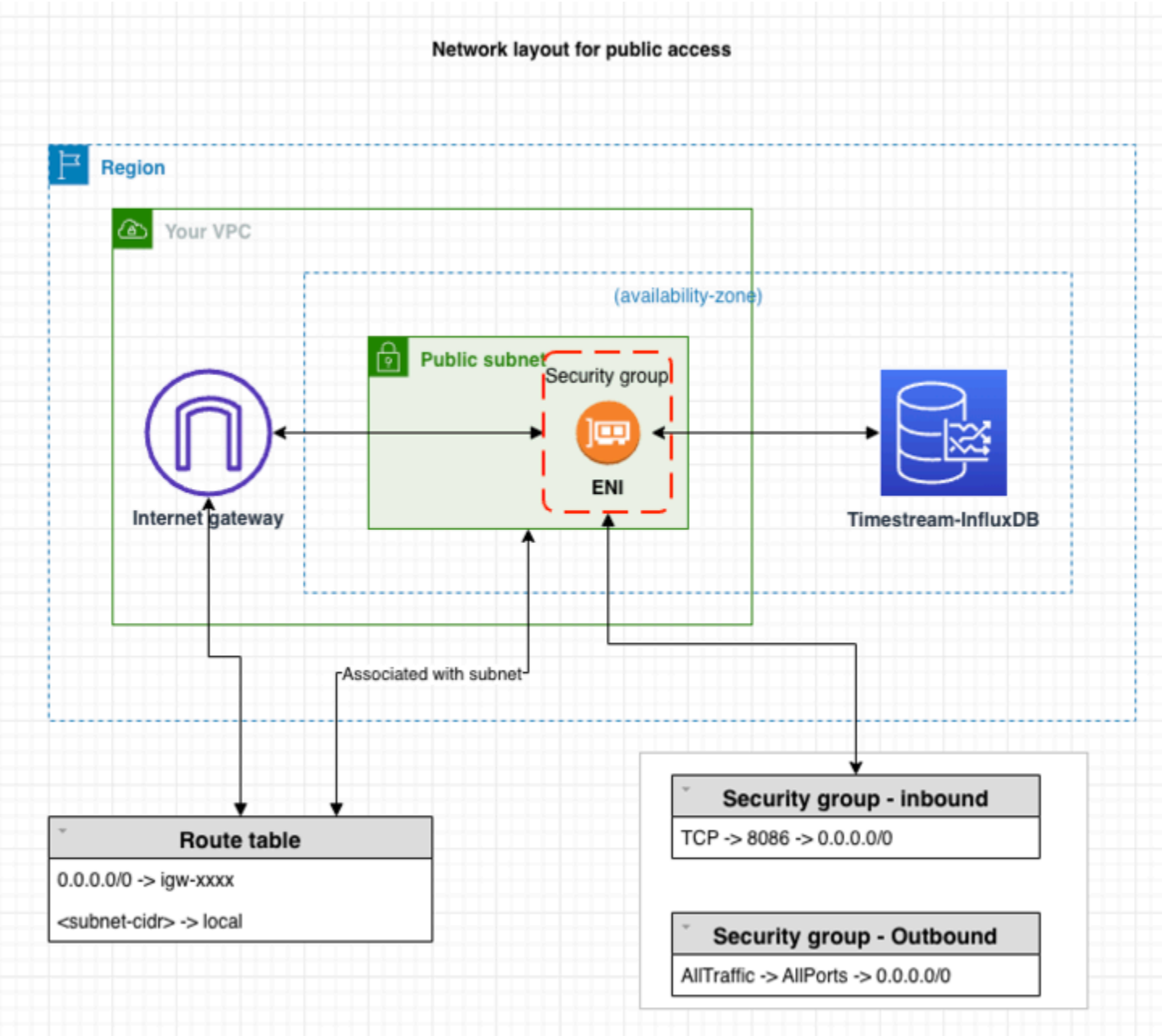
建立並連線至 Timestream for InfluxDB 執行個體

本教學課程會建立 Amazon EC2執行個體和 Amazon Timestream for InfluxDB 資料庫執行個體。本教學課程說明如何使用 Telegraf 用戶端，從執行個體將資料寫入資料庫EC2執行個體。最佳實務是，本教學課程會在虛擬私有雲端（ ）中建立私有資料庫執行個體VPC。在大多數情況下，相同 中的其他資源VPC，例如EC2執行個體，可以存取資料庫執行個體，但 以外的資源VPC無法存取。

完成教學課程後，您的 中的每個可用區域都會有一個公有和私有子網路VPC。在一個可用區域中，EC2執行個體位於公有子網路中，而資料庫執行個體位於私有子網路中。

> **Note**
>
> 建立 AWS 帳戶無需付費。不過，完成本教學課程後，您使用 AWS 的資源可能會產生費用。如果不再需要這些資源，您可以在完成教學課程後刪除這些資源。

下圖顯示可存取性為公開時的組態。



> **Warning**
>
> 我們不建議使用 0.0.0.0/0 進行HTTP存取，因為您可讓所有 IP 地址透過 存取您的公有 InfluxDB 執行個體HTTP。在測試環境中，此方法甚至在短時間內是不可接受的。僅授權特定 IP 地址或地址範圍，以使用 HTTP 進行 WebUI 或存取API來存取您的 InfluxDB 執行個體。

本教學課程會使用 建立執行 InfluxDB 的資料庫執行個體 AWS Management Console。我們只會專注於資料庫執行個體大小和資料庫執行個體識別符。我們將使用其他組態選項的預設設定。此範例建立的資料庫執行個體將為私有。

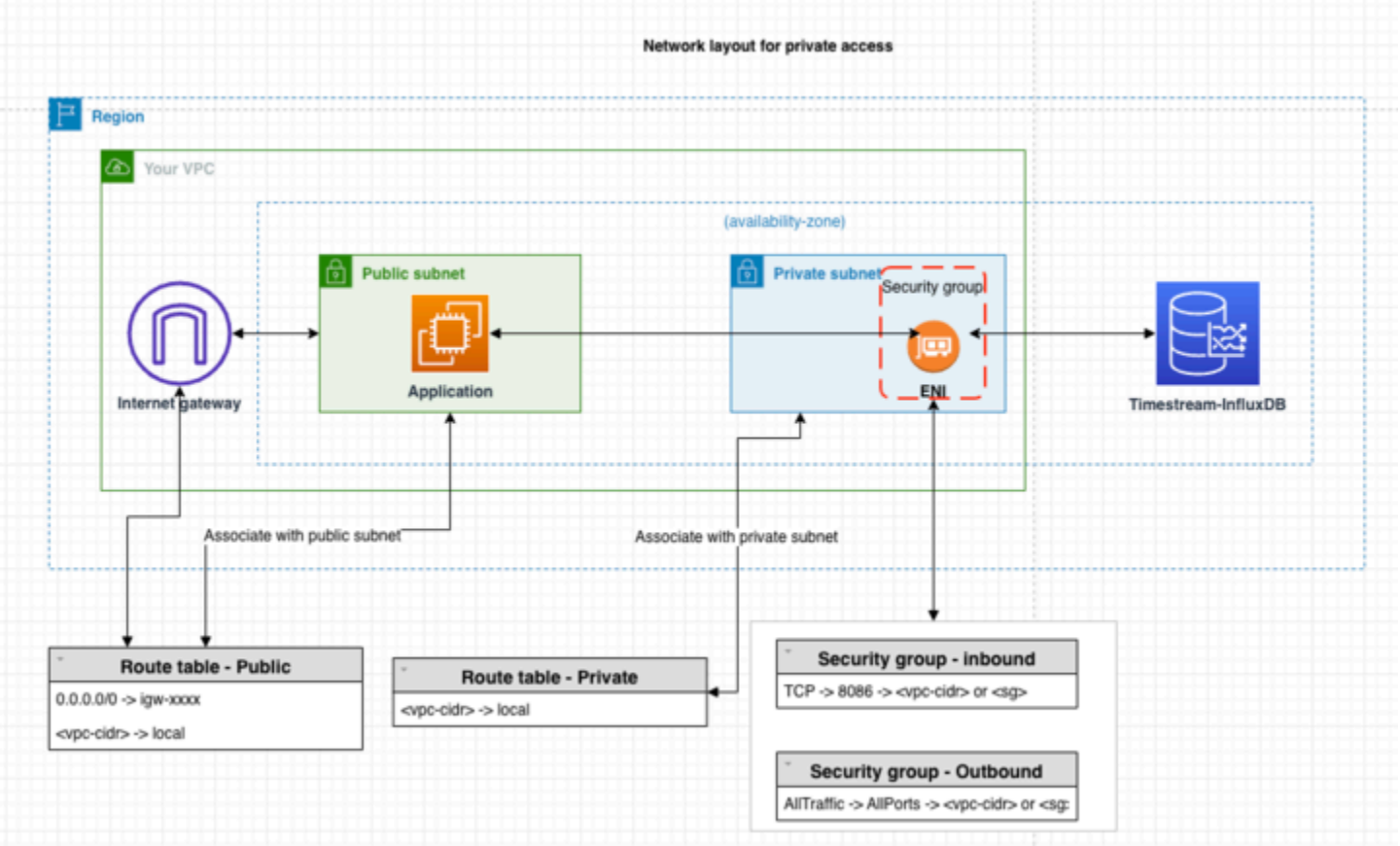
您可以設定的其他設定包括可用性、安全性和記錄。若要建立公有資料庫執行個體，您必須在連線組態區段中選擇讓執行個體「公有存取」。如需建立資料庫執行個體的相關資訊，請參閱 建立資料庫執行個體。

如果您的執行個體無法公開存取，請執行下列動作：

- 在執行個體VPC的 上建立主機，以便您透過它來通道流量。
- 將 SSH 通道設定為執行個體。如需詳細資訊，請參閱使用 AWS Systems Manager 轉送 Amazon EC2執行個體連接埠
- 若要讓憑證正常運作，請將以下行新增至用戶端機器`/etc/hosts`的檔案：`127.0.0.1`。這是執行個體DNS的地址。
- 使用完整網域名稱連線至執行個體，例如 https：//<DNS > ：8086。

> **Note**
>
> Localhost 無法驗證憑證，因為 localhost 不是憑證 的一部分SAN。

下圖顯示可存取性為私有時的組態：



必要條件

在開始之前，請先完成下節所含步驟：

- 註冊 AWS 帳戶。
- 建立管理使用者。

步驟 1：建立 Amazon EC2執行個體

建立您要用來連線至資料庫的 Amazon EC2執行個體。

1. 登入 AWS Management Console 並在 開啟 Amazon EC2主控台https://console.aws.amazon.com/ec2/。
2. 在 的右上角 AWS Management Console，選擇要 AWS 建立EC2執行個體的區域。
3. 選擇EC2儀表板 ，然後選擇啟動執行個體 。
4. 啟動執行個體頁面開啟時，請在啟動執行個體頁面上選擇下列設定。

    a. 在名稱和標籤下，針對名稱輸入 ec2-database-connect。

b. 在應用程式和作業系統映像（Amazon Machine Image）下，選擇 Amazon Linux，然後選擇 Amazon Linux 2023 AMI。保留其他選項的預設選擇。

c. 在Instance type (執行個體類型) 下，選擇 t2.micro。

d. 在 Key pair (login) (金鑰對 (登入)) 下，選擇 Key pair name (金鑰對名稱)，以使用現有金鑰對。若要為 Amazon EC2執行個體建立新的金鑰對，請選擇建立新的金鑰對，然後使用建立金鑰對視窗來建立金鑰對。如需建立新的金鑰對的詳細資訊，請參閱 Amazon EC2 適用於 Linux 執行個體的使用者指南 中的建立金鑰對。

e. 對於允許網路設定中的SSH流量，請選擇EC2執行個體的SSH連線來源。如果顯示的 IP 地址正確，您可以選擇我的 IP 進行SSH連線。否則，您可以使用 VPC Secure Shell（ ）來決定要用來連線至 中EC2執行個體的 IP 地址SSH。若要判斷公有 IP 地址，您可以在不同的瀏覽器視窗或索引標籤中使用 服務 https://checkip.amazonaws.com。IP 地址的範例為 192.0.2.1/32。在許多情況下，您可以透過網際網路服務供應商（ISP）或從防火牆後方連線，而不需要靜態 IP 地址。若是如此，請務必確定用戶端電腦所使用的 IP 地址範圍。

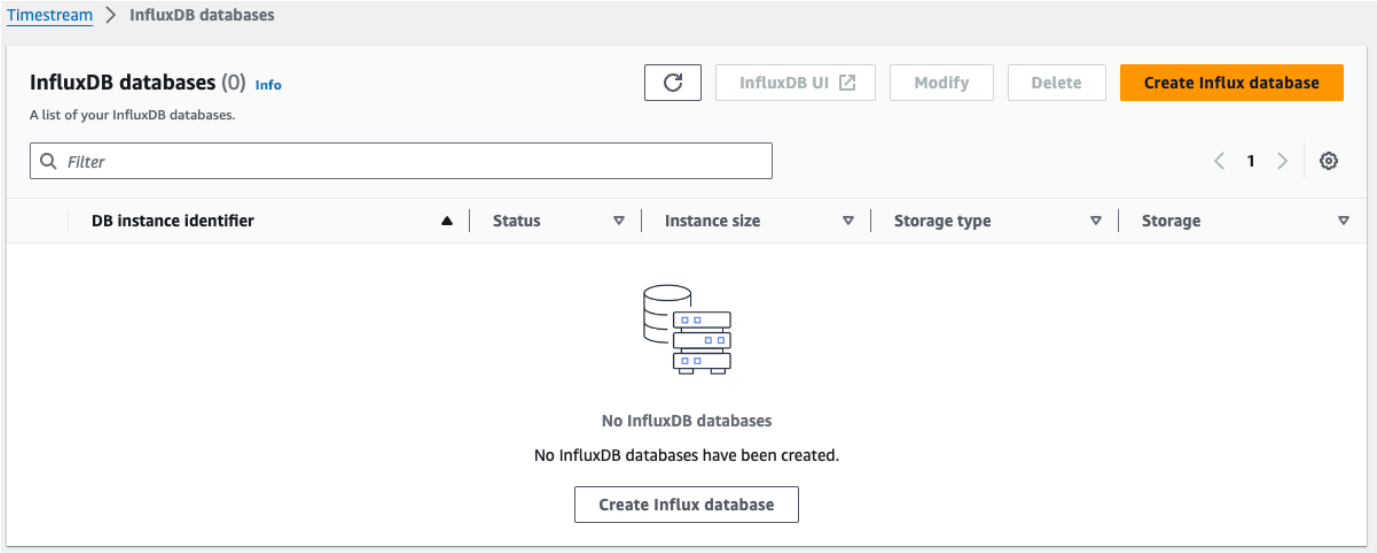
> **Warning**
>
> 我們不建議使用 0.0.0.0/0 進行SSH存取，因為您允許所有 IP 地址使用 存取您的公有EC2執行個體SSH。在測試環境中，此方法甚至在短時間內是不可接受的，請只授權特定 IP 地址或地址範圍來使用 存取您的EC2執行個體SSH。

步驟 2：建立 InfluxDB 資料庫執行個體

Amazon Timestream for InfluxDB 的基本建置區塊是資料庫執行個體。此環境可讓您執行 InfluxDB 資料庫。

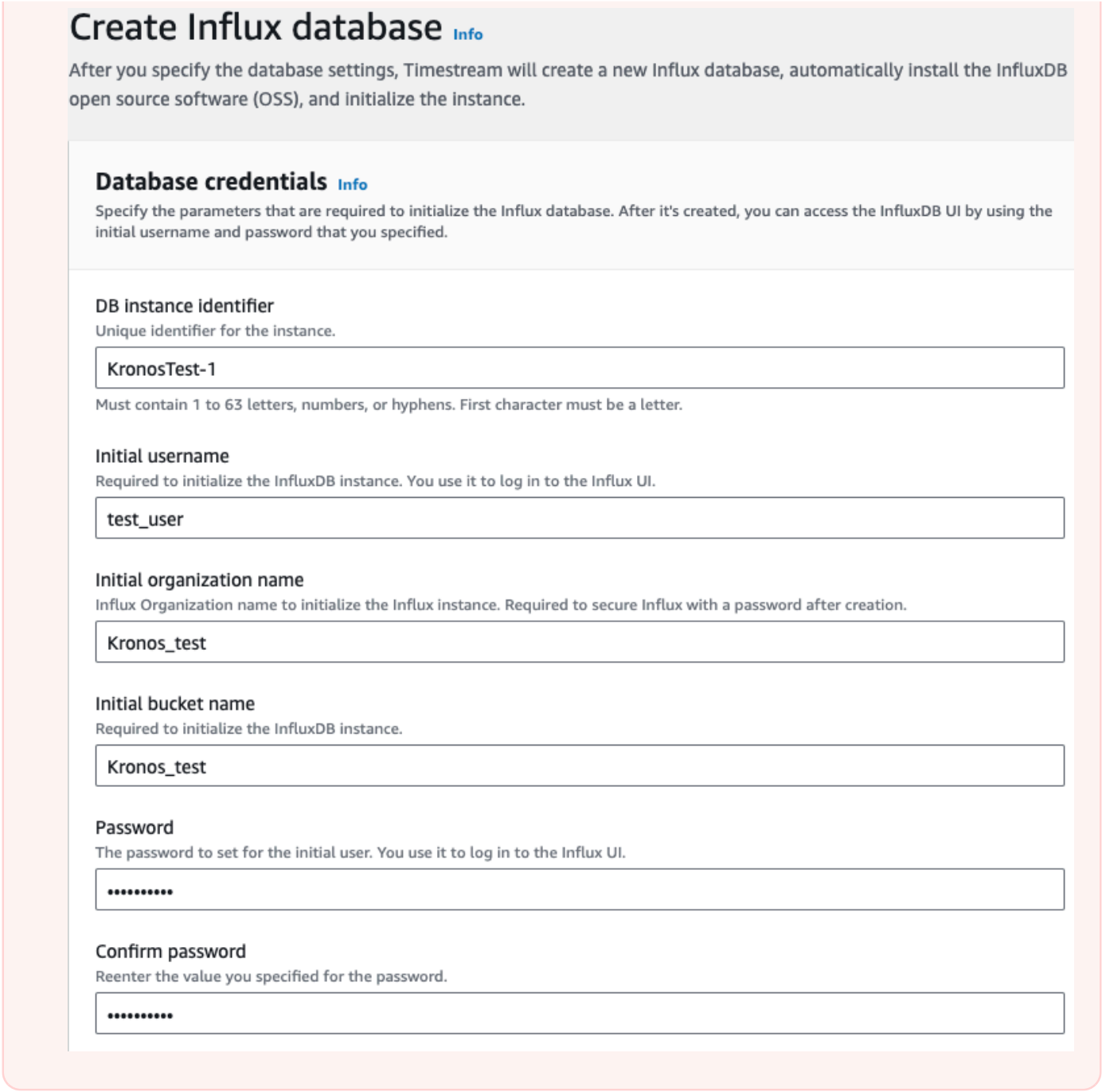
在此範例中，您將使用 db.influx.large 資料庫執行個體類別建立執行 InfluxDB 資料庫引擎的資料庫執行個體。

1. 登入 AWS Management Console ，並在 開啟 Amazon Timestream for InfluxDB 主控台https://console.aws.amazon.com/timestream/。
2. 在 Amazon Timestream for InfluxDB 主控台的右上角，選擇要 AWS 建立資料庫執行個體的區域。
3. 在導覽窗格中，選擇 InfluxDB Databases 。
4. 選擇建立 Influx 資料庫 。

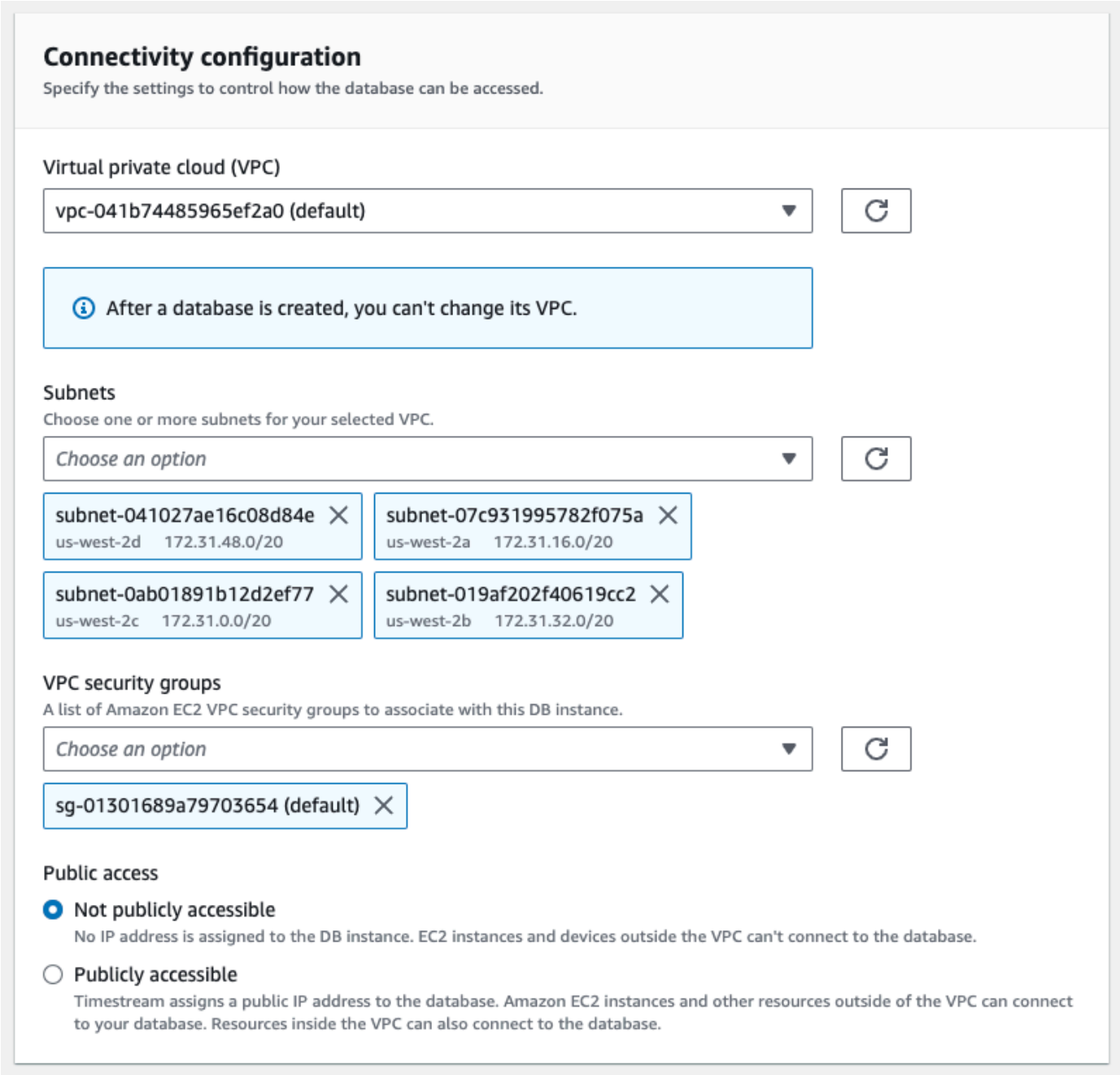


5. 對於資料庫執行個體識別符 ，輸入 KronosTest-1。
6. 提供 InfluxDB 基本組態參數：使用者名稱 、組織 、儲存貯體名稱 和密碼 。

> **Important**
>
> 您將無法再次檢視使用者密碼。如果沒有密碼，您將無法存取執行個體並取得運算子權杖。如果您沒有記錄下來，您可能需要進行變更。請參閱 為您的 InfluxDB 執行個體建立新的運算子權杖。
>
> 如果您需要在資料庫執行個體可用後變更使用者密碼，您可以修改資料庫執行個體以進行變更。如需修改 資料庫執行個體的詳細資訊，請參閱更新資料庫執行個體。



7. 針對資料庫執行個體類別 ，選取 db.influx.large 。
8. 針對資料庫儲存類別 ，選取包含 3K IOPS 的流入。
9. 設定您的日誌。如需詳細資訊，請參閱[設定以在 Timestream Influxdb 執行個體上檢視 InfluxDB 日誌]。
10. 在連線組態區段中，請確定您的 InfluxDB 執行個體與您新建立的EC2執行個體位於相同的子網路中。



11. 選擇建立 Influx 資料庫 。
12. 在資料庫清單中，選擇新 InfluxDB 執行個體的名稱以顯示其詳細資訊。資料庫執行個體在準備好使用之前的狀態為建立。

當狀態變更為可用時，您可以連線至資料庫執行個體。視資料庫執行個體類別和儲存體數量而定，可能需要最多 20 分鐘的時間，新執行個體才會可用。

> **Important**
>
> 目前，您無法修改現有執行個體的運算（執行個體類型）和儲存（儲存類型）組態。

步驟 3：將 Telegraf 資料傳送至您的 InfluxDB 執行個體

您現在可以開始使用 Telegraf 代理程式將遙測資料傳送至您的 InfluxDB 資料庫執行個體。在此範例中，您將安裝並設定 Telegraf 代理程式，以將效能指標傳送給 InfluxDB 資料庫執行個體。

1. 尋找資料庫執行個體的端點（DNS 名稱）和連接埠號碼。

   a. 登入 AWS 管理主控台，並在 開啟 Amazon Timestream 主控台https://console.aws.amazon.com/timestream/。

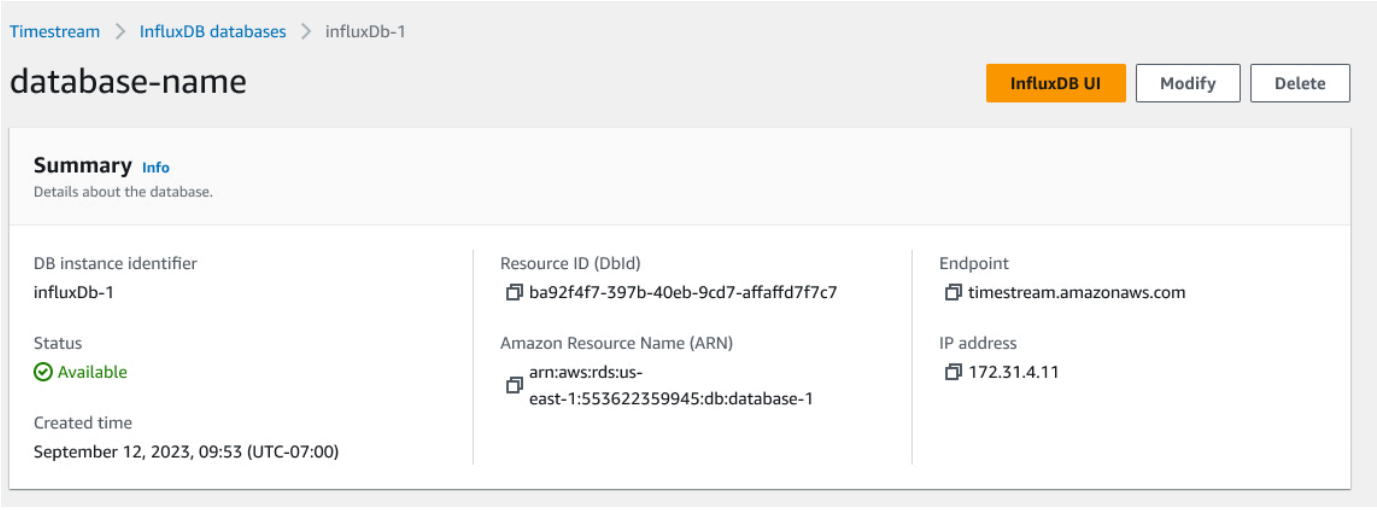
   b. 在 Amazon Timestream 主控台的右上角，選擇資料庫執行個體 AWS 的區域。

   c. 在導覽窗格中，選擇 InfluxDB Databases 。

   d. 選擇 InfluxDB 資料庫執行個體名稱以顯示其詳細資訊。

   e. 在摘要區段中，複製端點。另外，請記下連接埠號碼。您需要端點和連接埠號碼才能連線至資料庫執行個體（InfluxDB 的預設連接埠號碼為 8086）。

2. 接下來，選取 InfluxDB UI 。

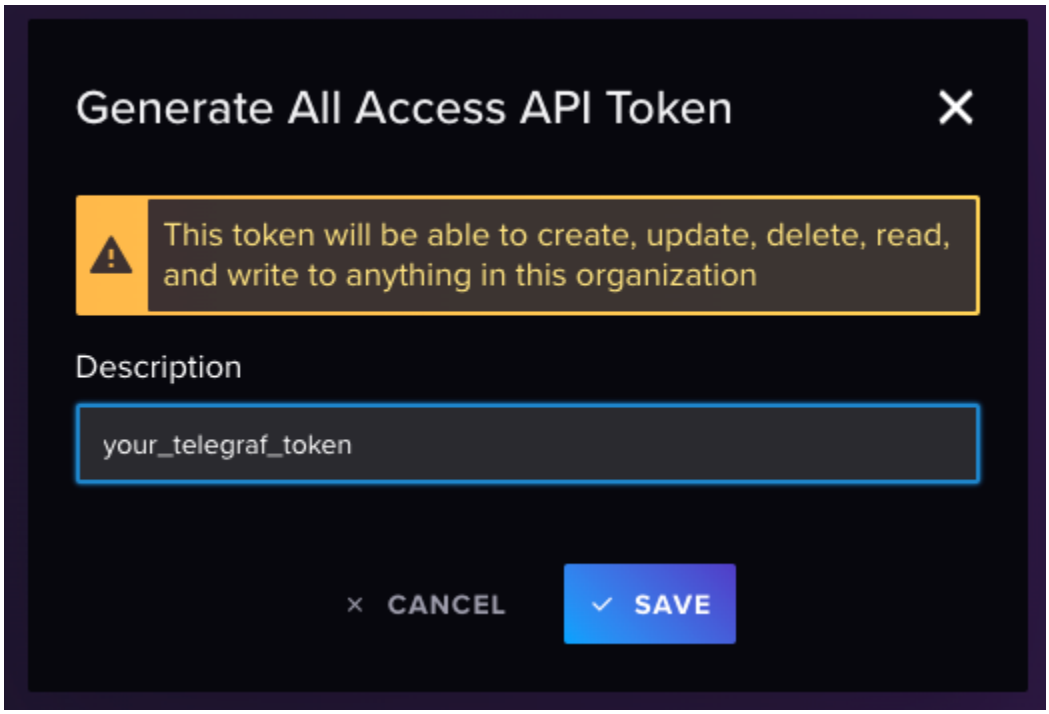


3. 這將開啟一個新的瀏覽器視窗，您應該會在其中看到登入提示。輸入您先前用來建立 InfluxDB Db 執行個體的憑證。

4. 在導覽窗格中，按一下箭頭，然後選取API權杖 。

5. 在此測試中， 會產生 All Access Token。

> **Note**
>
> 對於生產案例，我們建議您建立具有特定存取權的權杖，以存取為特定 Telegraf 需求建置的必要儲存貯體。



6. 您的權杖會顯示在畫面上。

> **Important**
>
> 請務必複製並儲存權杖，因為您將無法再次顯示權杖。

7. 請依照 Amazon 適用於 Linux EC2執行個體的EC2使用者指南 中的連線至 Linux 執行個體中的步驟，連線至您先前建立的執行個體。 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connect-to-linux-instance.html

   建議您使用 連線至EC2執行個體SSH。如果SSH用戶端公用程式安裝在 Windows、Linux 或 Mac 上，您可以使用下列命令格式連線至執行個體：

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

例如，假設 ec2-database-connect-key-pair.pem 存放在 Linux /dir1上的 ，且EC2執行個體IPv4DNS的公有 為 ec2-12-345-678-90.compute-1.amazonaws.com。您的SSH命令看起來如下所示：

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

8. 取得執行個體上安裝的最新版本 Telegraf。若要執行此操作，請使用下列命令：

```
cat <<EOF | sudo tee /etc/yum.repos.d/influxdata.repo
[influxdata]
name = InfluxData Repository - Stable
baseurl = https://repos.influxdata.com/stable/\$basearch/main
enabled = 1
gpgcheck = 1
gpgkey = https://repos.influxdata.com/influxdata-archive_compat.key
EOF

sudo yum install telegraf
```

9. 設定您的 Telegraf 執行個體。

> **Note**
>
> 如果 Telegraf.conf 不存在或包含區段，您可以產生具有下列內容的timestream區段：
>
> ```
> telegraf –section-filter agent:inputs:outputs –input-filter cpu:mem –output-
> filter timestream config > telegraf.conf
> ```

a. 編輯通常位於 的組態檔案/etc/telegraf。

```
sudo nano /etc/telegraf/telegraf.conf
```

b. 設定 CPU、 MEM和 的基本輸入DISK。

```
[[inputs.cpu]]
  percpu = true
  totalcpu = true
  collect_cpu_time = false
```

```
  report_active = false

[[inputs.mem]]

[[inputs.disk]]
  ignore_fs = ["tmpfs", "devtmpfs", "devfs"]
```

c. 設定輸出外掛程式，將資料傳送至您的 InfluxDB 資料庫執行個體，並儲存您的變更。

```
[[outputs.influxdb_v2]]
   urls = ["https://us-west-2-1.aws.cloud2.influxdata.com"]
   token = "<your_telegraf_token"
   organization = "your_org"
   bucket = "your_bucket"
   timeout = "5s"
```

d. 設定 Timestream 目標。

```
# Configuration for sending metrics to Amazon Timestream.
[[outputs.timestream]]

  ## Amazon Region and credentials
  region = "us-east-1"
  access_key = "<AWS key here>"
  secret_key = "<AWS secret key here>"
  database_name = "<timestream database name>" # needs to exist

  ## Specifies if the plugin should describe t start.
  describe_database_on_start = false
  mapping_mode = "multi-table" # allows multible tables for each input metrics

  create_table_if_not_exists = true
  create_table_magnetic_store_retention_period_in_days = 365
  create_table_memory_store_retention_period_in_hours = 24

  use_multi_measure_records = true # Important to use multi-measure records
  measure_name_for_multi_measure_records = "telegraf_measure"
  max_write_go_routines = 25
```

10. 啟用和啟動 Telegraf 服務。

```
$ sudo systemctl enable telegraf
$ sudo systemctl start telegraf
```

步驟 4：刪除 Amazon EC2執行個體和 InfluxDB 資料庫執行個體

使用 InfluxUI 的 InfluxDB 資料庫執行個體探索 Telegraf 產生的資料後，請同時刪除您的 EC2和 InfluxDB 資料庫執行個體，以便您不再需要支付這些執行個體的費用。 InfluxUI

若要刪除EC2執行個體：

1. 登入 AWS Management Console 並在 開啟 Amazon EC2主控台https://console.aws.amazon.com/ec2/。
2. 在導覽窗格中，選擇 Instances (執行個體)。
3. 選取EC2執行個體，選擇執行個體狀態 ，然後終止執行個體 。
4. 出現確認提示時，請選擇 Terminate (終止)。

如需刪除EC2執行個體的詳細資訊，請參閱 Amazon EC2使用者指南中的終止執行個體。

若要刪除沒有最終資料庫快照的資料庫執行個體：

1. 登入 AWS Management Console 並在 開啟 Amazon Timestream for InfluxDB 主控台https://console.aws.amazon.com/timestream/。
2. 在導覽窗格中，選擇 InfluxDB Databases 。
3. 選擇您要刪除的資料庫執行個體。
4. 對於 Actions (動作)，請選擇 Delete (刪除)。
5. 完成確認，然後選擇刪除。

（選用）使用 Amazon Managed Grafana 連線至資料庫執行個體

您可以使用 Amazon Managed Grafana 建立儀表板，並使用 Amazon Timestream for InfluxDB 監控EC2執行個體的效能。Amazon Managed Grafana 是 Grafana 的全面受管服務，這是熱門的開放原始碼分析平台，可讓您查詢、視覺化和警示指標、日誌和追蹤。

為您的 InfluxDB 執行個體建立新的運算子權杖

如果您需要取得新 InfluxDB 執行個體的運算子權杖，請執行下列步驟：

1. 若要變更您的運算子權杖，建議您使用 Influx CLI。如需說明，請參閱：安裝和使用 流入 CLI。
2. 將 CLI 設定為使用 `--username-password` 以建立運算子：

```
influx config create --config-name CONFIG_NAME1  --host-url "https://
yourinstanceid.eu-central-1.timestream-influxdb.amazonaws.com:8086" --org [YOURORG]
  --username-password [YOURUSERNAME] --active
```

3. 建立新的運算子權杖。系統會要求您提供密碼以確認此步驟。

```
influx auth create --org [YOURORG] --operator
```

> **Important**
>
> 建立新的運算子權杖後，您將需要更新目前使用舊權杖的任何用戶端。

從自我管理的 InfluxDB 將資料遷移至 InfluxDB 的 Timestream

Influx 遷移指令碼是 Python 指令碼，可在 InfluxDB OSS執行個體之間遷移資料，無論這些執行個體 AWS 是否由 管理。

InfluxDB 是時間序列資料庫。InfluxDB 包含點 ，其中包含數個鍵值對和時間戳記。當點依鍵值對分組時，它們會形成序列。系列由字串識別符分組，稱為測量 。InfluxDB 通常用於操作監控、IOT 資料和分析。儲存貯體是 InfluxDB 中儲存資料的容器。 AWS受管 InfluxDB 是 AWS 生態系統中的 InfluxDB。InfluxDB 提供 InfluxDB v2，API用於存取資料和變更資料庫。InfluxDB v2 API 是 Influx 遷移指令碼用來遷移資料的內容。

- Influx 遷移指令碼可以遷移儲存貯體及其中繼資料、從所有組織遷移所有儲存貯體，或執行完整遷移，以取代目的地執行個體上的所有資料。
- 指令碼會在任何系統執行指令碼的本機備份來源執行個體的資料，然後將資料還原至目的地執行個體。資料會保留在 code>influxdb-backup-<timestamp></timestamp> 目錄中，每次遷移各一個。
- 指令碼提供許多選項和組態，包括掛載 S3 儲存貯體以在遷移期間限制本機儲存用量，以及選擇要在遷移期間使用的組織。

主題

- 準備
- 如何使用指令碼
- 遷移概觀

準備

InfluxDB 的資料遷移使用 Python 指令碼完成，該指令碼使用 InfluxDB CLI功能和 InfluxDB v2 API。遷移指令碼的執行需要下列環境組態：

- 支援的版本：CLI支援最低 2.3 版本的 InfluxDB 和 Influx。
- 權杖環境變數
  - 建立`INFLUX_SRC_TOKEN`包含來源 InfluxDB 執行個體權杖的環境變數。
  - 建立`INFLUX_DEST_TOKEN`包含目的地 InfluxDB 執行個體字符的環境變數。
- Python 3
  - 檢查安裝：`python3 --version`。
  - 如果未安裝，請從 Python 網站安裝 。需要最低 3.7 版。在 Windows 上，預設 Python 3 別名只是 python。
  - Python 模組請求為必要項目。使用下列工具安裝： `shell python3 -m pip install requests`
  - TThe Python 模組 influxdb_client 為必填。使用下列工具安裝： `shell python3 -m pip install influxdb_client`
- InfluxDB CLI
  - 確認安裝：`influx version`。
  - 如果未安裝，請遵循 InfluxDB 文件 中的安裝指南。

    將輸入新增至 $PATH。
- S3 安裝工具（選用）

  使用 S3 掛載時，所有備份檔案都會存放在使用者定義的 S3 儲存貯體中。S3 掛載有助於節省執行中機器或需要共用備份檔案時的空間。如果未使用 S3 掛載，則省略 `--s3-bucket`選項，就會建立本機`influxdb-backup-<millisecond timestamp>`目錄，將備份檔案存放在執行指令碼的相同目錄中。

  對於 Linux： mountpoint-s3。

  對於 Windows：rclone （需要先前的 rclone 組態）。
- 磁碟空間
  - 遷移程序會自動建立唯一的目錄，以儲存一組備份檔案，並根據提供的程式引數將這些備份目錄保留在 S3 或本機檔案系統上。

  - 確保有足夠的磁碟空間進行資料庫備份，如果您選擇省略 `--s3-bucket`選項並使用本機儲存進行備份和還原，最好是現有 InfluxDB 資料庫的兩倍大小。
  - 在 Windows 上檢查磁碟機屬性，以 `df -h` (UNIX/Linux)或 檢查空間。
- 直接連線

  確保執行遷移指令碼的系統與來源和目的地系統之間存在直接網路連線。 `influx ping --host <host>`是驗證直接連線的一種方法。

如何使用指令碼

執行指令碼的簡單範例是 命令：

```
python3 influx_migration.py --src-host <source host> --src-bucket <source bucket> --
dest-host <destination host>
```

會遷移單一儲存貯體。

所有選項都可以透過執行下列動作來檢視：

```
python3 influx_migration.py -h
```

用途

```
shell   influx_migration.py [-h] [--src-bucket SRC_BUCKET] [--dest-bucket DEST_BUCKET]
 [--src-host SRC_HOST] --dest-host DEST_HOST [--full] [--confirm-full] [--src-org
 SRC_ORG] [--dest-org DEST_ORG] [--csv] [--retry-restore-dir RETRY_RESTORE_DIR] [--dir-
name DIR_NAME] [--log-level LOG_LEVEL] [--skip-verify] [--s3-bucket S3_BUCKET]
```

選項

- -confirm-full （選用）：使用 `--full` 而不 `--csv`會將目的地資料庫中的所有權杖、使用者、儲存貯體、儀表板和任何其他鍵值資料取代為來源資料庫中的權杖、使用者、儲存貯體、儀表板和任何其他鍵值資料。 `--full --csv`僅遷移所有儲存貯體和儲存貯體中繼資料，包括儲存貯體組織。此選項（`--confirm-full`）將確認完全遷移，並在不輸入使用者的情況下繼續進行。如果未提供此選項，且`--full`已提供且未`--csv`提供此選項，則指令碼將暫停執行，並等待使用者確認。這是關鍵動作，請謹慎處理。預設為 false。

- -csv （選用）：是否要使用 csv 檔案進行備份和還原。如果 同樣--full通過，則會遷移所有組織中所有使用者定義的儲存貯體，而不是系統儲存貯體、使用者、權杖或儀表板。如果目的地伺服器中的所有儲存貯體都需要單一組織，而不是其現有的來源組織，請使用 --dest-org。
- -dest-bucket DEST_BUCKET （選用）：目的地伺服器中 InfluxDB 儲存貯體的名稱不得為現有的儲存貯體。如果--src-bucket未提供 None，則預設為 --src-bucket或 的值。
- -dest-host DEST_HOST：目的地伺服器的主機。範例：http://localhost:8086.
- -dest-org DEST_ORG （選用）：在目的地伺服器中將儲存貯體還原至 的組織名稱。如果省略這種情況，則來源伺服器的所有已遷移儲存貯體都會保留其原始組織，而且在未建立和切換組織的情況下，遷移的儲存貯體可能不會出現在目的地伺服器中。無論單一儲存貯體、完整遷移還是使用 csv 檔案進行備份和還原的任何遷移，此值都會用於所有形式的還原。
- -dir-name DIR_NAME （選用）：要建立的備份目錄名稱。預設為 influxdb-backup-<timestamp>。不得已存在。
- -full （選用）：是否要執行完整還原，請將目的地伺服器上的所有資料取代為來自所有組織的來源伺服器的所有資料，包括所有鍵值資料，例如權杖、儀表板、使用者等。覆寫 --src-bucket和 --dest-bucket。如果與 搭配使用--csv， 只會遷移儲存貯體的資料和中繼資料。預設為 false。
- h， --help ：顯示說明訊息和結束。
- -log-level LOG_LEVEL（選用）：執行期間要使用的日誌層級。選項是偵錯、錯誤和資訊。預設為資訊。
- -retry-restore-dir RETRY_RESTORE_DIR（選用）：當先前的還原失敗時用於還原的目錄，會略過備份和目錄建立，如果目錄不存在，則會失敗，可以是 S3 儲存貯體中的目錄。如果還原失敗，則會根據目前的目錄指示可用於還原的備份目錄路徑。S3 儲存貯體的格式為 influxdb-backups/<s3 bucket>/<backup directory>。預設備份目錄名稱為 influxdb-backup-<timestamp>。
- -s3-bucket S3_BUCKET（選用）：用於儲存備份檔案的 S3 儲存貯體名稱。在 Linux 上，這只是 S3 儲存貯體的名稱，例如 my-bucket、指定的AWS_ACCESS_KEY_ID和AWS_SECRET_ACCESS_KEY環境變數已設定或${HOME}/.aws/credentials存在。在 Windows 上，這是rclone設定的遠端和儲存貯體名稱，例如 my-remote:my-bucket。在建立的influxdb-backups-<timestamp>目錄中遷移後，所有備份檔案都會保留在 S3 儲存貯體中。名為 的臨時掛載目錄influx-backups將在執行此指令碼的目錄中建立。如果未提供，則所有備份檔案都會儲存在本機中執行此指令碼的建立influxdb-backups-<timestamp>目錄中。
- -略過-驗證 （選用）：略過TLS憑證驗證。
- -src-bucket SRC_BUCKET（選用）：來源伺服器中 InfluxDB 儲存貯體的名稱。如果未提供，--full則必須提供 。
- -src-host SRC_HOST（選用）：來源伺服器的主機。預設為 http://localhost:8086.

如前所述，如果--s3-bucket要使用 mountpoint-s3，rclone則需要 ，但如果使用者未提供 的值，則可以忽略 --s3-bucket，在這種情況下，備份檔案將儲存在本機的唯一目錄中。

遷移概觀

符合先決條件之後：

1. 執行遷移指令碼：使用您選擇的終端應用程式，執行 Python 指令碼，將資料從來源 InfluxDB 執行個體傳輸至目的地 InfluxDB 執行個體。
2. 提供憑證：提供主機地址和連接埠作為CLI選項。
3. 驗證資料：確保資料正確傳輸者：

    a. 使用 InfluxDB UI 和檢查儲存貯體。

    b. 使用 列出儲存貯體`influx bucket list -t <destination token> --host <destination host address> --skip-verify`。

    c. 使用 `influx v1 shell -t <destination token> --host <destination host address> --skip-verify`和 執行 `SELECT * FROM <migrated bucket>.<retention period>.<measurement name> LIMIT 100` to view contents of a bucket or `SELECT COUNT(*) FROM <migrated bucket>.<retention period>.<measurment name>`來驗證已遷移的正確記錄數目。

Example 範例執行

1. 開啟您選擇的終端應用程式，並確保正確安裝必要的先決條件：

```
~ > python3 --version
Python 3.11.5
~ > influx version
Influx CLI 2.7.3 (git: 8b962c7e75) build_date: 2023-04-28T14:22:49Z
~ > s3fs --version
Amazon Simple Storage Service File System V1.92 (commit:unknown) with GnuTLS(gcrypt)
Copyright (C) 2010 Randy Rizun <rrizun@gmail.com>
License GPL2: GNU GPL version 2 <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
~ >
```

2. 導覽至遷移指令碼：

```
~ > cd sample-code/influxdb-sample/migration/influxdb
~/sample-code/influxdb-sample/migration/influxdb > ls *.py
influx_migration.py
~/sample-code/influxdb-sample/migration/influxdb >
```

3. 準備下列資訊：

   a. 要遷移的來源儲存貯體名稱。

   b. （選用）為目的地伺服器中的已遷移儲存貯體選擇新的儲存貯體名稱。

   c. 來源和目的地流入執行個體的根權杖。

   d. 來源和目的地流入執行個體的主機地址。

   e. （選用）S3 儲存貯體名稱和憑證； AWS Command Line Interface 憑證應在作業系統環境變數中設定。

```
# AWS credentials (for timestream testing)
    export AWS_ACCESS_KEY_ID="xxx"
    export AWS_SECRET_ACCESS_KEY="xxx"
```

   f. 將命令建構為：

```
python3 influx_migration.py --src-bucket [source-bucket-name]  --dest-bucket
 [dest-bucket-name] --src-host [source host] --dest-host [dest host] --s3-
bucket [s3 bucket name](optional) --log-level debug
```

   g. 執行指令碼：

```
~/sample-code/influxdb-sample/migration/influxdb > python3 influx_migration.py --src-bucket primary-bucket --src-host $INFLUXDB_1_HOST --dest-host $KRO
NOS_HOST --dest-bucket new-bucket-name
```

   h. 等待指令碼完成執行。

   i. 檢查新遷移的儲存貯體的資料完整性，`performance.txt`。此檔案位於執行指令碼的相同目錄下，包含每個步驟所採取時間的一些基本資訊。

遷移案例

Example 範例 1：使用本機儲存的簡易遷移

您想要將單一儲存貯體，主要儲存貯體，從來源伺服器遷移(`http://localhost:8086`)至目的地伺服器 (`http://dest-server-address:8086`)。

在確定您能夠TCP存取（用於HTTP存取）在連接埠 8086 上託管 InfluxDB 執行個體的這兩部機器，並且您同時擁有來源和目的地權杖，並分別將其儲存為環境變數 INFLUX_SRC_TOKEN和 INFLUX_DEST_TOKEN，以增強安全性：

```
python3 influx_migration.py --src-bucket primary-bucket --src-host http://
localhost:8086 --dest-host http://dest-server-address:8086
```

輸出格式應類似以下內容：

```
INFO: influx_migration.py: Backing up bucket data and metadata using the InfluxDB CLI
2023/10/26 10:47:15 INFO: Downloading metadata snapshot
2023/10/26 10:47:15 INFO: Backing up TSM for shard 1
2023/10/26 10:47:15 INFO: Backing up TSM for shard 8245
2023/10/26 10:47:15 INFO: Backing up TSM for shard 8263
[More shard backups . . .]
2023/10/26 10:47:20 INFO: Backing up TSM for shard 8240
2023/10/26 10:47:20 INFO: Backing up TSM for shard 8268
2023/10/26 10:47:20 INFO: Backing up TSM for shard 2
INFO: influx_migration.py: Restoring bucket data and metadata using the InfluxDB CLI
2023/10/26 10:47:20 INFO: Restoring bucket "96c11c8876b3c016" as "primary-bucket"
2023/10/26 10:47:21 INFO: Restoring TSM snapshot for shard 12772
2023/10/26 10:47:22 INFO: Restoring TSM snapshot for shard 12773
[More shard restores . . .]
2023/10/26 10:47:28 INFO: Restoring TSM snapshot for shard 12825
2023/10/26 10:47:28 INFO: Restoring TSM snapshot for shard 12826
INFO: influx_migration.py: Migration complete
```

目錄influxdb-backup-<timestamp>將建立並儲存在執行指令碼的目錄中，其中包含備份檔案。

Example 範例 2：使用本機儲存和偵錯記錄進行完整遷移

與上述相同，除了您要遷移所有儲存貯體、權杖、使用者和儀表板、刪除目的地伺服器中的儲存貯體，以及使用 --confirm-full選項繼續進行，而無需使用者確認完整的資料庫遷移。您也想要查看什麼是效能測量，以便啟用偵錯記錄。

```
python3 influx_migration.py --full --confirm-full --src-host http://localhost:8086 --
dest-host http://dest-server-address:8086 --log-level debug
```

輸出格式應類似以下內容：

```
INFO: influx_migration.py: Backing up bucket data and metadata using the InfluxDB CLI
```

```
2023/10/26 10:55:27 INFO: Downloading metadata snapshot
2023/10/26 10:55:27 INFO: Backing up TSM for shard 6952
2023/10/26 10:55:27 INFO: Backing up TSM for shard 6953
[More shard backups . . .]
2023/10/26 10:55:36 INFO: Backing up TSM for shard 8268
2023/10/26 10:55:36 INFO: Backing up TSM for shard 2
DEBUG: influx_migration.py: backup started at 2023-10-26 10:55:27 and took 9.41 seconds
 to run.
INFO: influx_migration.py: Restoring bucket data and metadata using the InfluxDB CLI
2023/10/26 10:55:36 INFO: Restoring KV snapshot
2023/10/26 10:55:38 WARN: Restoring KV snapshot overwrote the operator token, ensure
 following commands use the correct token
2023/10/26 10:55:38 INFO: Restoring SQL snapshot
2023/10/26 10:55:39 INFO: Restoring TSM snapshot for shard 6952
2023/10/26 10:55:39 INFO: Restoring TSM snapshot for shard 6953
[More shard restores . . .]
2023/10/26 10:55:49 INFO: Restoring TSM snapshot for shard 8268
2023/10/26 10:55:49 INFO: Restoring TSM snapshot for shard 2
DEBUG: influx_migration.py: restore started at 2023-10-26 10:55:36 and took 13.51
 seconds to run.
INFO: influx_migration.py: Migration complete
```

Example 範例 3：使用 CSV、目的地組織和 S3 儲存貯體進行完整遷移

與上一個範例相同，但使用 Linux 或 Mac 並將檔案儲存在 S3 儲存貯體 中`my-s3-bucket`。這可避免備份檔案超過本機儲存容量。

```
python3 influx_migration.py --full --src-host http://localhost:8086 --dest-host http://
dest-server-address:8086 --csv --dest-org MyOrg --s3-bucket my-s3-bucket
```

輸出格式應類似以下內容：

```
INFO: influx_migration.py: Creating directory influxdb-backups
INFO: influx_migration.py: Mounting influxdb-migration-bucket
INFO: influx_migration.py: Creating directory influxdb-backups/my-s3-bucket/influxdb-
backup-1698352128323
INFO: influx_migration.py: Backing up bucket data and metadata using the InfluxDB v2
 API
INFO: influx_migration.py: Restoring bucket data and metadata from csv
INFO: influx_migration.py: Restoring bucket some-bucket
INFO: influx_migration.py: Restoring bucket another-bucket
INFO: influx_migration.py: Restoring bucket primary-bucket
```

```
INFO: influx_migration.py: Migration complete
INFO: influx_migration.py: Unmounting influxdb-backups
INFO: influx_migration.py: Removing temporary mount directory
```

設定資料庫執行個體

本節說明如何設定 Amazon Timestream for InfluxDB 資料庫執行個體。建立一個資料庫執行個體之前，決定將執行資料庫執行個體的資料庫執行個體類別。此外，選擇 AWS 區域來決定資料庫執行個體的執行位置。接下來，建立資料庫執行個體。

您可以使用資料庫參數群組來設定資料庫執行個體。資料庫參數群組可充當套用到一或多個資料庫執行個體的引擎組態值的容器。

可用的參數取決於資料庫引擎和資料庫引擎版本。您可以在建立資料庫執行個體時指定資料庫參數群組。您也可以修改資料庫執行個體來指定它們。

> **Important**
>
> 目前，您無法修改現有執行個體的運算（執行個體類型）和儲存（儲存類型）組態。

建立資料庫執行個體

使用主控台

1. 登入 AWS Management Console 並開啟 Amazon Timestream for InfluxDB 。
2. 在 Amazon Timestream for InfluxDB 主控台的右上角，選擇要 AWS 建立資料庫執行個體的區域。
3. 在導覽窗格中，選擇 InfluxDB Databases 。
4. 選擇建立輸入資料庫 。
5. 針對資料庫執行個體識別碼 。輸入可識別執行個體的名稱。
6. 提供 InfluxDB 基本組態參數 使用者名稱、組織、儲存貯體名稱和密碼 。

> **Important**
>
> 您的使用者名稱、組織、儲存貯體名稱和密碼將儲存為 AWS Secrets Manager 中的機密，該機密將針對您的帳戶建立。

如果您需要在資料庫執行個體可用後變更使用者密碼，您可以使用 Influx CLI進行修改。

7.

8. 針對資料庫執行個體類別 ，選取更符合您工作負載需求的執行個體大小。
9. 針對資料庫儲存類別 ，選取符合您需求的儲存類別。在所有情況下，您只需要設定配置的儲存體。
10. 在連線組態區段中，請確定您的 InfluxDB 執行個體與需要連線至 Timestream for InfluxDB 資料庫執行個體的新用戶端位於相同的子網路中。您也可以選擇公開資料庫執行個體。
11. 選擇建立 Influx 資料庫 。
12. 在資料庫清單中，選擇新 InfluxDB 執行個體的名稱以顯示其詳細資訊。資料庫執行個體的狀態為建立，直到準備好使用為止。
13. 狀態變更為 Available (可用) 時，您便能連線至資料庫執行個體。視資料庫執行個體類別和儲存體數量而定，可能需要最多 20 分鐘的時間，新執行個體才會可用。

使用 CLI

若要使用 建立資料庫執行個體 AWS Command Line Interface，請使用下列參數呼叫 `create-db-instance`命令：

```
--name
--vpc-subnet-ids
--vpc-security-group-ids
--db-instance-type
--db-storage-type
--username
--organization
--password
--allocated-storage
```

如需每項設定的相關資訊，請參閱 資料庫執行個體的設定。

Example 範例：使用預設引擎組態

若為 Linux、macOS 或 Unix：

```
aws timestream-influxdb create-db-instance \
    --name myinfluxDbinstance \
```

```
    --allocated-storage 400 \
    --db-instance-type db.influx.4xlarge \
    --vpc-subnet-ids subnetid1 subnetid2
    --vpc-security-group-ids mysecuritygroup \
    --username masterawsuser \
    --password \
    --db-storage-type InfluxIOIncludedT2
```

針對 Windows：

```
aws timestream-influxdb create-db-instance \
    --name myinfluxDbinstance \
    --allocated-storage 400 \
    --db-instance-type db.influx.4xlarge \
    --vpc-subnet-ids subnetid1 subnetid2
    --vpc-security-group-ids mysecuritygroup \
    --username masterawsuser \
    --password \
    --db-storage-type InfluxIOIncludedT2
```

使用 API

若要使用 建立資料庫執行個體 AWS Command Line Interface，請使用下列參數呼叫 `CreateDBInstance`命令：

如需每項設定的相關資訊，請參閱 資料庫執行個體的設定。

> **Important**
>
> 您收到的DBInstance回應物件的一部分 influxAuthParametersSecretArn。這會將 保留ARN 在帳戶中的 SecretsManager 秘密。只有在您的 InfluxDB 資料庫執行個體可用後才會填入。秘密包含`CreateDbInstance`程序期間提供的輸入身分驗證參數。這是複本，因為updates/modifications/deletions此秘密的任何 READONLY 不會影響建立的資料庫執行個體。如果您刪除此秘密，我們的API回應仍會參考已刪除的秘密 ARN。

建立 Timestream for InfluxDB 資料庫執行個體後，建議您下載、安裝和設定 Influx CLI。

Influx CLI提供從命令列與 InfluxDB 互動的簡單方法。如需詳細的安裝和設定指示，請參閱使用 Influx CLI。

資料庫執行個體的設定

您可以使用主控台、 `create-db-instance`CLI命令或 `CreateDBInstance` Timestream for InfluxDB API操作來建立資料庫執行個體。

下表提供您在建立資料庫執行個體時選擇之設定的詳細資訊。

主控台設定	描述	CLI 選項和 Timestream API 參數
配置儲存	要配置給資料庫執行個體的儲存量 (以 GiB 為單位)。在部分情況下，配置給資料庫執行個體的儲存空間容量若高於資料庫的大小，可改善輸入/輸出效能。 如需詳細資訊，請參閱InfluxDB 執行個體儲存。	CLI: `allocated-storage` API: `allocated storage`
儲存貯體名稱	儲存貯體用於初始化 InfluxDb 執行個體的名稱	CLI: `bucket` API: `bucket`
資料庫執行個體類型	資料庫執行個體的組態。例如，db.influx.large 資料庫執行個體類別具有 16 GiB 記憶體、2 vCPUs、記憶體最佳化。 如果可能，請選擇資料庫執行個體類型，其大小足以讓一般查詢工作集保留在記憶體中。當工作集保留在記憶體中時，系統可以避免寫入至磁碟，因而可改善效能。如需詳細資訊，請參閱資料庫執行個體類別的類型。	CLI: `db-instance-type` API: `Dbinstancetype`
DB instance identifier (資料庫執行個體識別符) :	資料庫執行個體的名稱。以您命名現場部署伺服器的相同方式，命名您的資料庫執行個體。您的資料庫執行個體識別符最多可包含 63 個英數字元，且對於您選擇的 AWS 區域中的帳戶而言必須是唯一的。	CLI: `db-instance-identifier` API: `Dbinstanceidentifier`
DB parameter group (資料庫參數群組)	資料庫執行個體的參數群組。您可以選擇預設參數群組，或可以建立自訂參數群組。	CLI: `db-parameter-group-name`

主控台設定	描述	CLI 選項和 Timestream API 參數
	如需詳細資訊，請參閱 使用資料庫參數群組。	API: `DBParameterGroupName`
日誌傳遞設定	S3 儲存貯體的名稱是將儲存 InfluxDB 日誌。	CLI: `LogDeliveryConfiguration` API: `log-delivery-configuration`
Multi-AZ deployment (異地同步備份部署)	Create a standby instance (建立待命執行個體)，在另一個可用區域中建立資料庫執行個體的被動次要複本，以提供容錯移轉支援。我們建議針對生產工作負載使用 Multi-AZ 以維持高可用性。 針對開發和測試，您可以選擇 Do not create a standby instance (不要建立待命執行個體)。 如需詳細資訊，請參閱設定和管理多可用區部署。	CLI: `MultiAz` API: `multi-az`
密碼	這將是您主要使用密碼來初始化 InfluxDB Db 執行個體。您將使用此密碼登入 InfluxUI，以取得您的運算子權杖。	CLI: `password` API: `password`
公有存取	是，為資料庫執行個體提供公有 IP 地址，這表示其可在 外部存取VPC。若要公開存取，資料庫執行個體也必須位於 中的公有子網路中VPC。 否，讓資料庫執行個體只能從 內部存取VPC。 若要從其 外部連線至資料庫執行個體VPC，資料庫執行個體必須可公開存取。亦須使用資料庫執行個體安全群組的傳入規則授予存取權。此外，必須滿足其他要求。	CLI: `publicly-accessible` API: `PubliclyAccessible`

主控台設定	描述	CLI 選項和 Timestream API 參數
儲存類型	資料庫執行個體的儲存體類型 您可以根據您的工作負載需求，選擇 3 種不同類型的佈建流入IOPS包含的儲存體： * 已包含 Influx IOPS 3000 IOPS * 包含 12000 IOPS 的 Influx IOPS * INflux IOPS 包含 16000 IOPS 如需詳細資訊，請參閱InfluxDB 執行個體儲存。	CLI: `db-storage-type` API: `DbStorageType`
初始使用者名稱	這將是初始化您的 InfluxDB 資料庫執行個體的主要使用者。您將使用此使用者名稱登入 InfluxUI，以取得您的運算子權杖。	CLI: `username` API: `Username`
子網	與此資料庫執行個體關聯的 vpc 子網路。	CLI: `vpc-subnet-ids` API: `VPCSubnetIds`
VPC 安全群組（防火牆）	要與資料庫執行個體建立關聯的安全群組。	CLI: `vpc-security-group-ids` API: `VPCSecurityGroupIds`

連線至 Amazon Timestream for InfluxDB 資料庫執行個體

您必須先建立資料庫執行個體，才能連線到資料庫執行個體。如需相關資訊，請參閱 建立資料庫執行個體。Amazon Timestream 佈建資料庫執行個體後，請使用 influxDB API、Influx CLI或任何相容的用戶端或公用程式，讓 InfluxDB 連線至資料庫執行個體。

主題

尋找 Amazon Timestream for InfluxDB 資料庫執行個體的連線資訊

資料庫執行個體的連線資訊包括其端點、連接埠、使用者名稱、密碼和有效的存取權杖，例如運算子或所有存取權杖。例如，對於 InfluxDB 資料庫執行個體， 假設端點值為 `influxdb1-123456789.us-east-1.timestream-influxdb.amazonaws.com`。在此情況下，連接埠值為 8086，資料庫使用者為 admin 。藉由此資訊，您可以在連線字串中指定下列值：

- 對於主機或主機名稱或DNS名稱，請指定 `influxdb1-123456789.us-east-1.timestream-influxdb.amazonaws.com`。
- 針對連接埠，指定 8086。
- 對於使用者，請指定 admin。
- 針對密碼，請指定您在建立資料庫執行個體時提供的 。

> **Important**
>
> 當您為 InfluxDB Db 執行個體建立 Timestream 時，您收到的DBInstance回應物件的一部分`influxAuthParametersSecretArn`。這將保留您帳戶中 SecretsManager 秘密的 。只有在您的 InfluxDB 資料庫執行個體可用後才會填入。秘密包含`CreateDbInstance`程序期間提供的輸入身分驗證參數。這是複本updates/modifications/deletions，因為此秘密的任何 READONLY 不會影響建立的資料庫執行個體。如果您刪除此秘密，我們的API回應仍會參考已刪除的秘密。

每個資料庫執行個體的端點都是唯一的，連接埠和使用者的值可能會有所不同。若要連線至資料庫執行個體，您可以使用 Influx CLI、 Influx API或相容於 InfluxDB 的任何用戶端。

若要尋找資料庫執行個體的連線資訊，請使用 AWS 管理主控台。您也可以使用 AWS Command Line Interface （ AWS CLI ） `describe-db-instances`命令或 Timestream-influxdb APIGetDBInstance操作。

使用 AWS Management Console

1. 登入 AWS Management Console 並開啟 Amazon Timestream 主控台 。
2. 在導覽窗格中，選擇 InfluxDB 資料庫以顯示資料庫執行個體的清單。
3. 選擇資料庫執行個體的名稱以顯示其詳細資訊。
4. 在摘要區段 上，複製端點。另外，請記下連接埠號碼。您需要同時有端點和連接埠號碼，才能連線至資料庫執行個體。

如果您需要尋找使用者名稱和密碼資訊，請選擇組態詳細資訊索引標籤，然後選擇 `influxAuthParametersSecretArn`以存取 Secrets Manager。

使用 CLI

- 若要使用 尋找 InfluxDB 資料庫執行個體的連線資訊 AWS CLI，請呼叫 get-db-instance 命令。在通話中，查詢資料庫執行個體 ID、端點、連接埠和 influxAuthParameters秘密。

  若為 Linux、macOS 或 Unix：

  ```
  aws timestream-influxdb get-db-instance --identifier id \
   --query "[name,endpoint,influxAuthParametersSecretArn]"
  ```

  針對 Windows：

  ```
  aws timestream-influxdb get-db-instance --identifier id \
   --query "[name,endpoint,influxAuthParametersSecretArn]"
  ```

  輸出內容應如下所示：若要存取使用者名稱資訊，您需要檢查 `InfluxAuthParameterSecret`。

  ```
  [
      [
          "mydb",
          "mydb-123456789012.us-east-1.timestream-influxdb.amazonaws.com",
          8086,
      ]
  ]
  ```

建立存取權杖

透過此資訊，您將能夠為執行個體連線，以擷取或建立您的存取權杖。有幾種方法可以達成此目標：

使用 CLI

1. 如果您尚未下載、安裝和設定 流入 CLI。
2. 設定您的 Influx CLI 組態時，請使用 `--username-password`進行驗證。

```
influx config create --config-name YOUR_CONFIG_NAME  --host-url "https://
yourinstance.timestream-influxdb.amazonaws.com:8086"  --org yourorg --username-
password admin --active
```

3. 使用 influx auth create 命令來重新建立您的運算子權杖。考慮到此程序將使舊的運算子字符失效。

```
influx auth create --org kronos --operator
```

4. 擁有運算子權杖後，您可以使用輸入驗證清單命令來檢視權杖所有權杖。您可以使用 influx auth create 命令來建立所有存取權杖。

> **Important**
>
> 您必須先執行此步驟，才能取得您的運算子權杖，然後才能使用 InfluxDB API或 建立新的權杖 CLI。

使用 Influx UI

1. 使用建立的端點瀏覽至您的 Timestream for InfluxDB 執行個體，以登入和存取 InfluxDB 使用者介面。您需要使用用來建立 InfluxDB 資料庫執行個體的使用者名稱和密碼。您可以從 `influxAuthParametersSecretArn` 回應物件中指定的 擷取此資訊`CreateDbInstance`。

   或者，您可以從 Timestream for InfluxDB 管理主控台開啟 InfluxUI： InfluxDB

   a. 登入 AWS Management Console 並在 開啟 Amazon Timestream for InfluxDB 主控台https://console.aws.amazon.com/timestream/。
   b. 在 Amazon Timestream for InfluxDB 主控台的右上角，選擇您建立資料庫執行個體 AWS 的區域。

c. 在資料庫清單中，選擇 InfluxDB 執行個體的名稱以顯示其詳細資訊。在右上角，選擇 Open Influx UI 。

2. 登入您的 InfluxUI 後，請使用左側導覽列導覽至載入資料，然後導覽API至權杖。
3. 選擇 + GENERATEAPITOKEN並選取所有存取API字符 。
4. 輸入API權杖的描述，然後選擇 SAVE。
5. 複製產生的權杖並存放，以便安全保存。

> **Important**
>
> 從 InfluxUI 建立權杖時，新建立的權杖只會顯示一次。請務必複製這些內容，否則您將需要重新建立這些內容。

使用 InfluxDB API

- 使用請求方法將POST請求傳送至 InfluxDB API`/api/v2/authorizations`端點。

  在您的請求中包含下列項目：

  a. 標頭：

     i. 授權：權杖 <INFLUX_OPERATOR_TOKEN >

     ii. Content-Type：應用程式/json

  b. 請求內文：具有下列屬性的JSON內文：

     i. 狀態：「作用中」

     ii. 描述：API字符描述

     iii. orgIDInfluxDB 組織 ID

     iv. 許可：物件陣列，其中每個物件代表 InfluxDB 資源類型或特定資源的許可。每個許可都包含下列屬性：

        A. 動作：「讀取」或「寫入」

        B. 資源：代表要授予許可的 InfluxDB 資源的JSON物件。每個資源至少包含下列屬性：orgID：InfluxDB 組織 ID

        C. 類型：資源類型。如需 InfluxDB 資源類型的相關資訊，請使用 the /api/v2/resources 端點。

下列範例使用 curl和 InfluxDB API來產生所有存取權杖：

```
export INFLUX_HOST=https://influxdb1-123456789.us-east-1.timestream-
influxdb.amazonaws.com
export INFLUX_ORG_ID=<YOUR_INFLUXDB_ORG_ID>
export INFLUX_TOKEN=<YOUR_INFLUXDB_OPERATOR_TOKEN>

curl --request POST \
"$INFLUX_HOST/api/v2/authorizations" \
  --header "Authorization: Token $INFLUX_TOKEN" \
  --header "Content-Type: text/plain; charset=utf-8" \
  --data '{
    "status": "active",
    "description": "All access token for get started tutorial",
    "orgID": "'"$INFLUX_ORG_ID"'",
    "permissions": [
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "authorizations"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "authorizations"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "buckets"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "buckets"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "dashboards"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "dashboards"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "orgs"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "orgs"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "sources"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "sources"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "tasks"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "tasks"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "telegrafs"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "telegrafs"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "users"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "users"}},
```

```
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"variables"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"variables"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"scrapers"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"scrapers"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"secrets"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"secrets"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"labels"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"labels"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "views"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"views"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"documents"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"documents"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notificationRules"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notificationRules"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notificationEndpoints"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notificationEndpoints"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"checks"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"checks"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "dbrp"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "dbrp"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notebooks"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notebooks"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"annotations"}},
```

```
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "annotations"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "remotes"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "remotes"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "replications"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "replications"}}
    ]
  }
'
```

資料庫身分驗證選項

Amazon Timestream for InfluxDB 支援下列方法驗證資料庫使用者：

- 密碼身分驗證 – 您的資料庫執行個體會執行使用者帳戶的所有管理。您可以使用 InfluxUI 、Influx CLI或 Influx 建立使用者、指定密碼和管理權杖API。
- 權杖身分驗證 – 資料庫執行個體會執行使用者帳戶的所有管理。您可以使用 Influx 、 Influx 來建立使用者CLI、指定密碼和管理權杖API。

加密的連線

您可以從應用程式使用 Secure Socket Layer （ SSL ） 或 Transport Layer Security （ TLS ） 來加密與資料庫執行個體的連線。InfluxDB 與 Kronos 服務建立和管理的應用程式之間TLS交握所需的憑證。當憑證續約時，執行個體會自動更新為最新版本，而不需要任何使用者介入。

使用參數群組

資料庫參數指定資料庫的配置方式。例如，資料庫參數可以指定要配置給資料庫的資源量 (例如記憶體)。

您可以透過將資料庫執行個體與參數群組建立關聯來管理資料庫組態。Amazon Timestream for InfluxDB 使用預設設定定義參數群組。您也可以使用自訂設定，定義自己的參數群組。

參數群組概觀

資料庫參數群組扮演引擎組態值的容器，以套用至一或多個資料庫執行個體。

主題

- 預設和自訂參數群組
- 建立資料庫參數群組
- 靜態和動態資料庫執行個體參數
- 支援的參數和參數值

預設和自訂參數群組

資料庫執行個體會使用資料庫參數群組。以下各節介紹資料庫執行個體參數群組的設定和管理。

建立資料庫參數群組

您可以使用 AWS Management Console、 AWS Command Line Interface或 Timestream 建立新的資料庫參數群組API。

下列限制適用於資料庫參數群組名稱：

- 名稱必須為 1 到 255 個字母、數字或連字號。
- 預設參數群組名稱可以包括句點，例如 `default.InfluxDB.2.7`。不過，自訂參數群組名稱不能包括句點。
- 第一個字元必須是字母。
- 名稱不能以「dbpg-」開頭
- 名稱不能以連字號結尾或包含兩個連續連字號。
- 如果您未指定資料庫參數群組就建立資料庫執行個體，資料庫執行個體會使用 influxDB 引擎預設值。

您無法修改預設參數群組的參數設定。相反地，您可以執行下列作業：

1. 建立新的參數群組。
2. 變更所需參數的設定。並非參數群組中的所有資料庫引擎參數都有資格進行修改。
3. 更新您的資料庫執行個體以使用自訂參數群組。如需有關更新資料庫執行個體的資訊，請參閱 更新資料庫執行個體。

> **Note**
>
> 如果您已將資料庫執行個體修改為使用自訂參數群組，且啟動資料庫執行個體，則 Amazon Timestream for InfluxDB 會自動重新啟動資料庫執行個體，作為啟動程序的一部分。

目前，一旦建立自訂參數群組，您將無法對其進行修改。如果您需要變更參數，您必須建立新的自訂參數群組，並將其指派給需要此組態變更的執行個體。如果您更新現有的資料庫執行個體以指派新的參數群組，則它一律會立即套用並重新啟動執行個體。

靜態和動態資料庫執行個體參數

InfluxDB 資料庫執行個體參數一律為靜態 。其行為如下：

當您變更靜態參數、儲存資料庫參數群組，並將其指派給執行個體時，參數變更會在執行個體重新啟動後自動生效。

當您將新的資料庫參數群組與資料庫執行個體建立關聯時，Timestream 只會在資料庫執行個體重新啟動後套用修改後的靜態參數。目前唯一選項是立即套用。

如需變更資料庫叢集參數群組的詳細資訊，請參閱 更新資料庫執行個體。

支援的參數和參數值

若要判斷資料庫執行個體支援的參數，請檢視資料庫執行個體所使用的資料庫參數群組中的參數。如需詳細資訊，請參閱檢視資料庫參數群組的參數值。

如需 InfluxDB 開放原始碼版本支援的所有參數的詳細資訊，請參閱 InfluxDB 組態選項 。目前您只能修改下列 InfluxDB 參數：

參數	描述	預設值	Value	有效範圍	注意
flux-log-enabled	包含顯示 Flux 查詢詳細日誌的選項	FALSE	true、false	N/A	
日誌層級	記錄輸出層級。InfluxDB 輸出嚴重性層級大於或等於指定層級的日誌項目。	info	除錯、資訊、錯誤	N/A	
禁止任務	允許同時執行的查詢數目。	FALSE	true、false	N/A	

參數	描述	預設值	Value	有效範圍	注意
	將 設定為 0 允許無限數量的並行查詢。				
query-concurrency	停用任務排程器。如果有問題的任務阻止 InfluxDB 啟動，請使用此選項啟動 InfluxDB，而無需排程或執行任務。	1024		N/A	
query-queue-size	執行佇列中允許的查詢數量上限。達到佇列限制時，會拒絕新的查詢。將 設為 0 允許佇列中不限數量的查詢。	1024		N/A	
tracing-type	在 InfluxDB 中啟用追蹤，並指定追蹤類型。追蹤預設為停用。	""	日誌、jaeger	N/A	
停用指標	停用公開內部 InfluxDB 指標 的 HTTP /metrics 端點。	FALSE		N/A	

參數	描述	預設值	Value	有效範圍	注意
http-idle-timeout	在等待新請求時，伺服器應保持已建立連線的持續時間上限。設定為 0 ，不會逾時。	3m0s	單位 hours、minut 的持續時間milliseco nds 。範例：duration ype=minut es,value= 10	小時： -最小值：0 -上限：25620 5 分鐘： -最小值：0 -上限：15372 286 秒數： -最小值：0 -上限：92233 7203 毫秒： -最小值：0 -上限：92233 7203685	

參數	描述	預設值	Value	有效範圍	注意
http-read-header-timeout	伺服器應嘗試讀取新請求HTTP標頭的持續時間上限。設定為 0，不會逾時。	10 秒	單位 hours、minut 的持續時間milliseconds 。範例：duration ype=minutes,value=10	小時： -最小值：0 -上限：256205 分鐘： -最小值：0 -上限：15372286 秒數： -最小值：0 -上限：922337203 毫秒： -最小值：0 -上限：9223372036685	

參數	描述	預設值	Value	有效範圍	注意
http-read-timeout	伺服器應嘗試讀取整個新請求的持續時間上限。設定為 0 ，不會逾時。	0	單位 hours、minut 的持續時間milliseconds 。範例：duration ype=minutes,value=10	小時： -最小值：0 -上限：256205 分鐘： -最小值：0 -上限：15372286 秒數： -最小值：0 -上限：922337203 毫秒： -最小值：0 -上限：9223372036685	

參數	描述	預設值	Value	有效範圍	注意
http-write-timeout	伺服器應花費處理和回應寫入請求的持續時間上限。設定為 0 ，不會逾時。	0	單位 hours、minut 的持續時間milliseconds 。範例：duration ype=minutes,value=10	小時： -最小值：0 -上限：256205 分鐘： -最小值：0 -上限：15372286 秒數： -最小值：0 -上限：922337203 毫秒： -最小值：0 -上限：9223372036685	
influxql-max-select-buckets	SELECT 陳述式可以建立的依時間儲存貯體的群組數量上限。 0 允許不限數量的儲存貯體。	0	Long	下限：0 上限：9，223，372，036，854，775，807	

參數	描述	預設值	Value	有效範圍	注意
influxql-max-select-point	SELECT 陳述式可以處理的點數上限。 0 允許無限的點數。InfluxDB 每秒檢查點計數（因此超過上限的查詢不會立即中止）。	0	Long	下限：0 上限：9，223，372，036，854，775，807	
influxql-max-select-series	SELECT 陳述式可以傳回的系列數量上限。 0 允許無限數量的系列。	0	Long	下限：0 上限：9，223，372，036，854，775，807	
pprof-disabled	停用/debug/pprof HTTP 端點。此端點提供執行期分析資料，在除錯時很有幫助。	FALSE	Boolean	N/A	
query-initial-memory-bytes	為查詢配置的初始記憶體位元組。	0	Long	下限：0 最大值：query-memory-bytes	

參數	描述	預設值	Value	有效範圍	注意
query-max-memory-bytes	查詢允許的記憶體總位元組數上限。	0	Long	下限：0 上限：9，223，372，036，854，775，807	
query-memory-bytes	指定新建立的使用者工作階段的存留時間（TTL），以分鐘為單位。	0	Long	下限：0 上限：2，147，483，647	必須大於或等於 query-initial-memory-bytes。
工作階段長度	指定新建立的使用者工作階段的存留時間（TTL），以分鐘為單位。	60	Integer	下限：0 上限：2880	

參數	描述	預設值	Value	有效範圍	注意
session-renew-disabled	在每個請求TTL上停用自動擴展使用者的工作階段。根據預設，每個請求都會將工作階段的過期時間設定為從現在開始五分鐘。停用時，即使最近處於作用中狀態，工作階段仍會在指定的工作階段長度後過期，使用者也會重新導向至登入頁面。	FALSE	Boolean	N/A	
storage-cache-max-memory- 大小	碎片快取的大小上限（以位元組為單位）可在開始拒絕寫入之前達到。	1073741824	Long	下限：0 上限：549755813888	必須低於執行個體的總記憶體容量。 我們建議將其設定為低於總記憶體容量的15%。

參數	描述	預設值	Value	有效範圍	注意
storage-cache-snapshot-memory-大小	儲存引擎將快照快取並將其寫入TSM檔案的大小（以位元組為單位），以便提供更多記憶體。	26214400	Long	下限：0 上限：549755813888	必須小於storage-cache-max-memory大小。
storage-cache-snapshot-write-cold-duration	如果碎片未收到寫入或刪除，儲存引擎將快照快取並將其寫入新TSM檔案的持續時間。	10m0s	單位hours、minut的持續時間milliseconds 。範例：duration ype=minutes,value=10	小時： -最小值：0 -上限：256205 分鐘： -最小值：0 -上限：15372286 秒數： -最小值：0 -上限：922337203 毫秒： -最小值：0 -上限：9223372036685	

參數	描述	預設值	Value	有效範圍	注意
storage-compact-full-write-cold-duration	如果尚未收到寫入或刪除，儲存引擎將壓縮碎片中所有TSM檔案的持續時間。	4h0m0s	單位 hours、minut 的持續時間milliseconds 。範例：duration ype=minutes,value=10	小時： -最小值：0 -上限：256205 分鐘： -最小值：0 -上限：15372286 秒數： -最小值：0 -上限：922337203 毫秒： -最小值：0 -上限：9223372036854	
storage-compact-throughput-burst	TSM 壓縮可寫入磁碟的速率限制（以每秒位元組為單位）。	50331648	Long	下限：0 上限：9，223，372，036，854，775，807	

參數	描述	預設值	Value	有效範圍	注意
storage-max-concurrent-compactions	可同時執行的完整和層級壓縮數量上限。的值0會產生執行階段runtime.GOMAXPROCS(0) 使用的50%。任何大於零的數字都會將壓縮限制為該值。此設定不適用於快取快照。	0	Integer	下限：0 上限：64	
storage-max-index-log-file-size	索引預先寫入日誌（）檔案將壓縮為索引檔案的大小（以位元組為單位WAL）。較小的大小會導致日誌檔案更快壓縮，並降低堆積使用量，而犧牲寫入輸送量。	1048576	Long	下限：0 上限：9，223，372，036，854，775，807	
storage-no-validate-field-大小	略過傳入寫入請求的欄位大小驗證。	FALSE	Boolean	N/A	

參數	描述	預設值	Value	有效範圍	注意
storage-retention-check-interval	保留政策強制執行檢查的間隔。	30m0s	單位 hours、minut 的持續時間milliseconds 。範例：duration ype=minutes,value=10	N/A	小時： -最小值：0 -上限：2562005 分鐘： -最小值：0 -上限：15372286 秒數： -最小值：0 -上限：922337203 毫秒： -最小值：0 -上限：9223372036685
storage-series-file-max-concurrent-snapshot-compactions	可在資料庫中所有序列分割區同時執行的快照壓縮數量上限。	0	Integer	下限：0 上限：64	

參數	描述	預設值	Value	有效範圍	注意
storage-series-id-set-cache-size	TSI 索引中用於儲存先前計算之序列結果的內部快取大小。執行具有相同標籤索引鍵/值述詞的後續查詢時，會快速傳回快取結果，而不需要重新計算。將此值設定為 0 將停用快取，並可能會降低查詢效能。	100	Long	下限：0 上限：9，223，372，036，854，775，807	
storage-wal-max-concurrent-寫入	寫入WAL目錄以同時嘗試的最大數目。	0	Integer	下限：0 上限：256	

參數	描述	預設值	Value	有效範圍	注意
storage-wal-max-write-延遲	寫入WAL目錄請求WAL在達到目錄並行作用中寫入的最大數量時，等待的時間上限。設定為 0 以停用逾時。	10m	單位 hours、minut 的持續時間milliseconds 。範例：duration ype=minutes,value=10	小時： -最小值：0 -上限：256205 分鐘： -最小值：0 -上限：15372286 秒數： -最小值：0 -上限：922337203 毫秒： -最小值：0 -上限：9223372036685	
ui 停用	停用 InfluxDB 使用者介面（UI）。預設會啟用 UI。	FALSE	Boolean	N/A	

未正確設定參數群組中的參數，可能產生各種意外影響，包括效能降低和系統不穩定。修改資料庫參數時，請務必小心。在將參數群組變更套用至生產資料庫執行個體之前，嘗試對測試資料庫執行個體進行參數群組設定變更。

使用資料庫參數群組

資料庫執行個體會使用資料庫參數群組。以下各節介紹資料庫執行個體參數群組的設定和管理。

主題

建立資料庫參數群組

使用 AWS Management Console

1. 登入 AWS Management Console 並開啟 Amazon Timestream for InfluxDB 主控台 。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 選擇 Create parameter group (建立參數群組)。
4. 在 Group name (群組名稱) 方塊中，輸入新資料庫參數群組的名稱。
5. 在 Description (描述) 方塊中，輸入新資料庫參數群組的描述。
6. 選擇要修改和套用所需值的參數。如需支援參數的詳細資訊，請參閱 支援的參數和參數值。
7. 選擇 Save (儲存)。

使用 AWS Command Line Interface

- 若要使用 建立資料庫參數群組 AWS CLI，請使用下列參數呼叫 `create-db-parameter-group`命令：

```
--db-parameter-group-name <value>
--description <value>
--endpoint_url <value>
--region <value>
--parameters (list) (string)
```

Example 範例

如需每項設定的相關資訊，請參閱 資料庫執行個體的設定。此範例使用預設引擎組態。

```
aws timestream-influxdb create-db-parameter-group
    --db-parameter-group-name YOUR_PARAM_GROUP_NAME\
    --endpoint-url YOUR_ENDPOINT
    --region YOUR_REGION \
    --parameters
 "InfluxDBv2={logLevel=debug,queryConcurrency=10,metricsDisabled=true}" \" \
    --debug
```

將資料庫參數群組與資料庫執行個體建立關聯

您可以使用自訂設定，建立自己的資料庫參數群組。您可以使用 AWS Management Console、 AWS Command Line Interface或 Timestream-InfluxDB ，將資料庫參數群組與資料庫執行個體建立關聯 API。您可以在建立或修改資料庫執行個體時執行此動作。

如需建立資料庫參數群組的資訊，請參閱建立資料庫參數群組。如需建立資料庫執行個體的相關資訊，請參閱 建立資料庫執行個體。如需修改資料庫執行個體的相關資訊，請參閱 更新資料庫執行個體。

> **Note**
>
> 當您將新的資料庫參數群組與資料庫執行個體建立關聯時，修改後的靜態參數只會在資料庫執行個體重新啟動後套用。目前僅支援立即套用。InfluxDB 的 Timestream 僅支援靜態參數。

使用 AWS Management Console

1. 登入 AWS Management Console 並開啟 Amazon Timestream for InfluxDB 主控台 。
2. 在導覽窗格中，選擇 InfluxDB 資料庫 ，然後選擇您要修改的資料庫執行個體。
3. 選擇更新。隨即顯示更新資料庫執行個體頁面。
4. 變更 DB parameter group (資料庫參數群組) 設定。
5. 選擇 Continue (繼續)，並檢查修改的摘要。
6. 目前僅支援立即套用。此選項在某些情況下可能會導致中斷，因為它會重新啟動您的資料庫執行個體。
7. 在確認頁面上，檢閱您的變更。如果正確，請選擇更新資料庫執行個體以儲存變更並套用變更。或者，選擇 Back (上一步) 以編輯變更，或是選擇 Cancel (取消) 以取消變更。

使用 AWS Command Line Interface

若為 Linux、macOS 或 Unix：

```
aws timestream-influxdb update-db-instance
--identifier YOUR_DB_INSTANCE_ID \
--region YOUR_REGION \
--db-parameter-group-identifier YOUR_PARAM_GROUP_ID \
--log-delivery-configuration "{\"s3Configuration\": {\"bucketName\":
 \"${LOGGING_BUCKET}\", \"enabled\": false }}"
```

針對 Windows：

```
aws timestream-influxdb update-db-instance
--identifier YOUR_DB_INSTANCE_ID ^
--region YOUR_REGION ^
--db-parameter-group-identifier YOUR_PARAM_GROUP_ID ^
--log-delivery-configuration "{\"s3Configuration\": {\"bucketName\":
 \"${LOGGING_BUCKET}\", \"enabled\": false }}"
```

列出資料庫參數群組

您可以列出您為 AWS 帳戶建立的資料庫參數群組。

使用 AWS Management Console

1. 登入 AWS Management Console 並開啟 Amazon Timestream for InfluxDB 主控台 。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 資料庫參數群組隨即會出現在清單中。

使用 AWS Command Line Interface

若要列出 AWS 帳戶的所有資料庫參數群組，請使用 AWS Command Line Interface `list-db-parameter-groups`命令。

```
aws timestream-influxdb list-db-parameter-groups --region region
```

若要傳回 AWS 帳戶的特定資料庫參數群組，請使用 AWS Command Line Interface `get-db-parameter-group`命令。

```
aws timestream-influxdb get-db-parameter-group --region region --identifier identifier
```

檢視資料庫參數群組的參數值

您可以從資料庫參數群組取得所有參數與其值的清單。

使用 AWS Management Console

1. 登入 AWS Management Console 並開啟 Amazon Timestream for InfluxDB 主控台 。
2. 在導覽窗格中，選擇 Parameter groups (參數群組)。
3. 資料庫參數群組隨即會出現在清單中。
4. 選擇參數群組的名稱，以查看其參數清單。

使用 AWS Command Line Interface

若要檢視資料庫參數群組的參數值，請使用 AWS Command Line Interface `get-db-parameters`命令搭配下列必要參數。

```
--db-parameter-group-name
```

使用 API

若要檢視資料庫參數群組的參數值，請使用 Timestream API`GetDBParameters`命令搭配下列必要參數。

```
DBParameterGroupName
```

管理資料庫執行個體

本節涵蓋管理 Timestream for InfluxDB 執行個體的各種層面，以確保最佳效能、可用性和監控功能。它提供更新資料庫執行個體組態、處理多可用區域部署和容錯移轉程序的指引。它也說明如何刪除資料庫執行個體，並設定 InfluxDB 執行個體的日誌檢視。

主題

- 更新資料庫執行個體
- 維持資料庫執行個體
- 刪除資料庫執行個體

- 多可用區域資料庫執行個體部署
- 設定以在 Timestream Influxdb 執行個體上檢視 InfluxDB 日誌

更新資料庫執行個體

您可以更新 Timestream for InfluxDB 執行個體的下列組態參數：

- 執行個體類別
- 部署類型
- 參數群組
- 日誌傳遞組態

> **Important**
>
> 我們建議您在修改生產執行個體之前測試測試執行個體上的所有變更，以了解其影響，特別是在升級資料庫版本時。在更新設定之前，請先檢閱對資料庫和應用程式的影響。有些修改需要重新啟動資料庫執行個體，導致停機時間。

使用 AWS Management Console

1. 登入 AWS Management Console 並開啟 Amazon Timestream for InfluxDB 主控台 。
2. 在導覽窗格中，選擇 InfluxDB 資料庫 ，然後選擇您要修改的資料庫執行個體。
3. 選擇 Modify (修改)。
4. 在修改資料庫執行個體頁面上，進行所需的變更。
5. 選擇 Continue (繼續)，並檢查修改的摘要。
6. 選擇 Next (下一步)。
7. 檢閱您的變更。
8. 選擇修改執行個體以套用您的變更。

> **Note**
>
> 精簡修改需要重新啟動 Influx 資料庫執行個體，在某些情況下可能會導致中斷。

使用 AWS Command Line Interface

若要使用 更新資料庫執行個體 AWS Command Line Interface，請呼叫 `update-db-instance`命令。指定資料庫執行個體識別符，以及您要修改選項的值。如需每個選項的詳細資訊，請參閱資料庫執行個體的設定。

Example 範例

下列程式碼會透過設定不同的 dbparametergroup 來修改 mydbinstance。變更會立即套用。

針對 Linux、macOS 或 Unix：

```
aws timestream-influxdb update-db-instance \
    --identifier mydbinstance \
    --db-instance-type desired-instance-type \
    --deployment-type desired-deployment-type \
    --db-parameter-group-name newparamgroup \
    --port 8086
```

針對 Windows：

```
aws timestream-influxdb update-db-instance ^
    --identifier mydbinstance ^
    --db-instance-type desired-instance-type ^
    --deployment-type desired-deployment-type ^
    --db-parameter-group-name newparamgroup
    --port 8086
```

維持資料庫執行個體

Amazon Timestream for InfluxDB 會定期對 Amazon Timestream for InfluxDB 資源執行維護。維護通常涉及資料庫執行個體中下列資源的更新：

- 基礎硬體
- 基礎作業系統 (OS)
- 資料庫引擎版本

作業系統更新大多是因為安全性問題。

某些維護項目需要 Amazon Timestream for InfluxDB 短時間內讓您的資料庫執行個體離線。需要資源離線的維護項目包括必要的作業系統或資料庫修補。所需的修補程式僅會針對與安全性和執行個體可靠性相關的修補程式自動安排。這類修補不常發生，通常每隔幾個月進行一次。維護僅需片刻的時間即可完成。

- 維護時段設定為每天在本機時間凌晨 12：00 至凌晨 4：00 之間，針對執行個體託管的區域進行維護時段。
- 在一週的七個維護時段中的任何一個，客戶資源都可以每週修補一次。

刪除資料庫執行個體

刪除資料庫執行個體會影響執行個體復原能力和快照可用性。請考慮以下問題：

- 如果您想要刪除 InfluxDB 資源的所有 Timestream，請注意，資料庫執行個體資源會產生帳單費用。
- 刪除資料庫執行個體的狀態時，其 CA 憑證值不會出現在 Timestream for InfluxDB 主控台中，也不會出現在輸出中用於 AWS Command Line Interface 命令或 Timestream API操作。
- 刪除資料庫執行個體所需的時間取決於刪除的資料量，以及是否拍攝最終快照。

您可以使用 AWS Management Console、 AWS Command Line Interface或 Timestream 刪除資料庫執行個體API。您必須提供資料庫執行個體的名稱：

使用 AWS Management Console

1. 登入 AWS Management Console 並開啟 Amazon Timestream for InfluxDB 主控台 。
2. 在導覽窗格中，選擇 InfluxDB 資料庫 ，然後選擇要刪除的資料庫執行個體。
3. 選擇 刪除 。
4. 在方塊中輸入確認。
5. 選擇 刪除 。

使用 AWS Command Line Interface

若要尋找帳戶中資料庫執行個體IDs的執行個體，請呼叫 `list-db-instances`命令：

```
aws timestream-influxdb list-db-instances \
--endpoint-url YOUR_ENDPOINT \
```

```
--region YOUR_REGION
```

若要使用 刪除資料庫執行個體CLI，請使用下列選項呼叫 AWS `delete-db-instance`命令：

```
aws timestream-influxdb list-db-instances \
--identifier YOUR_DB_INSTANCE \
```

Example 範例

若為 Linux、macOS 或 Unix：

```
aws timestream-influxdb delete-db-instance \
    --identifier mydbinstance
```

針對 Windows：

```
aws timestream-influxdb delete-db-instance ^
    --identifier mydbinstance
```

多可用區域資料庫執行個體部署

Amazon Timestream for InfluxDB 使用具有單一待命資料庫執行個體的多可用區部署，為資料庫執行個體提供高可用性和容錯移轉支援。這種類型的部署稱為多可用區域資料庫執行個體部署。Amazon Timestream for InfluxDB 使用 Amazon 容錯移轉技術。

在多可用區域資料庫執行個體部署中，Amazon Timestream 會自動佈建並維護不同可用區域中的同步待命複本。主要資料庫執行個體會在待命複本的可用區域間進行同步複製，以提供資料備援。在資料庫執行個體失敗和可用區域中斷期間，執行高可用性的資料庫執行個體可以增強可用性。如需可用區域的詳細資訊，請參閱AWS 區域和可用區域。

> **Note**
>
> 高可用性選項不是唯讀案例的擴展解決方案。您無法使用待命複本來提供讀取流量。

使用 Amazon Timestream 主控台，您只需在建立資料庫執行個體時，在可用性和耐久性組態區段中指定建立待命執行個體選項，即可建立多可用區域資料庫執行個體部署。 您也可以使用 AWS Command Line Interface 或 Amazon Timestream 指定多可用區域資料庫執行個體部署API。使用 `create-db-instance`或 CLI命令，或 `CreateDBInstance`API操作。

相較於單一可用區域部署，使用多可用區域資料庫執行個體部署的資料庫執行個體會有增加的寫入和遞交延遲。這可能是因為發生的同步資料複寫。如果您的部署容錯移轉到待命複本，則延遲可能會有所變更，儘管 AWS 是使用可用區域之間的低延遲網路連線進行設計。對於生產工作負載，我們建議您使用IOPS隨附的 12K 或 16K 儲存體，IOPS以獲得快速、一致的效能。如需資料庫執行個體類別的詳細資訊，請參閱 資料庫執行個體類別。

設定和管理多可用區部署

InfluxDB Multi-AZ 部署的時間串流只能有一個待命。當部署具有一個備用資料庫執行個體時，稱為多可用區域資料庫執行個體部署。多可用區域資料庫執行個體部署有一個待命資料庫執行個體，可提供容錯移轉支援，但是不提供讀取流量。

> **Important**
>
> 您的執行個體必須至少有兩個與其相關聯的子網路，才能執行單一可用區到多可用區更新。執行個體建立後，您無法將其部署模式從單一可用區 修改為多可用區 。

您可以使用 AWS Management Console 來判斷資料庫執行個體是單一可用區還是多可用區部署。

使用 AWS Management Console

1. 登入 AWS Management Console 並開啟 Amazon Timestream for InfluxDB 主控台 。
2. 在導覽窗格中，選擇 InfluxDB 資料庫 ，然後選擇資料庫識別碼 。

多可用區域資料庫執行個體部署具有以下特徵：

- 資料庫執行個體只有一行。
- Role (角色) 的值為 Instance (執行個體) 或 Primary (主要)。
- Multi-AZ (多可用區域) 的值為 Yes (是)。

Amazon Timestream 的容錯移轉程序

如果因基礎設施缺陷而導致資料庫執行個體的計劃內或非計劃性中斷，如果您已開啟多可用區域，Amazon Timestream for InfluxDB 會自動切換到另一個可用區域中的待命複本。完成容錯移轉所需的時間取決於主要資料庫執行個體失效時的資料庫活動和其他條件。通常容錯移轉時間是 60–120 秒。不過，大型交易或冗長復原程序可能會增加容錯移轉時間。當容錯移轉完成時，Timestream 主控台可能需要額外的時間來反映新的可用區域。

> **Note**
>
> Amazon Timestream 會自動處理容錯移轉，因此您可以盡快恢復資料庫操作，而無需管理介入。如果發生下表所述的任何條件，主要資料庫執行個體會自動切換至待命複本。

容錯移轉原因	描述
Timestream 資料庫執行個體基礎的作業系統正在離線操作中修補。	作業系統修補或安全更新的維護期間觸發容錯移轉。
Timestream Multi-AZ 執行個體的主要主機運作狀態不佳。	多可用區域資料庫執行個體部署偵測到主要資料庫執行個體受損並容錯移轉。
由於網路連線中斷，因此無法連線 Timestream Multi-AZ 執行個體的主要主機。	時間流監控偵測到主要資料庫執行個體的網路可連線性失敗，並觸發容錯移轉。
客戶已修改 Timestream 執行個體。	InfluxDB 資料庫執行個體修改的 Timesteam 觸發容錯移轉。如需詳細資訊，請參閱更新資料庫執行個體。
Timestream Multi-AZ 主要執行個體忙碌且沒有回應。	主要資料庫執行個體沒有回應。建議您執行下列動作：* 檢查事件是否過度使用 CPU、記憶體或交換空間。* 評估工作負載，以判斷您是否使用適當的資料庫執行個體類別。如需更多詳細資訊，請參閱資料庫執行個體類別。
Timestream Multi-AZ 執行個體主要主機下方的儲存磁碟區發生故障。	多可用區域資料庫執行個體部署在主要資料庫執行個體上偵測到儲存問題並容錯移轉。

設定DNS名稱查詢JVMTTL的

容錯移轉機制會自動變更資料庫執行個體的網域名稱系統（DNS）記錄，以指向待命資料庫執行個體。因此，您必須重新建立資料庫執行個體任何現有的連線。在 Java 虛擬機器（JVM）環境中，由於 Java DNS 快取機制的運作方式，您可能需要重新設定JVM設定。

JVM 快取DNS名稱查詢。當 將主機名稱JVM解析為 IP 地址時，它會快取 IP 地址一段時間，稱為 time-to-live（TTL）。

由於 AWS 資源使用偶爾變更DNS的名稱項目，建議您將 的值設定為JVMTTL不超過 60 秒。這樣做可確保當資源的 IP 地址變更時，您的應用程式可以透過重新查詢 來接收和使用資源的新 IP 地址DNS。

在某些 Java 組態上，TTL會設定JVM預設值，使其在JVM重新啟動 之前永遠不會重新整理DNS項目。因此，如果 AWS 資源的 IP 地址在應用程式仍在執行時變更，則在您手動重新啟動 JVM並重新整理快取的 IP 資訊之前，無法使用該資源。在此情況下，請務必設定 JVM，TTL以便定期重新整理其快取的 IP 資訊。

您可以透過TTL擷取`networkaddress.cache.ttl`屬性值來取得JVM預設值：

```
String ttl = java.security.Security.getProperty("networkaddress.cache.ttl");
```

> **Note**
>
> 預設值TTL會因 的版本JVM和是否已安裝安全管理員而有所不同。許多 JVMs 提供的預設TTL時間少於 60 秒。如果您使用的是這類 JVM，而未使用安全管理器，則可以忽略本主題的其餘部分。
> 若要修改 JVM的 TTL，請設定 networkaddress.cache.ttl 屬性值。根據您的需求，使用下列其中一種方法：
>
> - 若要針對使用 的所有應用程式全域設定 屬性值JVM，請在 `$JAVA_HOME/jre/lib/security/java.security` `networkaddress.cache.ttl`檔案中設定 。
>
>   ```
>   networkaddress.cache.ttl=60
>   ```
>
> - 若要僅針對您的應用程式進行適當的本機設定，請在建立任何網路連線之前，在您應用程式的初始化程式碼中設定 `networkaddress.cache.ttl`。
>
>   ```
>   java.security.Security.setProperty("networkaddress.cache.ttl" , "60");
>   ```

設定以在 Timestream Influxdb 執行個體上檢視 InfluxDB 日誌

根據預設，InfluxDB 會產生前往 stdout 的日誌。如需詳細資訊，請參閱管理 InfluxDB 日誌

若要檢視您透過 Timestream InfluxDB 建立的執行個體所產生的 InfluxDB 日誌，我們提供提供每小時日誌的機會。這些日誌將移至您在建立執行個體之前必須建立的指定 S3 儲存貯體。

- 在建立執行個體之前，提供的 Amazon S3 儲存貯體還必須授予 Timestream-InfluxDB 許可，透過向 Timestream InfluxDB Service Principal 提供儲存貯體政策，將日誌傳送至此儲存貯體，如下所示（取代 *{BUCKET_NAME}* Amazon S3 儲存貯體的實際名稱：

```
{
    "Version": "2012-10-17",
    "Id": "PolicyForInfluxLogs",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "timestream-influxdb.amazonaws.com"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::{BUCKET_NAME}/InfluxLogs/*"
        }
    ]
}
```

- 提供的儲存貯體必須位於您建立的 Timestream InfluxDB 執行個體的相同帳戶和相同區域

以下是您可以呼叫 以讓執行個體接收流入日誌的命令：

```
aws timestream-influxdb create-db-instance \
    --name myinfluxDbinstance \
    --allocated-storage 400 \
    --db-instance-type db.influx.4xlarge \
    --vpc-subnet-ids subnetid1 subnetid2
    --vpc-security-group-ids mysecuritygroup \
    --username masterawsuser \
    --password \
    --db-storage-type InfluxIOIncludedT2
```

以下是此參數的格式。

```
-- log-delivery-configuration
{
    "S3Configuration": {
      "BucketName": "string",
      "Enabled": true|false
    }
}
```

- 此欄位不是必要欄位，且預設不會啟用記錄。
- 不設定此欄位與未啟用日誌相同。
- 日誌將傳送至具有 字首的指定儲存貯體`InfluxLogs/`。
- 建立執行個體後，您可以使用 `update-db-instance`API命令修改日誌交付組態。

InfluxDB 提供不同類型的日誌。您可以透過設定 InfluxDB 參數來設定這些參數。使用 flux-log-enabled 和 日誌層級參數來設定從執行個體發出的日誌類型。如需詳細資訊，請參閱支援的參數和參數值。

將標籤新增至資源

您可以使用標籤 標記 Amazon Timestream for InfluxDB 資源。標籤可讓您以不同的方式分類資源，例如，依用途、擁有者、環境或其他條件。標籤可協助您執行以下作業：

- 根據您指派給資源的標籤來快速識別資源。
- 請參閱依標籤細分的 AWS 帳單。

Amazon Elastic Compute Cloud （Amazon EC2）、Amazon Simple Storage Service （Amazon S3）、Timestream for InfluxDB 等 AWS 服務支援標記。有效標記可讓您跨帶有特定標籤的服務來建立報告，以提供成本深入資訊。

最後，最好遵循最佳標記策略。如需資訊，請參閱 AWS 標記策略。

標記限制

每個標記皆包含由您定義的索引鍵和值。將適用以下限制：

- InfluxDB 資料庫執行個體的每個 Timestream 只能有一個具有相同金鑰的標籤。如果您嘗試新增現有的標籤，現有的標籤值會更新為新的值。
- 值充當標籤類別內的描述符。在 InfluxDB 的 Timestream 中，值不能為空或 null。
- 標籤鍵與值皆區分大小寫。
- 鍵長度上限為 128 個 Unicode 字元。
- 值長度上限為 256 個 Unicode 字元。
- 允許的字元為字母、空格和數字，以及下列特殊字元：`+ - = . _ : /`
- 每一資源標籤數最多為 50。

- AWS- 指派的標籤名稱和值會自動指派您無法指派的aws:字首。 AWS指派的標籤名稱不會計入 50 的標籤限制。使用者指派的標籤名稱在成本分配報告中具有字首 user:。
- 標籤的套用不可回溯。

Timestream for InfluxDB 的安全最佳實務

最佳化 InfluxDB 的寫入

如同任何其他時間序列資料庫，InfluxDB 的建置目的是能夠即時擷取和處理資料。為了讓系統保持最佳效能，我們建議在將資料寫入 InfluxDB 時遵循下列最佳化：

- 批次寫入：將資料寫入 InfluxDB 時，請以批次方式寫入資料，將與每個寫入請求相關的網路額外負荷降至最低。每個寫入請求的最佳批次大小為 5000 行通訊協定。若要在一個請求中寫入多行，每行通訊協定都必須以新行分隔（\n）。
- 依索引鍵排序標籤：在將資料點寫入 InfluxDB 之前，請先依索引鍵以詞彙順序排序標籤。

```
measurement,tagC=therefore,tagE=am,tagA=i,tagD=i,tagB=think fieldKey=fieldValue
 1562020262

# Optimized line protocol example with tags sorted by key
measurement,tagA=i,tagB=think,tagC=therefore,tagD=i,tagE=am fieldKey=fieldValue
 1562020262
```

- 使用可能的最粗時間精確度：– InfluxDB 會以奈秒精確度寫入資料，但如果沒有以奈秒為單位收集您的資料，就不需要以該精確度寫入資料。為了獲得更好的效能，請針對時間戳記使用可能的最粗精確度。您可以在下列情況下指定寫入精確度：
  - 使用 時SDK，您可以在設定點的時間屬性 WritePrecision 時指定 。如需 InfluxDB 用戶端程式庫的詳細資訊，請參閱 InfluxDB 文件 。
  - 使用 Telegraf 時，您可以在 Telegraf 代理程式組態中設定時間精確度。精確度指定為整數 + 單位（例如 0s、10ms、2us、4s）的間隔。有效時間單位為「ns」、「us」、「ms」和「s」。

    ```
    [agent]
     interval ="10s"
     metric_batch_size="5000"
     precision = "0s"
    ```

- 使用 gzip 壓縮： – 使用 gzip 壓縮來加速對 InfluxDB 的寫入並減少網路頻寬。當資料壓縮時，基準顯示速度改善高達 5 倍。

- 使用 Telegraf 時，請在 Telegraf.conf 中的 Influxdb_v2 輸出外掛程式組態中，將 content_encoding 選項設定為 gzip：

```
[[outputs.influxdb_v2]]
  urls = ["http://localhost:8086"]
  # ...
  content_encoding = "gzip"
```

- 使用用戶端程式庫時，每個 InfluxDB 用戶端程式庫都會提供壓縮寫入請求的選項，或依預設強制執行壓縮。每個程式庫的啟用壓縮方法不同。如需特定指示，請參閱 InfluxDB 文件
- 使用 InfluxDB API `/api/v2/write`端點寫入資料時，請使用 gzip 壓縮資料，並將 Content-Encoding 標頭設定為 gzip。

效能設計

設計您的結構描述，以進行更簡單且效能更高的查詢。下列指導方針將確保您的結構描述易於查詢並最大限度地提高查詢效能：

- 設計查詢：選擇測量、標籤鍵和易於查詢的欄位鍵。若要實現此目標，請遵循下列原則：
  - 使用具有簡單名稱且準確描述結構描述的測量。
  - 避免在相同的結構描述中，對標籤金鑰和欄位金鑰使用相同的名稱。
  - 避免在標籤和欄位索引鍵中使用預留的 Flux 關鍵字和特殊字元。
  - 標籤會儲存描述欄位的中繼資料，而且在許多資料點之間都是常見的。
  - 欄位會儲存唯一或高度可變的資料，通常是數值資料點。
  - 測量和金鑰不應包含資料，但用於彙總或描述資料。資料將儲存在標籤和欄位值中。
- 將時間序列基數控制在 高序列基數 是 InfluxDB 中寫入和讀取效能降低的主要原因之一。在 InfluxDB 高基數的背景中，指的是存在非常大量的唯一標籤值。標籤值會在 InfluxDB 中編製索引，這表示非常大量的唯一值將產生更大的索引，以減緩資料擷取和查詢效能。

  若要進一步了解並解決潛在的高基數相關問題，您可以遵循下列步驟：
  - 了解高基數的原因
  - 測量儲存貯體的基數
  - 採取行動來解決高基數

- 高序列基數原因 InfluxDB 會根據測量結果和標籤索引資料，以加速資料讀取。每組索引資料元素都會形成系列索引鍵。包含高度可變資訊的標籤，例如唯一 IDs、雜湊和隨機字串會導致大量序列，也稱為高序列基數。高序列基數是 InfluxDB 中高記憶體用量的主要驅動因素。
- 測量序列基數如果您遇到效能下降，或看到 Timestream for InfluxDB 執行個體中的記憶體用量不斷增加，建議您測量儲存貯體的序列基數。

  InfluxDB 提供的函數可讓您在 Flux 和 InfluxQL 中測量序列基數。

  - 在 Flux 中使用 函數 `influxdb.cardinality()`
  - 在 FluxQL 中，使用 `SHOW SERIES CARDINALITY`命令

  在這兩種情況下，引擎都會傳回資料中唯一系列金鑰的數量。請記住，不建議在任何 Timestream for InfluxDB 執行個體上擁有超過 1，000 萬個序列金鑰。
- 高序列基數的原因 如果您遇到任何儲存貯體具有高基數，您可以採取一些修正步驟來修正：
  - 檢閱您的標籤：確保您的工作負載不會產生案例，而且大多數項目的標籤具有唯一的值。如果唯一標籤值的數量始終隨時間增加，或日誌類型訊息寫入資料庫，且每個訊息都有時間戳記、標籤等的唯一組合，則可能會發生這種情況。您可以使用下列 Flux 程式碼，協助您找出哪些標籤對高基數問題的影響最大：

```
// Count unique values for each tag in a bucketimport "influxdata/influxdb/schema"

cardinalityByTag = (bucket) => schema.tagKeys(bucket: bucket)
    |> map(
        fn: (r) => ({
            tag: r._value,
            _value: if contains(set: ["_stop", "_start"], value: r._value) then
                0
            else
                (schema.tagValues(bucket: bucket, tag: r._value)
                    |> count()
                    |> findRecord(fn: (key) => true, idx: 0))._value,
        }),
    )
    |> group(columns: ["tag"])
    |> sum()

cardinalityByTag(bucket: "example-bucket")
```

  如果您正經歷非常高的基數，上述查詢可能會逾時。如果您遇到逾時，請執行下列查詢 – 一次一個。

產生標籤清單：

```
// Generate a list of tagsimport "influxdata/influxdb/schema"

schema.tagKeys(bucket: "example-bucket")
```

計算每個標籤的唯一標籤值：

```
// Run the following for each tag to count the number of unique tag valuesimport
 "influxdata/influxdb/schema"

tag = "example-tag-key"

schema.tagValues(bucket: "my-bucket", tag: tag)
    |> count()
```

我們建議您在不同時間點執行這些操作，以識別哪些標籤正在快速增長。

- 改善您的結構描述：遵循我們 中討論的模型建議Timestream for InfluxDB 的安全最佳實務。
- 移除或彙總較舊的資料以減少基數：考慮您的使用案例是否需要導致高基數問題的所有資料。如果不再需要或經常存取這些資料，您可以彙總這些資料、將其刪除或匯出至其他引擎，例如 Timestream for Live Analytics，以進行長期儲存和分析。

疑難排解

無法辨識 "dev" 版本的警告

在遷移期間，可能會顯示伺服器報告的警告 'WARN：無法剖析版本 "dev"，假設APIs支援最新的備份/還原"。您可以忽略此警告。

在還原階段遷移失敗

如果在還原階段發生遷移失敗，使用者可以使用 `--retry-restore-dir`旗標重新嘗試還原。使用`--retry-restore-dir`旗標搭配先前備份目錄的路徑，略過備份階段並重試還原階段。如果遷移在還原期間失敗，則會顯示用於遷移的建立備份目錄。

還原失敗的可能原因包括：

- 無效的 InfluxDB 目的地權杖 – 在目的地執行個體中存在的儲存貯體，其名稱與來源執行個體中相同。對於個別儲存貯體遷移，請使用 `--dest-bucket`選項來設定已遷移儲存貯體的唯一名稱
- 連線失敗，無論是使用來源或目的地主機，或是使用選用的 S3 儲存貯體。

Amazon Timestream for InfluxDB 基本操作指南

以下是每個人在使用 Amazon Timestream for InfluxDB 時應遵循的基本操作準則。請注意，Amazon Timestream for InfluxDB 服務層級協議要求您遵循下列準則：

- 使用指標來監控您的記憶體CPU、 和儲存體用量。您可以設定 Amazon CloudWatch ，以便在使用模式變更或接近部署容量時通知您。如此一來，您就可以維護系統效能和可用性。
- 在您處理儲存容量限制時向上擴展資料庫執行個體。您應該在儲存體和記憶體中具有一些緩衝，以容納應用程式需求中未知的增加。請記住，目前您需要建立新的執行個體並遷移資料才能達成此目標。
- 如果資料庫工作負載所需的 I/O 較您佈建得多，容錯移轉或資料庫失敗之後的復原將會很緩慢。若要增加資料庫執行個體的 I/O 容量，請執行下列任何或所有動作：
  - 遷移至具有較高 I/O 容量的不同資料庫執行個體。
  - 如果您已經使用 Influx IOPS包含的儲存體儲存體，請佈建IOPS包含較高 的儲存體類型。
- 如果您的用戶端應用程式快取資料庫執行個體的網域名稱服務 （DNS） 資料，請將 time-to-live（TTL） 值設定為小於 30 秒。資料庫執行個體的基礎 IP 位址可能會在容錯移轉後變更。長時間快取DNS資料可能會導致連線失敗。您的應用程式可能會嘗試連線到不再提供服務的 IP 地址。

資料庫執行個體RAM建議

Amazon Timestream for InfluxDB 效能最佳實務是配置足夠的 ，RAM讓您的工作集幾乎完全存放在記憶體中。工作集是您的執行個體經常使用的資料和索引。您越常使用資料庫執行個體，工作集會成長越多。

適用於 InfluxDB 的 Timestream 中的安全性

的雲端安全 AWS 是最高優先順序。身為 AWS 客戶，您受益於為滿足最安全敏感組織的需求而建置的資料中心和網路架構。

安全性是 AWS 和 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。第三方稽核人員定期檢測及驗證安全的效率也是我們 AWS 合規計劃的一部分。若要了解適用於 Timestream for InfluxDB 的合規計劃，請參閱AWS 依合規計劃範圍中的服務。
- 雲端的安全性 – 您的責任取決於您使用 AWS 的服務。您也必須對其他因素負責，包括資料的敏感度、您組織的需求和適用的法律及法規。

本文件將協助您了解如何在使用 Timestream for InfluxDB 時套用共同責任模型。下列主題說明如何設定 InfluxDB 的 Timestream，以符合您的安全和合規目標。您也將了解如何使用其他服務AWS，以協助您監控和保護 Timestream for InfluxDB 資源。

主題

概觀

本文件可協助您了解如何在使用 Amazon Timestream for InfluxDB 時套用共同責任模型。下列主題說明如何設定 Amazon Timestream for InfluxDB 以符合您的安全和合規目標。您也會了解如何使用其他 AWS 服務來協助您監控和保護 Amazon Timestream for InfluxDB 資源。

您可以在資料庫執行個體上管理 Amazon Timestream for InfluxDB 資源和資料庫的存取權。您用來管理存取權的方法取決於使用者需要使用 Amazon Timestream for InfluxDB 執行的任務類型：

- 根據 Amazon VPC服務在虛擬私有雲端（VPC）中執行資料庫執行個體，以進行網路存取控制。
- 使用 AWS Identity and Access Management（IAM）政策來指派許可，以決定誰可以管理 Amazon Timestream for InfluxDB 資源。例如，您可以使用 IAM來判斷誰可以建立、描述、修改和刪除資料庫執行個體、標籤資源或修改安全群組。
- 使用安全群組來控制哪些 IP 地址或 Amazon EC2執行個體可以連線到資料庫執行個體上的資料庫。當您第一次建立資料庫執行個體時，只能透過關聯安全群組指定的規則存取。
- 搭配資料庫執行個體使用 Secure Socket Layer（SSL）或 Transport Layer Security（TLS）連線。
- 使用 InfluxDB 引擎的安全功能來控制誰可以登入資料庫執行個體上的資料庫。這項功能的運作方式就好像資料庫位在您的本機網路上。如需詳細資訊，請參閱適用於 InfluxDB 的 Timestream 中的安全性。

> **Note**
>
> 您只須針對您的使用案例設定安全。您不需要為 Amazon Timestream for InfluxDB 管理的程序設定安全存取權。這些包括建立備份、在主要資料庫執行個體和僅供讀取複本間複寫資料，以及其他程序。

主題

一般安全

主題

許可

應授予 InfluxDB 使用者最低權限許可。遷移期間只應使用授予特定使用者的權杖，而非運算子權杖。

InfluxDB 的 Timestream 使用IAM許可來控制使用者許可。我們建議使用者有權存取他們所需的特定動作和資源。如需詳細資訊，請參閱授予最低權限存取 。

網路存取

Influx 遷移指令碼可在本機運作，在相同系統上的兩個 InfluxDB 執行個體之間遷移資料，但假設遷移的主要使用案例是將資料遷移到本機或公有網路。有了安全考量。根據預設，Influx 遷移指令碼會驗證TLS已啟用 之執行個體的TLS憑證：我們建議使用者在 InfluxDB 執行個體TLS中啟用 ，且不要為指令碼使用 `--skip-verify`選項。

我們建議您使用允許清單來限制來自您預期來源的網路流量。您可以只從已知 限制網路流量至 InfluxDB 執行個體IPs。

相依性

應使用所有相依性的最新主要版本，包括 Influx CLI、InfluxDB 、Python、請求模組，以及選用的相依性，例如 `mountpoint-s3`和 `rclone`。

S3 儲存貯體

如果使用 S3 儲存貯體作為遷移的臨時儲存體，我們建議您啟用 TLS、版本控制和停用公有存取。

使用 S3 儲存貯體進行遷移

1. 開啟 AWS Management Console，導覽至 Amazon Simple Storage Service，然後選擇儲存貯體 。
2. 選擇您要使用的儲存貯體。
3. 選擇許可索引標籤標籤。
4. 在 Block public access (bucket settings) (封鎖公開存取 (儲存貯體設定)) (封鎖公開存取 (儲存貯體設定)) 下，選擇 Edit (編輯)。
5. 核取封鎖所有公有存取 。
6. 選擇 Save changes (儲存變更)。
7. 在 Bucket policy (儲存貯體政策) 下方，選擇 Edit (編輯)。
8. 輸入以下內容，將 <example-bucket> 取代為您的儲存貯體名稱，以強制使用 1.2 TLS版或更新版本進行連線：

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceTLSv12orHigher",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "s3:*"
            ],
            "Effect": "Deny",
            "Resource": [
                "arn:aws:s3:::<example bucket>/*",
                "arn:aws:s3:::<example bucket>"
            ],
            "Condition": {
                "NumericLessThan": {
                    "s3:TlsVersion": 1.2
                }
            }
        }
    ]
}
```

9. 選擇 Save changes (儲存變更)。
10. 選擇屬性索引標籤。
11. 在 Bucket Versioning (儲存貯體版本控制) 底下，選擇 Edit (編輯)。
12. 勾選啟用 。
13. 選擇 Save changes (儲存變更)。

如需 Amazon S3 儲存貯體最佳實務的相關資訊，請參閱 Amazon Simple Storage Service 的安全最佳實務。

使用 Amazon Timestream for InfluxDB 進行資料庫身分驗證

Amazon Timestream for InfluxDB 支援兩種驗證資料庫使用者的方式。

密碼和存取權杖資料庫身分驗證使用不同的方法來驗證資料庫。因此，特定使用者只能使用一種身分驗證方法登入資料庫。在這兩種情況下InfluxDB 都會執行使用者帳戶和API權杖的所有管理。

密碼身分驗證

在 InfluxDB 資料庫執行個體建立過程中，您已建立組織、使用者和密碼。使用者具有管理 Timestream for InfluxDB 資料庫執行個體中所有內容的許可。透過此使用者名稱和密碼組合，您將可以使用 InfluxUI LogIn 進入執行個體，也可以使用 InfluxCLI 產生運算子權杖。

建立使用者、刪除儲存貯體 、組織等需要操作員權杖。如需詳細資訊，請參閱資料庫身分驗證選項。

API 權杖

InfluxDB API權杖可確保 InfluxDB 與用戶端或應用程式等外部工具之間的安全互動。API 權杖屬於特定使用者，並識別使用者組織內的 InfluxDB 許可。

InfluxDB 中有三種API類型的權杖：

- 運算子權杖：授予 InfluxDB OSS 2.x 中所有組織和所有組織資源的完整讀取和寫入存取權。某些操作，例如擷取伺服器組態，需要操作員許可。若要在完成設定程序CLI後，使用 InfluxDB UIAPI、`api/v2` 或 Influx 手動建立運算子權杖，您必須使用現有的運算子權杖或使用者名稱和密碼。若要建立新的運算子權杖而不使用現有的權杖，請參閱流入復原驗證 CLI。

  > **Important**
  >
  > 由於運算子權杖對資料庫中的所有組織具有完整的讀取和寫入存取權，因此我們建議您為每個組織建立 All-Access 權杖，並使用這些權杖來管理 InfluxDB 。這有助於防止跨組織進行意外互動。

- All-Access API 權杖：授予組織中所有資源的完整讀取和寫入存取權。
- 讀取/寫入權杖：授予組織中特定儲存貯體的讀取存取權、寫入存取權或兩者。

所有 InfluxDb 權杖都是未設定過期日期的長活權杖，因此不建議使用您的運算子或所有存取權杖，從用戶端或 Telegraf 代理傳送監控資料，也不要在儀表板應用程式中內嵌這些權杖。對於這些應用程式，建立只具有完成任務所需許可的讀取/寫入權杖。Fo 有關如何建立 influxDB 權杖的詳細資訊，請參閱建立權杖 。

秘密

InfluxDB 運算子權杖會在執行個體設定時產生；其他類型的權杖，例如全存取和讀取/寫入權杖，可以使用 Influx CLI、Influx v2 API或 Timestream for InfluxDB 多使用者輪換函數建立。如需如何產生、檢視、指派和刪除API權杖，請參閱管理權杖。

我們建議您經常輪換 InfluxDB 權杖的 Timestream，並透過環境變數來 AWS Secrets Manager 儲存權杖。請參閱使用權杖用於環境變數中的權杖用量輪換秘密，以及如何輪換 InfluxDB 使用者和權杖的 Timestream。

另請參閱:

- Amazon Timestream for InfluxDB 中的基礎設施安全
- Timestream for InfluxDB 的安全最佳實務

Amazon Timestream for InfluxDB 如何使用秘密

InfluxDB 的 Timestream 支援透過使用者介面的使用者名稱和密碼驗證，以及最低權限用戶端和應用程式連線的權杖憑證。InfluxDB 使用者的 Timestream 在其組織內具有`allAccess`許可，而權杖可以具有任何一組許可。遵循安全API權杖管理的最佳實務，應建立使用者來管理權杖，以便在組織內精細存取。如需使用 Timestream for InfluxDB 管理最佳實務的詳細資訊，請參閱 Influxdata 文件 。

AWS Secrets Manager 是一項秘密儲存服務，可用來保護資料庫憑證、API金鑰和其他秘密資訊。然後，您可以在程式碼中將硬式編碼憑證取代為 Secrets Manager 的API呼叫。這有助於確保檢查程式碼的人員不會洩露機密，因為機密不存在。如需 Secrets Manager 的概觀，請參閱什麼是 AWS Secrets Manager。

建立資料庫執行個體時，InfluxDB 的 Timestream 會自動為您建立管理員秘密，以便與多使用者輪換 AWS Lambda 函數搭配使用。若要輪換 InfluxDB 使用者和權杖的 Timestream，您必須為要輪換的每個使用者或權杖手動建立新的秘密。每個秘密都可以設定為使用 Lambda 函數依排程輪換。設定新輪換秘密的程序包括上傳 Lambda 函數程式碼、設定 Lambda 角色、定義新秘密，以及設定秘密輪換排程。

機密中有什麼

當您將 Timestream for InfluxDB 使用者憑證存放在秘密中時，請使用下列格式。

單一使用者：

```
{
  "engine": "<required: must be set to 'timestream-influxdb'>",
  "username": "<required: username>",
  "password": "<required: password>",
  "dbIdentifier": "<required: DB identifier>"
}
```

當您建立 Timestream for InfluxDB 執行個體時，管理員秘密會自動儲存在 Secrets Manager 中，並具有要與多使用者 Lambda 函數搭配使用的憑證。將 adminSecretArn設定為資料庫執行個體摘要頁面上Authentication Properties Secret Manager ARN的值，或設定為管理員秘密ARN的 。若要建立新的管理員秘密，您必須已擁有相關聯的憑證，且憑證必須具有管理員權限。

當您將 Timestream for InfluxDB 字符憑證存放在秘密中時，請使用下列格式。

多使用者：

```
{
  "engine": "<required: must be set to 'timestream-influxdb'>",
  "org": "<required: organization to associate token with>",
  "adminSecretArn": "<required: ARN of the admin secret>",
  "type": "<required: allAccess or operator or custom>",
  "dbIdentifier": "<required: DB identifier>",
  "token": "<required unless generating a new token: token being rotated>",
  "writeBuckets": "<optional: list of bucketIDs for custom type token, must be input
 within plaintext panel, for example ['id1','id2']>",
  "readBuckets": "<optional: list of bucketIDs for custom type token, must be input
 within plaintext panel, for example ['id1','id2']>",
  "permissions": "<optional: list of permissions for custom type token, must be input
 within plaintext panel, for example ['write-tasks','read-tasks']>"
}
```

當您將 InfluxDB 管理員憑證的 Timestream 存放在秘密中時，請使用下列格式：

管理員秘密：

```
{
  "engine": "<required: must be set to 'timestream-influxdb'>",
  "username": "<required: username>",
  "password": "<required: password>",
  "dbIdentifier": "<required: DB identifier>",
  "organization": "<optional: initial organization>",
  "bucket": "<optional: initial bucket>"
}
```

若要開啟秘密的自動輪換，秘密必須位於正確的JSON結構中。如需如何輪換 InfluxDB 秘密的 Timestream輪換秘密，請參閱 。

修改機密

在 Timestream for InfluxDB 執行個體建立程序期間產生的憑證會儲存在您帳戶中的 Secrets Manager 秘密中。GetDbInstance 回應物件包含將 Amazon Resource Name （ ARN ） 保留為此類秘密`influxAuthParametersSecretArn`的 。只有在您的 InfluxDB 執行個體 Timestream 可用之後，才會填入秘密。這是複本，因為updates/modifications/deletions此秘密的任何 READONLY 不會影響建立的資料庫執行個體。如果您刪除此秘密，API回應仍會參考已刪除的秘密 ARN。

若要在 Timestream for InfluxDB 執行個體中建立新的權杖，而不是儲存現有的權杖憑證，您可以透過在秘密中將`token`值保留空白，並使用 `AUTHENTICATION_CREATION_ENABLED` Lambda 環境變數設定為 的多使用者輪換函數來建立非運算器權杖`true`。如果您建立新的權杖，則秘密中定義的許可會指派給權杖，且無法在第一次成功輪換後變更。如需輪換秘密的詳細資訊，請參閱輪換 AWS 秘密管理員秘密 。

如果刪除秘密，則不會刪除 Timestream for InfluxDB 執行個體中的相關使用者或權杖。

輪換秘密

您可以使用 Timestream for InfluxDB 單一和多使用者輪換 Lambda 函數來輪換 Timestream for InfluxDB 使用者和權杖憑證。使用單一使用者 Lambda 函數輪換 Timestream for InfluxDB 執行個體的使用者憑證，並使用多使用者 Lambda 函數輪換 Timestream for InfluxDB 執行個體的權杖憑證。

使用單一使用者和多使用者 Lambda 函數輪換使用者和權杖是選用的。InfluxDB 憑證的時間串流永不過期，任何公開的憑證都對您的資料庫執行個體構成惡意動作的風險。使用 Secrets Manager 輪換 Timestream for InfluxDB 憑證的優點是新增的安全層，可將公開憑證的攻擊向量限制在時間窗口，直到下一個輪換週期為止。如果您的資料庫執行個體沒有輪換機制，任何公開的憑證在手動刪除之前都是有效的。

您可以設定 Secrets Manager 根據指定的排程自動輪換秘密。這可讓您以短期秘密取代長期秘密，有助於大幅降低洩漏風險。如需使用 Secrets Manager 輪換秘密的詳細資訊，請參閱輪換 AWS Secrets Manager Secrets 。

輪換使用者

當您使用單一使用者 Lambda 函數輪換使用者時，每次輪換後都會將新的隨機密碼指派給使用者。如需如何啟用自動輪換的詳細資訊，請參閱設定非資料庫 AWS Secrets Manager 秘密的自動輪換。

輪換管理員秘密

若要輪換管理員秘密，請使用單一使用者輪換函數。您需要將 `engine`和 `dbIdentifier`值新增至秘密，因為這些值不會自動填入資料庫初始化。如需完整的秘密範本，機密中有什麼請參閱 。

若要尋找 Timestream for InfluxDB 執行個體的管理員秘密，請使用 Timestream for InfluxDB 執行個體摘要頁面ARN中的管理員秘密。建議您輪換 InfluxDB 管理員秘密的所有 Timestream，因為管理員使用者具有提升的 Timestream for InfluxDB 執行個體許可。

Lambda 輪換函數

您可以使用機密中有什麼具有新秘密的 來輪換具有單一使用者輪換函數的 Timestream for InfluxDB 使用者，並新增 Timestream for InfluxDB 使用者的必填欄位。如需秘密輪換 Lambda 函數的詳細資訊，請參閱依 Lambda 函數輪換 。

單一使用者輪換函數會使用秘密中定義的憑證，使用 Timestream for InfluxDB 資料庫執行個體進行驗證，然後產生新的隨機密碼，並為使用者設定新密碼。如需秘密輪換 Lambda 函數的詳細資訊，請參閱依 Lambda 函數輪換 。

Lambda 函數執行角色許可

使用下列IAM政策作為單一使用者 Lambda 函數的角色。此政策為 Lambda 函數提供必要的許可，以為 InfluxDB 使用者執行 Timestream 的秘密輪換。

將IAM政策中列出的所有項目取代為來自您 AWS 帳戶的值：

- {rotating_secret_arn} — 要在 Secrets Manager 秘密詳細資訊中找到所輪換秘密ARN的 。
- {db_instance_arn} — ARN可以在 Timestream for InfluxDB 執行個體摘要頁面上找到 InfluxDB 執行個體的 Timestream。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "secretsmanager:UpdateSecretVersionStage"
            ],
     "Resource": "{rotating_secret_arn}"
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "secretsmanager:GetRandomPassword"
            ],
            "Resource": "*"
        },
        {
            "Action": [
                "timestream-influxdb:GetDbInstance"
            ],
     "Resource": "{db_instance_arn}",
            "Effect": "Allow"
        }
    ]
}
```

輪換權杖

您可以使用機密中有什麼具有新秘密的 ，搭配多使用者輪換函數輪換 Timestream for InfluxDB 權杖，並新增 Timestream for InfluxDB 權杖的必要欄位。如需秘密輪換 Lambda 函數的詳細資訊，請參閱依 Lambda 函數輪換 。

您可以使用 Timestream for InfluxDB 多使用者 Lambda 函數來輪換 Timestream for InfluxDB 權杖。在 Lambda 組態`true`中將`AUTHENTICATION_CREATION_ENABLED`環境變數設定為 ，以啟用權杖建立。若要建立新的權杖，請將 機密中有什麼用於您的秘密值。省略新秘密中的`token`鍵值對，並將`type`設定為 `allAccess`，或定義特定許可並將類型設定為 `custom`。輪換函數會在第一個輪換週期期間建立新的權杖。您無法在輪換後編輯秘密來變更權杖許可，任何後續輪換都會使用資料庫執行個體中設定的許可。

Lambda 輪換函數

多使用者輪換函數會使用管理員秘密中的管理員憑證，建立新的許可來輪換權杖憑證。Lambda 函數會在建立取代權杖、將新權杖值儲存在秘密中，以及刪除舊權杖之前，驗證秘密中的權杖值。如果 Lambda 函數正在建立新的權杖，它首先會驗證`AUTHENTICATION_CREATION_ENABLED`環境變數是否設定為 `true`、秘密中沒有權杖值，以及權杖類型不是類型運算子。

Lambda 函數執行角色許可

使用下列IAM政策作為多使用者 Lambda 函數的角色。此政策為 Lambda 函數提供必要的許可，以針對 Timestream for InfluxDB 權杖執行秘密輪換。

將IAM政策中列出的所有項目取代為來自您 AWS 帳戶的值：

- {rotating_secret_arn} — 要在 Secrets Manager 秘密詳細資訊中找到所輪換秘密ARN的 。
- {authentication_properties_admin_secret_arn } — InfluxDB 管理秘密的 Timestream ARN可在 InfluxDB 執行個體摘要的 Timestream 頁面上找到。
- {db_instance_arn} — ARN可以在 Timestream for InfluxDB 執行個體摘要頁面上找到 InfluxDB 執行個體的 Timestream。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "secretsmanager:UpdateSecretVersionStage"
            ],
     "Resource": "{rotating_secret_arn}"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
     "Resource": "{authentication_properties_admin_secret_arn}"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword"
            ],
            "Resource": "*"
        },
        {
            "Action": [
                "timestream-influxdb:GetDbInstance"
            ],
     "Resource": "{db_instance_arn}",
            "Effect": "Allow"
        }
    ]
```

```
}
```

Timestream for InfluxDB 中的資料保護

AWS 共同責任模型適用於 Amazon Timestream for InfluxDB 中的資料保護。如本模型所述， AWS 負責保護執行所有 的全域基礎設施 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 AWS 服務 的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱資料隱私權。FAQ如需歐洲資料保護的相關資訊，請參閱AWS 安全部落格 上的AWS 共同責任模型和GDPR部落格文章。

為了資料保護目的，我們建議您保護 AWS 帳戶 憑證，並使用 AWS IAM Identity Center 或 AWS Identity and Access Management （ ） 設定個別使用者IAM。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 對每個帳戶使用多重要素驗證 （MFA）。
- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 和建議 TLS 1.3。
- 使用 設定 API和使用者活動日誌 AWS CloudTrail。如需使用 CloudTrail 線索擷取 AWS 活動的資訊，請參閱 AWS CloudTrail 使用者指南 中的使用 CloudTrail 線索。
- 使用 AWS 加密解決方案，以及 中的所有預設安全控制項 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在 AWS 透過命令列介面或 FIPS 存取 時需要 140-3 個經過驗證的密碼編譯模組API，請使用FIPS端點。如需可用FIPS端點的詳細資訊，請參閱聯邦資訊處理標準 （FIPS） 140-3。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用 Timestream for InfluxDB 或使用主控台API AWS CLI、 或 的其他 AWS 服務 時 AWS SDKs。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您將 URL提供給外部伺服器，強烈建議您在 中不要包含憑證資訊，URL以驗證您對該伺服器的請求。

如需 Timestream for InfluxDB 資料保護主題的詳細資訊，例如靜態加密和金鑰管理，請選取下列任何可用的主題。

主題

- 靜態加密
- 傳輸中加密

靜態加密

靜態 InfluxDB 加密的 Timestream 透過使用 AWS Key Management Service （ AWS KMS ） 中存放的加密金鑰加密您靜態所有資料，提供增強的安全性。此功能協助降低了保護敏感資料所涉及的操作負擔和複雜性。您可以透過靜態加密，建立符合嚴格加密合規和法規要求，而且對安全性要求甚高的應用程式。

- Timestream for InfluxDB 資料庫執行個體預設為開啟加密，且無法關閉。業界標準 AES-256 加密演算法是使用的預設加密演算法。
- AWS KMS 在 Timestream for InfluxDB 中用於靜態加密。
- 您不需要修改資料庫執行個體用戶端應用程式即可使用加密。

傳輸中加密

您的所有 Timestream for InfluxDB 資料都會在傳輸中加密。根據預設，所有與 Timestream for InfluxDB 之間的通訊都會使用 Transport Layer Security （ TLS ） 加密進行保護。

往返 Amazon Timestream for InfluxDB 的流量是使用支援的 1.2 或 1.3 TLS版進行安全保護。

Amazon Timestream for InfluxDB 的身分和存取管理

AWS Identity and Access Management （ IAM ） 是一種 AWS 服務 ，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員會控制誰可以進行身分驗證 （ 登入 ） 和授權 （ 具有許可 ），以使用 Timestream for InfluxDB 資源。IAM 是 AWS 服務 您可以免費使用的 。

主題

- 使用身分驗證
- 使用政策管理存取權
- Amazon Timestream for InfluxDB 如何與 搭配使用 IAM
- 適用於 InfluxDB 的 Amazon Timestream 身分型政策範例
- 對 Amazon Timestream for InfluxDB 身分和存取權進行故障診斷
- 在 中控制資料庫執行個體的存取 VPC
- 針對 Amazon Timestream for InfluxDB 使用服務連結角色
- AWS Amazon Timestream for InfluxDB 的 受管政策
- 透過VPC端點連線至 Timestream for InfluxDB

使用身分驗證

驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者身分、IAM使用者身分或擔任IAM角色來驗證 （登入 AWS）。

您可以使用透過身分來源提供的憑證，以聯合身分 AWS 身分登入 。 AWS IAM Identity Center （IAM Identity Center） 使用者、您公司的單一登入身分驗證，以及您的 Google 或 Facebook 憑證，都是聯合身分的範例。當您以聯合身分登入時，您的管理員先前會使用 IAM角色設定身分聯合。當您 AWS 使用聯合存取 時，您會間接擔任 角色。

您可以登入 AWS Management Console 或 AWS 存取入口網站，視您是的使用者類型而定。如需登入的詳細資訊 AWS，請參閱 使用者指南 中的如何登入 AWS 帳戶您的 。 AWS 登入

如果您以 AWS 程式設計方式存取 ， AWS 會提供軟體開發套件 （SDK） 和命令列介面 （CLI），以使用您的 憑證以密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，則必須自行簽署請求。如需使用建議方法自行簽署請求的詳細資訊，請參閱 IAM 使用者指南 中的AWS 簽章第 4 版以取得API請求。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如， AWS 建議您使用多重要素身分驗證 （MFA） 來提高帳戶的安全性。若要進一步了解，請參閱 AWS IAM Identity Center 使用者指南中的多重要素驗證，以及 IAM 使用者指南 AWS 中的多重要素驗證IAM。

IAM 使用者和群組

IAM 使用者是 中具有單一個人或應用程式特定許可 AWS 帳戶 的身分。如果可能，我們建議您依賴臨時憑證，而不是建立具有密碼和存取金鑰等長期憑證IAM的使用者。不過，如果您有特定的使用案例需要IAM使用者長期憑證，建議您輪換存取金鑰。如需詳細資訊，請參閱 IAM 使用者指南 中的針對需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是指定IAM使用者集合的身分。您無法以群組身分簽署。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的程序變得更為容易。例如，您可以擁有名為 的群組IAMAdmins，並授予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供暫時憑證。若要進一步了解，請參閱 IAM 使用者指南 中的IAM使用者使用案例。

IAM 角色

IAM 角色是 中具有特定許可 AWS 帳戶 的身分。它類似於IAM使用者，但與特定人員無關。若要暫時在 中擔任IAM角色 AWS Management Console，您可以從使用者切換至IAM角色 （主控台）。您可以

呼叫 AWS CLI 或 AWS API 操作，或使用自訂 來擔任角色URL。如需使用角色方法的詳細資訊，請參閱 IAM 使用者指南 中的擔任角色的方法。

IAM 具有臨時憑證的角色在下列情況下很有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需聯合角色的相關資訊，請參閱 IAM 使用者指南 中的為第三方身分提供者建立角色。如果您使用 IAM Identity Center，您可以設定許可集。若要控制身分在身分驗證後可存取的內容，IAMIdentity Center 會將許可集與 中的角色相關聯IAM。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的許可集。
- 臨時IAM使用者許可 – IAM使用者或角色可以擔任IAM角色，暫時接受特定任務的不同許可。
- 跨帳戶存取 – 您可以使用 IAM角色，允許不同帳戶中的某人（受信任的主體）存取您帳戶中的資源。角色是授予跨帳戶存取權的主要方式。不過，在某些 中 AWS 服務，您可以將政策直接連接至資源（而不是使用角色作為代理）。若要了解跨帳戶存取的角色與資源型政策之間的差異，請參閱 IAM 使用者指南 中的跨帳戶資源存取IAM。
- 跨服務存取 – 有些 AWS 服務 使用其他 中的功能 AWS 服務。例如，當您在 服務中撥打電話時，該服務通常會在 Amazon 中執行應用程式EC2或在 Amazon S3 中儲存物件。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉送存取工作階段（FAS）– 當您使用IAM使用者或角色在 中執行動作時 AWS，您會被視為主體。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫 的委託人許可 AWS 服務，並結合請求向下游服務 AWS 服務 提出請求的 。FAS 只有在服務收到需要與其他 AWS 服務 或 資源互動才能完成的請求時，才會發出請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出FAS請求的政策詳細資訊，請參閱轉送存取工作階段 。
  - 服務角色 – 服務角色是服務代表您執行動作時擔任IAM的角色。IAM 管理員可以從 內部建立、修改和刪除服務角色IAM。如需詳細資訊，請參閱 使用者指南 中的建立角色以將許可委派給 AWS 服務 。 IAM
  - 服務連結角色 – 服務連結角色是連結至 的服務角色類型 AWS 服務。服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的 中 AWS 帳戶 ，並由 服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon 上執行的應用程式 EC2 – 您可以使用 IAM角色來管理在EC2執行個體上執行之應用程式的臨時憑證，以及提出 AWS CLI 或 AWS API請求。最好將存取金鑰儲存在EC2執行個體中。若要將 AWS 角色指派給EC2執行個體並將其提供給其所有應用程式，您可以建立連接至執行個體的執行個體設定檔。執行個體設定檔包含 角色，並啟用執行個體上執行的程式EC2，以取得臨時憑證。如需詳細資訊，請參閱 IAM 使用者指南 中的使用 IAM角色將許可授予在 Amazon EC2執行個體上執行的應用程式。

使用政策管理存取權

您可以透過建立政策並將其連接至 AWS 身分或資源 AWS 來控制 中的存取。政策是 AWS 其中的物件，當與身分或資源相關聯時， 會定義其許可。當主體 （使用者、根使用者或角色工作階段） 發出請求時， 會 AWS 評估這些政策。政策中的許可決定是否允許或拒絕請求。大多數政策都以JSON文件 AWS 形式儲存在 中。如需JSON政策文件結構和內容的詳細資訊，請參閱 IAM 使用者指南 中的JSON政策概觀。

管理員可以使用 AWS JSON政策來指定誰可以存取什麼。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對所需資源執行動作的許可，IAM管理員可以建立IAM政策。然後，管理員可以將IAM政策新增至角色，使用者可以擔任角色。

IAM 無論您用來執行操作的方法為何，政策都會定義動作的許可。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該政策的使用者可以從 AWS Management Console、 AWS CLI或 AWS 取得角色資訊API。

身分型政策

身分型政策是您可以連接到身分的JSON許可政策文件，例如IAM使用者、使用者群組或角色。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分型政策，請參閱 IAM 使用者指南 中的使用客戶受管政策定義自訂IAM許可。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管政策是獨立的政策，您可以連接到 中的多個使用者、群組和角色 AWS 帳戶。受管政策包括 AWS 受管政策和客戶受管政策。若要了解如何在受管政策或內嵌政策之間進行選擇，請參閱 IAM 使用者指南 中的在受管政策與內嵌政策之間進行選擇。

資源型政策

資源型政策是您連接至資源JSON的政策文件。資源型政策的範例包括IAM角色信任政策和 Amazon S3 儲存貯體政策 。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。主體可以包括帳戶、使用者、角色、聯合使用者或 AWS 服務。

資源型政策是位於該服務中的內嵌政策。您無法在資源型政策IAM中使用來自 的 AWS 受管政策。

存取控制清單 （ACLs）

存取控制清單 （ACLs） 控制哪些主體 （帳戶成員、使用者或角色） 具有存取 資源的許可。ACLs 類似於資源型政策，雖然它們不使用JSON政策文件格式。

Amazon S3 AWS WAF和 Amazon VPC是支援 的服務範例ACLs。若要進一步了解 ACLs，請參閱 Amazon Simple Storage Service 開發人員指南 中的存取控制清單 （ACL） 概觀。

其他政策類型

AWS 支援其他較不常見的政策類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可界限是一項進階功能，您可以在其中設定身分型政策可授予IAM實體 （IAM使用者或角色） 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可界限的詳細資訊，請參閱 IAM 使用者指南 中的IAM實體許可界限。
- 服務控制政策 （SCPs） – SCPs是在 中指定組織或組織單位 （OU） 最大許可JSON的政策 AWS Organizations。 AWS Organizations 是一種用於分組和集中管理您企業擁有 AWS 帳戶 的多個的服務。如果您啟用組織中的所有功能，則可以將服務控制政策 （SCPs） 套用至任何或所有帳戶。SCP 限制成員帳戶中實體的許可，包括每個 AWS 帳戶根使用者。如需 Organizations 和 的詳細資訊SCPs，請參閱 AWS Organizations 使用者指南 中的服務控制政策。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合使用者的暫時工作階段時，作為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參閱 IAM 使用者指南 中的工作階段政策。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。若要了解如何 AWS 在涉及多種政策類型時決定是否允許請求，請參閱 IAM 使用者指南 中的政策評估邏輯。

Amazon Timestream for InfluxDB 如何與 搭配使用 IAM

IAM 您可以搭配 Amazon Timestream for InfluxDB 使用的功能

IAM 功能	InfluxDB 支援的時間串流
身分型政策	是
資源型政策	否
政策動作	是

IAM 功能	InfluxDB 支援的時間串流
政策資源	是
政策條件索引鍵	否
ACLs	否
ABAC （政策中的標籤）	是
暫時性憑證	是
主體許可	是
服務角色	否
服務連結角色	是

若要取得 Timestream for InfluxDB 和其他 AWS 服務如何與大多數 IAM 功能搭配使用的高階檢視，請參閱 IAM 使用者指南 中的 AWS 使用 的服務IAM。

Timestream for InfluxDB 的身分型政策

支援身分型政策：是

身分型政策是您可以連接到身分的JSON許可政策文件，例如IAM使用者、使用者群組或角色。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分型政策，請參閱 IAM 使用者指南 中的使用客戶受管政策定義自訂IAM許可。

透過身分IAM型政策，您可以指定允許或拒絕的動作和資源，以及允許或拒絕動作的條件。您無法在身分型政策中指定主體，因為這會套用至連接的使用者或角色。若要了解您可以在JSON政策中使用的所有元素，請參閱 IAM 使用者指南 中的IAMJSON政策元素參考。

Timestream for InfluxDB 的身分型政策範例

若要檢視 Timestream for InfluxDB 身分型政策的範例，請參閱 適用於 InfluxDB 的 Amazon Timestream 身分型政策範例。

Timestream for InfluxDB 中的資源型政策

支援資源型政策：否

資源型政策是您連接至資源JSON的政策文件。資源型政策的範例包括IAM角色信任政策和 Amazon S3 儲存貯體政策 。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。主體可以包括帳戶、使用者、角色、聯合使用者或 AWS 服務。

若要啟用跨帳戶存取，您可以將另一個帳戶中的整個帳戶或IAM實體指定為資源型政策中的主體。新增跨帳戶主體至資源型政策，只是建立信任關係的一半。當主體和資源位於不同的 時 AWS 帳戶，受信任帳戶中的IAM管理員也必須授予主體實體（使用者或角色）存取資源的許可。其透過將身分型政策連接到實體來授與許可。不過，如果資源型政策會為相同帳戶中的主體授予存取，這時就不需要額外的身分型政策。如需詳細資訊，請參閱 IAM 使用者指南 中的跨帳戶資源存取權IAM。

Timestream for InfluxDB 的政策動作

支援政策動作：是

管理員可以使用 AWS JSON政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 `Action`元素說明您可以用來允許或拒絕政策中存取的動作。政策動作通常具有與相關聯 AWS API操作相同的名稱。有一些例外狀況，例如沒有相符API操作的僅限許可動作。也有一些作業需要政策中的多個動作。這些額外的動作稱為相依動作。

政策會使用動作來授予執行相關聯動作的許可。

若要查看 InfluxDB 動作的 Timestream，請參閱服務授權參考 中的 Amazon Timestream for InfluxDB 定義的動作。

Timestream for InfluxDB 中的政策動作在動作之前使用下列字首：

```
timestream-influxdb
```

如需在單一陳述式中指定多個動作，請用逗號分隔。

```
"Action": [
      "timestream-influxdb:action1",
      "timestream-influxdb:action2"
         ]
```

您也可以使用萬用字元 (*) 來指定多個動作。例如，若要指定開頭是 Describe 文字的所有動作，請包含以下動作：

```
"Action": "timestream-influxdb:Describe*"
```

Timestream for InfluxDB 的政策資源

支援政策資源：是

管理員可以使用 AWS JSON政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Resource JSON 政策元素會指定動作套用的物件。陳述式必須包含 Resource 或 NotResource 元素。最佳實務是使用其 Amazon Resource Name （ ARN ） 指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出操作)，請使用萬用字元 (*) 來表示陳述式適用於所有資源。

```
"Resource": "*"
```

若要查看 InfluxDB 資源類型及其 的 TimestreamARNs，請參閱服務授權參考 中的 Amazon Timestream for InfluxDB 定義的資源。若要了解您可以使用哪些動作指定每個資源ARN的 ，請參閱 Amazon Timestream for InfluxDB 定義的動作。

InfluxDB Timestream 的政策條件索引鍵

支援服務特定的政策條件金鑰：否

管理員可以使用 AWS JSON政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Condition 元素 (或 Condition 區塊) 可讓您指定使陳述式生效的條件。Condition 元素是選用項目。您可以建立使用條件運算子的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。

若您在陳述式中指定多個 Condition 元素，或是在單一 Condition 元素中指定多個索引鍵， AWS 會使用邏輯 AND 操作評估他們。如果您為單一條件索引鍵指定多個值， 會使用邏輯OR操作 AWS 評估條件。必須符合所有條件，才會授與陳述式的許可。

您也可以在指定條件時使用預留位置變數。例如，只有在使用者使用其IAM使用者名稱加上標籤時，您才能授予IAM使用者存取資源的許可。如需詳細資訊，請參閱 IAM 使用者指南 中的IAM政策元素：變數和標籤。

AWS 支援全域條件索引鍵和服務特定條件索引鍵。若要查看所有 AWS 全域條件索引鍵，請參閱 IAM 使用者指南 中的AWS 全域條件內容索引鍵。

Timestream for InfluxDB 中的存取控制清單 （ACLs）

支援ACLs：否

存取控制清單 （ACLs） 控制哪些主體 （帳戶成員、使用者或角色） 具有存取 資源的許可。ACLs 類似於資源型政策，雖然它們不使用JSON政策文件格式。

使用 Timestream for InfluxDB 的屬性型存取控制 （ABAC）

支援 ABAC（政策中的標籤）：是

屬性型存取控制 （ABAC） 是根據屬性定義許可的授權策略。在 中 AWS，這些屬性稱為標籤 。您可以將標籤連接至IAM實體 （使用者或角色） 和許多 AWS 資源。標記實體和資源是 的第一步ABAC。然後，您可以設計ABAC政策，以便在主體的標籤與其嘗試存取之資源上的標籤相符時允許操作。

ABAC 有助於快速成長的環境，並有助於處理政策管理變得繁瑣的情況。

如需根據標籤控制存取，請使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 條件索引鍵，在政策的條件元素中，提供標籤資訊。

如果服務支援每個資源類型的全部三個條件金鑰，則對該服務而言，值為 Yes。如果服務僅支援某些資源類型的全部三個條件金鑰，則值為 Partial。

如需 的詳細資訊ABAC，請參閱 IAM 使用者指南 中的使用ABAC授權定義許可。若要檢視包含設定 之步驟的教學課程ABAC，請參閱 IAM 使用者指南 中的使用屬性型存取控制 （ABAC）。

搭配 Timestream for InfluxDB 使用暫時憑證

支援臨時憑證：是

當您使用臨時憑證登入時，有些 AWS 服務 無法使用。如需詳細資訊，包括 AWS 服務 使用哪些臨時憑證，請參閱 IAM 使用者指南 中的 AWS 服務 與 搭配使用IAM。

如果您 AWS Management Console 使用使用者名稱和密碼以外的任何方法登入 ，則表示您正在使用臨時憑證。例如，當您 AWS 使用公司的單一登入 （SSO） 連結存取 時，該程序會自動建立臨時憑

證。當您以使用者身分登入主控台，然後切換角色時，也會自動建立臨時憑證。如需切換角色的詳細資訊，請參閱 IAM 使用者指南 中的從使用者切換至IAM角色（主控台）。

您可以使用 AWS CLI 或 手動建立臨時憑證 AWS API。然後，您可以使用這些臨時憑證來存取 AWS. AWS recommends，讓您動態產生臨時憑證，而不是使用長期存取金鑰。如需詳細資訊，請參閱 中的臨時安全憑證IAM。

Timestream for InfluxDB 的跨服務主體許可

支援轉送存取工作階段（FAS）：是

當您使用IAM使用者或角色在 中執行動作時 AWS，您會被視為主體。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫 的委託人許可 AWS 服務，並結合請求向下游服務 AWS 服務 提出請求。FAS 只有在服務收到需要與其他 AWS 服務 或 資源互動才能完成的請求時，才會發出請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出FAS請求時的政策詳細資訊，請參閱轉送存取工作階段。

Timestream for InfluxDB 的服務角色

支援服務角色：否

服務角色是服務代表您執行動作時擔任IAM的角色。IAM 管理員可以從 內部建立、修改和刪除服務角色IAM。如需詳細資訊，請參閱 使用者指南 中的建立角色以將許可委派給 AWS 服務 。 IAM

> **Warning**
>
> 變更服務角色的許可可能會中斷 InfluxDB 功能的 Timestream。只有在 Timestream for InfluxDB 提供指引時，才能編輯服務角色。

Timestream for InfluxDB 的服務連結角色

支援服務連結角色：是

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

如需建立或管理服務連結角色的詳細資訊，請參閱AWS 使用 的服務IAM。在表格中尋找服務，其中包含服務連結角色欄中的 Yes。選擇是連結，以檢視該服務的服務連結角色文件。

適用於 InfluxDB 的 Amazon Timestream 身分型政策範例

根據預設，使用者和角色沒有建立或修改 InfluxDB 資源 Timestream 的許可。他們也無法使用 AWS Management Console、 AWS Command Line Interface （ AWS CLI ） 或 來執行任務 AWS API。若要授予使用者對所需資源執行動作的許可，IAM管理員可以建立IAM政策。然後，管理員可以將IAM政策新增至角色，使用者可以擔任角色。

若要了解如何使用這些範例政策文件來建立IAM身分型JSON政策，請參閱 IAM 使用者指南 中的建立IAM政策 （ 主控台 ）。

如需 Timestream for InfluxDB 所定義動作和資源類型的詳細資訊，包括ARNs每種資源類型的 格式，請參閱服務授權參考 中的適用於 Amazon Timestream for InfluxDB 的動作、資源和條件金鑰。

主題

政策最佳實務

身分型政策會判斷是否有人可以在您的帳戶中建立、存取或刪除 InfluxDB 資源的 Timestream。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用 AWS 受管政策，將許可授予許多常見使用案例。它們可在您的 中使用 AWS 帳戶。建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需詳細資訊，請參閱 IAM 使用者指南 中的 AWS 受管政策或 AWS 任務功能的受管政策。
- 套用最低權限許可 – 當您使用 IAM 政策設定許可時， 只會授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的詳細資訊，請參閱 IAM 使用者指南 中的政策和許可IAM。
- 使用IAM政策中的條件來進一步限制存取：您可以將條件新增至政策，以限制對動作和資源的存取。例如，您可以撰寫政策條件來指定所有請求都必須使用 傳送SSL。如果透過特定 使用服務動作，例如 AWS 服務，您也可以使用 條件來授予其存取權 AWS CloudFormation。如需詳細資訊，請參閱 IAM 使用者指南 中的IAMJSON政策元素：條件。

- 使用 IAM Access Analyzer 驗證您的IAM政策以確保安全且功能許可 – IAM Access Analyzer 會驗證新的和現有的政策，以便政策遵循IAM政策語言 （JSON） 和IAM最佳實務。IAM Access Analyzer 提供超過 100 個政策檢查和可操作的建議，協助您撰寫安全且實用的政策。如需詳細資訊，請參閱 IAM 使用者指南 中的使用 IAM Access Analyzer 驗證政策。
- 需要多重要素身分驗證 （MFA） – 如果您有需要IAM使用者或 根使用者的案例 AWS 帳戶，請開啟 MFA 以獲得額外的安全性。若要在呼叫API操作MFA時要求 ，請將MFA條件新增至您的政策。如需詳細資訊，請參閱 IAM 使用者指南 中的使用 安全API存取MFA。

如需 中最佳實務的詳細資訊IAM，請參閱 IAM 使用者指南 中的安全最佳實務IAM。

使用 Timestream for InfluxDB 主控台

若要存取 Amazon Timestream for InfluxDB 主控台，您必須具有一組最低許可。這些許可必須允許您列出和檢視 中有關 InfluxDB 資源 Timestream 的詳細資訊 AWS 帳戶。如果您建立比最基本必要許可更嚴格的身分型政策，則對於具有該政策的實體 (使用者或角色) 而言，主控台就無法如預期運作。

對於僅對 AWS CLI 或 進行呼叫的使用者，您不需要允許最低主控台許可 AWS API。相反地，僅允許存取與其嘗試執行API的操作相符的動作。

為了確保使用者和角色仍然可以使用 Timestream for InfluxDB 主控台，也請將 Timestream for InfluxDB `ConsoleAccess`或`ReadOnly` AWS 受管政策連接至實體。如需詳細資訊，請參閱 IAM 使用者指南 中的新增許可給使用者。

允許使用者檢視他們自己的許可

此範例示範如何建立政策，允許使用者檢視連接至其IAM使用者身分的內嵌和受管政策。此政策包含在主控台上完成此動作或使用 或 AWS CLI 以程式設計方式完成此動作的許可 AWS API。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
```

```
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

存取一個 Amazon S3 儲存貯體

在此範例中，您想要將 AWS 帳戶中IAM的使用者存取權授予其中一個 Amazon S3 儲存貯體 `examplebucket`。您也希望允許使用者新增、更新和刪除物件。

除了授予使用者 `s3:PutObject`、`s3:GetObject` 與 `s3:DeleteObject` 許可之外，政策也會授予 `s3:ListAllMyBuckets`、`s3:GetBucketLocation` 與 `s3:ListBucket` 許可。這些是主控台需要的額外許可。還需要 `s3:PutObjectAcl` 與 `s3:GetObjectAcl` 動作才能在主控台中複製、剪下與貼上物件。如需將許可授予使用者並使用主控台進行測試的範例逐步解說，請參閱[範例逐步解說：使用使用者政策來控制對儲存貯體 的存取](#)。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListBucketsInConsole",
         "Effect":"Allow",
         "Action":[
            "s3:ListAllMyBuckets"
         ],
         "Resource":"arn:aws:s3:::*"
      },
      {
```

```
         "Sid":"ViewSpecificBucketInfo",
         "Effect":"Allow",
         "Action":[
            "s3:ListBucket",
            "s3:GetBucketLocation"
         ],
         "Resource":"arn:aws:s3:::examplebucket"
      },
      {
         "Sid":"ManageBucketContents",
         "Effect":"Allow",
         "Action":[
            "s3:PutObject",
            "s3:PutObjectAcl",
            "s3:GetObject",
            "s3:GetObjectAcl",
            "s3:DeleteObject"
         ],
         "Resource":"arn:aws:s3:::examplebucket/*"
      }
   ]
}
```

允許所有操作

以下是允許 Timestream for InfluxDB 中的所有操作的範例政策。

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

建立、描述、刪除和更新資料庫執行個體

下列範例政策允許使用者建立、描述、刪除和更新資料庫執行個體sampleDB：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream-influxdb:CreateDbInstance",
                "timestream-influxdb:GetDbInstance",
                "timestream-influxdb:DeleteDbInstance",
                "timestream-influxdb:UpdateDbInstance"
            ],
            "Resource": "arn:aws:timestream-influxdb:us-east-1:<account_ID>:dbinstance/
sampleDB"
        }
    ]
}
```

對 Amazon Timestream for InfluxDB 身分和存取權進行故障診斷

使用下列資訊來協助您診斷和修正使用 Timestream for InfluxDB 和 時可能遇到的常見問題IAM。

主題

- 我無權在 Timestream for InfluxDB 中執行動作
- 我想要允許 AWS 帳戶外的人員存取我的 Timestream for InfluxDB 資源

我無權在 Timestream for InfluxDB 中執行動作

如果 AWS Management Console 告訴您未獲授權執行動作，則必須聯絡管理員尋求協助。您的管理員是提供您使用者名稱和密碼的人員。

下列範例錯誤會在 mateojackson 使用者嘗試使用主控台檢視一個虛構 *my-example-widget* 資源的詳細資訊，但卻無虛構 timestream-influxdb:*GetWidget* 許可時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 timestream-influxdb:GetWidget on resource: my-example-widget
```

在此情況下，Mateo 會請求管理員更新他的政策，允許他使用 *my-example-widget* 動作存取 timestream-influxdb:*GetWidget* 資源。

我想要允許 AWS 帳戶外的人員存取我的 Timestream for InfluxDB 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。對於支援資源型政策或存取控制清單（ACLs）的服務，您可以使用這些政策來授予人員對資源的存取權。

如需進一步了解，請參閱以下內容：

- 在 中控制資料庫執行個體的存取 VPC
- 若要了解 Timestream for InfluxDB 是否支援這些功能，請參閱 Amazon Timestream for InfluxDB 如何與 搭配使用IAM。
- 若要了解如何在您擁有 AWS 的帳戶中提供資源的存取權，請參閱 IAM 使用者指南 中的為您擁有的另一個 AWS 帳戶中IAM的使用者提供存取權。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱 IAM 使用者指南 中的提供第三方擁有 AWS 帳戶的存取權。
- 若要了解如何透過身分聯合提供存取權，請參閱 IAM 使用者指南 中的為外部驗證的使用者提供存取權（身分聯合）。
- 若要了解跨帳戶存取使用角色和資源型政策之間的差異，請參閱 IAM 使用者指南 中的IAM角色與資源型政策有何不同。

在 中控制資料庫執行個體的存取 VPC

使用 Amazon Virtual Private Cloud（Amazon VPC），您可以將 Amazon Timestream for InfluxDB 資料庫執行個體等 AWS 資源啟動至虛擬私有雲端（VPC）。使用 Amazon 時VPC，您可以控制虛擬聯網環境。您可以選擇自己的 IP 地址範圍、建立子網路，以及設定路由和存取控制清單。

VPC 安全群組控制對 內資料庫執行個體的存取VPC。每個VPC安全群組規則都會讓特定來源存取與該VPC安全群組VPC相關聯的 中的資料庫執行個體。來源可以是一組地址（例如 203.0.113.0/24）或其他VPC安全群組。透過將VPC安全群組指定為來源，您可以允許來自使用來源VPC安全群組之所有執行個體（通常是應用程式伺服器）的傳入流量。在嘗試連線至資料庫執行個體之前，請VPC為您的使用案例設定 。以下是在 中存取資料庫執行個體的常見案例VPC：

Amazon 執行個體在相同 中VPC存取的 中的資料庫EC2執行個體 VPC

> 資料庫執行個體在 中的常見用途，VPC是與在相同 中EC2執行個體中執行的應用程式伺服器共用資料VPC。EC2 執行個體可能會使用與資料庫執行個體互動的應用程式執行 Web 伺服器。

由不同 執行個體VPC存取的 中的資料庫EC2執行個體 VPC

在某些情況下，您的資料庫執行個體與您用來存取該執行個體的EC2執行個體VPC不同。如果是這樣，您可以使用VPC對等來存取資料庫執行個體。

用戶端應用程式透過網際網路VPC存取的 中的資料庫執行個體

若要透過網際網路VPC從用戶端應用程式存取 中的資料庫執行個體，您可以使用VPC單一公有子網路設定 ，並使用公有子網路來建立資料庫執行個體。您也可以在 中設定網際網路閘道VPC，以啟用網際網路通訊。若要從其 外部連線至資料庫執行個體VPC，資料庫執行個體必須可公開存取。此外，必須使用資料庫執行個體安全群組的入站規則授予存取權，且必須符合其他需求。

如需VPC安全群組的詳細資訊，請參閱 Amazon Virtual Private Cloud 使用者指南 中的安全群組。

如需如何連線至 Timestream for InfluxDB 資料庫執行個體的詳細資訊，請參閱 連線至 Amazon Timestream for InfluxDB 資料庫執行個體。

安全群組案例

資料庫執行個體在 中的常見用途，VPC是與相同 Amazon EC2執行個體中執行的應用程式伺服器共用資料VPC，該伺服器由 外部的用戶端應用程式存取VPC。在此案例中，您可以使用 InfluxDB 的 Timestream 和 上的VPC頁面， AWS Management Console 或 InfluxDB 的 Timestream 和 EC2 API 操作來建立必要的執行個體和安全群組：

1. 建立VPC安全群組（例如 `sg-0123ec2example`），並定義使用用戶端應用程式的 IP 地址作為來源的傳入規則。此安全群組可讓您的用戶端應用程式連線至使用此安全群組VPC的 EC2 執行個體。
2. 為應用程式建立EC2執行個體，並將EC2執行個體新增至您在上一個步驟中建立VPC的安全群組（`sg-0123ec2example`）。
3. 建立第二個VPC安全群組（例如 `sg-6789rdsexample`），並將您在步驟 1（`sg-0123ec2example`）中建立VPC的安全群組指定為來源，以建立新的規則。
4. 建立新的資料庫執行個體，並將資料庫執行個體新增至您在上一個步驟中建立VPC的安全群組（`sg-6789rdsexample`）。當您建立資料庫時，請使用與您在步驟 3 中建立VPC的安全群組（`sg-6789rdsexample`）規則指定的連接埠號碼相同的連接埠號碼。

建立VPC安全群組

您可以使用 VPC主控台為資料庫執行個體建立VPC安全群組。如需建立安全群組的相關資訊，請參閱 Amazon Virtual Private Cloud 使用者指南 中的安全群組。

將安全群組與資料庫執行個體建立關聯

您可以使用 Timestream for InfluxDB 主控台上的更新、UpdateDBInstanceTimestream for InfluxDB API或 update-db-instance AWS CLI 命令，將安全群組與資料庫執行個體建立關聯。

下列CLI範例會建立特定VPC安全群組的關聯，並從資料庫執行個體中移除資料庫安全群組

```
aws timestream-influxdb update-db-instance --identifier dbName --vpc-security-group-
ids sg-ID
```

如需修改資料庫執行個體的相關資訊，請參閱更新資料庫執行個體。

針對 Amazon Timestream for InfluxDB 使用服務連結角色

Amazon Timestream for InfluxDB 使用 AWS Identity and Access Management （ IAM ） 服務連結角色 。服務連結角色是直接連結至 AWS 服務的唯一IAM角色類型，例如 Amazon Timestream for InfluxDB 。Amazon Timestream for InfluxDB 服務連結角色是由 Amazon Timestream for InfluxDB 預先定義。它們包含服務代表 dbinstance 呼叫 AWS 服務所需的所有許可。

服務連結角色可讓您更輕鬆地為 InfluxDB 設定 Amazon Timestream，因為您不需要手動新增必要的許可。這些角色已存在於 AWS 您的帳戶中，但會連結至 Amazon Timestream for InfluxDB 使用案例，並具有預先定義的許可。只有 Amazon Timestream for InfluxDB 可以擔任這些角色，而且只有這些角色可以使用預先定義的許可政策。您必須先刪除角色的相關資源，才能刪除角色。這可保護您的 Amazon Timestream for InfluxDB 資源，因為您不會不小心移除存取資源的必要許可。

如需支援服務連結角色的其他 服務的資訊，請參閱AWS 服務連結角色欄中的 服務，IAM並尋找具有是的服務。 選擇具有連結的是，以檢視該服務的服務連結角色文件。

內容

- Amazon Timestream for InfluxDB 的服務連結角色許可
- 建立服務連結角色 （ IAM ）
- 編輯 Amazon Timestream for InfluxDB 的服務連結角色描述
  - 編輯服務連結角色描述 （ IAM 主控台 ）
  - 編輯服務連結角色描述 （ IAM CLI ）
  - 編輯服務連結角色描述 （ IAM API ）
- 刪除 Amazon Timestream for InfluxDB 的服務連結角色
  - 清除服務連結角色
  - 刪除服務連結角色 （ IAM 主控台 ）

- 刪除服務連結角色（IAM CLI）
- 刪除服務連結角色（IAM API）
- Amazon Timestream for InfluxDB Service 連結角色的支援區域

Amazon Timestream for InfluxDB 的服務連結角色許可

Amazon Timestream for InfluxDB 使用名為 的服務連結角色 AmazonTimestreamInfluxDBServiceRolePolicy – 此政策允許 Timestream for InfluxDB 根據需要代表您管理 AWS 資源，以管理您的叢集。

AmazonTimestreamInfluxDBServiceRolePolicy 服務連結角色許可政策允許 Amazon Timestream for InfluxDB 在指定的資源上完成下列動作：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "DescribeNetworkStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:DescribeSubnets",
    "ec2:DescribeVpcs",
    "ec2:DescribeNetworkInterfaces"
   ],
   "Resource": "*"
  },
  {
   "Sid": "CreateEniInSubnetStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:CreateNetworkInterface"
   ],
   "Resource": [
    "arn:aws:ec2:*:*:subnet/*",
    "arn:aws:ec2:*:*:security-group/*"
   ]
  },
  {
   "Sid": "CreateEniStatement",
   "Effect": "Allow",
   "Action": [
```

```
    "ec2:CreateNetworkInterface"
   ],
   "Resource": "arn:aws:ec2:*:*:network-interface/*",
   "Condition": {
    "Null": {
     "aws:RequestTag/AmazonTimestreamInfluxDBManaged": "false"
    }
   }
  },
  {
   "Sid": "CreateTagWithEniStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:CreateTags"
   ],
   "Resource": "arn:aws:ec2:*:*:network-interface/*",
   "Condition": {
    "Null": {
     "aws:RequestTag/AmazonTimestreamInfluxDBManaged": "false"
    },
    "StringEquals": {
     "ec2:CreateAction": [
      "CreateNetworkInterface"
     ]
    }
   }
  },
  {
   "Sid": "ManageEniStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:CreateNetworkInterfacePermission",
    "ec2:DeleteNetworkInterface"
   ],
   "Resource": "arn:aws:ec2:*:*:network-interface/*",
   "Condition": {
    "Null": {
     "aws:ResourceTag/AmazonTimestreamInfluxDBManaged": "false"
    }
   }
  },
  {
   "Sid": "PutCloudWatchMetricsStatement",
   "Effect": "Allow",
```

```
    "Action": [
     "cloudwatch:PutMetricData"
    ],
    "Condition": {
     "StringEquals": {
      "cloudwatch:namespace": [
       "AWS/Timestream/InfluxDB",
       "AWS/Usage"
      ]
     }
    },
    "Resource": [
     "*"
    ]
   },
   {
    "Sid": "ManageSecretStatement",
    "Effect": "Allow",
    "Action": [
     "secretsmanager:CreateSecret",
     "secretsmanager:DeleteSecret"
    ],
    "Resource": [
     "arn:aws:secretsmanager:*:*:secret:READONLY-InfluxDB-auth-parameters-*"
    ],
    "Condition": {
     "StringEquals": {
      "aws:ResourceAccount": "${aws:PrincipalAccount}"
     }
    }
   }
  ]
}
```

允許IAM實體建立 AmazonTimestreamInfluxDBServiceRolePolicy服務連結角色

將下列政策陳述式新增至該IAM實體的許可：

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole",
        "iam:PutRolePolicy"
    ],
```

```
    "Resource": "arn:aws:iam::*:role/aws-service-role/
timestreamforinfluxdb.amazonaws.com/AmazonTimestreamInfluxDBServiceRolePolicy*",
    "Condition": {"StringLike": {"iam:AWS ServiceName":
 "timestreamforinfluxdb.amazonaws.com"}}
}
```

允許IAM實體刪除 AmazonTimestreamInfluxDBServiceRolePolicy服務連結角色

將下列政策陳述式新增至該IAM實體的許可：

```
{
    "Effect": "Allow",
    "Action": [
        "iam:DeleteServiceLinkedRole",
        "iam:GetServiceLinkedRoleDeletionStatus"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/
timestreamforinfluxdb.amazonaws.com/AmazonTimestreamInfluxDBServiceRolePolicy*",
    "Condition": {"StringLike": {"iam:AWS ServiceName":
 "timestreamforinfluxdb.amazonaws.com"}}
}
```

或者，您可以使用 AWS 受管政策來提供 Amazon Timestream for InfluxDB 的完整存取權。

建立服務連結角色（IAM）

您不需要手動建立一個服務連結角色。當您建立資料庫執行個體時，Amazon Timestream for InfluxDB 會為您建立服務連結角色。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。當您建立資料庫執行個體時，Amazon Timestream for InfluxDB 會再次為您建立服務連結角色。

編輯 Amazon Timestream for InfluxDB 的服務連結角色描述

Amazon Timestream for InfluxDB 不允許您編輯 AmazonTimestreamInfluxDBServiceRolePolicy服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。不過，您可以使用 編輯角色的描述IAM。

編輯服務連結角色描述（IAM 主控台）

您可以使用IAM主控台編輯服務連結角色描述。

編輯服務連結角色的說明 (主控台)

1. 在IAM主控台的左側導覽窗格中，選擇角色 。
2. 選擇要修改之角色的名稱。
3. 在 Role description (角色說明) 的最右邊，選擇 Edit (編輯)。
4. 在方塊中輸入新的描述，然後選擇 Save (儲存)。

編輯服務連結角色描述 （ IAM CLI ）

您可以使用 中的 IAM 操作 AWS Command Line Interface 來編輯服務連結角色描述。

若要變更服務連結角色的描述 （ CLI ）

1. （選用 ）若要檢視角色的目前描述，請將 AWS CLI 用於IAM操作 get-role。

   Example

   ```
   $ aws iam get-role --role-name AmazonTimestreamInfluxDBServiceRolePolicy
   ```

   使用角色名稱，而非 ARN，以參考具有 CLI操作的角色。例如，如果角色具有下列 ARN：`arn:aws:iam::123456789012:role/myrole`，請將角色稱為 **myrole**。
2. 若要更新服務連結角色的描述，請將 AWS CLI 用於IAM操作 update-role-description。

   Linux 和 MacOS

   ```
   $ aws iam update-role-description \
       --role-name AmazonTimestreamInfluxDBServiceRolePolicy \
       --description "new description"
   ```

   Windows

   ```
   $ aws iam update-role-description ^
       --role-name AmazonTimestreamInfluxDBServiceRolePolicy ^
       --description "new description"
   ```

編輯服務連結角色描述 （ IAM API ）

您可以使用 IAM API 編輯服務連結角色描述。

若要變更服務連結角色的描述（API）

1. （選用）若要檢視角色的目前描述，請使用 IAMAPI操作 GetRole.

   Example

   ```
   https://iam.amazonaws.com/
      ?Action=GetRole
      &RoleName=AmazonTimestreamInfluxDBServiceRolePolicy
      &Version=2010-05-08
      &AUTHPARAMS
   ```

2. 若要更新角色的描述，請使用 IAMAPI操作 UpdateRoleDescription.

   Example

   ```
   https://iam.amazonaws.com/
      ?Action=UpdateRoleDescription
      &RoleName=AmazonTimestreamInfluxDBServiceRolePolicy
      &Version=2010-05-08
      &Description="New description"
   ```

刪除 Amazon Timestream for InfluxDB 的服務連結角色

若您不再使用需要服務連結角色的功能或服務，我們建議您刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。然而，務必清除您的服務連結角色，之後才能將其刪除。

Amazon Timestream for InfluxDB 不會刪除您的服務連結角色。

清除服務連結角色

在您使用 IAM 刪除服務連結角色之前，請先確認角色沒有與其相關聯的資源（叢集）。

檢查服務連結角色在IAM主控台中是否具有作用中工作階段

1. 登入 AWS Management Console 並在 開啟IAM主控台https://console.aws.amazon.com/iam/。
2. 在IAM主控台的左側導覽窗格中，選擇角色 。然後選擇角色的名稱 AmazonTimestreamInfluxDBServiceRolePolicy（而非核取方塊）。
3. 在所選角色的 Summary (摘要) 頁面中，選擇 Access Advisor (存取 Advisor) 分頁。
4. 在 Access Advisor (存取 Advisor) 分頁中，檢閱服務連結角色的近期活動。

刪除服務連結角色 （ IAM 主控台 ）

您可以使用 IAM 主控台來刪除服務連結角色。

刪除服務連結角色 (主控台)

1. 登入 AWS Management Console 並在 開啟IAM主控台https://console.aws.amazon.com/iam/。
2. 在IAM主控台的左側導覽窗格中，選擇角色 。然後，選擇您要刪除的角色名稱旁的核取方塊，而非名稱或資料列本身。
3. 在頁面頂端的 Role (角色) 動作中選擇 Delete (刪除) 角色。
4. 在確認頁面中，檢閱服務上次存取的資料，顯示每個所選角色上次存取 AWS 服務的時間。這可協助您確認角色目前是否作用中。如果您想要繼續進行，請選擇 Yes, Delete (是，刪除) 來提交服務連結角色以進行刪除。
5. 觀看IAM主控台通知，以監控服務連結角色刪除的進度。由於IAM服務連結角色刪除是非同步的，因此在您提交角色以供刪除之後，刪除任務可能會成功或失敗。如果任務失敗，您可以從通知中選擇 View details (檢視詳細資訊) 或 View Resources (檢視資源)，以了解刪除失敗的原因。

刪除服務連結角色 （ IAM CLI ）

您可以從 使用 IAM 操作 AWS Command Line Interface 來刪除服務連結角色。

若要刪除服務連結角色 （ CLI ）

1. 如果您不知道想要刪除的服務連結角色名稱，請輸入以下命令。此命令會列出您帳戶中的角色及其 Amazon Resource Names （ ARNs ）。

   ```
   $ aws iam get-role --role-name role-name
   ```

   使用角色名稱，而非 ARN，以參考具有 CLI操作的角色。例如，如果角色具有 ARN `arn:aws:iam::123456789012:role/myrole`，您會將角色稱為 **myrole**。
2. 因為無法刪除正在使用或具有相關聯資源的服務連結角色，所以您必須提交刪除要求。如果不符合這些條件，則可以拒絕該請求。您必須從回應中擷取 `deletion-task-id`，以檢查刪除任務的狀態。輸入下列內容，提交服務連結角色刪除請求。

   ```
   $ aws iam delete-service-linked-role --role-name role-name
   ```
3. 輸入下列內容來檢查刪除任務的狀態。

```
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

刪除任務的狀態可以是 NOT_STARTED、IN_PROGRESS、SUCCEEDED 或 FAILED。如果刪除失敗，則呼叫會傳回失敗原因，以進行疑難排解。

刪除服務連結角色（IAM API）

您可以使用 IAMAPI刪除服務連結角色。

刪除服務連結角色（API）

1. 若要提交服務連結卷的刪除請求，請呼叫 DeleteServiceLinkedRole。 在請求中，指定角色名稱。

   因為無法刪除正在使用或具有相關聯資源的服務連結角色，所以您必須提交刪除要求。如果不符合這些條件，則可以拒絕該請求。您必須從回應中擷取 DeletionTaskId，以檢查刪除任務的狀態。
2. 若要檢查刪除的狀態，請呼叫 GetServiceLinkedRoleDeletionStatus。 在請求中，指定 DeletionTaskId。

   刪除任務的狀態可以是 NOT_STARTED、IN_PROGRESS、SUCCEEDED 或 FAILED。如果刪除失敗，則呼叫會傳回失敗原因，以進行疑難排解。

Amazon Timestream for InfluxDB Service 連結角色的支援區域

Amazon Timestream for InfluxDB 支援在提供服務的所有 區域中使用服務連結角色。如需詳細資訊，請參閱 AWS 服務端點。

AWS Amazon Timestream for InfluxDB 的 受管政策

若要將許可新增至使用者、群組和角色，使用 AWS 受管政策比自行撰寫政策更容易。建立IAM客戶受管政策需要時間和專業知識，為您的團隊提供他們所需的許可。若要快速開始使用，您可以使用我們的 AWS 受管政策。這些政策涵蓋常見的使用案例，可在您的帳戶中使用 AWS 。如需 AWS 受管政策的詳細資訊，請參閱 IAM 使用者指南 中的AWS 受管政策。

AWS 服務會維護和更新 AWS 受管政策。您無法變更 AWS 受管政策中的許可。服務偶爾會在 AWS 受管政策中新增其他許可以支援新功能。此類型的更新會影響已連接政策的所有身分識別 (使用者、群

組和角色)。當新功能啟動或新操作可用時，服務很可能會更新 AWS 受管政策。服務不會從 AWS 受管政策中移除許可，因此政策更新不會破壞您現有的許可。

此外， AWS 支援跨多個 服務的任務函數的受管政策。例如， ReadOnlyAccess AWS 受管政策提供所有 AWS 服務和資源的唯讀存取權。服務啟動新功能時， 會為新操作和資源 AWS 新增唯讀許可。如需任務函數政策的清單和描述，請參閱 IAM 使用者指南 中的AWS 任務函數的受管政策。

AWS 受管政策： AmazonTimestreamInfluxDBServiceRolePolicy

您無法將 AmazonTimestreamInfluxDBServiceRolePolicy AWS 受管政策連接至帳戶中的身分。此政策是 AWS TimestreamforInflux資料庫服務連結角色的一部分。此角色允許 服務管理您帳戶中的網路介面和安全群組。

InfluxDB 的 Timestream 使用此政策中的許可來管理EC2安全群組和網路介面。這是管理 InfluxDB 資料庫執行個體的 Timestream 所需。

若要以 JSON 格式檢閱此政策，請參閱 AmazonTimestreamInfluxDBServiceRolePolicy。

AWS Amazon Timestream for InfluxDB 的 受管政策

AWS 提供由 建立和管理的獨立IAM政策，以解決許多常見的使用案例 AWS。受管政策授與常見使用案例中必要的許可，讓您免於查詢需要哪些許可。如需詳細資訊，請參閱 IAM 使用者指南 中的 AWS 受管政策。

下列 AWS 受管政策可連接至您帳戶中的使用者，其專屬於 Timestream for InfluxDB ：

AmazonTimestreamInfluxDBFullAccess

您可以將`AmazonTimestreamInfluxDBFullAccess`政策連接至身分IAM。此政策會授予管理許可，允許完全存取所有 Timestream for InfluxDB 資源。

您也可以建立自己的自訂IAM政策，以允許 Amazon Timestream for InfluxDB API動作的許可。您可以將這些自訂政策連接至需要這些許可IAM的使用者或群組。

若要以 JSON 格式檢閱此政策，請參閱 AmazonTimestreamInfluxDBFullAccess。

InfluxDB 更新受 AWS 管政策的時間串流

檢視自此服務開始追蹤這些變更以來，Timestream for InfluxDB 受 AWS 管政策更新的詳細資訊。如需此頁面變更的自動提醒，請訂閱 Timestream for InfluxDB 文件歷史記錄頁面上的RSS摘要。

變更	描述	日期
AmazonTimestreamInfluxDBFullAccess – 更新現有政策	已將 `ec2:DescribeRouteTables` 動作新增至現有的 `AmazonTimestreamInfluxDBFullAccess` 受管政策。此動作用於描述您的路由表	10/08/2024
AWS 受管政策： AmazonTimestreamInfluxDBServiceRolePolicy – 新政策	Amazon Timestream for InfluxDB 新增了一項新政策，允許服務管理您帳戶中的網路介面和安全群組。	03/14/2024
AmazonTimestreamInfluxDBFullAccess – 新政策	Amazon Timestream for InfluxDB 新增了新的政策，以提供建立、更新、刪除和列出 Amazon Timestream InfluxDB 執行個體，以及建立和列出參數群組的完整管理存取權。	03/14/2024

透過VPC端點連線至 Timestream for InfluxDB

您可以透過虛擬私有雲端（ ）中的私有介面端點，直接連線至 Timestream for InfluxDBVPC。當您使用介面VPC端點時，您的 VPC與 Timestream for InfluxDB 之間的通訊會完全在 AWS 網路中執行。

InfluxDB 的 Timestream 支援 支援的 Amazon Virtual Private Cloud（Amazon VPC）端點AWS PrivateLink。每個VPC端點都由一個或多個具有私有 IP 地址的彈性網路介面（ENIs）VPC 表示。

介面VPC端點會將您的 VPC直接連線至 Timestream for InfluxDB，無需網際網路閘道、NAT裝置、VPN連線或 AWS Direct Connect 連線。您 中的執行個體VPC不需要公有 IP 地址，即可與 Timestream for InfluxDB 通訊。

區域

InfluxDB 的 Timestream 支援所有支援 InfluxDB 的 Timestream AWS 區域 的VPC端點和VPC端點政策。

主題

- InfluxDB VPC端點的 Timestream 考量事項
- 建立 Timestream for InfluxDB 的VPC端點
- 連線至 Timestream for InfluxDB VPC端點
- 控制VPC端點的存取
- 在政策陳述式中使用VPC端點
- 記錄您的VPC端點

InfluxDB VPC端點的 Timestream 考量事項

在您為 Timestream for InfluxDB 設定介面VPC端點之前，請檢閱AWS PrivateLink 指南 中的介面端點屬性和限制主題。

VPC 端點的 InfluxDB 支援時間串流包括以下內容。

- 您可以使用VPC端點從 呼叫所有 Timestream for InfluxDB API操作VPC。
- 您可以使用 AWS CloudTrail 日誌，透過VPC端點稽核 Timestream for InfluxDB 資源的使用。如需詳細資訊，請參閱 記錄您的VPC端點。

建立 Timestream for InfluxDB 的VPC端點

您可以使用 Amazon VPC主控台或 Amazon 為 Timestream for InfluxDB 建立VPC端點VPCAPI。如需詳細資訊，請參閱《AWS PrivateLink 指南》中的建立介面端點。

- 若要為 Timestream for InfluxDB 建立VPC端點，請使用下列服務名稱：

  ```
  com.amazonaws.region.timestream-influxdb
  ```

  例如，在美國西部 (奧勒岡) 區域 (`us-west-2`)，服務名稱為：

  ```
  com.amazonaws.us-west-2.timestream-influxdb
  ```

若要更輕鬆地使用VPC端點，您可以為VPC端點啟用私有DNS名稱。如果您選取啟用DNS名稱選項，InfluxDB DNS 主機名稱的標準時間串流會解析為您的VPC端點。例如， `https://timestream-influxdb.us-west-2.amazonaws.com`會解析為連線至服務名稱 的VPC端點`com.amazonaws.us-west-2.timestream-influxdb`。

此選項可讓您更輕鬆地使用VPC端點。根據 AWS SDKs預設， 和 AWS CLI 會使用 InfluxDB DNS主機名稱的標準 Timestream，因此您不需要在應用程式和命令URL中指定VPC端點。

如需詳細資訊，請參閱《AWS PrivateLink 指南》中的透過介面端點存取服務。

連線至 Timestream for InfluxDB VPC端點

您可以使用 AWS SDK、 AWS CLI 或 ，透過VPC端點連線至 Timestream for InfluxDB AWS Tools for PowerShell。若要指定VPC端點，請使用其DNS名稱。

如果您在建立VPC端點時啟用了私有主機名稱，則不需要在CLI命令或應用程式組態URL中指定VPC端點。InfluxDB DNS主機名稱的標準 Timestream 會解析為您的VPC端點。根據預設， AWS CLI 和 SDKs使用此主機名稱，因此您可以開始使用 VPC 端點連線到 Timestream for InfluxDB 區域端點，而不需要變更指令碼和應用程式中的任何內容。

若要使用私有主機名稱，您的 `enableDnsHostnames`和 `enableDnsSupport` 屬性VPC必須設定為 `true`。若要設定這些屬性，請使用 ModifyVpcAttribute操作。如需詳細資訊，請參閱 Amazon VPC使用者指南 中的檢視和更新 的DNS屬性VPC。

控制VPC端點的存取

若要控制 Timestream for InfluxDB 對VPC端點的存取，請將VPC端點政策連接至您的VPC端點。端點政策會決定主體是否可以使用VPC端點在 Timestream for InfluxDB 資源上呼叫 Timestream for InfluxDB 操作。

您可以在建立VPC端點時建立端點政策，而且您可以隨時變更VPC端點政策。使用 VPC 管理主控台，或 CreateVpcEndpoint或 ModifyVpcEndpoint操作。您也可以使用 AWS CloudFormation 範本 建立和變更VPC端點政策。如需使用 VPC 管理主控台的說明，請參閱 AWS PrivateLink 指南 中的建立介面端點和修改介面端點。

> **Note**
>
> InfluxDB 的 Timestream 支援從 2020 年 7 月開始的VPC端點政策。VPC 在該日期之前建立的 Timestream for InfluxDB 端點具有預設VPC端點政策 ，但您可以隨時變更它。

主題

關於VPC端點政策

對於使用VPC端點來成功的 Timestream for InfluxDB 請求，主體需要兩個來源的許可：

- IAM 政策必須授予主體許可，才能呼叫資源上的操作。
- VPC 端點政策必須授予主體使用端點提出請求的許可。

預設VPC端點政策

每個VPC端點都有VPC端點政策，但您不需要指定政策。如果您未指定政策，則預設端點政策會允許端點上所有資源的所有委託人進行所有操作。

不過，對於 Timestream for InfluxDB 資源，委託人也必須具有從IAM政策呼叫操作的許可。因此，實際上，預設政策會指出，如果委託人有許可來呼叫資源上的操作，也可以使用端點呼叫它。

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Principal": "*",
      "Resource": "*"
    }
  ]
}
```

若要允許主體僅將VPC端點用於其允許操作的子集，請建立或更新VPC端點政策 。

建立VPC端點政策

VPC 端點政策會判斷主體是否具有使用VPC端點對資源執行操作的許可。對於 Timestream for InfluxDB 資源，委託人也必須具有從IAM政策 執行操作的許可，

每個VPC端點政策陳述式都需要下列元素：

- 可執行動作的委託人
- 可執行的動作
- 可在其中執行動作的資源

政策陳述式不會指定VPC端點。相反地，它適用於附加政策的任何VPC端點。如需詳細資訊，請參閱 Amazon VPC使用者指南 中的使用VPC端點控制對 服務的存取。

AWS CloudTrail 會記錄使用VPC端點的所有操作。

檢視VPC端點政策

若要檢視VPC端點的端點政策，請使用 VPC 管理主控台或 DescribeVpcEndpoints操作。

下列 AWS CLI 命令會取得具有指定端點 ID 的VPC端點政策。

使用此命令之前，請將範例端點 ID 取代為您帳戶的有效 ID。

```
$ aws ec2 describe-vpc-endpoints \
--query 'VpcEndpoints[?VpcEndpointId==`vpc-endpoint-id`].[PolicyDocument]'
--output text
```

在政策陳述式中使用VPC端點

當請求來自VPC或使用VPC端點時，您可以控制對 Timestream for InfluxDB 資源和操作的存取。若要這麼做，請在IAM政策 中使用下列其中一個全域條件金鑰。

- 使用aws:sourceVpce條件索引鍵，根據VPC端點授予或限制存取權。
- 使用 aws:sourceVpc 條件金鑰，根據託管私有端點的 VPC 來授予或限制存取權。

> **Note**
>
> 根據您的VPC端點建立金鑰政策和IAM政策時請小心。如果政策陳述式要求請求來自特定 VPC 或VPC端點，則代表您使用 Timestream for InfluxDB 資源的整合 AWS 服務發出的請求可能會失敗。
>
> 此外，當請求來自 Amazon VPC端點 時，aws:sourceIP條件金鑰無效。若要限制對VPC端點的請求，請使用 aws:sourceVpce或 aws:sourceVpc 條件索引鍵。如需詳細資訊，請參閱 AWS PrivateLink 指南 中的VPC端點和VPC端點服務的身分和存取管理。

您可以使用這些全域條件金鑰來控制對不依賴任何特定資源之操作CreateDbInstance的存取。

記錄您的VPC端點

AWS CloudTrail 會記錄使用VPC端點的所有操作。當對 Timestream for InfluxDB 的請求使用VPC端點時，VPC端點 ID 會出現在記錄請求的AWS CloudTrail 日誌項目中。您可以使用端點 ID 稽核 Timestream for InfluxDB VPC端點的使用。

不過，您的 CloudTrail 日誌不會包含其他帳戶中的主體所請求的操作，也不會包含在其他帳戶中的 Timestream for InfluxDB 操作，以及其他帳戶中的 Timestream for InfluxDB 資源和別名的請求。此外，為了保護您的 VPC，VPC端點政策 拒絕，但在其他情況下已允許 的請求不會記錄在 中AWS CloudTrail。

在 Timestream for InfluxDB 中記錄和監控

監控是維護 Timestream for InfluxDB 和您的 AWS 解決方案可靠性、可用性和效能的重要部分。您應該從 AWS 解決方案的所有部分收集監控資料，以便在發生多點故障時更輕鬆地進行偵錯。不過，在開始監控 Timestream for InfluxDB 之前，您應該建立監控計畫，其中包含下列問題的答案：

- 監控目標是什麼？
- 要監控哪些資源？
- 監控這些資源的頻率為何？
- 要使用哪些監控工具？
- 誰將執行監控任務？
- 發生問題時應該通知誰？

下一個步驟是建立您環境中 InfluxDB 效能正常時間串流的基準，方法是測量不同時間和不同負載條件下的效能。當您監控 InfluxDB 的 Timestream 時，請儲存歷史監控資料，以便將其與目前的效能資料進行比較、識別正常效能模式和效能異常，並設計方法來解決問題。

若要建立基準，您至少必須監控下列項目：

- 系統錯誤，以便您可以判斷是否有任何請求導致錯誤。

主題

- 監控工具

- 使用 記錄 InfluxDB API呼叫的時間串流 AWS CloudTrail

監控工具

AWS 提供各種工具，您可以用來監控 InfluxDB 的 Timestream。您可以設定其中一些工具來進行監控，但有些工具需要手動介入。建議您盡可能自動化監控任務。

主題

- 自動化監控工具
- 手動監控工具

自動化監控工具

您可以使用下列自動監控工具來觀看 InfluxDB 的 Timestream，並在發生問題時報告：

- Amazon CloudWatch Alarms – 在您指定的時段內觀察單一指標，並根據指標的值在多個時段內相對於指定閾值執行一或多個動作。動作是傳送至 Amazon Simple Notification Service （Amazon SNS） 主題或 Amazon EC2 Auto Scaling Policy. CloudWatch alarms 的通知，不會單純因為動作處於特定狀態而叫用動作；狀態必須已變更並維持在指定的期間數。如需詳細資訊，請參閱使用 Amazon 監控 CloudWatch。

手動監控工具

監控 Timestream for InfluxDB 的另一個重要部分是手動監控 CloudWatch 警示未涵蓋的項目。InfluxDB CloudWatch Trusted Advisor、 和其他 AWS Management Console 儀表板的 Timestream at-a-glance提供 AWS 環境狀態的檢視。

- CloudWatch 首頁顯示以下內容：
  - 目前警示與狀態
  - 警示與資源的圖表
  - 服務運作狀態

  此外，您可以使用 CloudWatch 執行下列動作：
  - 建立自定儀表板來監控您注重的服務
  - 用於疑難排解問題以及探索驅勢的圖形指標資料。
  - 搜尋和瀏覽您的所有 AWS 資源指標

- 建立與編輯要通知發生問題的警示

使用 記錄 InfluxDB API呼叫的時間串流 AWS CloudTrail

Timestream for InfluxDB 與 整合 AWS CloudTrail，此服務提供 Timestream for InfluxDB AWS . CloudTrail captures Data Definition Language （ DDL ） API呼叫 Timestream for InfluxDB 作為事件的使用者、角色或服務所採取動作的記錄。擷取的呼叫包括從 Timestream for InfluxDB 主控台的呼叫，以及對 Timestream for InfluxDB API操作的程式碼呼叫。如果您建立追蹤，則可以啟用事件持續交付 CloudTrail至 Amazon Simple Storage Service （ Amazon S3 ） 儲存貯體，包括 Timestream for InfluxDB 的事件。如果您未設定追蹤，仍然可以在 事件歷史記錄 中檢視 CloudTrail 主控台上的最新事件。 使用 所收集的資訊 CloudTrail，您可以判斷向 Timestream for InfluxDB 提出的請求、提出請求的 IP 地址、提出請求的人員、提出時間，以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱 AWS CloudTrail 使用者指南 。

InfluxDB 資訊的時間串流 CloudTrail

CloudTrail 會在您建立 AWS 帳戶時於您的帳戶啟用 。當活動在 Timestream for InfluxDB 中發生時，該活動會與 CloudTrail 事件歷史記錄 中的其他 AWS 服務事件一起記錄在事件中。您可以檢視、搜尋和下載 AWS 帳戶的最新事件。如需詳細資訊，請參閱檢視具有事件歷史記錄 CloudTrail 的事件。

若要持續記錄您 AWS 帳戶中的事件，包括 Timestream for InfluxDB 的事件，請建立追蹤。追蹤可讓 CloudTrail 將日誌檔案傳遞至 Amazon S3 儲存貯體。根據預設，當您在主控台中建立追蹤時，追蹤會套用至所有AWS區域。追蹤會記錄 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，您可以設定其他服務 AWS ，以進一步分析 CloudTrail 日誌中收集的事件資料並對其採取行動。

如需詳細資訊，請參閱 AWS CloudTrail 使用者指南中的以下主題：

- 建立追蹤的概觀
- CloudTrail 支援的服務和整合
- 設定 的 Amazon SNS Notifications CloudTrail
- 從多個區域接收 CloudTrail 日誌檔案
- 從多個帳戶接收 CloudTrail 日誌檔案
- 記錄資料事件

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：

- 請求是使用根還是 AWS Identity and Access Management （ IAM ） 使用者憑證提出
- 提出該請求時，是否使用了特定角色或聯合身分使用者的臨時安全憑證
- 該請求是否由其他 AWS 服務提出

如需詳細資訊，請參閱CloudTrail userIdentity元素 。

Amazon Timestream for InfluxDB 的合規驗證

第三方稽核人員會在多個合規計畫中評估 Amazon Timestream for InfluxDB 的安全性和 AWS 合規性。這些索引標籤包括以下項目：

- GDPR
- HIPAA
- PCI
- SOC

若要了解 是否 AWS 服務 在特定合規計劃的範圍內，請參閱AWS 服務 合規計劃範圍內然後選擇您感興趣的合規計劃。如需一般資訊，請參閱 AWS Compliance Programs 。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱在 下載報告 AWS Artifact。

使用 時的合規責任 AWS 服務 取決於資料的敏感度、您公司的合規目標，以及適用的法律和法規。AWS 提供下列資源以協助合規：

- 安全與合規快速入門指南 – 這些部署指南討論架構考量，並提供以 AWS 安全與合規為重心的基準環境部署步驟。
- Amazon Web Services 上HIPAA安全和合規的架構 – 本白皮書描述了公司如何使用 AWS 來建立HIPAA符合 資格的應用程式。

> **Note**
>
> 並非所有 AWS 服務 都HIPAA符合資格。如需詳細資訊，請參閱HIPAA合格服務參考 。

- AWS 合規資源 – 此工作手冊和指南集可能適用於您的產業和位置。
- AWS 客戶合規指南 – 透過合規的角度了解共同的責任模型。本指南摘要說明跨多個架構 （ 包括國家標準和技術研究所 （ ）NIST、支付卡產業安全標準委員會 （ PCI ） 和國際標準化組織 （ ISO ） ） 保護 AWS 服務 指南並映射至安全控制的最佳實務。

- AWS Config 開發人員指南中的使用規則評估資源 – AWS Config 服務會評估資源組態是否符合內部實務、產業準則和法規。
- AWS Security Hub – 這 AWS 服務 提供 內安全狀態的全面檢視 AWS。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱 Security Hub controls reference。
- Amazon GuardDuty – 透過監控您的環境是否有可疑和惡意活動，藉此 AWS 服務 偵測 AWS 帳戶、工作負載、容器和資料的潛在威脅。 GuardDuty 可以透過滿足某些合規架構強制要求的入侵偵測需求，協助您解決各種合規要求DSS，例如 PCI 。
- AWS Audit Manager – 這 AWS 服務 可協助您持續稽核 AWS 用量，以簡化您管理風險的方式，以及遵守法規和產業標準的方式。

Amazon Timestream for InfluxDB 中的彈性

AWS 全域基礎設施是以 AWS 區域和可用區域為基礎建置。 AWS 區域提供多個實體隔離和隔離的可用區域，這些區域與低延遲、高輸送量和高度備援的網路連接。透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 AWS 區域和可用區域的詳細資訊，請參閱AWS 全域基礎設施。

Amazon Timestream for InfluxDB 會定期進行內部備份，並保留 24 小時以支援可用性和耐久性。快照會在刪除期間擷取，並保留 30 天以支援還原。若要存取或使用這些資料，請透過 AWS 支援 提交票證。

您可以使用多可用區域復原功能建立執行個體。如需詳細資訊，請參閱 Multi-AZ 資料庫執行個體部署。

Amazon Timestream for InfluxDB 中的基礎設施安全

作為受管服務，Amazon Timestream for InfluxDB 受到 Amazon Web Services：安全程序概觀白皮書中所述 AWS 的全球網路安全程序保護。

您可以使用 AWS 發佈的控制平面API呼叫，透過網路存取 Timestream for InfluxDB。如需詳細資訊，請參閱控制平面和資料平面。用戶端必須支援 Transport Layer Security （ TLS ） 1.2 或更新版本。我們建議TLS使用 1.2 或 1.3。用戶端還必須支援具有完美正向保密性 （ PFS ） 的密碼套件，例如 Ephemeral Diffie-Hellman （ DHE ） 或 Elliptic Curve Ephemeral Diffie-Hellman （ ECDHE ）。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

此外，必須使用與IAM委託人相關聯的存取金鑰 ID 和秘密存取金鑰來簽署請求。或者，您可以透過 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

InfluxDB 的 Timestream 經過架構，因此您的流量會隔離到 Timestream for InfluxDB 執行個體所在的特定 AWS 區域。

安全群組

安全群組會控制進出資料庫執行個體流量的存取權限。資料庫執行個體的網路存取預設是關閉的。您可以在允許從 IP 地址範圍、連接埠或安全群組存取的安全群組中指定規則。設定傳入規則後，相同規則就會套用到與該安全群組相關聯的所有資料庫執行個體。

如需詳細資訊，請參閱在 中控制資料庫執行個體的存取 VPC。

Timestream for InfluxDB 中的組態和漏洞分析

組態和 IT 控制是 AWS 與身為我們客戶的您共同的責任。如需詳細資訊，請參閱 AWS 共同責任模型。除了共同責任模型之外，InfluxDB 使用者的 Timestream 應注意下列事項：

- 客戶有責任修補他們的用戶端應用程式與相關的用戶端相依性。
- 客戶應考慮在適當情況下進行滲透測試（請參閱https://aws.amazon.com/security/滲透測試/。）

Timestream for InfluxDB 中的事件回應

Amazon Timestream for InfluxDB 服務事件會在 Personal Health Dashboard 中報告。您可以在這裡進一步了解儀表板和 AWS Health https://docs.aws.amazon.com/health/latest/ug/what-is-aws-health.html。

InfluxDB 的 Timestream 支援使用 報告 AWS CloudTrail。如需詳細資訊，請參閱使用 記錄 InfluxDB API呼叫的時間串流 AWS CloudTrail。

InfluxDB API和介面VPC端點的 Amazon Timestream （ AWS PrivateLink ）

您可以透過建立介面API端點 ，在 VPC和 Amazon Amazon Timestream for InfluxDB 控制平面端點之間建立私有連線。 VPC 介面端點由 提供支援AWS PrivateLink。 AWS PrivateLink 可讓您在沒有網際網路閘道、NAT裝置、VPN連線或 AWS Direct Connect 連線的情況下，私下存取 Amazon Timestream for InfluxDB API操作。

您 中的執行個體VPC不需要公有 IP 地址即可與 Amazon Timestream for InfluxDB API端點通訊。您的執行個體也不需要公有 IP 地址，即可使用任何可用的 Timestream for InfluxDB API操作。您的 VPC

與 Amazon Timestream for InfluxDB 之間的流量不會離開 Amazon 網路。每個介面端點都由子網路中的一個或多個彈性網路介面表示。如需彈性網路介面的詳細資訊，請參閱 Amazon EC2使用者指南 中的彈性網路介面。

- 如需VPC端點的詳細資訊，請參閱 Amazon VPC使用者指南 中的介面VPC端點（AWS PrivateLink）。
- 如需 InfluxDB API操作的 Timestream 詳細資訊，請參閱 InfluxDB API操作的 Timestream。

建立介面VPC端點之後，如果您為端點啟用私有DNS主機名稱，則 InfluxDB 端點的預設 Timestream（https://timestream-influxb.：//*Region*.amazonaws.com）會解析至您的VPC端點。如果您未啟用私有DNS主機名稱，Amazon VPC會提供DNS端點名稱，您可以使用下列格式：

```
VPC_Endpoint_ID.timestream-influxb.Region.vpce.amazonaws.com
```

如需詳細資訊，請參閱 Amazon VPC使用者指南 中的介面VPC端點（AWS PrivateLink）。InfluxDB 的 Timestream 支援呼叫 內的所有API動作VPC。

> **Note**
>
> 只能為 中的一個VPC端點啟用私有DNS主機名稱VPC。如果您想要建立其他VPC端點，則應停用私有DNS主機名稱。

VPC 端點的考量事項

在您為 Amazon Timestream for InfluxDB VPC端點設定介面API端點之前，請務必檢閱 Amazon VPC使用者指南 中的介面端點屬性和限制。您可以使用 來管理 Amazon Timestream for InfluxDB 資源的所有 InfluxDB VPC API操作 AWS PrivateLink。VPC Timestream for InfluxDB 端點支援API端點政策。根據預設，允許透過端點完整存取 Timestream for InfluxDB API操作。如需詳細資訊，請參閱 Amazon VPC使用者指南 中的使用VPC端點控制對 服務的存取。

為 Timestream for InfluxDB 建立介面VPC端點 API

您可以使用 Amazon VPC主控台API或 為 Amazon Timestream for InfluxDB 建立VPC端點 AWS CLI。如需詳細資訊，請參閱 Amazon VPC使用者指南 中的建立介面端點。

建立介面VPC端點之後，您可以啟用端點的私有DNS主機名稱。執行此操作時，預設的 Amazon Timestream for InfluxDB 端點（https://timestream-influxb.*Region*.amazonaws.com）會解析至您的VPC端點。如需詳細資訊，請參閱 Amazon VPC使用者指南 中的透過介面端點存取服務。

為 Amazon Timestream for InfluxDB 建立VPC端點政策 API

您可以將端點政策連接至VPC端點，以控制對 Timestream for InfluxDB 的存取API。此政策會指定以下項目：

- 可執行動作的主體。
- 可執行的動作。
- 可供執行動作的資源。

如需詳細資訊，請參閱 Amazon VPC使用者指南 中的使用VPC端點控制對 服務的存取。

Example VPC Timestream for InfluxDB API動作的端點政策

以下是 Timestream for InfluxDB 的端點政策範例API。連接至端點時，此政策會授予所有資源上所有主體對列出的 Timestream for InfluxDB API動作的存取權。

```
{
 "Statement": [{
  "Principal": "*",
  "Effect": "Allow",
  "Action": [
   "timestream-influxb:CreateDbInstance",
   "timestream-influxb:UpdateDbInstance"
  ],
  "Resource": "*"
 }]
}
```

Example VPC 拒絕來自指定 AWS 帳戶的所有存取的端點政策

下列VPC端點政策拒絕 AWS 帳戶 *123456789012* 使用端點對資源的所有存取。此政策允許來自其他帳戶的所有動作。

```
{
 "Statement": [{
   "Action": "*",
   "Effect": "Allow",
   "Resource": "*",
   "Principal": "*"
  },
```

```
  {
   "Action": "*",
   "Effect": "Deny",
   "Resource": "*",
   "Principal": {
    "AWS": [
     "123456789012"
    ]
   }
  }
 ]
}
```

Timestream for InfluxDB 的安全最佳實務

Amazon Timestream for InfluxDB 提供許多安全功能，供您在開發和實作自己的安全政策時考慮。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，您可以決定誰取得適用於 InfluxDB 資源的 Timestream 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具有有效的憑證，才能存取 Timestream for InfluxDB 資料庫執行個體。您不應將 AWS 憑證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。

相反地，您應該使用 IAM角色來管理生產者和用戶端應用程式的臨時憑證，以存取 Timestream for InfluxDB 資料庫執行個體。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 中的下列主題：

- IAM 角色
- 常見的角色方案：使用者、應用程式和服務

使用 AWS Identity and Access Management （ IAM ） 帳戶來控制對 Amazon Timestream for InfluxDB API操作的存取，特別是建立、修改或刪除 Amazon Timestream for InfluxDB 資源的操作。這些資源包括資料庫執行個體、安全群組和參數群組。

- 為管理 Amazon Timestream for InfluxDB 資源的每個人建立個別使用者，包括您自己。請勿使用 AWS 根憑證來管理 Amazon Timestream for InfluxDB 資源。
- 授予每個使用者執行其職責所需最低程度的許可。
- 使用IAM群組有效管理多個使用者的許可。
- 定期輪換您的 IAM 登入資料。
- 設定 AWS Secrets Manager 自動輪換 Amazon Timestream for InfluxDB 的秘密。如需詳細資訊，請參閱 AWS Secrets Manager 使用者指南中的輪換 AWS Secrets Manager 秘密。您也可以從 AWS Secrets Manager 以程式設計方式擷取憑證。如需詳細資訊，請參閱 AWS Secrets Manager 使用者指南中的擷取秘密值。
- 使用 來保護您的 Timestream for InfluxDB Influx API權杖API 權杖。

在相依資源實作伺服器端加密

靜態資料和傳輸中的資料可以在 Timestream for InfluxDB 中加密。如需詳細資訊，請參閱傳輸中加密。

CloudTrail 使用 監控API通話

InfluxDB 的 Timestream 與 整合 AWS CloudTrail，此服務提供使用者、角色或在 Timestream for InfluxDB 中 AWS 服務所採取動作的記錄。

使用 收集的資訊 CloudTrail，您可以判斷向 Timestream for InfluxDB 提出的請求、提出請求的 IP 地址、提出請求的人員、提出請求的時間，以及其他詳細資訊。

如需詳細資訊，請參閱the section called “使用 記錄通話的時間串流 LiveAnalytics API AWS CloudTrail”。

Amazon Timestream for InfluxDB 支援控制平面 CloudTrail 事件，但不支援資料平面。如需詳細資訊，請參閱控制平面和資料平面。

Public accessibility (公開存取性)

當您根據 Amazon VPC服務在虛擬私有雲端 （ VPC ） 中啟動資料庫執行個體時，您可以開啟或關閉該資料庫執行個體的公有可存取性。若要指定您建立的資料庫執行個體是否具有解析為公有 IP 地址DNS 的名稱，請使用公有可存取性參數。使用此參數，您可以指定是否有資料庫執行個體的公有存取權

如果您的資料庫執行個體位於 中，VPC但無法公開存取，您也可以使用 AWS Site-to-Site VPN連線或 AWS Direct Connect 連線從私有網路存取。

如果您的資料庫執行個體可公開存取，請務必採取步驟來防止或協助減少服務相關威脅的拒絕。如需詳細資訊，請參閱拒絕服務攻擊簡介和保護網路 。

API 參考

如需 Amazon Timestream for InfluxDB 的完整清單和詳細資訊APIs，請參閱 Amazon Timestream for InfluxDB APIs。

如需所有 AWS 服務常見的錯誤碼，請參閱AWS 支援區段 。

文件歷史記錄

變更	描述	日期
僅文件更新	更新配額主題以隔離預設配額和系統限制。	2024 年 10 月 22 日
Amazon Timestream 現在支援查詢洞察	Timestream 現在包含對查詢洞察功能的支援，可協助您最佳化查詢、改善其效能並降低成本。	2024 年 10 月 22 日
Amazon Timestream for InfluxDB 更新至現有政策。	Amazon Timestream for InfluxDB 已將 `ec2:DescribeRouteTables` 動作新增至現有`AmazonTimestreamInfluxDBFullAccess` 受管政策，以描述您的路由表	2024 年 10 月 8 日
`AmazonTimestreamReadOnlyAccess` – 更新現有政策	的 Timestream LiveAnalytics 已將`DescribeAccountSettings` 許可新增至 `AmazonTimestreamRe`	2024 年 6 月 3 日

	adOnlyAccess 受管政策，以描述 AWS 帳戶 設定。	
的 Amazon Timestream LiveAnalytics 現在支援 Timestream Compute Units （TCUs）	適用於 的 Amazon Timestream LiveAnalytics 現在包含對 Timestream Compute Units （TCUs） 的支援，以測量為您的查詢需求配置的運算容量。	2024 年 4 月 29 日
已新增政策	Amazon Timestream for InfluxDB 新增了兩個新政策：一個政策，可讓服務管理您帳戶中的網路介面和安全群組。如需詳細資訊，請參閱 AmazonTimestreamInfluxDBServiceRolePolicy。提供建立、更新、刪除和列出 Amazon Timestream InfluxDB 執行個體，以及建立和列出參數群組的完整管理存取權的另一個。如需詳細資訊，請參閱 AmazonTimestreamInfluxDBFullAccess。	2024 年 3 月 14 日
Amazon Timestream for InfluxDB 現在已全面推出。	本文件涵蓋 Amazon Timestream for InfluxDB 的初始版本。	2024 年 3 月 14 日

Amazon Timestream for LiveAnalytics Query 事件可在中使用 AWS CloudTrail	的 Amazon Timestream LiveAnalytics 現在會將查詢API資料事件發佈至 AWS CloudTrail。客戶可以稽核其 AWS 帳戶中提出的所有查詢API請求，並查看相關資訊，例如IAM使用者/角色提出請求的對象、提出請求的時間、查詢的資料庫和資料表，以及請求的查詢 ID。	2023 年 9 月 12 日
的 Amazon Timestream LiveAnalytics UNLOAD	適用於 的 Amazon Timestream LiveAnalytics 現在支援UNLOAD將查詢結果匯出至S3。	2023 年 5 月 12 日
Amazon Timestream 用於 LiveAnalytics 更新現有政策。	批次載入許可已新增至受管政策。	2023 年 2 月 24 日
適用於 LiveAnalytics 批次載入的 Amazon Timestream。	適用於 的 Amazon Timestream LiveAnalytics 現在支援批次載入功能。	2023 年 2 月 24 日
適用於 的 Amazon Timestream LiveAnalytics 現在支援 AWS Backup。	適用於 的 Amazon Timestream LiveAnalytics 現在支援 AWS Backup。	2022 年 12 月 14 日
Amazon Timestream 提供受 AWS 管政策的 LiveAnalytics 更新	受 AWS 管政策和 Amazon Timestream 的新資訊 LiveAnalytics，包括現有受管政策的更新。	2021 年 11 月 29 日
的 Amazon Timestream LiveAnalytics 支援排程查詢	適用於 的 Amazon Timestream LiveAnalytics 現在支援根據排程代表您執行查詢。	2021 年 11 月 29 日

的 Amazon Timestream LiveAnalytics 支援磁性存放區。	適用於 的 Amazon Timestream LiveAnalytics 現在支援使用磁性儲存進行資料表寫入。	2021 年 11 月 29 日
LiveAnalytics 適用於多測量記錄的 Amazon Timestream。	適用於 的 Amazon Timestream LiveAnalytics 現在支援更精簡的格式來儲存您的時間序列資料。	2021 年 11 月 29 日
Amazon Timestream 可 LiveAnalytics 更新AWS受管政策	受 AWS 管政策和 Amazon Timestream 的新資訊 LiveAnalytics，包括現有受管政策的更新。	2021 年 5 月 24 日
適用於 的 Amazon Timestream LiveAnalytics 現在可在歐洲（法蘭克福）區域使用。	適用於 的 Amazon Timestream LiveAnalytics 現在已全面在歐洲（法蘭克福）區域（ ）推出`eu-central-1` 。	2021 年 4 月 23 日
的 Amazon Timestream LiveAnalytics 現在支援VPC端點（AWS PrivateLink）。	適用於 的 Amazon Timestream LiveAnalytics 現在支援使用VPC端點（AWS PrivateLink）。	2021 年 3 月 23 日
Amazon Timestream 現在支援跨資料表查詢。	您可以使用 的 Amazon Timestream LiveAnalytics 來執行跨資料表查詢。	2021 年 2 月 10 日
適用於 的 Amazon Timestream LiveAnalytics 現在支援增強型查詢執行統計資料。	適用於 的 Amazon Timestream LiveAnalytics 現在支援增強型查詢執行統計資料，例如掃描的資料量。	2021 年 2 月 10 日
適用於 的 Amazon Timestream LiveAnalytics 現在支援進階時間序列函數。	您可以使用 的 Amazon Timestream LiveAnalytics 來執行具有進階時間序列函數的SQL查詢，例如導數、積分和關聯。	2021 年 2 月 10 日

的 Amazon Timestream 現在 LiveAnalytics 已PCI符合 HIPAA、 ISO和 的要求。	您現在可以將 Amazon Timestream LiveAnalytics 用於需要 HIPAA、 ISO和 PCI合規基礎設施的工作負載。	2021 年 1 月 27 日
適用於 的 Amazon Timestream LiveAnalytics 現在支援開放原始碼 Telegraf 和 grafana。	您現在可以使用開放原始碼外掛程式驅動的伺服器代理程式 Telegraf 來收集和報告指標，以及使用資料庫的開放原始碼分析和監控平台 Grafana 搭配適用於 的 Amazon Timestream LiveAnalytics。	2020 年 11 月 25 日
適用於 的 Amazon Timestream LiveAnalytics 現在已全面推出。	本文件涵蓋 Amazon Timestream 的初始版本 LiveAnalytics。	2020 年 9 月 30 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。