使用者指南

AWS Glue

AWS Glue: 使用者指南

Table of Contents

什麼是 AWS Glue？

AWS Glue 是無伺服器資料整合服務，讓分析使用者可從多個來源輕鬆探索、準備、移動和整合資料。您可以將其用於分析、機器學習和應用程式開發。它還包括用於編寫、執行任務和實作業務工作流程的額外生產力和資料操作工具。

透過 AWS Glue，您可以探索並連線到 70 多種不同的資料來源，並在集中式資料目錄中管理資料。您可以直觀地建立、執行和監控擷取、轉換和載入 (ETL) 管道，以將資料載入資料湖。此外，您還可以使用 Amazon Athena，Amazon EMR 和 Amazon Redshift Spectrum 立即搜尋和查詢已編目的資料。

AWS Glue 將主要資料整合功能整合到單一服務中。其中包括資料探索、現代 ETL、清理、轉換和集中編目。這也是無伺服器服務，即無需管理基礎結構。AWS Glue 在單一服務中靈活支援 ETL、ELT 和串流等所有工作負載，支援各種工作負載和使用者類型的使用者。

此外，AWS Glue 可讓您輕鬆整合架構中的資料。它與 AWS 分析服務和 Amazon S3 資料湖整合。AWS Glue具有集成界面和工作創作工具，從開發人員到商業用戶，所有用戶都可以輕鬆使用，並為各種技術技能提供量身定制的解決方案。

AWS Glue 具有根據需求擴展的能力，能協助您專注於最大化資料價值的高價值活動。它可以根據任何資料大小進行擴展，並支持所有資料類型和結構描述變化。為了提高靈活性並最佳化成本，AWS Glue 提供內建的高可用性和 pay-as-you-go 計費功能。

如需定價資訊，請參閱 AWS Glue 定價。

AWS Glue Studio

AWS Glue Studio 是圖形介面，讓您能在 AWS Glue 中輕鬆建立、執行和監控資料整合任務。您可以用視覺化方式撰寫資料轉換工作流程，並在 AWS Glue 中的 Apache Spark 型無伺服器 ETL 引擎上順暢地執行它們。

有了 AWS Glue Studio，您可以建立和管理用於收集、轉換和清理資料的任務。您可以使用 AWS Glue Studio 來疑難排解和編輯任務指令碼。

主題

- AWS Glue 功能
- 學習創新 AWS Glue
- AWS Glue 入門

AWS Glue 功能

AWS Glue 功能分為三個主要類別：

- 探索和整理資料
- 轉換、準備和清理資料以進行分析
- 建立和監控資料管道

探索和整理資料

- 統一和搜尋多個資料存放區 — 透過將所有資料編目在中，跨多個資料來源和接收器進行儲存、索引和搜尋。 AWS
- 自動探索資料：使用 AWS Glue 爬蟲程式可自動推斷結構描述資訊，並將其整合至您的 AWS Glue Data Catalog。
- 管理結構描述和權限：驗證和控制對資料庫和資料表的存取。
- Connect 到各種資料來源 — 使用AWS Glue連線建立資料湖，利用內部部署和內部部署的多個資料來源。 AWS

轉換、準備和清理資料以進行分析

- 使用作業畫布介面以視覺化方式轉換資料 — 在視覺化作業編輯器中定義 ETL 程序，並自動產生程式碼以擷取、轉換和載入資料。
- 使用簡單的任務排程建立複雜的 ETL 管道：根據排程、需求或基於事件呼叫 AWS Glue 任務。
- 清理和轉換傳輸中的資料：啟用持續的資料消耗，並在傳輸過程中加以清理和轉換。這使得它在幾秒鐘內可在目標資料存放區中進行分析。
- 利用內建的機器學習來刪除重複資料並清除資料：使用 `FindMatches` 功能，無需成為機器學習專家即可清理和準備資料以進行分析。此功能會刪除重複資料，並尋找彼此不完美相符的記錄。
- 內建任務筆記本：AWS Glue 任務筆記本提供無伺服器筆記本，在 AWS Glue 中只需最少的設定，您就可以快速開始使用。
- 編輯、偵錯和測試 ETL 程式碼：您可以藉由 AWS Glue 互動式工作階段，以互動方式探索和準備資料。您可以使用自己選擇的 IDE 或筆記本，以互動方式探索、實驗和處理資料。

- 定義、偵測及修復敏感資料：AWS Glue敏感資料偵測可讓您定義、辨識和處理資料管道和資料湖中的敏感資料。

建立和監控資料管道

- 根據工作負載自動擴展：根據工作負載動態擴展和縮減資源。這只會在需要時才將工作者指派給任務。
- 使用事件型觸發器將任務自動化：啟動爬蟲程式或具有事件型觸發器的 AWS Glue 任務，並設計相依任務和爬蟲程式鏈結。
- 執行和監控任務：使用您選擇的引擎 Spark 或 Ray 來執行 AWS Glue 任務。使用自動化監控工具、AWS Glue 任務執行洞見和 AWS CloudTrail來監控任務。使用 Apache Spark UI 改善您對 Spark 支援任務的監控。
- 定義 ETL 和整合活動的工作流程：為多個爬蟲程式、任務和觸發器定義 ETL 和整合活動的工作流程。

學習創新 AWS Glue

瞭解中的最新創新， AWS Glue 並瞭解客戶如何在其組織中使用 AWS Glue 自助式資料準備工作。

瞭解客戶如何擴充 AWS Glue 超越傳統設定，以及他們如何針 AWS Glue 對工作監控和效能進行配置。

AWS Glue 入門

我們建議您從下列各節開始著手：

- 使用 AWS Glue 的概觀
- AWS Glue 概念
- 設定 AWS Glue 的 IAM 許可
- AWS Glue Data Catalog 入門
- 在 AWS Glue 中編寫任務
- AWS Glue 互動式工作階段入門
- 在 AWS Glue 中的協同運作

存取 AWS Glue

您可以使用下列任一介面來建立、檢視和管理 AWS Glue：

- AWS Glue主控台：提供 Web 介面，讓您可建立、檢視和管理 AWS Glue 任務。若要存取主控台，請參閱 AWS Glue。
- AWS Glue Studio：提供圖形介面，供您直觀地建立和編輯 AWS Glue 任務。如需詳細資訊，請參閱什麼是 AWS Glue Studio。
- AWS Glue AWS CLI 參考區段 — 提供您可以搭配使用的 AWS CLI 指令AWS Glue。如需詳細資訊，請參閱 AWS CLI 參考適用於 AWS Glue。
- AWS GlueAPI：為開發人員提供完整的 API 參考。如需詳細資訊，請參閱 AWS Glue API。

相關服務

AWS Glue 使用者也可以使用：

- AWS Lake Formation ：一種服務，是對 AWS Glue Data Catalog 中的資源提供精細存取控制的授權層。
- AWS Glue DataBrew— 視覺化資料準備工具，您可以使用它來清理和規範化資料，而無需編寫任何程式碼。

AWS Glue：運作方式

AWS Glue 運用其他 AWS 服務協調您的 ETL (擷取、轉換、載入) 任務以建置資料倉儲或資料湖，並產生輸出串流。AWS Glue 會呼叫 API 操作來轉換您的資料、建立執行時間日誌、存放您的任務邏輯，並建立通知以協助您監控任務執行。AWS Glue 主控台會將這些服務連接至受管應用程式，讓您可以專注於建立和監控 ETL 任務。主控台會代表您執行管理與任務開發的操作。您可向 AWS Glue 提供登入資料和其他屬性，以存取資料來源和寫入資料目標。

AWS Glue 會負責佈建和管理執行任務負載所需的資源。您不必為 ETL 工具建立基礎設施，AWS Glue 會為您處理。需要資源時，為了減少啟動時間，AWS Glue 會從其執行個體暖集區中使用一個執行個體來執行您的任務負載。

有了 AWS Glue，您就可以使用資料目錄中的資料表定義來建立任務。任務由指令碼組成，其中包含執行轉換的程式設計邏輯。您可使用觸發，以排程或指定事件的結果啟動任務。您可決定目標資料存放的位置，以及將何種來源資料填入目標。AWS Glue 會根據您的輸入，產生將資料從來源轉換至目標所需的程式碼。您也可以在 AWS Glue 主控台或 API 提供指令碼以處理資料。

資料來源和目的地

AWS Glue for Spark 允許您從多個系統和資料庫讀取和寫入資料，包括：

- Amazon S3
- Amazon DynamoDB
- Amazon Redshift
- Amazon Relational Database Service (Amazon RDS)
- 第三方 JDBC 可存取的資料庫
- MongoDB 和 Amazon DocumentDB (with MongoDB compatibility)
- 其他 Marketplace 連接器和 Apache Spark 外掛程式

資料串流

AWS Glue for Spark 可串流下列系統中的資料：

- Amazon Kinesis Data Streams
- Apache Kafka

AWS Glue 可在多個 AWS 區域中使用。如需詳細資訊，請參閱 AWS 中的 Amazon Web Services 一般參考 區域與端點。

主題

隔離執行的無伺服器 ETL 任務

AWS Glue 會透過您選擇的引擎、Spark 或 Ray，在無伺服器環境中執行 ETL 任務。AWS Glue 會在其服務帳戶中佈建和管理的虛擬資源上執行這些任務。

AWS Glue 旨在執行以下項目：

- 區域客戶資料。
- 保護客戶傳輸中和靜態的資料。
- 僅在回應客戶要求且必要時存取客戶資料，而使用的是暫時、縮減範圍的登入資料，或經客戶同意使用其帳戶中的 IAM 角色。

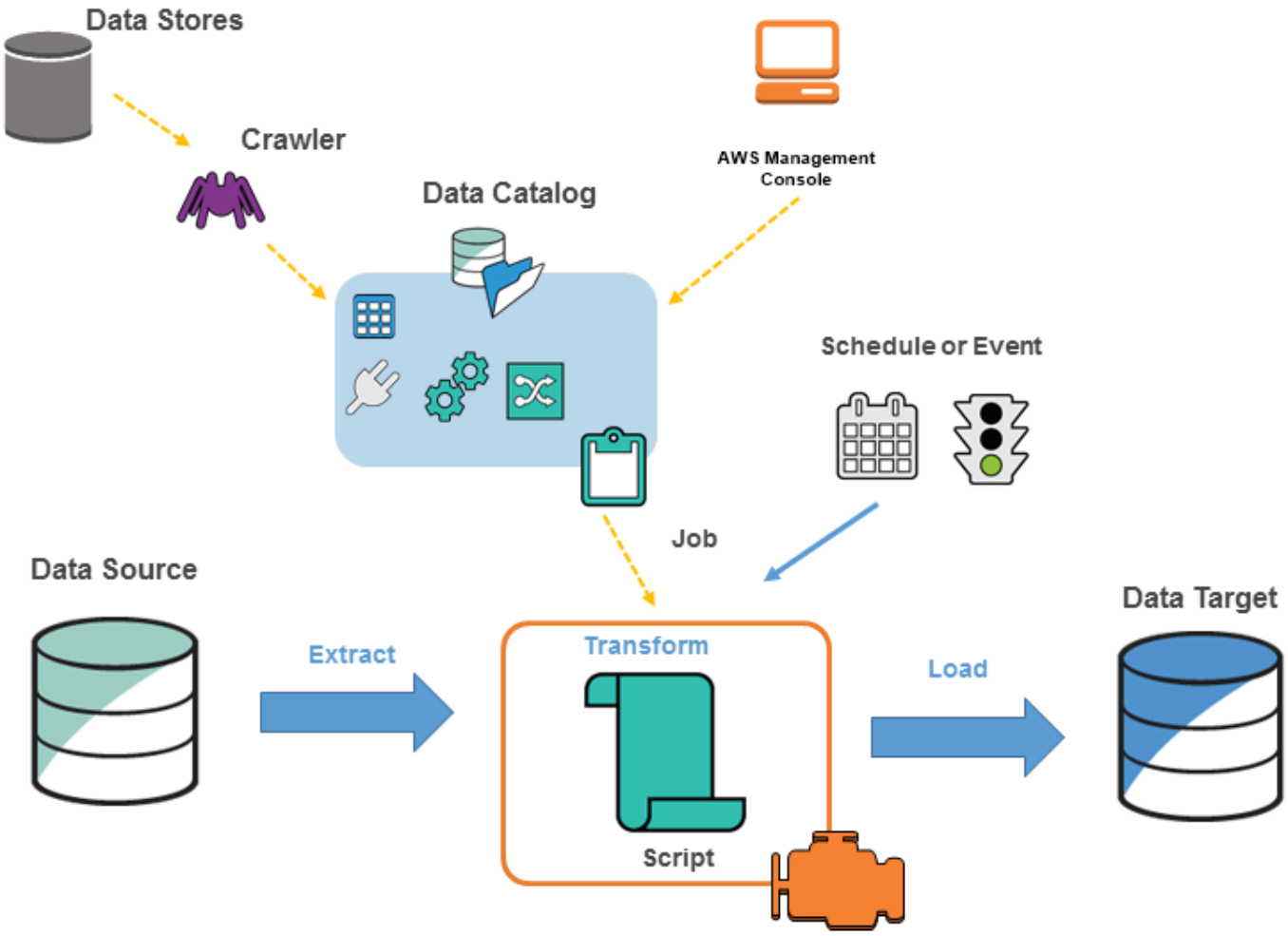
在佈建 ETL 任務時，您應提供 Virtual Private Cloud (VPC) 中的輸入資料來源和輸出資料目標。此外，您應提供存取資料來源和目標所需的 IAM 角色、VPC ID、子網路 ID 和安全群組。AWS Glue 會針對每個值組 (客戶帳戶 ID、IAM 角色、子網路 ID、安全群組) 建立新的環境，並在網路和管理層級與 AWS Glue 服務帳戶中的其他所有環境隔離。

AWS Glue 會使用私有 IP 地址在您的子網路中建立彈性網路界面。任務會使用這些彈性網路介面存取您的資料來源和資料目標。任務執行環境傳入、傳出以及在環境內部往來的流量，都由您的 VPC 與聯網政策控管，只有一項例外：對 AWS Glue 程式庫的呼叫，可透過 AWS Glue VPC 將流量代理至 AWS Glue API 操作。所有的 AWS Glue API 都會加以記錄；因此，資料擁有者可啟用 AWS CloudTrail 以稽核 API 存取，並將稽核記錄傳送至您的帳戶。

執行 ETL 任務的 AWS Glue 受管環境在防護上採用與其他 AWS 服務相同的安全實務。如需作法和共用安全性責任的概觀，請參閱 AWS 安全程序簡介白皮書。

AWS Glue 概念

下圖顯示 AWS Glue 環境的架構。



您可以在 中定義工作AWS Glue，以完成將資料從資料來源擷取、轉換和載入 (ETL) 到資料目標所需的工作。您通常會執行以下動作：

- 針對資料存放區來源，定義一個爬蟲程式，將中繼資料資料表定義填入 AWS Glue Data Catalog。您將爬蟲程式指向資料存放區，爬蟲程式在 Data Catalog 中建立資料表定義。針對串流來源，手動定義 Data Catalog 資料表並指定資料串流屬性。

  除了資料表定義之外，AWS Glue Data Catalog 還包含定義 ETL 任務所需要的其他中繼資料。當您要定義一個任務以轉換資料時，您使用此中繼資料。

- AWS Glue 可以產生指令碼來轉換您的資料。或者，您可以在 AWS Glue 主控台或 API 提供指令碼。
- 您可以隨需執行任務，也可以設定在發生指定的觸發時開始執行任務。觸發可以是以時間為基礎的排程或事件。

  當您的任務執行時，指令碼會從資料來源擷取資料、轉換資料，然後將資料載入到您的資料目標。指令碼在 AWS Glue 中的 Apache Spark 環境執行。

> **Important**
>
> AWS Glue 的資料表和資料庫為 AWS Glue Data Catalog 中的物件。它們包含中繼資料，不包含來自資料存放區的資料。

必須使用 **UTF-8** 將 CSV 之類的文字型資料編碼，AWS Glue 才能順利處理這類資料。如需詳細資訊，請參閱 Wikipedia 中的 UTF-8。

AWS Glue 術語

AWS Glue 依賴數個元件的互動來建立和管理擷取、轉換和載入 (ETL) 工作流程。

AWS Glue Data Catalog

存放於 AWS Glue 的持久性中繼資料。它包含資料表定義、任務定義及其他控制資訊，可管理您的 AWS Glue 環境。每個 AWS 帳戶在每個區域都有一個 AWS Glue Data Catalog。

分類器

判斷資料的結構描述。AWS Glue 提供常見檔案類型 (如 CSV、JSON、AVRO、XML 等) 的分類器。它還提供常見的使用 JDBC 連線的關聯式資料庫管理系統的分類器。您可以撰寫自己的分類器，方法是使用 grok 模式或在 XML 文件中指定資料列標籤。

連線

包含連接到資料存放區所需屬性的 Data Catalog 物件。

爬蟲程式

連接到資料存放區 (來源或目標) 的程式，它會執行已排定優先順序的分類器清單，判斷您資料的結構描述，然後在 AWS Glue Data Catalog 建立中繼資料資料表。

資料庫

一組相關 Data Catalog 資料表定義會整理在邏輯群組中。

資料存放區、資料來源、資料目標

資料存放區是一種適用於永久存放資料的儲存庫。範例包括 Amazon S3 儲存貯體和關聯式資料庫。資料來源是做為程序或轉換輸入的資料存放區。資料目標是寫入程序或轉換的目標資料存放區。

開發端點

可用於開發及測試 AWS Glue ETL 指令碼的環境。

動態框架

支援巢狀資料 (例如結構和陣列) 的分散式資料表。每個記錄都是自我描述，專為具有半結構化資料的結構描述彈性而設計。每筆記錄都包含資料和描述該資料的結構描述。您可以在 ETL 指令碼中使用動態框架和 Apache Spark DataFrames，並在它們之間進行轉換。動態框架為資料清除和 ETL 提供了一組進階轉換。

任務

執行 ETL 任務時所需的商業邏輯。它由轉換指令碼、資料來源和資料目標所組成。任務的執行由可透過排程或事件觸發的觸發器起始。

任務效能儀表板

AWS Glue 為您的 ETL 任務提供全面的執行儀表板。儀表板會顯示特定時間範圍內之任務執行的相關資訊。

筆記本介面

更優異的筆記本體驗，只要按一下即可輕鬆撰寫任務和探索資料。自動設定筆記本和連線。您可以使用以 Jupyter 筆記本為基礎的筆記本介面，以互動方式使用 AWS Glue 無伺服器 Apache Spark ETL 基礎設施開發、偵錯和部署指令碼與工作流程。您也可以在筆記本環境中執行臨機操作查詢、資料分析和視覺化 (例如，資料表和圖形)。

指令碼

用於從來源擷取資料、轉換資料，並將資料載入到目標的程式碼。AWS Glue 會產生 PySpark 或 Scala 指令碼。

資料表

呈現您資料的中繼資料定義。無論您的資料位於 Amazon Simple Storage Service (Amazon S3) 檔案、Amazon Relational Database Service (Amazon RDS) 資料表或其他資料集，皆由資料表定義資料的結構描述。AWS Glue Data Catalog 中的資料表包含欄位名稱、資料類型定義、分割區資訊，以及有關基本資料集的其他中繼資料。您資料的結構描述呈現在 AWS Glue 資料表定義中。實際資料仍保留在其原始資料存放區，無論位於檔案或關聯式資料庫資料表。AWS Glue 在 AWS Glue Data Catalog 中分類您的檔案和關聯式資料庫資料表。當您建立 ETL 時，它們會做為來源和目標。

轉換

用於將資料處理為不同格式的程式碼邏輯。

觸發條件

啟動 ETL 任務。觸發可根據排程時間或事件加以定義。

視覺化任務編輯器

視覺化任務編輯器是一個新的圖形介面，有助於在 AWS Glue 更加輕鬆地內建立、執行、監控擷取、轉換和載入 (ETL) 任務。您可以用視覺化方式撰寫資料轉換工作流程，在 AWS Glue 的 Apache Spark 型無伺服器 ETL 引擎上順暢地執行它們，並在任務的每個步驟中檢查結構描述與資料結果。

工作者

藉助 AWS Glue，您只需依 ETL 任務執行所需的時間付費。您無需管理資源，不用預付款項，也不必為啟動或關機時間付費。我們會根據您執行 ETL 任務所使用的 資料處理單位 (或 DPU) 數量，以小時費率計費。單一資料處理單位 (DPU) 也稱為工作者。AWS Glue 隨附三種工作者類型，可協助您選取符合任務延遲和成本需求的組態。工作者提供標準、G.1X、G.2X 和 G.025X 組態。

AWS Glue 元件

AWS Glue 提供主控台和 API 操作來設定和管理您的擷取、轉換和載入 (ETL) 任務負載。您可以透過多種語言專屬的 SDK 和 AWS Command Line Interface (AWS CLI)，使用 API 操作。如需有關使用 AWS CLI 的詳細資訊，請參閱 AWS CLI 命令參考。

AWS Glue 使用 AWS Glue Data Catalog 來存放有關資料來源、轉換和目標的中繼資料。資料目錄可用於替換 Apache Hive 中繼存放區。AWS Glue Jobs system 提供受管的基礎設施，可用於在您的資料上定義、排程並執行 ETL 操作。如需 AWS Glue API 的詳細資訊，請參閱AWS Glue API。

AWS Glue 主控台

您可以使用 AWS Glue 主控台來定義和協調您的 ETL 任務流程。主控台會呼叫 AWS Glue Data Catalog 和 AWS Glue Jobs system 中的幾個 API 操作來執行下列任務：

- 定義 AWS Glue 物件，例如工作、資料表、爬蟲程式和連線。
- 排程爬蟲程式執行的時間。
- 定義事件或排程以進行任務觸發。
- 搜尋和篩選 AWS Glue 物件清單。
- 編輯轉換指令碼。

AWS Glue Data Catalog

AWS Glue Data Catalog 是您在 AWS 雲端中的持久性技術中繼資料存放區。

每個 AWS 帳戶在每個 AWS 區域都有一個 AWS Glue Data Catalog。每個資料目錄都是組織成資料庫的高度可擴展資料表集合。資料表是存放在 Amazon RDS、Apache Hadoop 分散式檔案系統、Amazon OpenSearch Service 等來源中的結構化或半結構化資料集合的中繼資料表示。AWS Glue Data Catalog 提供統一的儲存庫，讓不同系統存放及搜尋中繼資料，以追蹤資料筒倉中的資料。然後，您可以使用中繼資料在各個應用程式中以一致的方式來查詢和轉換資料。

您可以搭配使用資料目錄與 AWS Identity and Access Management 政策和 Lake Formation 來控制對資料表和資料庫的存取。這樣一來，您可讓企業中不同的群組將資料安全地發佈給更廣泛的組織，同時以更精細的方式保護敏感資訊。

資料目錄、CloudTrail 和 Lake Formation 也可為您提供完整的稽核和管理功能，包括結構描述變更追蹤以及資料存取控制。這有助於確保資料不會受到不適當的修改或意外遭到共享。

如需有關保護和稽核 AWS Glue Data Catalog 的資訊，請參閱：

- AWS Lake Formation – 如需詳細資訊，請參閱《AWS Lake Formation 開發人員指南》https://docs.aws.amazon.com/lake-formation/latest/dg/what-is-lake-formation.html中的什麼是 AWS Lake Formation？。

- CloudTrail – 如需詳細資訊，請參閱 AWS CloudTrail 使用者指南中的什麼是 CloudTrail？。

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https://docs.aws.amazon.com/athena/latest/ug/understanding-tables-databases-and-the-data-catalog.html《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》https://docs.aws.amazon.com/redshift/latest/dg/c-using-spectrum.html中的使用 Amazon Redshift Spectrum 以查詢外部資料。
- Amazon EMR - 如需詳細資訊，請參閱《Amazon EMR 管理指南》中的針對 AWS Glue Data Catalog 的 Amazon EMR 存取使用資源類型政策。
- 適用於 Apache Hive 中繼存放區的 AWS Glue Data Catalog 用戶端 - 如需關於此 GitHub 專案的詳細資訊，請參閱適用於 Apache Hive 中繼存放區的 AWS Glue Data Catalog 用戶端。

AWS Glue 爬蟲程式和分類器

AWS Glue 也可讓您設定爬蟲程式，它可掃描所有類型儲存庫中的資料，進行分類並從資料中擷取結構描述資訊，然後自動將中繼資料存放在 AWS Glue Data Catalog。然後，即可將 AWS Glue Data Catalog 用於引導 ETL 操作。

如需有關如何設定爬蟲程式和分類器的詳細資訊，請參閱 使用編目器填入資料目錄 。如需有關如何使用 AWS Glue API 為爬蟲程式和分類器編寫程式的詳細資訊，請參閱 爬蟲程式和分類器 API。

AWS Glue ETL 操作

使用資料目錄中的中繼資料，AWS Glue 可自動產生 Scala 或 PySpark (Apache Spark 的 Python API) 指令碼，其中具備您可使用並修改的 AWS Glue 延伸模組，藉此執行各種 ETL 操作。例如，您可以擷取、清理和轉換原始資料，然後將結果存放在不同的儲存庫，以供查詢和分析。這種指令碼可能會將 CSV 檔案轉換為關聯形式，並將其儲存在 Amazon Redshift。

如需如何使用 AWS Glue ETL 功能的詳細資訊，請參閱 Spark 指令碼程式設計。

AWS Glue 的串流 ETL

AWS Glue 可讓您使用持續執行的任務，對串流資料執行 ETL 操作。AWS Glue 串流 ETL 是建立在 Apache Spark 結構化串流引擎之上，並且可以從 Amazon Kinesis Data Streams、Apache Kafka 和

Amazon Managed Streaming for Apache Kafka (Amazon MSK) 中擷取串流。串流 ETL 可以清理和轉換串流資料，並將其載入 Amazon S3 或 JDBC 資料存放區。在 AWS Glue 中使用串流 ETL 可處理 IoT 串流、點擊流和網路日誌等事件資料。

如果您知道串流資料來源的結構描述，您可以在資料目錄資料表中指定它。如果沒有，您可以啟用串流 ETL 任務中的結構描述偵測。然後，任務會自動從傳入資料判斷結構描述。

您的 ETL 指令碼可以使用 AWS Glue 內建轉換和 Apache Spark 結構化串流的原生轉換。如需詳細資訊，請參閱 Apache Spark 網站上的串流 DataFrames/資料集上的操作。

如需更多詳細資訊，請參閱the section called “串流 ETL 任務”。

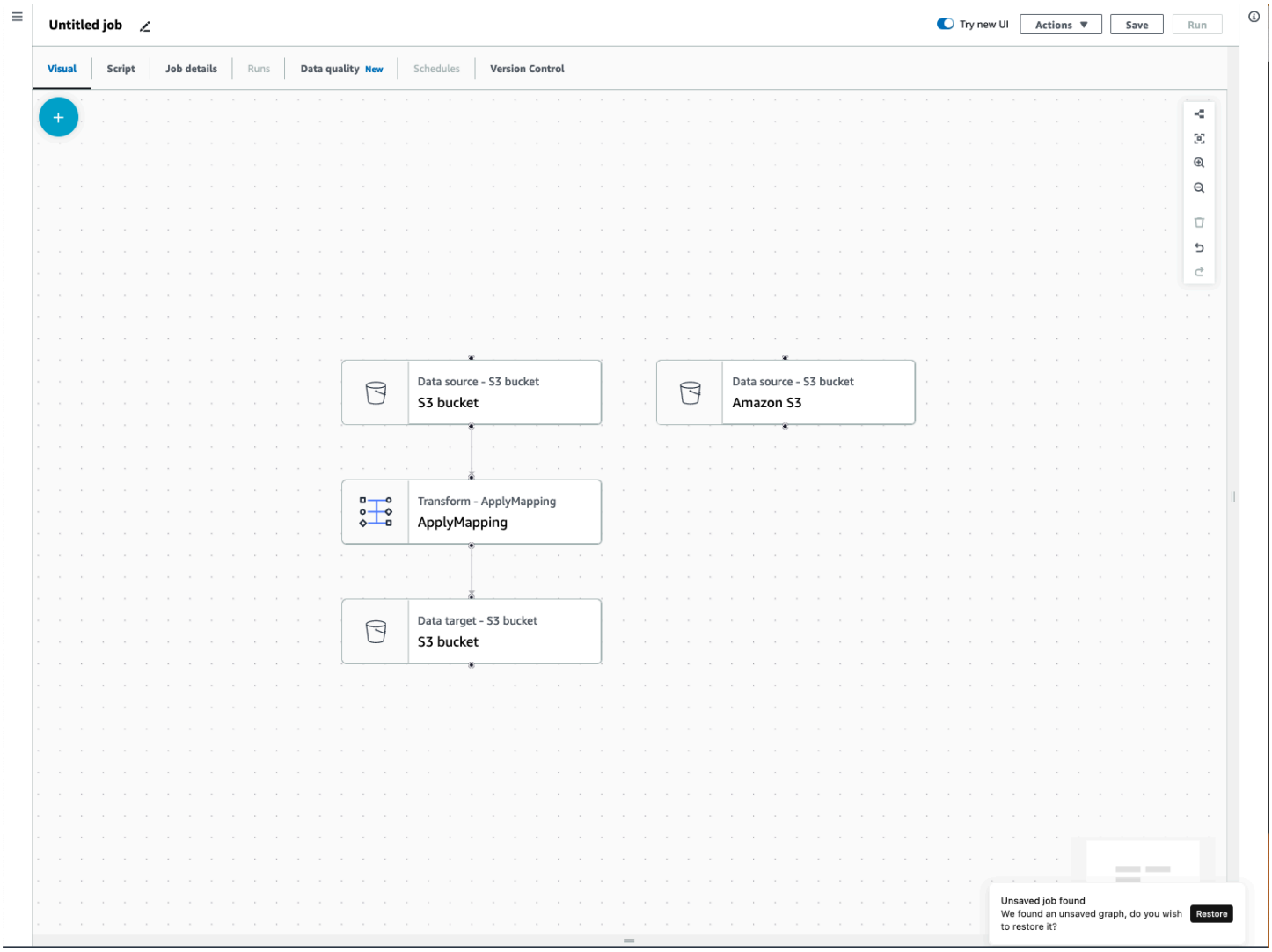
AWS Glue 任務系統

AWS Glue Jobs system 提供受管的基礎設施來協調您的 ETL 任務流程。您在 AWS Glue 中建立的任務，可將資料擷取、轉換和傳送至不同位置所用之指令碼進行自動化。您可以排程和鏈結任務，或透過新資料到達等事件予以觸發。

如需使用 AWS Glue Jobs system 的詳細資訊，請參閱監控 AWS Glue。如需使用 AWS Glue Jobs system API 編寫程式的詳細資訊，請參閱任務 API。

視覺化 ETL 元件

AWS Glue 可讓您透過可操作的視覺化畫布建立 ETL 任務。



ETL 任務選單

畫布頂端的選單選項允許您存取有關任務的各種檢視和組態詳細資訊。

- 視覺化：視覺化任務編輯器畫布。您可以在此處新增節點以建立任務。
- 指令碼：ETL 任務的指令碼展示。AWS Glue 根據任務的視覺化展示產生指令碼。您也可以編輯或下載指令碼。

> **Note**
>
> 如果您選擇編輯指令碼，任務撰寫體驗會永久轉換為僅指令碼模式。之後，您將無法再使用視覺化編輯器來編輯任務。在選擇編輯指令碼之前，您應該新增所有任務來源、轉換和目標，並使用視覺化編輯器進行全部所需變更。

- 任務詳細資訊：「任務詳細資訊」索引標籤可讓您透過設定任務屬性來設定任務。有一些基本屬性，例如任務的名稱和描述、IAM 角色、任務類型、AWS Glue 版本、語言、工作者類型、工作者數量、任務書籤、彈性執行、淘汰數量和任務逾時；還有進階屬性，例如連線、程式庫、任務參數和索引標籤。
- 執行：任務執行後，可存取此索引標籤以檢視過去的任務執行情況。
- 資料品質：資料品質會評估和監控資料資產的品質。您可以在此索引標籤上進一步了解如何使用資料品質，並將資料品質轉換新增至您的任務。
- 排程：您已排程的任務會顯示在此索引標籤中。如果沒有連接到此任務的排程，則無法存取此索引標籤。
- 版本控制：您可以將任務設定至 Git 儲存庫，以便將 Git 與任務搭配使用。

視覺化 ETL 面板

當您在畫布中工作時，有多個面板可協助您設定節點，或協助您預覽資料並檢視輸出結構描述。

- 屬性：當您在畫布上選擇節點時，會顯示「屬性」面板。
- 資料預覽：「資料預覽」面板提供資料輸出的預覽，讓您可以在執行任務和檢查輸出之前做出決定。
- 輸出結構描述：「輸出結構描述」索引標籤可讓您檢視和編輯轉換節點的結構描述。

調整面板大小

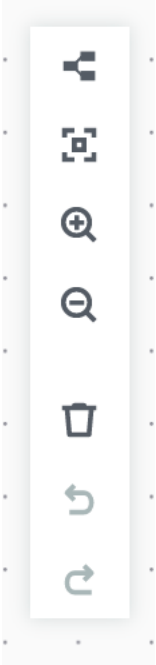
您可以調整畫面右側的「屬性」面板以及包含「資料預覽」和「輸出結構描述」索引標籤的底部面板的大小，方法是按一下面板邊緣，然後左右或上下拖曳它。

- 屬性面板：按一下並拖曳畫面右側的畫布邊緣，然後將其向左拖曳以擴大寬度，即可調整屬性面板的大小。依預設，面板會收合，而當選取節點時，屬性面板會以預設大小開啟。
- 資料預覽和輸出結構描述面板：按一下並拖曳畫面底部的畫布底部邊緣，然後將底部面板向上拖曳以擴大其高度，即可調整底部面板的大小。依預設，面板會收合，而當選取節點時，底部面板會以預設大小開啟。

任務畫布

您可以直接在視覺化 ETL 畫布上新增、移除和移動/重新排序節點。您可以將其視為工作區，以建立功能齊全的 ETL 任務，該任務從資料來源開始，且能以資料目標結束。

當您使用畫布上的節點時，工具列可以協助您放大和縮小、移除節點、建立或編輯節點之間的連線、變更任務流程方向，以及復原或重做動作。



浮動工具列會錨定至畫布右上角的大小，並包含多個執行動作的影像：

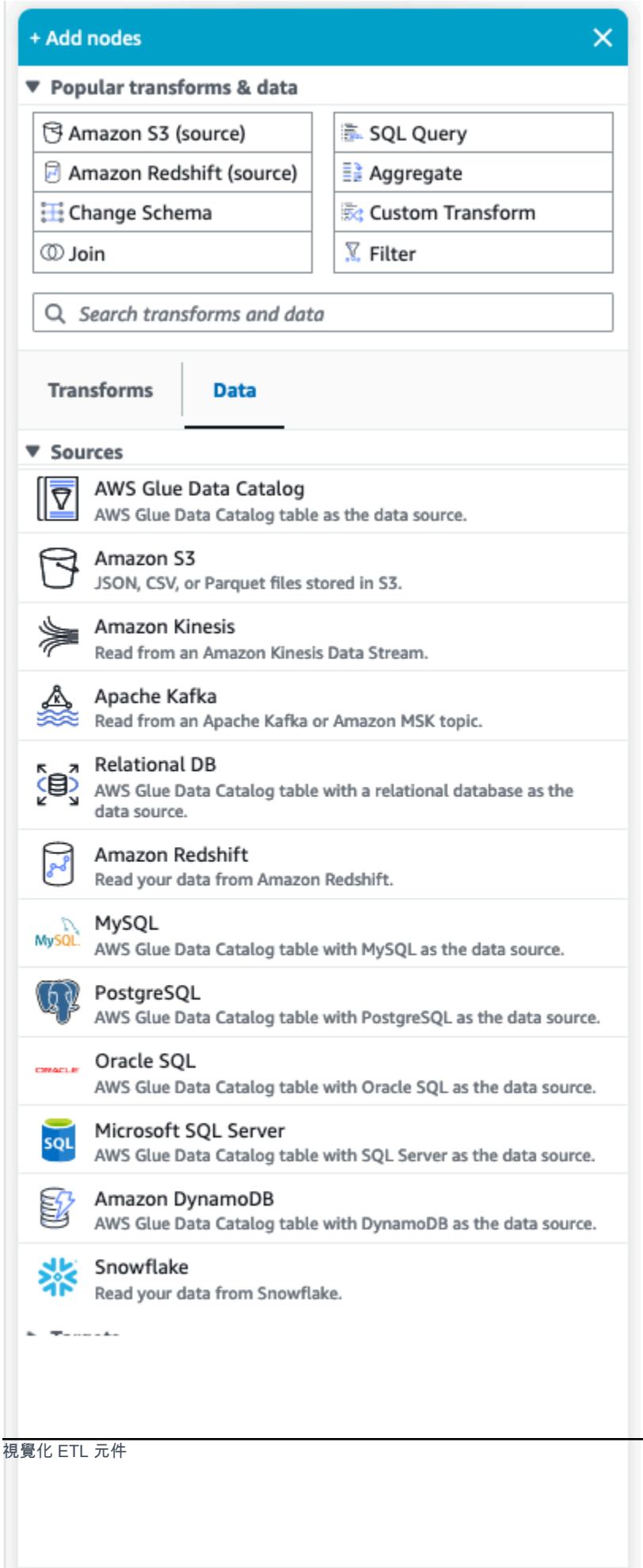
- 版面圖示：工具列中的第一個圖示是版面圖示。依預設，視覺任務的方向是從上到下。它透過從左到右水平排列節點來重新排列視覺化任務的方向。再次按一下版面圖示可將方向變更回從上到下。
- 重新置中圖示：重新置中圖示會透過將畫布檢視置中來變更畫布檢視。您可以將其與大型任務一起使用，以返回到中心位置。
- 放大圖示：放大圖示可放大畫布上節點的大小。
- 縮小圖示：縮小圖示可縮小畫布上節點的大小。
- 垃圾桶圖示：垃圾桶圖示會從視覺化任務中移除節點。您必須先選取節點。
- 復原圖示：復原圖示可回復對視覺化任務執行的最後一個動作。
- 重做圖示：重做圖示可重複對視覺化任務執行的最後一個動作。

使用迷你地圖

資源面板

資源面板包含您可用的所有資料來源、轉換動作和連線。透過按一下 "+" 圖示開啟畫布上的資源面板。這將開啟資源面板。

若要關閉資源面板，按一下資源面板右上角的 X。這樣會隱藏面板，直到您準備好再次開啟面板為止。


+ Add nodes
Popular transforms & data
Amazon S3 (source)
Amazon Redshift (source)
Change Schema
Join
SQL Query
Aggregate
Custom Transform
Filter
Search transforms and data
Transforms
Data
Sources
AWS Glue Data Catalog
AWS Glue Data Catalog table as the data source.
Amazon S3
JSON, CSV, or Parquet files stored in S3.
Amazon Kinesis
Read from an Amazon Kinesis Data Stream.
Apache Kafka
Read from an Apache Kafka or Amazon MSK topic.
Relational DB
AWS Glue Data Catalog table with a relational database as the data source.
Amazon Redshift
Read your data from Amazon Redshift.
MySQL
AWS Glue Data Catalog table with MySQL as the data source.
PostgreSQL
AWS Glue Data Catalog table with PostgreSQL as the data source.
Oracle SQL
AWS Glue Data Catalog table with Oracle SQL as the data source.
Microsoft SQL Server
AWS Glue Data Catalog table with SQL Server as the data source.
Amazon DynamoDB
AWS Glue Data Catalog table with DynamoDB as the data source.
Snowflake
Read your data from Snowflake.

熱門轉換與資料

面板頂部有一個熱門轉換與資料的集合。這些節點通常在 AWS Glue 中使用。選擇一個節點以將其新增至畫布。您也可以按一下熱門轉換與資料標題旁的三角形，隱藏熱門轉換與資料。

在熱門轉換與資料區段下方，您可以搜尋轉換和資料來源節點。結果會在您輸入時顯示。您新增至搜尋查詢的字母越多，結果清單就會越小。搜尋結果會根據節點名稱和/或描述填入。選擇對應節點以將其新增至畫布。

轉換與資料

有兩個索引標籤可將節點組織為轉換與資料。

轉換：當您選擇轉換索引標籤時，可以選取所有可用的轉換。選擇一個轉換以將其新增至畫布。您也可以選擇轉換清單底部的新增轉換，這樣會開啟文件的新頁面，以建立自訂視覺化轉換。按照這些步驟操作可建立自己的轉換。然後，您的轉換就會出現在可用轉換清單中。

資料：資料索引標籤包含來源和目標的所有節點。您可以按一下「來源」或「目標」標題旁的三角形，隱藏來源和目標。再次按一下三角形可取消隱藏來源和目標。選擇來源或目標節點以將其新增至畫布。您也可以選擇管理連線來新增連線。這將打開主控台中的「連接器」頁面。

AWS Glue for Spark 與 AWS Glue for Ray

在 AWS Glue on Apache Spark (AWS Glue ETL) 中，您可使用 PySpark 撰寫 Python 程式碼來大規模處理資料。Spark 是解決此問題的常見解決方案，但若資料工程師的工作背景是以 Python 為主，可能會發現轉換不直觀。Spark DataFrame 模型並非極具 Python 風格 (Pythonic)，其在建置時反映 Scala 語言與 Java 執行階段。

在 AWS Glue 中，您可使用 Python Shell 任務執行原生 Python 資料整合。這些任務會在單一 Amazon EC2 執行個體上運作，並受該執行個體容量的限制。這會限制您可以處理的資料輸送量，讓處理大數據時的維護成本變得昂貴。

AWS Glue for Ray 允許您縱向擴展 Python 工作負載，而無需對學習 Spark 進行大量投資。您可善用 Ray 表現更加出色的某些情況。您可以借助提供給您的選擇同時運用 Spark 和 Ray 的優勢。

AWS Glue ETL 和 AWS Glue for Ray 的底層有所不同，因此其支援不同的功能。請查看文件判斷支援的功能。

什麼是 AWS Glue for Ray？

Ray 是開放原始碼的分散式運算架構，可用於縱向擴展工作負載並專注處理 Python。如需有關 Ray 的詳細資訊，請參閱 Ray 網站。AWS GlueRay 任務和互動式工作階段可讓您在 AWS Glue 中使用 Ray。

您可以使用 AWS Glue for Ray 撰寫 Python 指令碼來進行運算，運算會在多台電腦上平行執行。在 Ray 任務與互動式工作階段中，您可以使用熟悉的 Python 程式庫 (例如 pandas)，讓您的工作流程易於撰寫和執行。如需有關 Ray 資料集的詳細資訊，請參閱 Ray 文件中的 Ray 資料集。如需有關 pandas 的詳細資訊，請參閱 Pandas 網站。

當您使用 AWS Glue for Ray 時，僅需幾行程式碼即可針對企業規模的大數據執行 pandas 工作流程。您可以透過 AWS Glue 主控台或 AWS SDK 建立 Ray 任務。您也可以開啟 AWS Glue 互動式工作階段，在無伺服器 Ray 環境中執行程式碼。尚不支援 AWS Glue Studio 中的視覺化任務。

AWS Glue for Ray 任務可讓您根據排程執行指令碼，或回應來自 Amazon EventBridge 的事件。任務會存放 CloudWatch 中的日誌資訊和監控統計資訊，讓您了解指令碼的運作狀態和可靠性。如需有關 AWS Glue 任務系統的詳細資訊，請參閱the section called “使用 Ray 任務”。

AWS Glue for Ray 互動式工作階段 (預覽版) 可讓您針對相同的佈建資源逐一執行程式碼片段。您可以使用它來有效率地建立原型並開發指令碼，或建置專屬互動式應用程式。您可以在 AWS Management Console 中使用來自 AWS Glue Studio 筆記本的 AWS Glue 互動式工作階段。如需詳細資訊，請參閱搭配 AWS Glue Studio 和 AWS Glue 使用筆記本。您也可以透過 Jupyter 核心使用它們，其可讓您從支援 Jupyter 筆記本的現有程式碼編輯工具 (例如 VSCode) 執行互動式工作階段。如需詳細資訊，請參閱 the section called “AWS Glue 用於 Ray 互動式工作階段 (預覽)”。

Ray 會根據負載，將處理程序分配至一組機器叢集，藉此自動調整 Python 程式碼的擴展任務。這會改善某些工作負載的每美元效能。我們已透過 Ray 任務將自動擴展功能原生建置至 AWS Glue 工作模型中，因此您可以充分利用此功能。Ray 任務會在 AWS Graviton 上執行，從整體上提升價格/效能比。

除了節省成本，您還可以使用原生自動擴展功能來執行 Ray 任務負載，而無須耗費時間進行叢集維護、調整和管理。您可以立即使用熟悉的開放原始碼程式庫，例如 pandas 和 pandas 版 AWS SDK。這些程式庫可提高您在 AWS Glue for Ray 上進行開發時的反覆運算速度。在使用 AWS Glue for Ray 時，您可以快速開發和執行具成本效益的資料整合工作負載。

將半結構化結構描述轉換為具有 AWS Glue 的關聯式結構描述

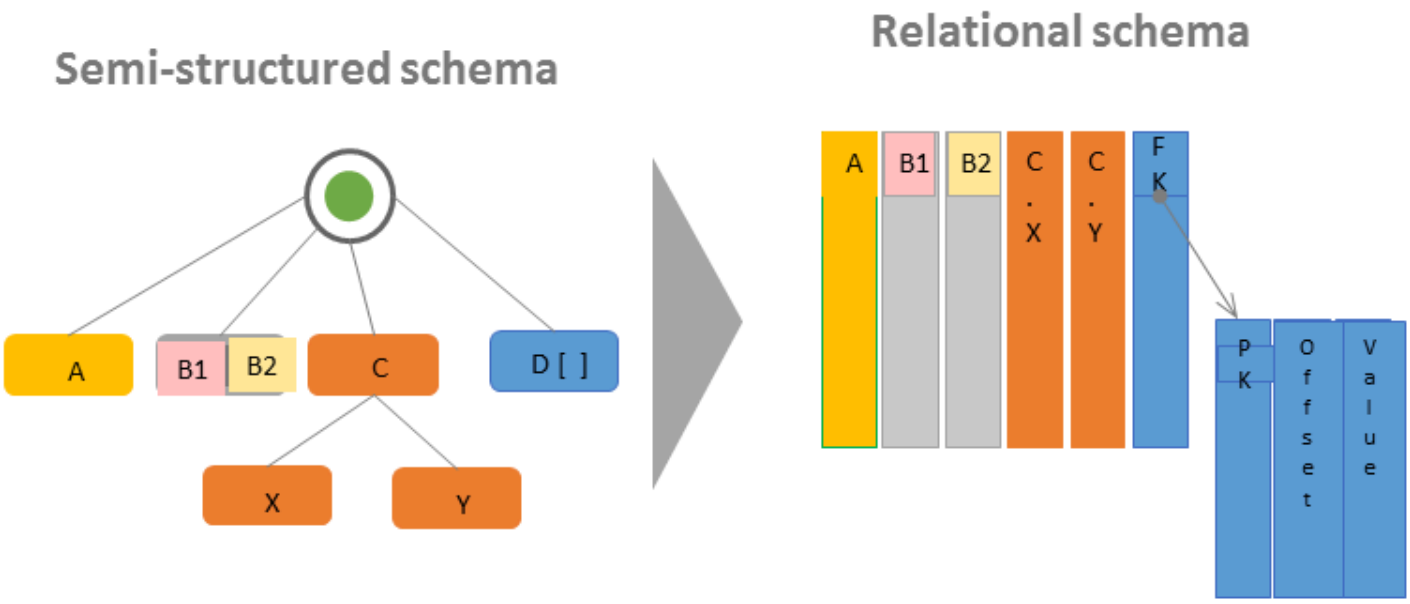
您很可能會想將半結構化資料轉換為關聯式資料表。在概念上，這是將階層式結構描述展平為關聯式結構描述。AWS Glue 可以迅速為您執行此轉換作業。

半結構化資料通常包含用以識別資料內實體的標記。這種資料有巢狀資料結構，而無固定式結構描述。如需半結構化資料的詳細資訊，請參閱 Wikipedia 的 Semi-structured data (半結構化資料)。

關聯式資料即為列和欄組成的資料表。資料表之間的關係可以用主金鑰 (PK) 至外部金鑰 (FK) 的關係表示。如需詳細資訊，請參閱 Wikipedia 的 Relational database (關聯式資料庫)。

AWS Glue 使用爬蟲程式推斷半結構化資料的結構描述。接著，資料會藉由擷取、轉換和載入 (ETL) 任務轉換為關聯式結構描述。例如，您可能想將 Amazon Simple Storage Service (Amazon S3) 來源檔案的 JSON 資料剖析為 Amazon Relational Database Service (Amazon RDS) 資料表。了解 AWS Glue 如何處理結構描述之間的差異，可協助您了解轉換程序。

此圖表說明 AWS Glue 如何將半結構化結構描述轉換為關聯式結構描述。



此圖展示了以下要點：

- 單值 A 直接轉換為關聯式欄。

- B1 和 B2 值對轉換為兩個關聯式欄。
- 結構 C 以及子結構 X 和 Y 轉換為兩個關聯式欄。
- 陣列 D[] 轉換為關聯式欄以及指向另一個關聯式資料表的外部金鑰 (FK)。除了主金鑰 (PK)，第二個關聯式資料表還有包含陣列項目之位移和值的欄位。

AWS Glue 型系統

AWS Glue 使用多種類型的系統來提供資料系統的多功能介面，這些資料系統會以非常不同的方式儲存本文件消除 AWS Glue 類型系統和資料標準的歧義。

AWS Glue 資料型錄類型

Data Catalog 是存放於各種資料系統 (中繼存放區) 的資料表和欄位登錄檔。當 AWS Glue 元件 (例如 AWS Glue 爬蟲程式和 AWS Glue to Spark 工作) 寫入資料目錄時，它們會使用內部類型系統來追蹤欄位類型。這些值會顯示在 AWS Glue Console 中資料表結構定義的「資料類型」欄中。這種類型的系統是以 Apache Hive 的類型系統為基礎。如需有關 Apache Hive 類型系統的詳細資訊，請參閱 Apache Hive 維基中的類型。如需有關特定類型和支援的詳細資訊， AWS Glue Console 提供範例，做為結構描述產生器的一部分。

驗證、相容性和其他用途

Data Catalog 不會驗證寫入至類型欄位的類型。當 AWS Glue 元件讀取和寫入資料目錄時，它們會彼此相容。 AWS Glue 組件還旨在保持與 Hive 類型的高度兼容性。但是， AWS Glue 組件不保證與所有 Hive 類型的兼容性。這可讓您在使用 Data Catalog 中的資料表時，實現與 Athena DDL 等工具互通性。

由於 Data Catalog 不會驗證類型，因此其他服務可能會使用 Data Catalog，透過與 Hive 類型系統或任何其他系統高度符合的系統來追蹤類型。

在 AWS Glue 與火花腳本類型

當 AWS Glue in Spark 指令碼解譯或轉換資料集時，我們會提供DynamicFrame資料集在指令碼中使用的記憶體內表示法。DynamicFrame 的目標與 Spark DataFrame 類似 – 其會為您的資料集建模，以便 Spark 可對您的資料排程和執行轉換。我們保證，DynamicFrame 的類型表示法可透過提供 toDF 和 fromDF 方法與 DataFrame 相互相容。

如果可推斷類型資訊或提供給 DataFrame，除非另有記錄，否則可進行推斷或提供給 DynamicFrame。當我們針對特定資料格式提供最佳化的讀取器或寫入器時，如果 Spark 可讀取或寫

入您的資料，則我們提供的讀取器或寫入器將受到文件限制。如需有關讀取器或寫入器的詳細資訊，請參閱 the section called “資料格式選項”。

選項類型

DynamicFrames 提供一種機制，用於在資料集中進行欄位建模，該資料集的值可能在磁碟上具有不一致的資料列類型。例如，一個欄位可在某些資料列中保留做為字串存放的數字，而在其他資料列中則保留做為整數存放的數字。這種機制是一種稱為 Choice 的記憶體類型。我們提供諸如ResolveChoice方法之類的轉換，以將 Choice 列解析為具體類型。 AWS Glue ETL 不會在一般操作過程中將 Choice 類型寫入資料目錄；選擇類型只存在於資料集的 DynamicFrame 記憶體模型內容中。如需選項類型用量的範例，請參閱 the section called “資料準備範例”。

AWS Glue 履帶類型

爬蟲的目標是為您的資料集產生一致且可用的結構描述，然後將其儲存在資料目錄中，以便用於其他 AWS Glue 元件和 Athena。爬蟲程式根據 Data Catalog 上一節內容 the section called “AWS Glue 資料型錄類型” 中所述來處理類型。若要在「選項」類型案例中產生可用的類型，其中一個資料欄包含兩個或多個類型的值，爬蟲程式將會建立 struct 類型來對潛在類型進行建模。

AWS Glue 入門

下列各節提供了有關設定 AWS Glue 的資訊。並非所有的設定部分都是開始使用 AWS Glue 的必要選項。您可以根據需要使用指示來設定 IAM 許可、加密和 DNS (如果您使用 VPC 環境來存取資料存放區或如果您使用互動式工作階段)。

主題

使用 AWS Glue 的概觀

有了 AWS Glue，您可將中繼資料存放在 AWS Glue Data Catalog 內。您可以使用這個中繼資料來協調 ETL 任務，轉換資料來源及載入資料倉儲或資料湖。下列步驟說明使用 AWS Glue 時的一般任務流程和您會用到的部分選項。

> **Note**
>
> 您可以使用下列步驟，或建立自動執行步驟 1 到 3 的工作流程。如需詳細資訊，請參閱 the section called “使用藍圖和工作流程來執行複雜的 ETL 活動”。

1. 用資料表定義填入 AWS Glue Data Catalog。

   在主控台中，您可以針對持久性資料存放區新增爬蟲程式，以填入 AWS Glue Data Catalog。您可以從資料表清單或爬蟲程式清單中啟動 Add crawler (新增爬蟲程式) 精靈。您可以選擇一或多個資料存放區供爬蟲程式存取。也可以建立排程，以決定執行爬蟲程式的頻率。您可以為資料串流手動建立資料表定義，並定義串流屬性。

   也可以選擇性提供自訂分類器，以推斷資料的結構描述。建立自訂分類器時可用 Grok 模式。不過，AWS Glue 提供了內建的分類器，可在自訂分類器無法識別您的資料時自動供爬蟲程式使用。

定義爬蟲程式時，您不需要選擇分類器。如需 AWS Glue 中分類器的詳細資訊，請參閱將分類器新增至 AWS Glue 中的爬蟲程式。

爬取某些類型的資料存放區需要連線，以提供驗證和位置資訊。如有需要，您可以建立連線，在 AWS Glue 主控台中提供這項必要資訊。

爬蟲程式會讀取您的資料存放區，以及在 AWS Glue Data Catalog 中建立資料定義和具名資料表。這些資料表會整理到您選擇的資料庫內。您也可以用手動建立的資料表填入資料目錄。透過這個方法提供結構描述和其他中繼資料，以在資料目錄中建立資料表定義。由於此方法有點複雜且容易出錯，因此最好由爬蟲程式建立資料表定義。

如需在 AWS Glue Data Catalog 填入資料表定義的詳細資訊，請參閱建立資料表。

2. 定義任務，描述資料從來源到目標的轉換。

   一般而言，若要建立任務，您需要進行以下選擇：

   - 從 AWS Glue Data Catalog 選擇要成為任務來源的資料表。您的任務將使用此資料表定義來存取資料來源及解譯資料的格式。
   - 從 AWS Glue Data Catalog 選擇要成為任務目標的資料表或位置。您的任務將使用此資訊來存取資料存放區。
   - 要求 AWS Glue 產生指令碼，將來源轉換為目標。AWS Glue 將產生程式碼，呼叫內建的轉換，將資料從來源結構描述轉換為目標結構描述格式。這些轉換會視需要執行像是資料複製、重新命名欄和篩選資料等操作來轉換資料。您可以在 AWS Glue 主控台修改指令碼。

   如需在 AWS Glue 定義任務的詳細資訊，請參閱使用 AWS Glue Studio 建立視覺化 ETL 任務。

3. 執行任務，以轉換資料。

   您可以隨需執行任務，或在發生下列其中一種觸發時開始執行：

   - 以 Cron 排程為基礎的觸發程式。
   - 以事件為基礎的觸發；例如，成功完成另一個任務便能開始 AWS Glue 任務。
   - 可隨需開始任務的觸發。

   關於 AWS Glue 中觸發條件的詳細資訊，請參閱使用觸發啟動任務和爬蟲程式。

4. 監控排程的爬蟲程式和觸發的工作。

   AWS Glue 主控台可用來檢視下列內容：

   - 任務執行詳細資訊和錯誤。
   - 爬蟲程式執行詳細資訊和錯誤。

- 有關 AWS Glue 活動的任何通知

如需在 AWS Glue 中監控爬蟲程式和工作的詳細資訊，請參閱 監控 AWS Glue。

設定下列項目的 IAM 許可 AWS Glue

本主題中的指示可協助您快速設定 AWS Identity and Access Management (IAM) 的許可 AWS Glue。您將完成下列任務：

- 授予您的 IAM 身分對 AWS Glue 資源的存取權限。
- 建立用於執行工作、存取資料和執行資 AWS Glue 料品質工作的服務角色。

如需可用來自訂 IAM 許可的詳細指示 AWS Glue，請參閱設定 AWS Glue 的 IAM 許可。

若要 AWS Glue 在中設定 IAM 許可 AWS Management Console

1. 請登入 AWS Management Console 並開啟 AWS Glue 主控台，網址為 https://console.aws.amazon.com/glue/。
2. 選擇 Getting started (入門)。
3. 在 [準備您的帳戶] 下方 AWS Glue，選擇 [設定 IAM 許可]。
4. 選擇您要 AWS Glue 授予權限的 IAM 身分 (角色或使用者)。 AWS Glue 將`AWSGlueConsoleFullAccess`受管理的原則附加至這些身分識別。如果您想要手動設定這些許可，或只想設定預設的服務角色，則可略過此步驟。
5. 選擇下一步。
6. 選擇您的角色和使用者所需的 Amazon S3 存取層級。您在此步驟中選擇的選項會套用至您選取的所有身分。

   a. 在選擇 S3 位置下，選擇您要授予存取權的 Amazon S3 位置。

   b. 接下來，選取您的身分對先前選取的位置應具有「唯讀」(建議) 或「讀寫」存取權。 AWS Glue 根據您選取的位置和讀取或寫入權限的組合，將權限原則新增至您的身分識別。

      下表顯示針對 Amazon S3 存取 AWS Glue 附加的許可。

如果選擇...	AWS Glue 附加...
未變更	沒有權限。 AWS Glue 不會對您的身份權限進行任何更改。
授予對特定 Amazon S3 位置的存取權 (唯讀)	嵌入在您選取的 IAM 身分中的內嵌政策。如需詳細資訊，請參閱《IAM 使用者指南》中的內嵌政策。 AWS Glue 使用下列慣例來命名策略：`AWSGlueConsole` *`<Role/User>`* `InlinePolicy-read-specific-access-` *`<UUID>`*例如：`AWSGlueConsoleRole InlinePolicy-read-specific-access-123456780123` 。 以下是內嵌政策的範例，該政策 AWS Glue 附加以授與指定 Amazon S3 位置的唯讀存取權限。 `{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Action": [ "s3:Get*", "s3:List*" ], "Resource": [ "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*" ] } ] }`

如果選擇...	AWS Glue 附加...
授予對指定 Amazon S3 位置的存取權 (讀取和寫入)	嵌入在您選取的 IAM 身分中的內嵌政策。如需詳細資訊，請參閱《IAM 使用者指南》中的內嵌政策。 AWS Glue 使用下列慣例來命名策略：`AWSGlueConsole` *`<Role/User>`* `InlinePolicy-read-and-write-specific-access-` *`<UUID>`*例如：`AWSGlueConsoleRole InlinePolicy-read-and-write-specific-access-123456780123` 。 以下是內嵌政策的範例，該政策 AWS Glue 附加以授與指定 Amazon S3 位置的讀取和寫入存取權限。 `{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Action": [ "s3:Get*", "s3:List*", "s3:*Object*" ], "Resource": [ "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*", "arn:aws:s3:::DOC-EXAMPLE-BUCKET2/*" ] } ] }`

如果選擇...	AWS Glue 附加...
授予對 Amazon S3 的完整存取權 (唯讀)	`AmazonS3ReadOnlyAccess` 受管 IAM 政策。若要深入了解，請參閱AWS 受管政策：AmazonS3 ReadOnlyAccess。
授予對 Amazon S3 的完整存取權 (讀取和寫入)	`AmazonS3FullAccess` 受管 IAM 政策。若要深入了解，請參閱AWS 受管政策：AmazonS3 FullAccess。

7. 選擇下一步。
8. 選擇帳戶的預設 AWS Glue 服務角色。服務角色是 IAM 角色， AWS Glue 用於代表您存取其他 AWS 服務中的資源。如需詳細資訊，請參閱 AWS Glue 的服務角色。

   - 當您選擇標準 AWS Glue 服務角色時，請在您的 AWS 帳戶 指名`AWSGlueServiceRole`中 AWS Glue 建立新的 IAM 角色，並附加下列受管政策。如果您的帳戶已具有名為的 IAM 角色`AWSGlueServiceRole`，請 AWS Glue 將這些政策附加到現有角色。
     - AWSGlueServiceRole
     - 亞馬遜 FullAccess
   - 當您選擇現有的 IAM 角色時， AWS Glue 請將該角色設定為預設角色，但不會為其新增任何許可。請確定您已設定要用作服務角色的角色 AWS Glue。如需詳細資訊，請參閱 步驟 1：建立適用於 AWS Glue 服務的 IAM 政策 及 步驟 2：為 AWS Glue 建立 IAM 角色。

9. 選擇下一步。
10. 最後，檢閱您選取的許可，然後選擇套用變更。套用變更時， AWS Glue 將 IAM 許可新增至您選取的身分。您可以在 IAM 主控台中檢視或修改新許可，網址為 https://console.aws.amazon.com/iam/。

您現在已完成的最低 IAM 許可設定 AWS Glue。在生產環境中，我們建議您熟悉AWS Glue 中的安全性並AWS Glue 的身分識別與存取管理協助您保護使用案例的 AWS 資源。

後續步驟

現在您已設定 IAM 許可，您可以瀏覽下列主題以開始使用 AWS Glue：

- AWS Glue 在 AWS 技能建置器中開始使用

- AWS Glue Data Catalog入門

設定 AWS Glue Studio

第一次使用視覺化 ETL 的 AWS Glue 時，請完成本節中的工作：

主題

- 檢閱 AWS Glue Studio 使用者所需的 IAM 許可
- 檢閱 ETL 任務所需的 IAM 許可
- 設定 AWS Glue Studio 的 IAM 許可
- 為您的 ETL 任務設定 VPC

檢閱 AWS Glue Studio 使用者所需的 IAM 許可

若要使用 AWS Glue Studio，使用者必須能夠存取各種 AWS 資源。使用者必須能夠檢視和選取 Amazon S3 儲存貯體、IAM 政策和角色，以及 AWS Glue Data Catalog 物件。

AWS Glue 服務許可

AWS Glue Studio 使用 AWS Glue 服務的動作和資源。您的使用者需要這些動作和資源的許可，才能有效地使用 AWS Glue Studio。您可以授予 AWS Glue Studio 使用者 `AWSGlueConsoleFullAccess` 受管政策，或使用較小的許可集建立自訂政策。

> **Important**
>
> 根據安全性最佳實務，建議透過收緊政策來限制存取，進一步限制對 Amazon S3 儲存貯體和 Amazon CloudWatch 日誌群組的存取。如需範例 Amazon S3 政策，請參閱編寫 IAM 政策：如何授予 Amazon S3 儲存貯體的存取。

為 AWS Glue Studio 建立自訂 IAM 政策

您可以為 AWS Glue Studio 建立具有較小許可集的自訂政策。政策可以針對物件或動作的子集授予許可。建立自訂政策時，請使用下列資訊。

若要使用 AWS Glue Studio API，請在您 IAM 許可的動作政策中納入 `glue:UseGlueStudio`。使用 `glue:UseGlueStudio` 將讓您能存取所有 AWS Glue Studio 動作，即使隨著時間的推移會有更多動作新增到 API 中也能存取。

有向無環圖 (DAG) 動作

- CreateDag
- UpdateDag
- GetDag
- DeleteDag

任務動作

- SaveJob
- GetJob
- CreateJob
- DeleteJob
- GetJobs
- UpdateJob

任務執行動作

- StartJobRun
- GetJobRuns
- BatchStopJobRun
- GetJobRun
- QueryJobRuns
- QueryJobs
- QueryJobRunsAggregated

結構描述動作

- GetSchema
- GetInferredSchema

資料庫動作

- GetDatabases

計畫動作

- GetPlan

資料表動作

- SearchTables
- GetTables
- GetTable

連線動作

- CreateConnection
- DeleteConnection
- UpdateConnection
- GetConnections
- GetConnection

映射動作

- GetMapping

S3 代理動作

- ListBuckets
- ListObjectsV2
- GetBucketLocation

安全組態動作

- GetSecurityConfigurations

指令碼動作

- CreateScript (不同於 AWS Glue 中同名的 API)

存取 AWS Glue Studio API

若要存取 AWS Glue Studio，將 `glue:UseGlueStudio` 新增到 IAM 許可的動作政策中。

在以下範例中，將 `glue:UseGlueStudio` 納入動作政策中，但未單獨識別 AWS Glue Studio API。這是因為當您納入 `glue:UseGlueStudio` 時，會自動授予您對內部 API 的存取權，而無需在 IAM 許可中指定個別 AWS Glue Studio API。

在範例中，列出的其他動作政策 (例如 `glue:SearchTables`) 不是 AWS Glue Studio API，因此要視需要將其納入 IAM 許可中。您可能還希望納入 Amazon S3 代理動作，以指定要授予的 Amazon S3 存取等級。以下範例政策提供存取權，能打開 AWS Glue Studio、建立視覺任務，並在選定的 IAM 角色具有足夠的存取權時儲存/執行該任務。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "glue:UseGlueStudio",
                "iam:ListRoles",
                "iam:ListUsers",
                "iam:ListGroups",
                "iam:ListRolePolicies",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "glue:SearchTables",
                "glue:GetConnections",
                "glue:GetJobs",
                "glue:GetTables",
                "glue:BatchStopJobRun",
                "glue:GetSecurityConfigurations",
                "glue:DeleteJob",
                "glue:GetDatabases",
                "glue:CreateConnection",
                "glue:GetSchema",
                "glue:GetTable",
                "glue:GetMapping",
                "glue:CreateJob",
                "glue:DeleteConnection",
                "glue:CreateScript",
                "glue:UpdateConnection",
```

```
                "glue:GetConnection",
                "glue:StartJobRun",
                "glue:GetJobRun",
                "glue:UpdateJob",
                "glue:GetPlan",
                "glue:GetJobRuns",
                "glue:GetTags",
                "glue:GetJob",
                "glue:QueryJobRuns",
                "glue:QueryJobs",
                "glue:QueryJobRunsAggregated"
            ],
            "Resource": "*"
        },
        {
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iam::*:role/AWSGlueServiceRole*",
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "glue.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

筆記本和資料預覽許可

資料預覽和筆記本可讓您在任務的任何階段 (讀取、轉換、寫入) 檢視資料範例，而無需實際執行任務。您可以指定 AWS Identity and Access Management (IAM) 角色，以便 AWS Glue Studio 在存取資料時使用。IAM 角色應可擔任，並且沒有與之關聯的標準長期憑證，例如密碼或存取金鑰。反之，當 AWS Glue Studio 擔任角色時，IAM 會為其提供臨時安全性憑證。

若要確保資料預覽和記事本命令正常運作，請使用名稱以 `AWSGlueServiceRole` 字串開頭的角色。如果您選擇使用不同角色名稱，則必須新增 `iam:passrole` 許可，並設定 IAM 中角色的政策。如需詳細資訊，請參閱 為未命名為 "AWSGlueServiceRole*" 的角色建立 IAM 政策。

> **Warning**
>
> 如果角色為基本授予 `iam:passrole` 許可，並且您實作角色鏈結，則使用者可能會無意中取得筆記本的存取權。目前沒有已實作的稽核，稽核可讓您監控哪些使用者已被授予筆記本的存取權。

如果您想拒絕給予 IAM 身分建立資料預覽工作階段的能力，請參閱下列範例 the section called “拒絕給予身分建立資料預覽工作階段的能力”。

Amazon CloudWatch 許可

您可以使用 Amazon CloudWatch 監控您的 AWS Glue Studio 任務，其會收集來自 AWS Glue 的原始資料，並處理為可讀且近乎即時的指標。依預設，系統會自動將 AWS Glue 指標資料傳送至 CloudWatch。如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的什麼是 Amazon CloudWatch？，以及AWS Glue 開發人員指南中的 AWS Glue 指標。

若要存取 CloudWatch 儀表板，存取 AWS Glue Studio 的使用者需要以下其中一個項目：

- `AdministratorAccess` 政策
- `CloudWatchFullAccess` 政策
- 自訂政策，其中包含一或多個特定許可：
  - `cloudwatch:GetDashboard` 和 `cloudwatch:ListDashboards` 以檢視儀表板
  - `cloudwatch:PutDashboard` 以建立或修改儀表板
  - `cloudwatch:DeleteDashboards` 以刪除儀表板

如需使用政策變更 IAM 使用者許可的詳細資訊，請參閱 IAM 使用者指南中的變更 IAM 使用者的許可。

檢閱 ETL 任務所需的 IAM 許可

當您使用 AWS Glue Studio 建立任務時，任務會承擔您在建立時指定之 IAM 角色的許可。這個 IAM 角色必須具有許可，才能從您的資料來源擷取資料、將資料寫入至目標，以及存取 AWS Glue 資源。

您為任務建立的角色名稱必須以字串 `AWSGlueServiceRole` 開頭，AWS Glue Studio 才能正確使用。例如，您可以將角色命名為 `AWSGlueServiceRole-FlightDataJob`。

資料來源和資料目標許可

對於您用於任務中的任何來源、目標、指令碼和臨時目錄，AWS Glue Studio 任務必須可以存取 Amazon S3。您可以建立政策，對特定 Amazon S3 資源提供精細存取。

- 資料來源需要 `s3:ListBucket` 和 `s3:GetObject` 許可。
- 資料目標需要 `s3:ListBucket`、`s3:PutObject` 和 `s3:DeleteObject` 許可。

如果您選擇 Amazon Redshift 做為資料來源，您可以為叢集許可提供角色。針對 Amazon Redshift 叢集執行的任務發出使用臨時憑證存取 Amazon S3 的暫時儲存命令。如果您的任務執行超過一小時，這些憑證將會過期，導致任務失敗。若要避免此問題，您可以將角色指派給 Amazon Redshift 叢集本身，可將必要的許可授予使用臨時憑證的任務。如需詳細資訊，請參閱 AWS Glue 開發人員指南中的將資料移入及移出 Amazon Redshift。

如果任務使用 Amazon S3 以外的資料來源或目標，則必須將必要的許可連接到任務所使用的 IAM 角色以存取這些資料來源和目標。如需詳細資訊，請參閱 AWS Glue 開發人員指南中的設定您的環境，以存取資料存放區。

如果您為資料存放區使用連接器和連線，則需要其他許可，如the section called “使用連接器所需的許可”中所述。

刪除任務所需的許可

在 AWS Glue Studio 中，您可以在主控台中選取多個任務來刪除。若要執行此動作，您必須具有 `glue:BatchDeleteJob` 許可。這有別於 AWS Glue 主控台，在此處需要 `glue:DeleteJob` 許可才能刪除任務。

AWS Key Management Service 許可

如果您計劃存取使用搭配 AWS Key Management Service (AWS KMS) 伺服器端加密的 Amazon S3 來源和目標，則將政策連接至任務所使用的 AWS Glue Studio 角色，以讓任務解密資料。任務角色需要 `kms:ReEncrypt`、`kms:GenerateDataKey` 以及 `kms:DescribeKey` 許可。此外，任務角色需要`kms:Decrypt`許可，以上傳或下載使用 AWS KMS 客戶主金鑰 (CMK) 加密的 Amazon S3 物件。

使用 AWS KMS CMK 需另外付費。如需詳細資訊，請參閱 AWS Key Management Service 開發人員指南中的 AWS Key Management Service 概念 - 客戶主金鑰 (CMK)和 AWS Key Management Service 定價。

使用連接器所需的許可

如果您使用 AWS Glue Custom Connector 和連線來存取資料存放區，則用於執行 AWS Glue ETL 任務的角色需要連接額外的許可：

- 用於存取從 AWS Marketplace 購買之連接器的 AWS 受管政策 `AmazonEC2ContainerRegistryReadOnly`。
- `glue:GetJob` 和 `glue:GetJobs` 許可。
- 存取與連線搭配使用之秘密的 AWS Secrets Manager 許可。請參閱 Example: Permission to retrieve secret values (範例：擷取秘密值的許可) 以了解 IAM 政策範例。

如果您的 AWS Glue ETL 任務在執行 Amazon VPC 的 VPC 內執行，則必須按照the section called “為您的 ETL 任務設定 VPC”所述設定 VPC。

設定 AWS Glue Studio 的 IAM 許可

您可以建立角色，並將政策指派給使用者和任務角色，方法是使用 AWS 管理員使用者。

您可以使用 AWSGlueConsoleFullAccess AWS 受管政策，以提供使用 AWS Glue Studio 主控台所需的必須許可。

若要建立您自己的政策，請遵循 AWS Glue 開發人員指南中的建立適用於 AWS Glue 服務的 IAM 政策。包含先前 檢閱 AWS Glue Studio 使用者所需的 IAM 許可 中描述的 IAM 許可。

主題

- 將政策連接至 AWS Glue Studio 使用者。
- 為未命名為 "AWSGlueServiceRole*" 的角色建立 IAM 政策

將政策連接至 AWS Glue Studio 使用者。

登入 AWS Glue Studio 主控台的任何 AWS 使用者必須具有存取特定資源的許可。您可透過將 IAM 政策指派給使用者來提供這些許可。

將 AWSGlueConsoleFullAccess 受管政策連接至使用者

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在導覽窗格中，選擇 政策 。

3. 在政策清單中，選取 AWSGlueConsoleFullAccess 旁的核取方塊。您可用篩選功能表和搜尋方塊來篩選政策清單。
4. 選擇政策動作，再選擇附加。
5. 選擇要附加政策的使用者。您可用篩選功能表和搜尋方塊來篩選主體實體清單。選擇要附加的使用者後，選擇附加政策。
6. 視需要重複上述步驟，將其他政策連接至使用者。

為未命名為 "AWSGlueServiceRole*" 的角色建立 IAM 政策

為 AWS Glue Studio 使用的角色設定 IAM 政策

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 新增新的 IAM 政策。您可以新增至現有政策，或建立新的 IAM 內嵌政策。建立 IAM 政策：
   1. 選擇政策，然後選擇建立政策。出現開始使用按鈕時，選擇它，然後選擇建立政策。
   2. 在建立您自己的政策旁邊，選擇選取。
   3. 針對政策名稱，輸入您稍後容易參考的任何值。(選用) 在說明中輸入描述性文字。
   4. 針對政策文件，使用下列格式輸入政策文件，然後選擇建立政策：
3. 將下列區塊複製並貼入「陳述式」陣列下的政策中，並將 *my-interactive-session-role-prefix* 取代為要與 AWS Glue 的許可相關聯的所有常用角色的字首。

```
{
    "Action": [
        "iam:PassRole"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/my-interactive-session-role-prefix*",
    "Condition": {
        "StringLike": {
            "iam:PassedToService": [
                "glue.amazonaws.com "
            ]
        }
    }
}
```

以下是政策中包含的版本和陳述式陣列的完整範例

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Action": [
            "iam:PassRole"
        ],
        "Effect": "Allow",
        "Resource": "arn:aws:iam::*:role/my-interactive-session-role-prefix*",
        "Condition": {
            "StringLike": {
                "iam:PassedToService": [
                    "glue.amazonaws.com "
                ]
            }
        }
    }
  ]
}
```

4. 若要為使用者啟用政策，請選擇使用者。
5. 選擇要連接政策的 使用者。

為您的 ETL 任務設定 VPC

您可使用 Amazon Virtual Private Cloud (Amazon VPC) 在 AWS 雲端 內的邏輯隔離區域中定義虛擬網路，也就是虛擬私有雲端 (VPC)。您可以在您的 VPC 中啟動 AWS 資源 (例如執行個體)。VPC 近似於您在自有資料中心內運作的傳統網路，卻能提供 AWS 可擴展性基礎設施的效益。您可以設定您的 VPC；您可以選取其 IP 地址範圍、建立子網，以及設定路由表、網路閘道與安全設定。您可以將 VPC 中的執行個體連線至網際網路。您可以將 VPC 連線至自己的公司資料中心，讓 AWS 雲端 成為您資料中心的延伸。為了保護各個子網路的資源，您可使用多個安全性層級，包括安全群組及網路存取控制清單。如需詳細資訊，請參閱 Amazon VPC 使用者指南。

您可以設定您的 AWS Glue ETL 任務在使用連接器時在 VPC 內執行。您必須根據需要為下列項目設定 VPC：

- 不在 AWS 中之資料存放區的公有網路存取。任務可存取的所有資料存放區必須可從 VPC 子網路使用。

- 如果您的任務需要同時存取 VPC 資源和公有網際網路，則 VPC 需要在 VPC 中有一個網路位址轉譯 (NAT) 閘道。

  如需詳細資訊，請參閱 AWS Glue 開發人員指南中的設定您的環境，以存取資料存放區。

AWS Glue Studio 中的筆記本入門

透過 AWS Glue Studio 啟動筆記本，所有的設定步驟都會自動完成，以便您在幾秒鐘之後探索資料並開始開發任務指令碼。

下列各節介紹如何建立角色並授予適當許可，以在 AWS Glue Studio 中將筆記本用於 ETL 任務。

主題

- 授予 IAM 角色的許可

授予 IAM 角色的許可

設定 AWS Glue Studio 是使用筆記本的先決條件。

若要在 AWS Glue 中使用筆記本，您的角色需要滿足下列條件：

- 與 AWS Glue 之間的信任關係，用於 `sts:AssumeRole` 動作；若您想標記，則為 `sts:TagSession` 動作。
- IAM 政策，其中包含筆記本、AWS Glue 和互動式工作階段的所有 API 操作。
- 傳遞角色的 IAM 政策，原因是角色需要能將自身從筆記本傳遞至互動式工作階段。

例如，當您建立新角色時，可以新增標準 AWS 受管政策 (如 `AWSGlueConsoleFullAccessRole`) 至該角色，然後新增筆記本操作的新政策和 IAM PassRole 政策。

與 AWS Glue 之間信任關係的所需操作

在開始筆記本工作階段時，您必須將 `sts:AssumeRole` 新增至已傳遞至筆記本的角色信任關係。如果您的工作階段包含標籤，則您也必須傳遞 `sts:TagSession` 動作。如果沒有這些動作，將無法開始筆記本工作階段。

例如：

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

包含筆記本的 API 操作的政策

下列範例政策描述筆記本的所需 AWS IAM 許可。如果您要建立新角色，請建立包含下列項目的政策：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:StartNotebook",
                "glue:TerminateNotebook",
                "glue:GlueNotebookRefreshCredentials",
                "glue:DeregisterDataPreview",
                "glue:GetNotebookInstanceStatus",
                "glue:GlueNotebookAuthorize"
            ],
            "Resource": "*"
        }
    ]
}
```

您可以使用以下 IAM 政策來允許存取特定資源：

- AwsGlueSessionUserRestrictedNotebookServiceRole：提供存取所有 AWS Glue 資源的完整權限 (工作階段除外)。允許使用者僅建立並使用與使用者相關聯的筆記本工作階段。此政策也包含 AWS Glue 管理其他 AWS 服務中 AWS Glue 資源所需的其他許可。

- AwsGlueSessionUserRestrictedNotebookPolicy：提供允許使用者僅建立和使用與該使用者相關聯的筆記本工作階段的許可。此政策也包括明確允許使用者傳遞受限制 AWS Glue 工作階段角色的許可。

傳遞角色的 IAM 政策

在您使用角色建立筆記本時，系統會將該角色傳遞至互動式工作階段，以便在兩個位置皆可使用同一個角色。因此，iam:PassRole 許可需要成為角色政策的一部分。

使用下列範例為角色建立新政策。用您的帳號與角色名稱取代範例中的值。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::090000000210:role/<role_name>"
        }
    ]
}
```

設定 AWS Glue 使用情況設定檔

使用雲平台的主要優點之一是它的靈活性。不過，如此輕鬆建立運算資源，就會在不受管理且沒有護欄的情況下，雲端成本上漲的風險。因此，管理員需要在避免高額基礎設施成本之間取得平衡，同時允許用戶工作而不會受到不必要的摩擦。

透過 AWS Glue 使用情況設定檔，管理員可以為帳戶內的各種使用者類別 (例如開發人員、測試人員和產品團隊) 建立不同的設定檔。每個設定檔都是一組唯一的參數，可以指派給不同類型的使用者。例如，開發人員可能需要更多的工作人員，並且可以擁有更多的最大工作人員數量，而產品團隊可能需要較少的工作人員和較低的逾時或閒置逾時值。

工作和工作執行行為的範例

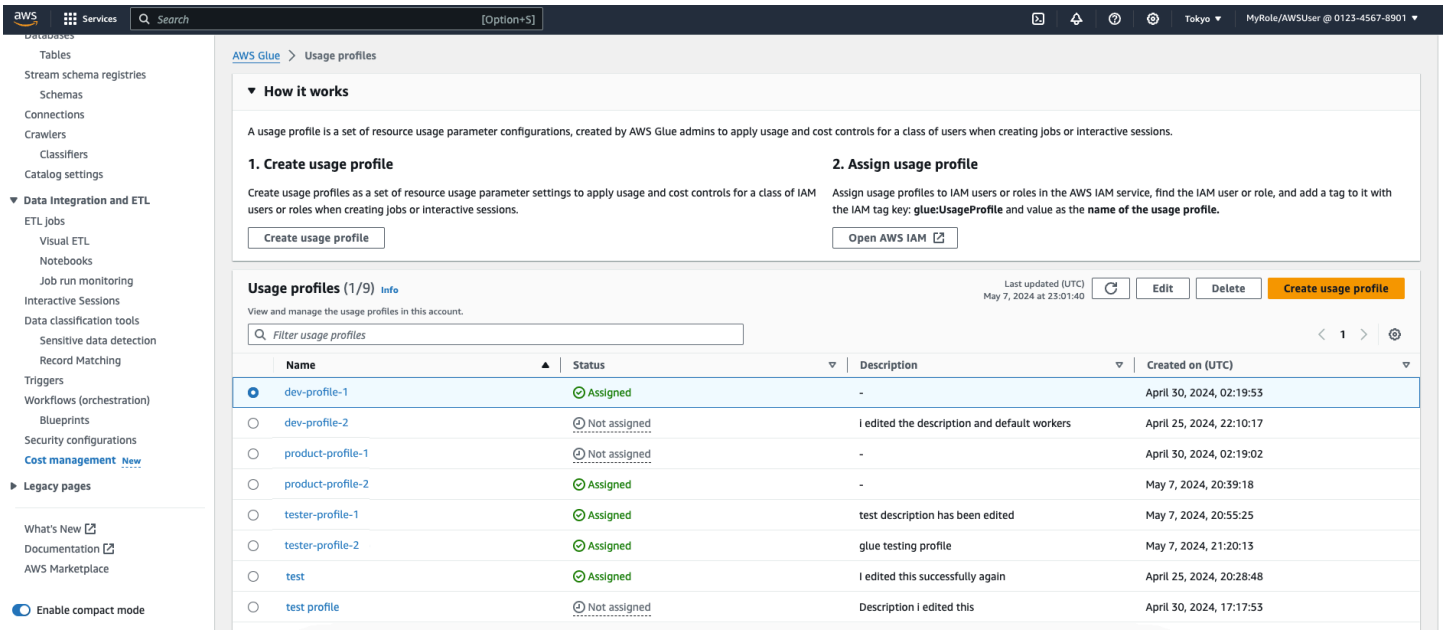
假設工作是由使用者 A 與設定檔 A 建立的。工作會以某些參數值儲存。具有設定檔 B 的使用者 B 將嘗試執行工作。

當用戶 A 編寫作業時，如果他沒有設置特定數量的 Worker，則應用了用戶 A 配置文件中的默認設置，並與作業的定義一起保存。

當用戶 B 運行作業時，它會使用為其保存的任何值運行。如果使用者 B 自己的設定檔限制較嚴格，且不允許使用該許多 Worker 執行，則作業執行將會失敗。

使用情況設定檔作為資源

AWS Glue 使用情況設定檔是以 Amazon 資源名稱 (ARN) 識別的資源。所有預設的 IAM (Identity and Access Management) 控制都適用，包括以動作為基礎和資源型授權。管理員應更新建立 AWS Glue 資源之使用者的 IAM 政策，授予他們使用設定檔的存取權限。



主題

- 建立和管理使用量設定檔
- 使用設定檔和工作

建立和管理使用量設定檔

建立 AWS Glue 使用情況設定檔

管理員應該創建使用情況配置文件，然後將其分配給各種用戶。建立使用情況設定檔時，您可以為各種工作和階段作業參數指定預設值以及允許的值範圍。您必須為工作或互動式工作階段設定至少一個參數。如果使用者在使用此設定檔時提供參數值，您可以自訂未為工作提供參數值時要使用的預設值，和/或設定範圍限制或一組允許的值以進行驗證。

預設值是由管理員設定協助工作作者的最佳做法。當使用者建立新作業且未設定逾時值時，將套用使用設定檔的預設逾時。如果作者沒有設定檔，則會套用 AWS Glue 服務預設值，並儲存在工作的定義中。在執行階段， AWS Glue 強制執行設定檔中設定的限制 (最小值、最大值、允許的 Worker)。

一旦配置了參數，所有其他參數都是可選的。可針對工作或互動式工作階段自訂的參數如下：

- 工作程式數量 — 限制 Worker 的數量，以避免過度使用運算資源。您可以設定預設值、最小值和最大值。最小值為 1。
- Worker 類型 — 限制工作負載的相關 Worker 類型。您可以為使用者設定檔設定預設類型並允許 Worker 類型。
- 逾時 — 定義工作或互動式工作階段在終止前可以執行和耗用資源的時間上限。設定逾時值以避免長時間執行的工作。

  您可以設定預設值、最小值和最大值 (以分鐘為單位)。最小值為 1 (分鐘)。雖然 AWS Glue 預設逾時為 2880 分鐘，但您可以在使用情況設定檔中設定任何預設值。

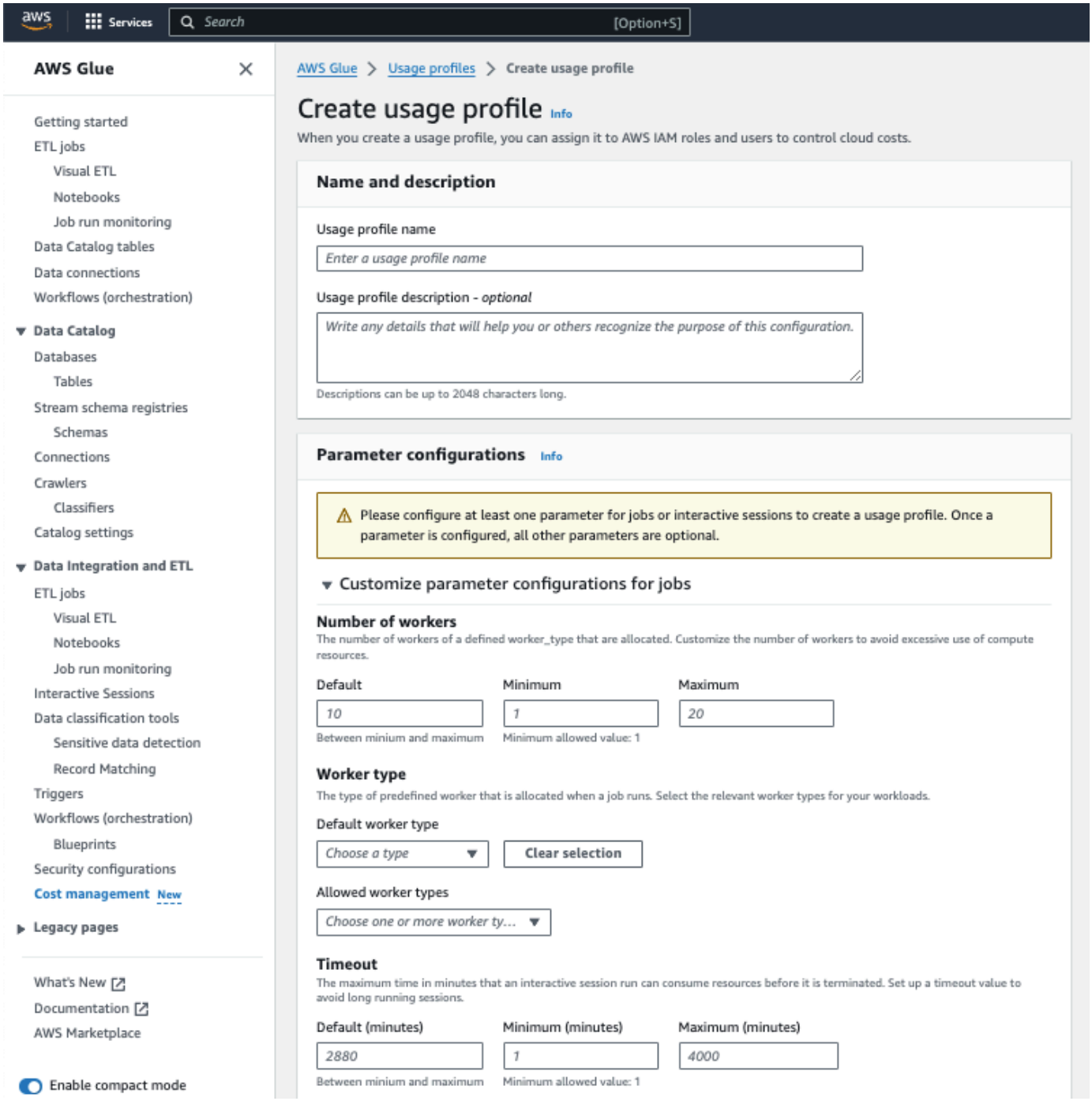
  為「默認」設置值是最佳實踐。如果使用者未設定任何值，則此值將用於建立工作或階段作業。
- 閒置逾時 — 定義在執行儲存格之後，互動式工作階段在逾時之前處於非作用中狀態的分鐘數。定義互動式工作階段在工作完成後終止的閒置逾時。閒置逾時範圍應在逾時限制內。

  您可以設定預設值、最小值和最大值 (以分鐘為單位)。最小值為 1 (分鐘)。雖然 AWS Glue 預設逾時為 2880 分鐘，但您可以在使用情況設定檔中設定任何預設值。

  為「默認」設置值是最佳實踐。如果使用者未設定任何值，則此值將用於建立工作階段。

以管理員身分建立 AWS Glue 使用情況設定檔 (主控台)

1. 在左側導覽功能表中，選擇 [成本管理]。
2. 選擇 [建立使用設定檔]
3. 輸入使用情況設定檔的「使用」設定檔名稱。
4. 輸入可選描述，以幫助其他人識別使用情況設定檔的目的。
5. 在輪廓中至少定義一個參數。表單中的任何欄位都是參數。例如，工作階段閒置逾時最小值。
6. 定義任何套用至使用情況設定檔的選用標籤。
7. 選擇儲存。



若要建立使用情況設定檔 (AWS CLI)

1. 輸入以下命令。

```
aws glue create-usage-profile --name profile-name --configuration file://
config.json --tags list-of-tags
```

其中 config.json 可以為交互式會話（SessionConfiguration）和作業（ ）定義參數值：JobConfiguration

```
//config.json (There is a separate blob for session/job configuration
{
    "SessionConfiguration": {
        "timeout": {
            "DefaultValue": "2880",
            "MinValue": "100",
            "MaxValue": "4000"
        },
        "idleTimeout": {
            "DefaultValue": "30",
            "MinValue": "10",
            "MaxValue": "4000"
        },
        "workerType": {
            "DefaultValue": "G.2X",
            "AllowedValues": [
                "G.2X",
                "G.4X",
                "G.8X"
            ]
        },
        "numberOfWorkers": {
            "DefaultValue": "10",
            "MinValue": "1",
            "MaxValue": "10"
        }
    },
    "JobConfiguration": {
        "timeout": {
            "DefaultValue": "2880",
            "MinValue": "100",
            "MaxValue": "4000"
        },
        "workerType": {
            "DefaultValue": "G.2X",
            "AllowedValues": [
                "G.2X",
                "G.4X",
                "G.8X"
```

```
            ]
        },
        "numberOfWorkers": {
            "DefaultValue": "10",
            "MinValue": "1",
            "MaxValue": "10"
        }
    }
}
```

2. 輸入下列指令以查看建立的使用情況設定檔：

```
aws glue get-usage-profile --name profile-name
```

回應：

```
{
    "ProfileName": "foo",
    "Configuration": {
        "SessionConfiguration": {
            "numberOfWorkers": {
                "DefaultValue": "10",
                "MinValue": "1",
                "MaxValue": "10"
            },
            "workerType": {
                "DefaultValue": "G.2X",
                "AllowedValues": [
                    "G.2X",
                    "G.4X",
                    "G.8X"
                ]
            },
            "timeout": {
                "DefaultValue": "2880",
                "MinValue": "100",
                "MaxValue": "4000"
            },
            "idleTimeout": {
                "DefaultValue": "30",
                "MinValue": "10",
                "MaxValue": "4000"
            }
```

```
        },
        "JobConfiguration": {
            "numberOfWorkers": {
                "DefaultValue": "10",
                "MinValue": "1",
                "MaxValue": "10"
            },
            "workerType": {
                "DefaultValue": "G.2X",
                "AllowedValues": [
                    "G.2X",
                    "G.4X",
                    "G.8X"
                ]
            },
            "timeout": {
                "DefaultValue": "2880",
                "MinValue": "100",
                "MaxValue": "4000"
            }
        }
    },
    "CreatedOn": "2024-01-19T23:15:24.542000+00:00"
}
```

用於管理使用情況設定檔的其他 CLI 命令：

- AWS 膠水 list-usage-profiles
- *AWS ## update-usage-profile -##########-#####//config. json*
- *aws ## delete-usage-profile -#######*

編輯使用情況設定檔

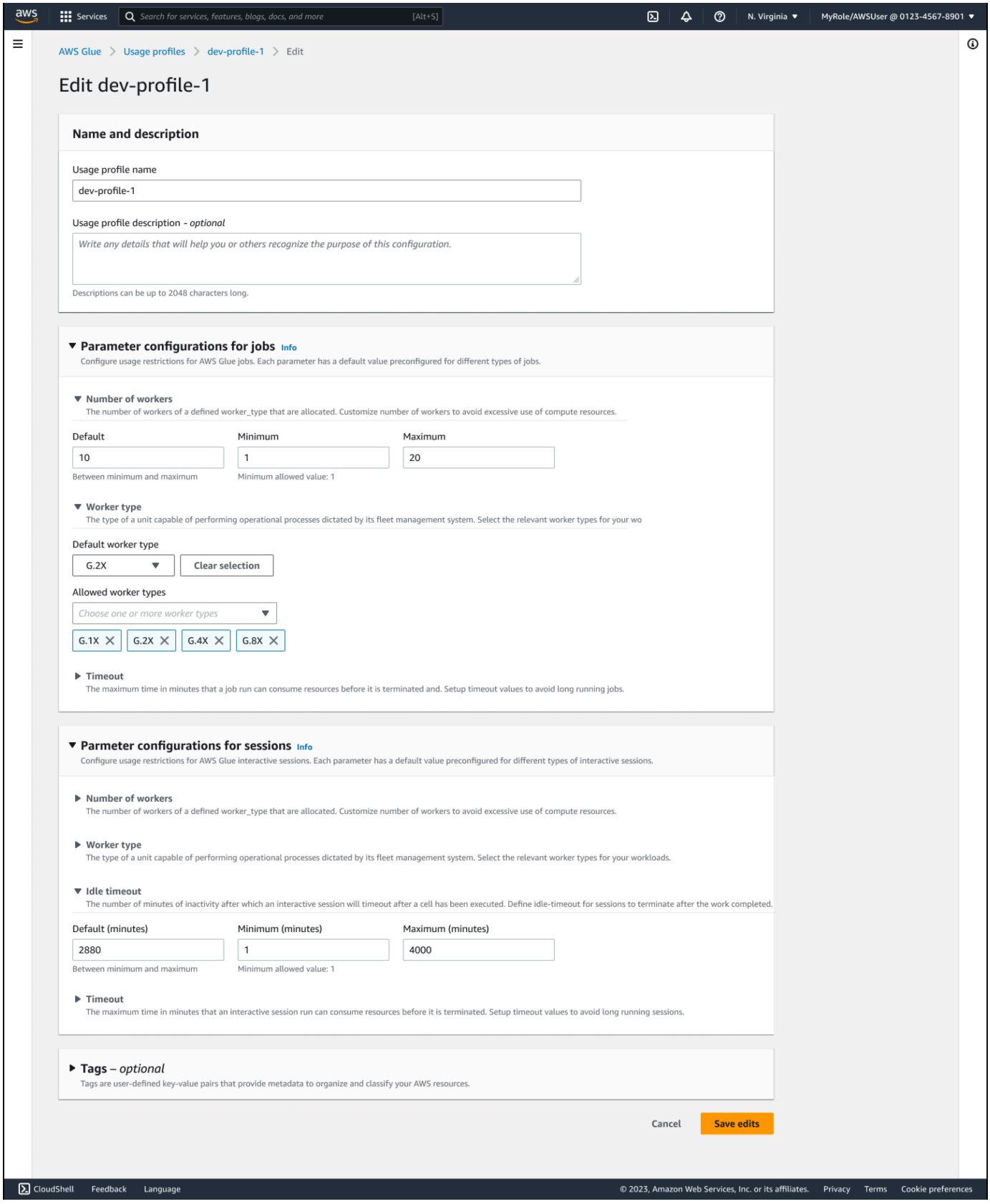
管理員可以編輯已建立的使用情況設定檔，以變更工作和互動式工作階段的設定檔參數值。

若要編輯使用情況設定檔：

以管理員身分編輯 AWS Glue 使用情況設定檔 (主控台)

1. 在左側導覽功能表中，選擇 [成本管理]。

2. 選擇您具有編輯權限的使用情況設定檔，然後選擇 [編輯]。
3. 視需要對設定檔進行變更。依預設，會展開已有值的參數。
4. 選擇「儲存編輯」。


Services
Search for services, features, blogs, docs, and more [Alt+S]
N. Virginia
MyRole/AWSUser @ 0123-4567-8901
AWS Glue > Usage profiles > dev-profile-1 > Edit
Edit dev-profile-1
Name and description
Usage profile name
dev-profile-1
Usage profile description - optional
Write any details that will help you or others recognize the purpose of this configuration.
Descriptions can be up to 2048 characters long.
Parameter configurations for jobs Info
Configure usage restrictions for AWS Glue jobs. Each parameter has a default value preconfigured for different types of jobs.
Number of workers
The number of workers of a defined worker_type that are allocated. Customize number of workers to avoid excessive use of compute resources.
Default
10
Between minimum and maximum
Minimum
1
Minimum allowed value: 1
Maximum
20
Worker type
The type of a unit capable of performing operational processes dictated by its fleet management system. Select the relevant worker types for your wo
Default worker type
G.2X
Clear selection
Allowed worker types
Choose one or more worker types
G.1X
G.2X
G.4X
G.8X
Timeout
The maximum time in minutes that a job run can consume resources before it is terminated and. Setup timeout values to avoid long running jobs.
Parmeter configurations for sessions Info
Configure usage restrictions for AWS Glue interactive sessions. Each parameter has a default value preconfigured for different types of interactive sessions.
Number of workers
The number of workers of a defined worker_type that are allocated. Customize number of workers to avoid excessive use of compute resources.
Worker type
The type of a unit capable of performing operational processes dictated by its fleet management system. Select the relevant worker types for your workloads.
Idle timeout
The number of minutes of inactivity after which an interactive session will timeout after a cell has been executed. Define idle-timeout for sessions to terminate after the work completed.
Default (minutes)
2880
Between minimum and maximum
Minimum (minutes)
1
Minimum allowed value: 1
Maximum (minutes)
4000
Timeout
The maximum time in minutes that an interactive session run can consume resources before it is terminated. Setup timeout values to avoid long running sessions.
Tags – optional
Tags are user-defined key-value pairs that provide metadata to organize and classify your AWS resources.
Cancel
Save edits
CloudShell
Feedback
Language
© 2023, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

若要編輯使用情況設定檔 (AWS CLI)

- 輸入以下命令。使用相同的`--configuration`文件語法如上所示的 create 命令。

```
aws glue update-usage-profile --name profile-name --configuration file://
config.json
```

其中 config.json 定義了交互式會話（`SessionConfiguration`）和作業（ ）的參數值：`JobConfiguration`

指派使用情況設定檔

「使用情況」設定檔頁面中的「使用率狀態」欄會顯示使用情況設定檔是否已指派給使用 將游標暫留在狀態上會顯示指派的 IAM 實體。

管理員可以將 AWS Glue 使用情況設定檔指派給建立 AWS Glue 資源的使用者/角色。指派設定檔是兩個動作的組合：

- 然後使用`glue:UsageProfile`密鑰更新 IAM 用戶/角色標籤
- 更新使用者/角色的IAM 政策。

對於使用 AWS Glue Studio 建立工作/互動式工作階段的使用者，管理員會標記下列角色：

- 對於工作的限制，管理員將登錄的控制台角色標記
- 對於互動式工作階段的限制，管理員標記使用者在建立記事本時提供的角色

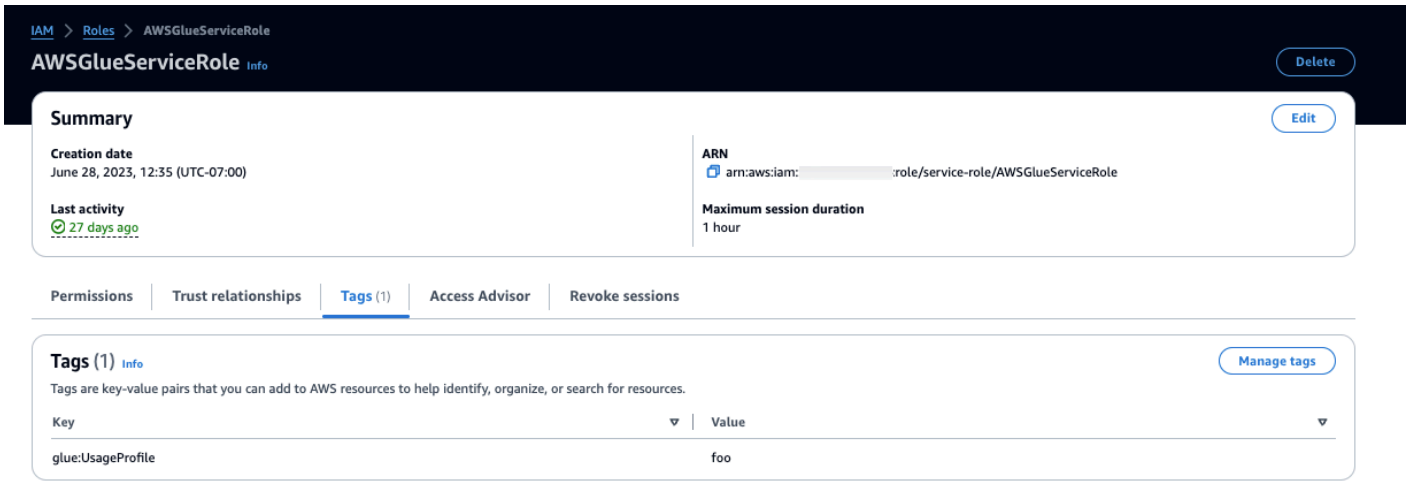
以下是管理員在建 AWS Glue 立資源的 IAM 使用者/角色上需要更新的範例政策：

```
{
    "Effect": "Allow",
    "Action": [
        "glue:GetUsageProfile"
    ],
    "Resource": [
        "arn:aws:glue:us-east-1:123456789012:usageProfile/foo"
    ]
}
```

AWS Glue 根據使用情況設定檔中指定的值來驗證工作、工作執行和工作階段要求，如果不允許請求，則會引發例外狀 AWS Glue 況。對於同步 API，將向用戶拋出錯誤。對於非同步路徑，會建立失敗的作業執行，並顯示錯誤訊息，指出輸入參數超出使用者/角色指派設定檔的允許範圍。

若要將使用情況設定檔指派給使用者/角色：

1. 開啟 (Identity and Access Management) IAM 主控台。
2. 在左側導覽列中，選擇 [使用者] 或 [角色]。
3. 選擇使用者或角色。
4. 選擇 Tags (標籤) 索引標籤。
5. 選擇「新增標籤」
6. 新增標籤，其中包含使用情況設定檔名稱的「鍵」`glue:UsageProfile` 和「值」。
7. 選擇 Save changes (儲存變更)



檢視指派的使用情況設定

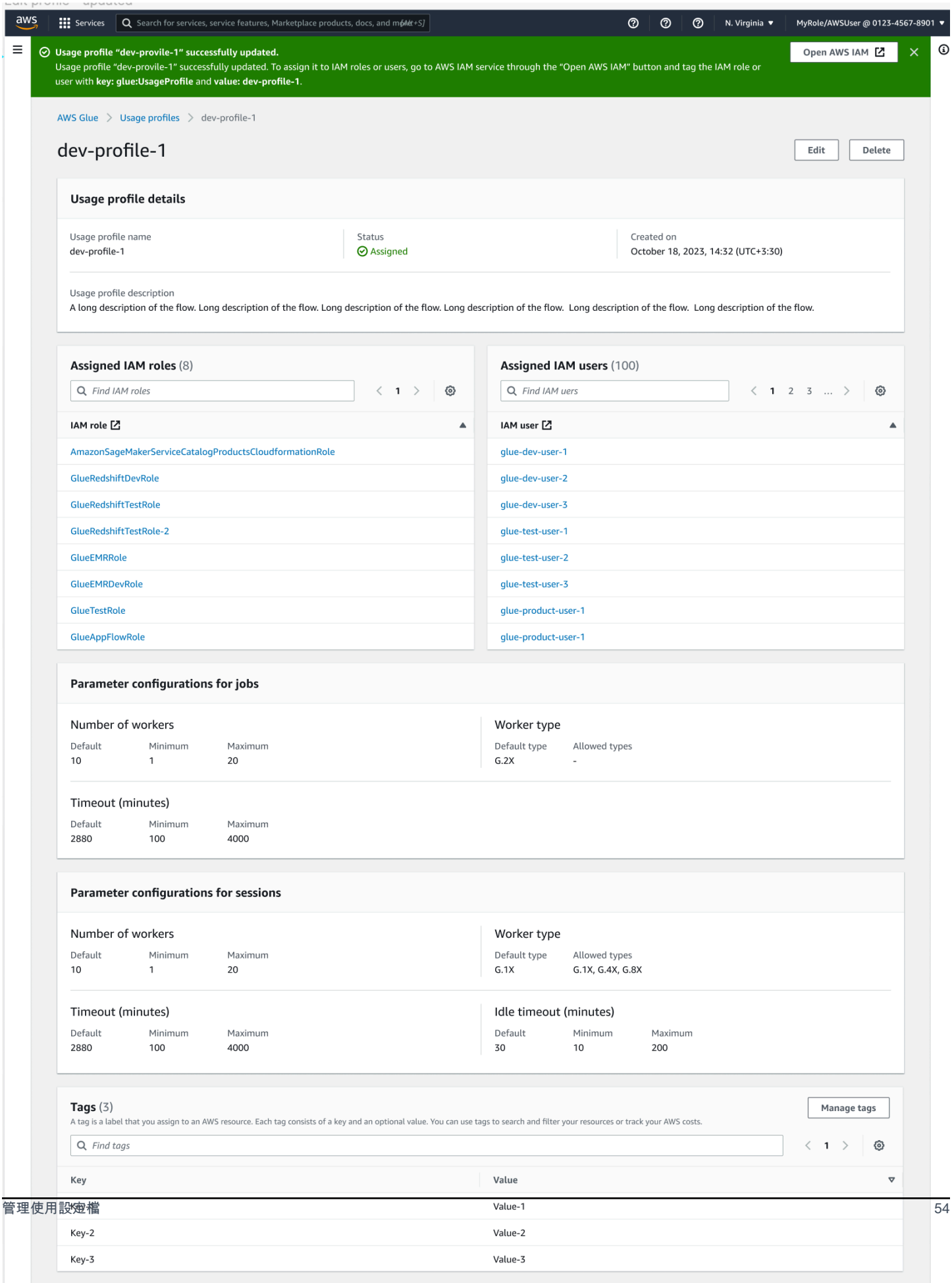
使用者可以檢視其指派的使用情況設定檔，並在進行 API 呼叫以建立 AWS Glue 工作和工作階段資源或開始工作時使用這些設定檔。

IAM 政策中提供了設定檔許可。只要呼叫者原則具有`glue:UsageProfile`權限，使用者就可以看到設定檔。否則，您將收到拒絕訪問錯誤。

若要檢視指派的使用情況設定檔：

1. 在左側導覽功能表中，選擇 [成本管理]。

2. 選擇您有權檢視的使用情況設定檔。


Services
Search for services, service features, Marketplace products, docs, and more [Alt+S]
N. Virginia
MyRole/AWSUser @ 0123-4567-8901
Usage profile "dev-provile-1" successfully updated.
Usage profile "dev-provile-1" successfully updated. To assign it to IAM roles or users, go to AWS IAM service through the "Open AWS IAM" button and tag the IAM role or user with key: glue:UsageProfile and value: dev-profile-1.
Open AWS IAM
AWS Glue > Usage profiles > dev-profile-1
dev-profile-1
Edit
Delete
Usage profile details
Usage profile name
dev-profile-1
Status
Assigned
Created on
October 18, 2023, 14:32 (UTC+3:30)
Usage profile description
A long description of the flow. Long description of the flow. Long description of the flow. Long description of the flow. Long description of the flow. Long description of the flow.
Assigned IAM roles (8)
Find IAM roles
IAM role
AmazonSageMakerServiceCatalogProductsCloudformationRole
GlueRedshiftDevRole
GlueRedshiftTestRole
GlueRedshiftTestRole-2
GlueEMRRole
GlueEMRDevRole
GlueTestRole
GlueAppFlowRole
Assigned IAM users (100)
Find IAM uers
IAM user
glue-dev-user-1
glue-dev-user-2
glue-dev-user-3
glue-test-user-1
glue-test-user-2
glue-test-user-3
glue-product-user-1
glue-product-user-1
Parameter configurations for jobs
Number of workers
Default 10
Minimum 1
Maximum 20
Worker type
Default type G.2X
Allowed types -
Timeout (minutes)
Default 2880
Minimum 100
Maximum 4000
Parameter configurations for sessions
Number of workers
Default 10
Minimum 1
Maximum 20
Worker type
Default type G.1X
Allowed types G.1X, G.4X, G.8X
Timeout (minutes)
Default 2880
Minimum 100
Maximum 4000
Idle timeout (minutes)
Default 30
Minimum 10
Maximum 200
Tags (3)
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.
Manage tags
Find tags
Key
Value
Key-1
Value-1
Key-2
Value-2
Key-3
Value-3

使用設定檔和工作

使用使用設定檔編寫工作

編寫工作時，將套用在使用情況設定檔中設定的限制和預設值。您的設定檔將在儲存時指派給工作。

使用使用設定檔執行工作

當您開始工作執行時，會 AWS Glue 強制執行呼叫者設定檔中設定的限制。如果沒有直接呼叫者，Glue 就會套用其作者指派給工作的設定檔中的限制。

> **Note**
>
> 按照排程執行工作時 (依工作 AWS Glue 流程或 AWS Glue 觸發器)，指派給作者將套用的工作設定檔。
> 當工作由外部服務（Step Functions，MWAA）或 `StartJobRun` API 運行時，將強制執行呼叫者的配置文件限制。

對於工作 AWS Glue 流程或 AWS Glue 觸發程序：需要更新預先存在的工作以儲存新的設定檔名稱，以便在執行時期針對排定的執行強制設定檔的限制 (最小、上限和允許的 Worker)。

檢視指派給工作的使用情況設定檔

若要檢視指派給工作的設定檔 (將在執行階段與排定的工作 AWS Glue 流程或 AWS Glue 觸發器搭配使用)，您可以查看工作詳細資訊標籤。您也可以在工作執行詳細資料索引標籤中查看過去執行中使用的設定檔。

更新或刪除附加至工作的使用情況設定檔

指派給工作的設定檔會在更新時變更。如果未指派作者使用情況設定檔，先前附加至該工作的任何描述檔都會從該作業中移除。

AWS Glue Data Catalog入門

AWS Glue Data Catalog 是您持久性技術中繼資料的存放區。這是一項受管服務，您可以使用此服務在 AWS 雲端中存放、標註和共享中繼資料。如需詳細資訊，請參閱 AWS Glue Data Catalog。

最近更新了AWS Glue主控台和部分使用者介面。

概要

您可以使用此教學來建立您的第一個 AWS Glue Data Catalog，而此目錄使用 Amazon S3 儲存貯體作為資料來源。

在本教學課程中，您將使用 AWS Glue 主控台執行下列操作：

1. 建立資料庫
2. 建立資料表
3. 使用 Amazon S3 儲存貯體作為資料來源

在完成這些步驟之後，您將成功地使用 Amazon S3 儲存貯體作為資料來源來填入 AWS Glue Data Catalog。

步驟 1：建立資料庫

若要開始使用，請登入AWS Management Console並開啟主AWS Glue控台。

使用 AWS Glue 主控台建立資料庫：

1. 在 AWS Glue 主控台中，從左側選單中，選擇 Data catalog (資料型錄) 下的 Databases (資料庫)。
2. 選擇 Add database (新增資料庫)。
3. 在「建立資料庫」頁面中輸入資料庫的名稱。在位置 - 選用區段中，設定 URI 位置以供資料型錄的用戶端使用。如果您不知道此資訊，則可以繼續建立資料庫。
4. (選用)。輸入資料庫的說明。
5. 選擇建立資料庫。

恭喜您，您成功地使用 AWS Glue 主控台設定了第一個資料庫。您的新資料庫將會顯示在可用資料庫清單中。您可以透過從 Databases (資料庫) 儀表板中選擇資料庫的名稱，來編輯資料庫。

後續步驟

建立資料庫的其他方法：

您剛剛使用 AWS Glue 主控台建立了資料庫，不過還有其他方法可以建立資料庫：

- 您可以使用爬蟲程式來自動為您建立資料庫和資料表。要使用爬蟲程式設定資料庫，請參閱在 AWS Glue 主控台上使用爬蟲程式。

- 您可以使用 AWS CloudFormation 範本。請參閱使用 AWS Glue Data Catalog 範本建立 AWS Glue 資源。
- 您也可以使用 AWS Glue 資料庫 API 操作建立資料庫。

  若要使用 `create` 操作建立資料庫，可透過包括 `DatabaseInput` (必要) 參數來建構請求。

  例如：

  以下範例說明了如何使用 CLI、Boto3 或 DDL 根據您在教學中使用的 S3 儲存貯體中的相同 flights_data.csv 檔案定義資料表。

  CLI

  ```
  aws glue create-database --database-input "{\"Name\":\"clidb\"}"
  ```

  Boto3

  ```
  glueClient = boto3.client('glue')

  response = glueClient.create_database(
      DatabaseInput={
          'Name': 'boto3db'
      }
  )
  ```

如需有關資料庫 API 資料類型、結構和操作的詳細資訊，請參閱資料庫 API。

後續步驟

在下一節中，您將建立資料表，並將該資料表新增至您的資料庫。

您也可以探索 Data Catalog 的設定和許可。請參閱使用 AWS Glue 主控台上的 Data Catalog 設定。

步驟 2. 建立資料表

在此步驟中，可使用 AWS Glue 主控台來建立資料表。

1. 在 AWS Glue 主控台的左側選單中，選擇 Tables (資料表)。

2. 選擇 Add table (新增資料表)。
3. 在 Table details (資料表詳細資訊) 中，輸入資料表的名稱以設定資料表的屬性。
4. 在 Database (資料庫) 區段中，從下拉式選單選擇您在步驟 1 中建立的資料庫。
5. 在 Add a data store (新增資料存放區) 區段中，預設將選取 S3 作為來源類型。
6. 對於 Data is located in (資料位置)，選擇 Specified path in another account (其他帳戶中的已指定路徑)。
7. 在 Include path (包含路徑) 輸入欄位中複製並貼上路徑：

   ```
   s3://crawler-public-us-west-2/flight/2016/csv/
   ```
8. 在 Data format (資料格式) 區段中，對於 Classification (分類)，選擇 CSV，對於 Delimiter (分隔符號)，選擇 comma (,) (逗號 (,))。選擇下一步。
9. 系統會要求您定義結構描述。結構描述定義資料記錄的結構和格式。選擇 Add column (新增資料欄)。(如需詳細資訊，請參閱結構描述登錄檔。)
10. 指定資料欄屬性：

    a. 輸入資料欄名稱。
    b. 對於 Column type (資料欄類型)，依預設已選取 'string' (字串)。
    c. 對於 Column number (欄號)，依預設已選取 '1'。
    d. 選擇新增。
11. 系統會要求您新增分割區索引。這是選用的。要略過此步驟，請選擇 Next (下一步)。
12. 此時將顯示資料表屬性的摘要。如果一切看起來如預期，請選擇 [建立]。否則，請選擇 Back (返回) 並視需要進行編輯。

恭喜您，您已成功手動建立資料表並將其關聯到資料庫。您最新新建的資料表將顯示在 Table (表) 儀表板中。對於儀表板，您可以修改和管理所有資料表。

如需詳細資訊，請參閱在 AWS Glue 主控台上使用資料表。

後續步驟

後續步驟

現在已填入 Data Catalog，您可以開始在 AWS Glue 中編寫任務。請參閱使AWS Glue Studio用建立視覺化 ETL 工作。

除了使用主控台之外，還有其他方法可以在 Data Catalog 中定義資料表，包括：

- 建立和執行爬蟲程式
- 將分類器新增至搜尋器 AWS Glue
- 使用 AWS Glue 資料表 API
- 使用 AWS Glue Data Catalog 範本
- 遷移 Apache Hive 中繼存放區
- 使用 AWS CLI、Boto3 或資料定義語言 (DDL)

以下範例說明了如何使用 CLI、Boto3 或 DDL 根據您在教學中使用的 S3 儲存貯體中的相同 flights_data.csv 檔案定義資料表。

請參閱有關如何構建 AWS CLI 命令的文件。CLI 範例包含 'aws glue create-table --table-input' 值的 JSON 語法。

CLI

```
{
        "Name": "flights_data_cli",
        "StorageDescriptor": {
            "Columns": [
                {
                    "Name": "year",
                    "Type": "bigint"
                },
                {
                    "Name": "quarter",
                    "Type": "bigint"
                }
            ],
            "Location": "s3://crawler-public-us-west-2/flight/2016/csv",
            "InputFormat": "org.apache.hadoop.mapred.TextInputFormat",
            "OutputFormat":
 "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
            "Compressed": false,
            "NumberOfBuckets": -1,
            "SerdeInfo": {
                "SerializationLibrary":
 "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
                "Parameters": {
                    "field.delim": ",",
                    "serialization.format": ","
                }
```

```
            }
        },
        "PartitionKeys": [
            {
                "Name": "mon",
                "Type": "string"
            }
        ],
        "TableType": "EXTERNAL_TABLE",
        "Parameters": {
            "EXTERNAL": "TRUE",
            "classification": "csv",
            "columnsOrdered": "true",
            "compressionType": "none",
            "delimiter": ",",
            "skip.header.line.count": "1",
            "typeOfData": "file"
        }
    }
```

Boto3

```
import boto3

glue_client = boto3.client("glue")

response = glue_client.create_table(
    DatabaseName='sampledb',
    TableInput={
        'Name': 'flights_data_manual',
    'StorageDescriptor': {
      'Columns': [{
        'Name': 'year',
        'Type': 'bigint'
      }, {
        'Name': 'quarter',
        'Type': 'bigint'
      }],
      'Location': 's3://crawler-public-us-west-2/flight/2016/csv',
      'InputFormat': 'org.apache.hadoop.mapred.TextInputFormat',
      'OutputFormat':
 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat',
```

```
      'Compressed': False,
      'NumberOfBuckets': -1,
      'SerdeInfo': {
        'SerializationLibrary':
 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe',
        'Parameters': {
          'field.delim': ',',
          'serialization.format': ','
        }
      },
    },
    'PartitionKeys': [{
      'Name': 'mon',
      'Type': 'string'
    }],
    'TableType': 'EXTERNAL_TABLE',
    'Parameters': {
      'EXTERNAL': 'TRUE',
      'classification': 'csv',
      'columnsOrdered': 'true',
      'compressionType': 'none',
      'delimiter': ',',
      'skip.header.line.count': '1',
      'typeOfData': 'file'
    }
    }
)
```

DDL

```
CREATE EXTERNAL TABLE `sampledb`.`flights_data` (
  `year` bigint,
  `quarter` bigint)
PARTITIONED BY (
  `mon` string)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
```

```
  's3://crawler-public-us-west-2/flight/2016/csv/'
TBLPROPERTIES (
  'classification'='csv',
  'columnsOrdered'='true',
  'compressionType'='none',
  'delimiter'=',',
  'skip.header.line.count'='1',
  'typeOfData'='file')
```

設定對資料存放區的網路存取

若要執行擷取、轉換和載入 (ETL) 任務，AWS Glue 必須能夠存取您的資料存放區。如果工作不需要在虛擬私有雲端 (VPC) 子網路中執行，例如，從 Amazon S3 到 Amazon S3 轉換資料，無需額外組態。

如果任務需要在 VPC 子網路內執行—例如，在私有子網路中轉換 JDBC 資料存放區內的資料—AWS Glue 會設定彈性網路界面，讓您的任務能安全地連接到 VPC 內的其他資源。每個彈性網路界面都會指派一個私有 IP 地址，而此地址來自您在子網路中指定的 IP 地址範圍。未指派公有 IP 地址。AWS Glue 連線中指定的安全群組將套用至每個彈性網路界面。如需詳細資訊，請參閱 為 Amazon RDS 資料存放區的 JDBC 連線設定 Amazon VPC AWS Glue。

任務可存取的所有 JDBC 資料存放區必須可從 VPC 子網路使用。要從 VPC 存取 Amazon S3，需有 VPC 端點。如果您的任務需要同時存取 VPC 資源和公有網際網路，則 VPC 需要在 VPC 中有一個網路位址轉譯 (NAT) 閘道。

任務或開發端點一次只能存取一個 VPC (和子網路)。如果您需要存取其他 VPC 內的資料存放區，您有下列選項：

- 使用 VPC 對等存取資料存放區。如需有關 VPC 對等的詳細資訊，請參閱 VPC 對等基本概念
- 使用 Amazon S3 儲存貯體做為媒介儲存位置。將任務分割成兩個任務，以任務 1 的 Amazon S3 輸出做為任務 2 的輸入。

如需有關如何使用 Amazon VPC 連線至 Amazon Redshift 資料存放區的詳細資訊，請參閱 the section called “設定 Redshift”。

如需有關如何使用 Amazon VPC 連線至 Amazon RDS 資料存放區的詳細資訊，請參閱 the section called “設置 Amazon VPC 以連接到 Amazon RDS 數據存儲區”。

在 Amazon VPC 中設定所需規則後，您可以在 AWS Glue 中建立具有必要屬性的連線，以連線至資料存放區。如需有關連線的詳細資訊，請參閱 連線至資料。

> **Note**
>
> 務必針對 AWS Glue 設定您的 DNS 環境。如需詳細資訊，請參閱 設定 VPC 中的 DNS。

主題

設定 VPC 以連線至 PyPI for AWS Glue

Python Package Index (PyPI) 是 Python 程式設計語言的軟體儲存庫。本主題介紹支援使用 pip 安裝的套件 (由工作階段建立者使用 `--additional-python-modules` 旗標指定) 所需的詳細資訊。

若將 AWS Glue 互動式工作階段與連接器搭配使用，會導致透過為連接器指定的子網路使用 VPC 網路。因此，除非您設定特殊組態，否則無法使用 AWS 服務和其他網路目的地。

此問題的解決方案包括：

- 使用工作階段可存取的網際網路閘道。
- 設定和使用具有 PyPI/Simple 儲存庫的 S3 儲存貯體，其中包含套件集相依性的傳遞關閉。
- 使用鏡像 PyPI 並連接至 VPC 的 CodeArtifact 儲存庫。

設定網際網路閘道

NAT 閘道使用案例中詳細介紹了技術方面的內容，但請注意這些使用 `--additional-python-modules` 的要求。具體而言，`--additional-python-modules` 需要存取 pypi.org，這由您的 VPC 組態決定。請注意以下要求：

1. 需要透過 pip 安裝為使用者工作階段安裝額外的 python 模組。如果工作階段使用連接器，您的組態可能會受到影響。
2. 當連接器與 `--additional-python-modules` 一起使用，在工作階段啟動時，與連接器的 `PhysicalConnectionRequirements` 關聯的子網路必須提供用於連接 pypi.org 的網路路徑。

3. 您必須確定您的組態是否正確。

設定 Amazon S3 儲存貯體以託管目標 PyPI/Simple 儲存庫

此範例會在 Amazon S3 中針對一組套件及其相依性設定 PyPI 鏡像。

若要為一組套件設定 PyPI 鏡像：

```
# pip download all the dependencies
pip download -d s3pypi --only-binary :all: plotly gglplot
pip download -d s3pypi --platform manylinux_2_17_x86_64 --only-binary :all: psycopg2-
binary
# create and upload the pypi/simple index and wheel files to the s3 bucket
s3pypi -b test-domain-name --put-root-index -v s3pypi/*
```

如果您已有現有的成品儲存庫，其將有一個供 pip 使用的索引 URL，您可以提供該索引 URL 來取代上述 Amazon S3 儲存貯體的範例 URL。

若要透過一些範例套件使用自訂索引 URL：

```
%%configure
{
    "--additional-python-modules": "psycopg2_binary==2.9.5",
    "python-modules-installer-option": "--no-cache-dir --verbose --index-url https://
test-domain-name.s3.amazonaws.com/ --trusted-host test-domain-name.s3.amazonaws.com"
}
```

設定連接至 VPC 之 pypi 的 CodeArtifact 鏡像

若要設定鏡像：

1. 在與連接器所用之子網路相同的區域中建立儲存庫。

   選取 `Public upstream repositories` 並選擇 `pypi-store`。
2. 提供從子網路的 VPC 對儲存庫的存取權。
3. 使用 `python-modules-installer-option` 指定正確的 `--index-url`。

```
%%configure
{
```

```
    "--additional-python-modules": "psycopg2_binary==2.9.5",
    "python-modules-installer-option": "--no-cache-dir --verbose --index-url https://
test-domain-name.s3.amazonaws.com/ --trusted-host test-domain-name.s3.amazonaws.com"
}
```

如需詳細資訊，請參閱 Use CodeArtifact from a VPC。

設定 VPC 中的 DNS

網域名稱系統 (DNS) 是一種在網際網路上用於解析為對應 IP 地址的標準名稱。DNS 主機名稱為獨特的電腦名稱並由主機名稱和網域名稱組成。DNS 伺服器會將 DNS 主機名稱解析為對應的 IP 地址。

若要在 VPC 中設定 DNS，請確保在您 VPC 中的 DNS 主機名稱和 DNS 解析均已啟用。VPC 網路屬性 `enableDnsHostnames` 和 `enableDnsSupport` 必須設定為 `true`. 要查看和修改這些屬性，請移至位於 https://console.aws.amazon.com/vpc/ 的 VPC 主控台。

如需詳細資訊，請參閱以 VPC 使用 DNS。此外，您也可以使用 AWS CLI 並呼叫 modify-vpc-attribute 命令來設定 VPC 網路屬性。

> **Note**
>
> 如果您使用的是 Route 53，請確認您的組態不會覆寫 DNS 網路屬性。

設定 AWS Glue 中的加密

以下範例任務流程重點說明，您在 AWS Glue 中使用加密時應設定的選項。範例中示範特定 AWS Key Management Service (AWS KMS) 金鑰的用法，但您可能會根據自己的特殊需要選擇其他設定。此任務流程僅重點說明與設定 AWS Glue 時的加密相關的選項。

1. 如果 AWS Glue 主控台的使用者未使用允許所有 AWS Glue API 操作 (例如 `"glue:*"`) 的許可政策，請確認允許以下動作：
   - `"glue:GetDataCatalogEncryptionSettings"`
   - `"glue:PutDataCatalogEncryptionSettings"`
   - `"glue:CreateSecurityConfiguration"`
   - `"glue:GetSecurityConfiguration"`
   - `"glue:GetSecurityConfigurations"`

- "glue:DeleteSecurityConfiguration"

2. 存取或寫入加密目錄的任何用戶端 - 也就是任何主控台使用者、爬蟲程式、任務或開發端點，都需要以下許可。

```
{
 "Version": "2012-10-17",
  "Statement": {
     "Effect": "Allow",
     "Action": [
           "kms:GenerateDataKey",
           "kms:Decrypt",
           "kms:Encrypt"
      ],
     "Resource": "<key-arns-used-for-data-catalog>"
   }
}
```

3. 存取加密連線密碼的任何使用者或角色皆需要以下許可。

```
{
 "Version": "2012-10-17",
  "Statement": {
     "Effect": "Allow",
     "Action": [
           "kms:Decrypt"
         ],
     "Resource": "<key-arns-used-for-password-encryption>"
         }
}
```

4. 將加密資料寫入 Amazon S3 的任何擷取、轉換和載入 (ETL) 任務角色都需要以下許可。

```
{
 "Version": "2012-10-17",
  "Statement": {
     "Effect": "Allow",
     "Action": [
           "kms:Decrypt",
           "kms:Encrypt",
           "kms:GenerateDataKey"
      ],
     "Resource": "<key-arns-used-for-s3>"
```

```
    }
}
```

5. 任何撰寫加密 Amazon CloudWatch Logs 的 ETL 任務或爬蟲程式都需要在金鑰和 IAM 政策中具有以下許可。

   在金鑰政策 (而非 IAM 政策) 中：

```
{
  "Effect": "Allow",
  "Principal": {
   "Service": "logs.region.amazonaws.com"
  },
  "Action": [
   "kms:Encrypt*",
   "kms:Decrypt*",
   "kms:ReEncrypt*",
   "kms:GenerateDataKey*",
   "kms:Describe*"
  ],
  "Resource": "<arn of key used for ETL/crawler cloudwatch encryption>"
 }
```

   如需金鑰政策的相關詳細資訊，請參閱 AWS Key Management Service 開發人員指南中的在 AWS KMS 中使用金鑰政策。

   在 IAM 政策中連接 `logs:AssociateKmsKey` 許可：

```
{
  "Effect": "Allow",
  "Principal": {
   "Service": "logs.region.amazonaws.com"
  },
  "Action": [
   "logs:AssociateKmsKey"
  ],
  "Resource": "<arn of key used for ETL/crawler cloudwatch encryption>"
 }
```

6. 任何使用加密任務書籤的 ETL 任務都需要以下許可。

```
{
```

```
 "Version": "2012-10-17",
  "Statement": {
     "Effect": "Allow",
     "Action": [
           "kms:Decrypt",
           "kms:Encrypt"
      ],
     "Resource": "<key-arns-used-for-job-bookmark-encryption>"
   }
}
```

7. 在 AWS Glue 主控台上的導覽窗格中，選擇 Settings (設定)。

   a. 在 Data catalog settings (資料目錄設定) 頁面上，選取 Metadata encryption (中繼資料加密)。此選項會使用您選擇的 AWS KMS 金鑰將資料目錄中的所有物件加密。

   b. 如為 AWS KMS 金鑰，選擇 aws/glue。您也可以選擇您建立的 AWS KMS 索引鍵。

   > **Important**
   >
   > AWS Glue 僅支援對稱客戶主金鑰 (CMK)。AWS KMS key (KMS 金鑰) 清單僅會顯示對稱金鑰。不過，如果您選取 Choose a AWS KMS key ARN (選擇 KMS 金鑰 ARN)，主控台可讓您輸入任何金鑰類型的 ARN。請確定您僅輸入對稱金鑰的 ARN。

   啟用加密時，存取資料目錄的用戶端必須具有 AWS KMS 許可。

8. 在導覽窗格中，選擇 Security configurations (安全組態)。安全組態是一組安全屬性，可用來設定 AWS Glue 程序。然後選擇 Add security configuration (新增安全組態)。在組態中，選擇以下任一個選項：

   a. 選取 S3 encryption (S3 加密)。針對 Encryption mode (加密模式) 選擇 SSE-KMS。針對 AWS KMS key (金鑰)，選擇 aws/s3 (確認使用者具有使用此金鑰的許可)。這樣就可讓任務寫入 Amazon S3 的資料使用 AWS 受管 AWS Glue AWS KMS 金鑰。

   b. 選取 CloudWatch logs encryption (CloudWatch Logs 加密)，然後選擇某個 CMK。(請確認使用者具有使用此金鑰的許可)。如需詳細資訊，請參閱 AWS Key Management Service 開發人員指南中的使用 AWS KMS 在 CloudWatch Logs 中加密日誌資料。

> **Important**
>
> AWS Glue 僅支援對稱客戶主金鑰 (CMK)。AWS KMS key (KMS 金鑰) 清單僅會顯示對稱金鑰。不過，如果您選取 Choose a AWS KMS key ARN (選擇 KMS 金鑰 ARN)，主控台可讓您輸入任何金鑰類型的 ARN。請確定您僅輸入對稱金鑰的 ARN。

c. 選擇 Advanced properties (進階屬性)，並選取 Job bookmark encryption (任務書籤加密)。針對 AWS KMS key (金鑰)，選擇 aws/glue (確認使用者具有使用此金鑰的許可)。這樣就可利用 AWS Glue AWS KMS 金鑰將寫入 Amazon S3 的任務書籤加密。

9. 在導覽窗格中，選擇 Connections (連線)。

   a. 選擇 Add connection (新增連線) 建立與 Java Database Connectivity (JDBC) 資料存放區的連線，這是您 ETL 任務的目標。

   b. 若要強制使用 Secure Sockets Layer (SSL) 加密，請選取 Require SSL connection (需要 SSL 連接) 並測試連線。

10. 在導覽窗格中，選擇 Jobs (任務)。

    a. 選擇 Add job (新增任務) 建立轉換資料的任務。

    b. 在任務定義中，選擇您建立的安全組態。

11. 在 AWS Glue 主控台上，隨需執行您的任務。確認任務寫入的任何 Amazon S3 資料、任務寫入的 CloudWatch Logs 以及任務書籤都已全部加密。

專為 AWS Glue 的開發設定聯網

若要用 AWS Glue 執行擷取、轉換和載入 (ETL) 指令碼，可以使用開發端點開發和測試您的指令碼。開發端點不支援用於 AWS Glue 2.0 版任務。對於 2.0 及更高版本，慣用的開發方法是使用 Jupyter 筆記本與其中一個 AWS Glue 核心。如需詳細資訊，請參閱 the section called “AWS Glue 互動式工作階段入門”。

專為開發端點設定您的網路

設定開發端點時，應指定 Virtual Private Cloud (VPC)、子網路和安全群組。

> **Note**
>
> 務必針對 AWS Glue 設定您的 DNS 環境。如需詳細資訊，請參閱 設定 VPC 中的 DNS。

若要啟用 AWS Glue 存取所需的資源，在您的子網路路由表中新增資料列，將 Amazon S3 字首清單與 VPC 端點建立關聯。字首清單 ID 為建立傳出安全群組規則，以允許從 VPC 透過 VPC 端點存取 AWS 服務所需。若要輕鬆連接到與此開發端點關聯的筆記型電腦伺服器，請在本機電腦上新增資料列至路由表，以新增網際網路閘道 ID。如需詳細資訊，請參閱 VPC 端點。更新子網路路由表為類似以下資料表：

目的地	目標		
10.0.0.0/16	本機		
適用於 Amazon S3 的 pl-id	vpce-id		
0.0.0.0/0	igw-xxxx		

要讓 AWS Glue 在其元件之間通訊，請指定一個安全群組並為所有 TCP 連接埠建立自我參考的傳入規則。建立自我參考規則後，您就可以將資源限制給 VPC 中相同的安全群組，不對所有網路開放。VPC 的預設安全群組可能已經有了 ALL Traffic 的自我參考傳入規則。

若要設定安全群組

1. 請登入 AWS Management Console，並在 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
2. 在左導覽窗格中，選擇 Security Groups (安全群組)。
3. 您可以從清單中選擇現有的安全群組，或 Create Security Group (建立安全群組)，以用於開發端點。
4. 在安全群組窗格中，導覽至 Inbound (傳入) 標籤。
5. 新增自我參考規則，以允許 AWS Glue 元件進行通訊。具體來說，新增或確認有類型 `All TCP`、通訊協定為 `TCP`，連接埠範圍包含所有連接埠，且其來源與群組 ID 為相同安全群組名稱的規則。

   傳入規則類似：

類型	通訊協定	連接埠範圍	來源
所有 TCP	TCP	0–65535	*security-group*

以下顯示自我參考傳入規則的範例：

6. 新增一個規則，以用於傳出流量。可以開啟傳出流量至所有連接埠，或建立 類型All TCP、通訊協定為 TCP、連接埠範圍包含所有連接埠，且其來源與群組 ID 為相同安全群組名稱的自我參考規則。

   傳出規則類似下列其中一個規則：

類型	通訊協定	連接埠範圍	目的地
所有 TCP	TCP	0–65535	*security-group*
所有流量	ALL	ALL	0.0.0.0/0

設定筆記本伺服器的 Amazon EC2

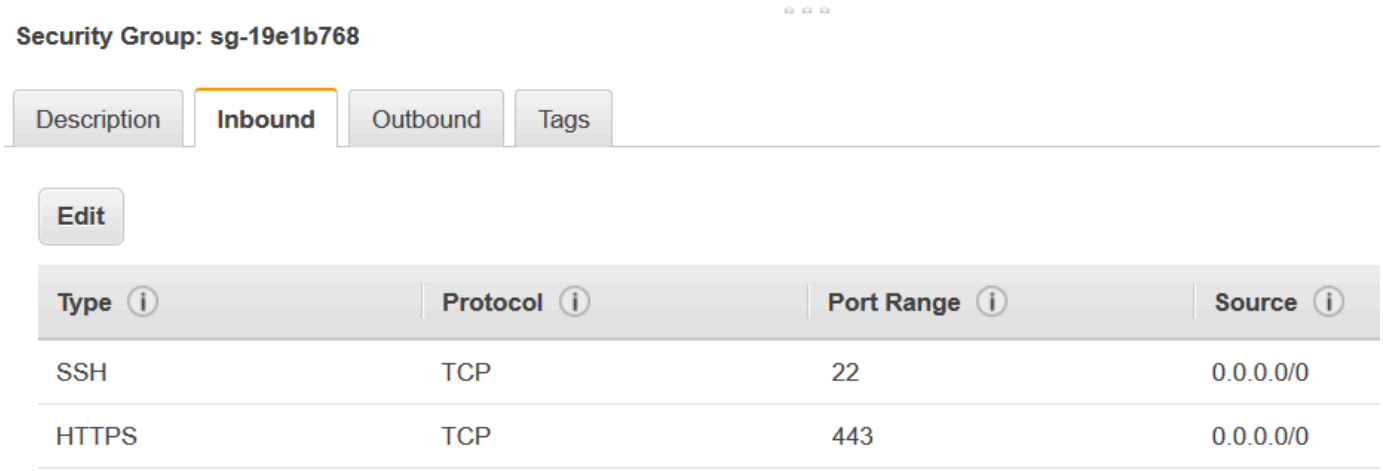
透過開發端點，您可以建立筆記本伺服器，以使用 Jupyter 筆記本測試 ETL 指令碼。若要啟用與您的筆記本的通訊，請指定含 HTTPS (連接埠 443) 和 SSH (連接埠 22) 規則傳入的安全群組。請確定規則的來源為 0.0.0.0/0 或連接至筆記本的機器 IP 地址。

若要設定安全群組

1. 請登入 AWS Management Console，並在 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
2. 在左導覽窗格中，選擇 Security Groups (安全群組)。
3. 您可以從清單中選擇現有的安全群組，或 Create Security Group (建立安全群組)，以搭配您的筆記本伺服器使用。與您的開發端點關聯的安全群組也可用來建立您的筆記本伺服器。
4. 在安全群組窗格中，導覽至 Inbound (傳入) 標籤。
5. 新增傳入規則，類似如下：

類型	通訊協定	連接埠範圍	來源
SSH	TCP	22	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0

以下顯示安全群組傳入規則的範例：



數據發現和編目 AWS Glue

這 AWS Glue Data Catalog 是一個集中儲存組織資料集的中繼資料的儲存庫。它充當資料來源位置、結構描述和執行階段度量的索引。中繼資料儲存在中繼資料表格中，其中每個表格代表單一資料存放區。

您可以使用爬蟲 (Crawler) 填入資料目錄，它會自動掃描您的資料來源並擷取中繼資料。爬行者程式可以連線到內部 (AWS基於) 和外部的資料來源。 AWS

如需有關支援資料來源的詳細資訊，請參閱 我可以網路爬取哪個資料存放區？

您也可以根據特定需求定義資料表結構、結構描述和資料分割結構，在「資料目錄」中手動建立資料表。

如需手動建立中繼資料表格的詳細資訊，請參閱手動定義元資料。

您可以使用資料目錄中的資訊來建立和監視 ETL 工作。資料型錄與其他分 AWS 析服務整合，提供資料來源的統一檢視，讓您更輕鬆地管理和分析資料。

- Amazon Athena — 使用 SQL 在 Amazon S3 資料的資料目錄中存放和查詢表格中繼資料。
- AWS Lake Formation — 集中定義和管理精細的數據訪問策略和審核數據訪問。
- Amazon EMR — 存取資料目錄中定義的資料來源以進行大數據處理。
- Amazon SageMaker — 快速且自信地建置、訓練和部署機器學習模型。

資料目錄的主要功能

以下是資料目錄的關鍵方面。

元數據存儲

「資料目錄」充當中央描述資料儲存庫，可儲存有關資料來源位置、結構描述和屬性的資訊。此中繼資料會組織成資料庫和表格，類似於傳統的關聯式資料庫目錄。

自動資料可探索性

AWS Glue 編目程式 s 可以自動探索和編目新的或更新的資料來源，減少手動中繼資料管理的額外負荷，並確保您的資料目錄保留 up-to-date。透過對資料來源進行編目，資料目錄可讓使用者和應用程式更輕鬆地探索和瞭解組織內可用的資料資產，進而促進資料重複使用和協同合作。

資料型錄支援各種資料來源，包括 Amazon S3、Amazon RDS、Amazon Redshift、阿帕奇蜂巢等。它可以使用 AWS Glue 編目程式 s 自動推斷和存儲這些來源的元數據。

若要取得更多資訊，請參閱使用編目器填入資料目錄 。

綱要管理

「資料目錄」會自動擷取和管理資料來源的結構描述，包括結構描述推論、演進和版本控制。您可以使用 AWS Glue ETL 工作更新「資料目錄」中的結構描述和分割區。

表格最佳化

為了透過 Amazon Athena 和 Amazon EMR 和 AWS Glue ETL 任務等 AWS 分析服務獲得更好的讀取效能，資料型錄為資料目錄中的冰山表提供受管壓縮 (將小型 Amazon S3 物件壓縮為較大物件的程序)。您可以使用 AWS Glue 主控 AWS Lake Formation 台 AWS CLI、主控台或 AWS API 來啟用或停用資料目錄中個別 Iceberg 資料表的壓縮功能。

如需詳細資訊，請參閱 最佳化處理 Iceberg 資料表。

資料欄統計資料

您可以計算資料目錄資料表的資料格式 (例如實木複合地板、ORC、JSON、ION、CSV 和 XML) 的欄層級統計資料，而無需設定其他資料管線。資料欄統計資料可協助您透過深入了解資料欄內的值，了解資料設定檔。資料目錄支援產生資料行值的統計資料，例如最小值、最大值、總 Null 值、總不同值、值的平均長度，以及真實值的總發生次數。

如需詳細資訊，請參閱 使用資料欄統計資料來最佳化。

資料歷程

「資料目錄」會維護對資料執行的轉換和作業的記錄，以提供資料歷程資訊。此歷程資訊對於稽核、法規遵循和瞭解資料的來源非常有用。

與其他 AWS 服務整合

資料型錄可與其他 AWS 服務無縫整合，例如 AWS Lake Formation Amazon Athena、Amazon Redshift Spectrum 和 Amazon EMR。此整合可讓您使用單一一致的中繼資料層，查詢和分析各種資料存放區的資料。

安全性和存取控制

AWS Glue 與整合 AWS Lake Formation 以支援資料目錄資源的精細存取控制，讓您可以根據組織的政策和需求來管理權限並保護對資料資產的存取。 AWS Glue 與 AWS Key Management Service (AWS KMS) 整合以加密儲存在資料目錄中的中繼資料。

主題

- 填入 AWS Glue 資料目錄

- 填入及管理交易資料表
- 管理資料目錄
- 存取資料目錄
- AWS Glue 資料目錄最佳做法
- AWS Glue 結構描述登錄檔

填入 AWS Glue 資料目錄

您可以 AWS Glue Data Catalog 使用下列方法填入：

- AWS Glue 編目程式 — AWS Glue 編目程式 可以自動探索和編目資料來源，例如資料庫、資料湖和串流資料。檢索器是填入資料目錄的最常用且建議使用的方法，因為它們可以自動探索和推斷各種資料來源的中繼資料。
- 手動新增中繼資料 — 您可以手動定義資料庫、表格和連線詳細資料，並使用 AWS Glue 主控台、Lake Formation 主控台或 AWS Glue API 將其新增至「資料目錄」。 AWS CLI當您要編目無法編目的資料來源時，手動輸入非常有用。
- 與其他 AWS 服務整合 — 您可以使用來自 Amazon Athena 等 AWS Lake Formation 服務的中繼資料填入資料目錄。這些服務可以探索和註冊資料目錄中的資料來源。
- 從現有的中繼資料存放庫填入 — 如果您擁有現有的中繼資料存放區 (例如 Apache Hive 中繼資料存放區)，則可以使用 AWS Glue 將該中繼資料匯入資料目錄。如需詳細資訊，請參閱 Hive 中繼存放區和 AWS Glue Data Catalog上 GitHub之間的遷移。

主題

- 使用編目器填入資料目錄
- 手動定義元資料
- 與其他 AWS 服務整合
- 資料目錄設定

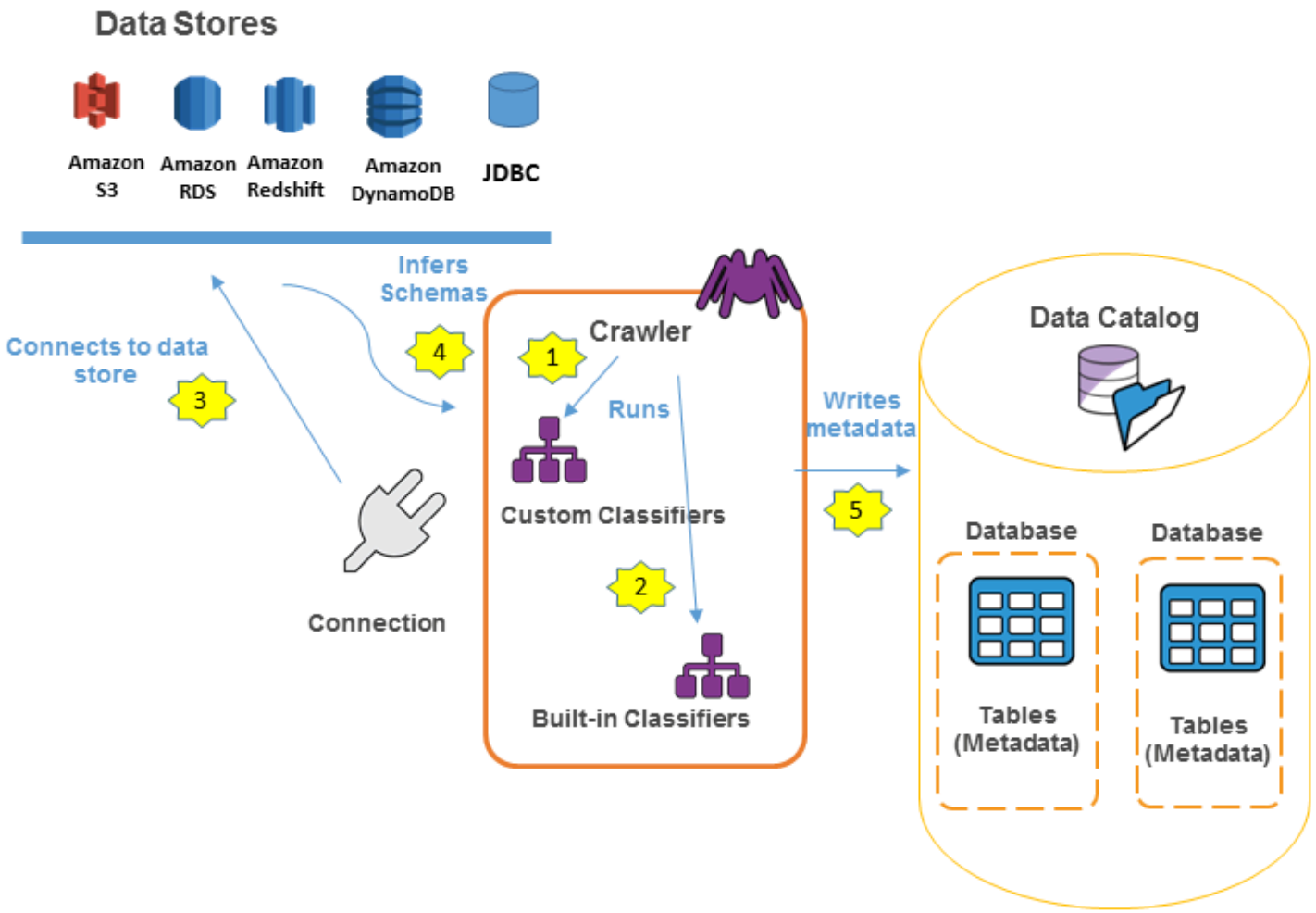
使用編目器填入資料目錄

您可以使用 AWS Glue 編目程式 來填入資 AWS Glue Data Catalog 料庫和資料表。這是大多數用 AWS Glue 戶使用的主要方法。爬蟲程式可以在單一執行中抓取多個資料存放區。一旦完成，爬蟲程式即會在 Data Catalog 中建立或更新一或多個資料表。您在 AWS Glue 中定義的擷取、轉換和載入

(ETL) 任務，會將這些 Data Catalog 資料表做為來源和目標使用。ETL 任務可讀取和寫入來源及目標 Data Catalog 資料表中指定的資料存放區。

工作流程

以下任務流程圖顯示 AWS Glue 爬蟲程式如何與資料存放區和其他元素互動以填入資料目錄。



以下是爬蟲程式填入 AWS Glue Data Catalog的一般任務流程：

1. 爬蟲程式會執行您選擇的任何自訂「分類器」以推斷資料的結構描述。您為自訂分類器提供程式碼，分類器依照您指定的順序執行。

   第一個自訂分類器成功識別資料結構可用來建立結構描述。清單中較低的自訂分類器會被略過。
2. 如果沒有自訂分類器符合您資料的結構描述，內建分類器將嘗試識別您資料的結構描述。可識別 JSON 的分類器即是一種內建分類器的範例。

3. 爬蟲程式連接到資料存放區。有些資料存放區需要連線屬性才能讓爬蟲程式存取。
4. 為您的資料建立經過推斷的結構描述。
5. 爬蟲程式將中繼資料寫入資料目錄。資料表定義包含與資料存放區中的資料有關的中繼資料。資料表寫入資料庫，該資料庫是資料目錄中的資料表容器。資料表的屬性包含分類，它是由推斷資料表結構描述的分類器所建立的標籤。

主題

爬蟲程式的運作方式

爬蟲程式執行時，它會進行以下動作來詢問資料存放區：

- 分類器資料會判斷格式、結構描述及原始資料的相關屬性 – 您可以透過建立自訂分類器來設定分類的結果。
- 將資料分組至資料表或分區中 – 資料是根據爬蟲程式啟發來加以分組。
- 將中繼資料寫入到 Data Catalog – 您可以設定爬蟲程式如何新增、更新和刪除資料表和分區。

定義爬蟲程式時，您可選擇一個或多個分類器評估資料格式以推斷結構描述。爬蟲程式執行時，清單中第一個成功辨識資料存放區的分類器會用於為您的資料表建立結構描述。您可以使用內建的分類器或自行定義。您可以先在個別操作中定義自訂分類器，之後再定義爬蟲程式。AWS Glue 提供內建的分類器，可從常見檔案格式 (包括 JSON、CSV 和 Apache Avro) 來推斷結構描述。關於 AWS Glue 目前的內建分類器清單，請參閱AWS Glue 中的內建分類器。

爬蟲程式建立的中繼資料資料表，會包含在您定義爬蟲程式時的資料庫裡。如果爬蟲程式未指定資料庫，資料表會存放於預設的資料庫。此外，每個資料表都有分類欄，會由第一個成功辨識資料存放區的分類器填寫。

如果抓取的檔案已壓縮，爬蟲程式就必須下載檔案才能處理。當爬蟲程式執行時，其會詢問檔案以判斷格式和壓縮類型，並將這些屬性寫入到 Data Catalog 。Apache Parquet 等部分檔案格式，可讓您在進行寫入時壓縮部分檔案。對於這些檔案，壓縮資料是檔案內部元件，且 AWS Glue 不會在將資料表寫入 Data Catalog 時填入 `compressionType` 屬性。反之，如果「整個檔案」是以壓縮演算法進行壓縮 (例如 gzip)，則會在將資料表寫入 Data Catalog 時填入 `compressionType` 屬性。

爬蟲程式會為其建立的資料表產生名稱。儲存在中的資料表名稱 AWS Glue Data Catalog 遵循下列規則：

- 只能使用英數字元和底線 (_)。
- 任何自訂的字首都不能超過 64 個字元。
- 名稱的長度上限不能超過 128 個字元。爬蟲程式會截斷產生的名稱以符合限制。
- 如果遇到重複的資料表名稱，爬蟲程式會為名稱加上雜湊字串尾碼。

如果爬蟲程式執行超過一次 (或許是按照排程)，則會在資料存放區中尋找新的或變更過的檔案或資料表。爬蟲程式的輸出會包含前一次執行時找到的新資料表和分割區。

我可以網路爬取哪個資料存放區？

爬蟲程式可以抓取下列檔案型和資料表型資料存放區。

爬蟲程式使用的存取類型	資料存放區
原生用戶端	• Amazon Simple Storage Service (Amazon S3) • Amazon DynamoDB • Delta Lake 2.0.x • 阿帕奇冰山 1.5 • Apache Hudi 0.14
JDBC	Amazon Redshift Snowflake

爬蟲程式使用的存取類型	資料存放區
	在 Amazon Relational Database Service (Amazon RDS) 中或 Amazon RDS 外部： • Amazon Aurora • MariaDB • Microsoft SQL Server • MySQL • Oracle • PostgreSQL
MongoDB 用戶端	• MongoDB • MongoDB Atlas • Amazon DocumentDB (with MongoDB compatibility)

> **Note**
>
> 目前 AWS Glue 不支援資料串流的爬蟲程式。

對於 JDBC、MongoDB、MongoDB Atlas 和 Amazon DocumentDB (with MongoDB compatibility) 資料存放區，您必須指定爬蟲程式可用來連線到資料存放區的 AWS Glue 連線爬蟲程式。對於 Amazon S3，您可以選擇性地指定網路類型的連線。連線是儲存連線資訊的 Data Catalog 物件，例如憑證、URL、Amazon Virtual Private Cloud 資訊等。如需詳細資訊，請參閱 連線至資料。

下列是爬行者程式支援的驅動程式版本：

產品	爬蟲支持的驅動程序
PostgreSQL	42.2.1
Amazon Aurora	與原生爬蟲驅動程式相同
MariaDB	8.0.13
Microsoft SQL Server	6.1.0

產品	爬蟲支持的驅動程序
MySQL	8.0.13
Oracle	11.2.2
Amazon Redshift	4.1
Snowflake	3.13.20
MongoDB	4.7.2
MongoDB Atlas	4.7.2

以下是有關各種資料存放區的注意事項。

Amazon S3

您可以選擇在您的帳戶或在另一個帳戶網路爬取路徑。如果資料夾中的所有 Amazon S3 檔案都具有相同的結構描述，爬蟲程式就會建立一個資料表。此外，如果 Amazon S3 物件已分割，則只會建立一個中繼資料資料表，並將分割資訊新增到該資料表的 Data Catalog 。

Amazon S3 和 Amazon DynamoDB

爬蟲使用 AWS Identity and Access Management (IAM) 角色取得存取資料存放區的權限。您傳遞至爬蟲程式的角色必須擁有許可，才能存取已抓取的 Amazon S3 路徑和 Amazon DynamoDB 資料表。

Amazon DynamoDB

使用 AWS Glue 主控台定義爬蟲程式時，需指定一個 DynamoDB 資料表。若您是使用 AWS Glue API，可指定資料表清單。您可以選擇只網路爬取一小部分的資料範例，以減少爬蟲程式的執行時間。

Delta Lake

針對每個 Delta Lake 資料存放區，請指定如何建立 Delta 資料表：

- 建立原生資料表：允許與支援直接查詢 Delta 交易日誌的查詢引擎整合。如需詳細資訊，請參閱查詢 Delta Lake 資料表。
- 建立符號連結資料夾：根據特定組態參數，使用由分割區索引鍵分割的資訊清單檔案建立 `_symlink_manifest` 資料夾。

Iceberg

對於每個 Iceberg 資料存放區，您可以指定包含 Iceberg 資料表中繼資料的 Amazon S3 路徑。如果爬蟲程式發現 Iceberg 資料表中繼資料，其會在資料型錄中註冊該中繼資料。您可以為爬蟲程式設定排程，讓資料表持續更新。

您可以為資料存放區定義下列參數：

- 排除：可讓您略過某些資料夾。
- 周遊深度上限：設定爬蟲程式可在 Amazon S3 儲存貯體中爬取的深度限制。預設的周遊深度上限為 10，而您可以設定的深度上限為 20。

Hudi

對於每個 Hudi 資料存放區，您可以指定包含 Hudi 資料表中繼資料的 Amazon S3 路徑。如果爬蟲程式發現 Hudi 資料表中繼資料，其會在資料型錄中註冊該中繼資料。您可以為爬蟲程式設定排程，讓資料表持續更新。

您可以為資料存放區定義下列參數：

- 排除：可讓您略過某些資料夾。
- 周遊深度上限：設定爬蟲程式可在 Amazon S3 儲存貯體中爬取的深度限制。預設的周遊深度上限為 10，而您可以設定的深度上限為 20。

> **Note**
>
> 由於與 Hudi 0.13.1 和時間戳記類型不相容，邏輯類型為 `millis` 的時間戳記資料欄將解譯為 `bigint`。即將發布的 Hudi 版本中可能會提供解決方案。

Hudi 資料表的分類如下，每個資料表都具有特定的含義：

- 寫入時複製 (CoW)：資料會以單欄式格式 (Parquet) 存放，每次更新都會在寫入期間建立新版檔案。
- 讀取時合併 (MoR)：資料的存放是使用單欄式格式 (Parquet) 和以資料列為基礎的格式 (Avro) 組合進行。更新會記錄到以資料列為基礎的 delta 檔案，並視需要壓縮以建立新版本的直欄式檔案。

使用 CoW 資料集，每次有記錄進行更新時，包含記錄的檔案就會以更新的值重寫。若使用 MoR 資料集，每次有更新時，Hudi 只會寫入已變更之記錄的資料行。MoR 更適合較少讀取，而寫入或變更較繁重的工作負載。CoW 更適合資料變更較不頻繁，而讀取作業較為繁重的工作負載。

Hudi 提供三個可用於資料存取的查詢類型：

- 快照查詢：查詢會查看截至指定遞交或壓縮動作之資料表的最新快照。對於 MoR 資料表，快照查詢會公開資料表的最新狀態，方法是合併查詢時最新檔案切片的基底和 delta 檔案。
- 增量查詢：查詢只會看到自指定遞交/壓縮以來在資料表中寫入的新資料。這會有效地提供變更串流，以啟用增量資料管道。
- 讀取最佳化查詢：對於 MoR 資料表，查詢會看到壓縮的最新資料。對於 CoW 資料表，查詢會看到遞交的最新資料。

對於「寫入時複製」資料表，爬行者程式會在「資料目錄」中建立含有 Serde 的單一資料表。ReadOptimized `org.apache.hudi.hadoop.HoodieParquetInputFormat`

對於「讀取時合併」資料表，爬蟲程式會在資料型錄中為相同的資料表位置建立兩個資料表：

- 帶有後綴的表_ro，它使用 ReadOptimized SERDE `org.apache.hudi.hadoop.HoodieParquetInputFormat`。
- 帶有後綴的表_rt，它使用 RealTime Serde 允許快照查詢：`org.apache.hudi.hadoop.realtime.HoodieParquetRealtimeInputFormat`。

MongoDB 和 Amazon DocumentDB (with MongoDB compatibility)

支援 MongoDB 3.2 版和更新版本。您可以選擇只網路爬取一小部分的資料範例，以減少爬蟲程式的執行時間。

關聯式資料庫

身分驗證是使用資料庫使用者名稱和密碼。根據資料庫引擎的類型，您可以選擇要探索哪些物件 (例如資料庫、結構描述和資料表)。

Snowflake

Snowflake JDBC 爬蟲程式支援網路爬取資料表、外部資料表、視觀表以及具體化視觀表。不會填入具體化視觀表定義。

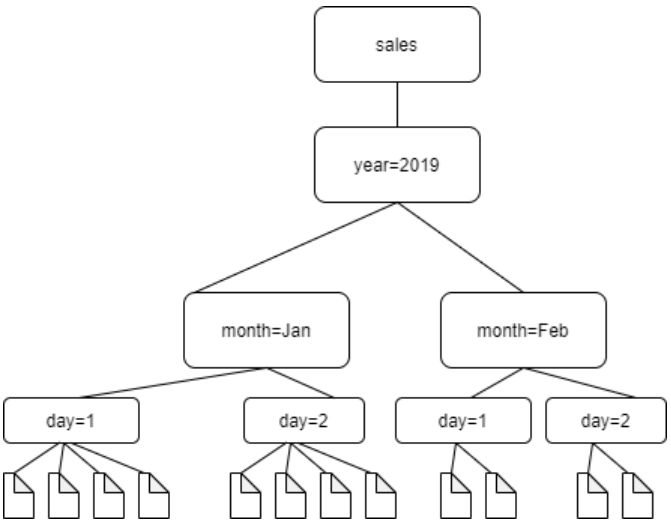
對於 Snowflake 外部資料表，爬蟲程式只有在指向 Amazon S3 位置時才會進行網路爬取。除了資料表結構描述，爬蟲程式還會網路爬取 Amazon S3 位置、檔案格式和作為 Data Catalog 資料表中資料表參數的輸出內容。請注意，不會填入已分割外部資料表的分割區資訊。

使用 Snowflake 爬蟲程式建立的 Data Catalog 資料表目前不支援 ETL。

爬蟲程式如何決定何時建立分割區？

當 AWS Glue 爬蟲程式掃描 Amazon S3 並在儲存貯體中偵測到多個資料夾時，它會決定資料夾結構中資料表的根以及哪些資料夾是資料表分區。資料表名稱是根據 Amazon S3 字首或資料夾名稱。您提供的包含路徑會指向要探索的資料夾層級。當大部分資料夾層級的結構描述都很類似時，爬蟲程式會建立資料表分區，而不是不同的資料表。若要影響爬蟲程式來建立不同的資料表，請當您定義爬蟲程式時，新增每個資料表的根資料夾做為個別的資料存放區。

例如，請試想有以下 Amazon S3 資料夾結構。



四個最低層級資料夾的路徑如下：

```
S3://sales/year=2019/month=Jan/day=1
S3://sales/year=2019/month=Jan/day=2
S3://sales/year=2019/month=Feb/day=1
S3://sales/year=2019/month=Feb/day=2
```

假設爬蟲程式目標設為 `Sales`，並且 `day=`*n* 資料夾中的所有檔案具有相同的格式 (例如 JSON，未加密)，並且具有相同或非常相似的結構描述。爬蟲程式將使用分割區索引鍵 `year`、`month` 以及 `day`，建立包含四個分割區的單一資料表。

在下一個範例中，請設想以下 Amazon S3 結構：

```
s3://bucket01/folder1/table1/partition1/file.txt
s3://bucket01/folder1/table1/partition2/file.txt
s3://bucket01/folder1/table1/partition3/file.txt
s3://bucket01/folder1/table2/partition4/file.txt
```

```
s3://bucket01/folder1/table2/partition5/file.txt
```

如果 table1 和 table2 下的檔案結構描述相似，且您是透過 Include path (包括路徑) s3://bucket01/folder1/ 在爬蟲程式中加以定義單一資料存放區，爬蟲程式即會建立含兩個分割區索引鍵欄的單一資料表。第一個分割區索引鍵欄包含 table1 和 table2，而第二個分割區索引鍵欄包含 partition1 到 partition3 (針對 table1 分割區) 和 partition4 和 partition5 (針對 table2 分割區)。若要建立兩個不同的資料表，定義含有兩個資料存放區的爬蟲程式。在這個範例中，定義第一個包含路徑為 s3://bucket01/folder1/table1/ 和第二個包含路徑為 s3://bucket01/folder1/table2。

> **Note**
>
> 在 Amazon Athena，每個資料表會與資料表中所有物件的 Amazon S3 字首相對應。如果物件有不同的結構描述，Athena 無法將相同字首中的不同的物件識別為個別資料表。若爬蟲程式透過相同 Amazon S3 字首建立資料表，就可能會發生上述問題。這可能會導致 Athena 中的查詢傳回零結果。對於 Athena，若要正確識別和查詢資料表，使用個別 Include path (包含路徑)，針對 Amazon S3 資料夾結構中的每個不同資料表結構描述建立爬蟲程式。如需詳細資訊，請參閱使用 Athena 搭配 AWS Glue 的最佳實務以及此 AWS 知識中心文章。

爬蟲程式的先決條件

爬行者程式會假設您在定義該角色時指定的 AWS Identity and Access Management (IAM) 角色的許可。這個 IAM 角色必須具有許可，來從您的資料存放區擷取資料，以及將資料寫入至 Data Catalog 。AWS Glue 主控台只會列出已經連接 AWS Glue 主要服務適用信任政策的 IAM 角色。從主控台，您也可以建立 IAM 角色與 IAM 政策，以存取爬蟲程式所存取的 Amazon S3 資料存放區。如需為 AWS Glue 提供角色的詳細資訊，請參閱 Glue 的身分識別原則 AWS。

> **Note**
>
> 網路爬取 Delta Lake 資料儲存時，您必須擁有讀/寫 Amazon S3 位置的權限。

對於爬蟲程式，您可以建立角色並連接下列政策：

- 受AWSGlueServiceRole AWS 管理的原則，授與資料目錄的必要權限
- 授予資料來源許可的內嵌政策。

- 授與角色iam:PassRole權限的內嵌政策。

更快的方法是讓 AWS Glue 主控台爬蟲程式精靈為您建立角色。它所建立的角色是專門針對爬行者程式，包含AWSGlueServiceRole AWS 受管理的原則以及指定資料來源所需的內嵌原則。

如果您指定爬蟲程式的現有角色，請確定它包含 AWSGlueServiceRole 政策或同等政策 (或此政策的範圍縮減版本)，以及必要的內嵌政策。例如，對於 Amazon S3 資料存放區，內嵌政策至少需要如下：

```
{
   "Version": "2012-10-17",
    "Statement": [
        {
          "Effect": "Allow",
          "Action": [
              "s3:GetObject",
              "s3:PutObject"
          ],
          "Resource": [
              "arn:aws:s3:::bucket/object*"
          ]
        }
    ]
}
```

對於 Amazon DynamoDB 資料存放區，政策至少需要如下：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DescribeTable",
        "dynamodb:Scan"
      ],
      "Resource": [
        "arn:aws:dynamodb:region:account-id:table/table-name*"
      ]
    }
  ]
```

```
}
```

此外，如果爬蟲讀取 AWS Key Management Service (AWS KMS) 加密的 Amazon S3 資料，則 IAM 角色必須具有 AWS KMS 金鑰的解密權限。如需更多詳細資訊，請參閱 步驟 2：為 AWS Glue 建立 IAM 角色。

設定爬行者程式

爬蟲程式會存取您的資料存放區、擷取中繼資料，以及在 AWS Glue Data Catalog中建立資料表定義。 AWS Glue 主控台中的 [爬行者程式] 窗格會列出您建立的所有爬行者程式。該清單會針對爬蟲程式最近一次的執行作業，顯示其狀態與指標。

> **Note**
>
> 如果您選擇引入自己的 JDBC 驅動程式版本，AWS Glue 爬蟲程式將使用 AWS Glue 任務和 Amazon S3 儲存貯體中的資源，以確保您提供的驅動程式在環境中執行。帳戶中將反映資源的額外使用量。此外，提供您的 JDBC 驅動程式，並不代表爬蟲程式能夠運用驅動程式的所有功能。驅動程式僅限於新增 AWS Glue 連線中所述的屬性。

設定爬行者程式

1. 請登入 AWS Management Console 並開啟AWS Glue主控台，網址為 https://console.aws.amazon.com/glue/。在導覽窗格中選擇 Crawlers (爬蟲程式)。
2. 選擇建立爬蟲程式，然後依照新增爬蟲程式精靈中的說明進行操作。精靈會引導您建立爬行者程式所需的步驟。如果您想要新增自訂的呼叫符號來定義結構描述，請參閱。將分類器新增至 AWS Glue 中的爬蟲程式

步驟 1：設定爬蟲程式屬性

輸入爬蟲程式的名稱和描述 (選用)。或者，您可以使用 Tag key (標籤金鑰) 和選用的 Tag value (標籤值)，標記您的爬蟲程式。建立之後，標籤金鑰為唯讀。應用標籤至某些資源，有助於您對其進行整理和識別。如需詳細資訊，請參閱中的 AWS 標籤 AWS Glue。

名稱

名稱可包含字母 (A-Z)、數字 (0-9)、連字號 (-) 或底線 (_)，並且長度上限為 255 個字元。

描述

說明最多可有 2,048 個字元。

標籤

您可以使用標籤整理和識別資源。如需詳細資訊，請參閱下列內容：

- AWS 中的標籤 AWS Glue

步驟 2：選擇資料來源和分類器

資料來源組態

為您的資料是否已對映至AWS Glue表格？ 選擇「尚未」或「是」。依預設已選取「尚未」。

爬蟲程式可以直接存取資料存放區做為抓取來源，也可以使用現有的 Data Catalog 資料表做為來源。如果爬蟲程式使用現有的目錄資料表，即會抓取這些目錄資料表指定的資料存放區。

- 尚未：選取一或多個要網路爬取的資料來源。爬蟲程式可以在單一執行中抓取多個不同類型的資料存放區 (Amazon S3,、JDBC 等)。

  您一次只可設定一個資料存放區。在您提供連線資訊並包含路徑和排除樣式之後，接著可以選擇加入其他資料存放區。

- 是：從 AWS Glue Data Catalog 中選取現有資料表。指定要抓取之資料存放區的目錄資料表。爬蟲程式只能在單一執行中抓取目錄資料表，而無法混合其他來源類型。

  指定目錄資料表做為來源的常見原因是您已手動建立資料表 (因為您已經知道資料存放區的結構)，並希望爬蟲程式讓資料表持續更新，包括新增新的分割區。如需了解其他原因的說明，請參閱使用爬蟲程式更新手動建立的資料目錄資料表。

  當您指定現有資料表做為爬蟲程式來源類型時，則適用以下條件：

  - 資料庫名稱是選用的。
  - 只允許指定 Amazon S3 或 Amazon DynamoDB 資料存放區的目錄資料表。
  - 爬蟲程式執行時，不會建立任何新的目錄資料表。系統會視需要更新現有的資料表，包括新增新的分割區。
  - 系統會忽略資料存放區中找到的已刪除物件，而不會刪除任何目錄資料表。反之，爬蟲程式會寫入日誌訊息。(`SchemaChangePolicy.DeleteBehavior=LOG`)
  - 依預設，系統會啟用爬蟲程式組態選項以建立每個 Amazon S3 路徑的單一結構描述，且無法停用。(`TableGroupingPolicy=CombineCompatibleSchemas`) 如需詳細資訊，請參閱如何為每個 Amazon S3 包含路徑建立單一結構描述。

- 您無法將含有任何其他來源類型的目錄資料表混合為來源 (例如 Amazon S3 或 Amazon DynamoDB)。

資料來源

選取或新增爬蟲程式要掃描的資料來源清單。

(選用) 如果您選擇 JDBC 作為資料來源，則在指定存放驅動程式資訊的連線存取時，可以使用自己的 JDBC 驅動程式。

包含路徑

在評估抓取要包含或排除的內容時，爬蟲程式會先評估必要的包含路徑。對於 Amazon S3、MongoDB、MongoDB Atlas、Amazon DocumentDB (with MongoDB compatibility) 和關聯式資料存放區，您必須指定包含路徑。

對於 Amazon S3 資料存放區

選擇是否要在此帳戶或在其他帳戶中指定路徑，然後瀏覽以選擇 Amazon S3 路徑。

對於 Amazon S3 資料存放區，包含路徑語法是 `bucket-name/folder-name/file-name.ext`。要探索儲存貯體中的所有物件，僅指定在包含路徑的儲存貯體名稱。排除模式是相對於包含路徑。

對於 Delta Lake 的資料儲存

將一個或多個指向 Delta 資料表的 Amazon S3 路徑指定為：s3://*bucket*/*prefix*/*object*。

對於 Iceberg 或 Hudi 資料存放區

將一個或多個包含資料夾且資料夾中具有 Iceberg 或 Hudi 資料表中繼資料的 Amazon S3 路徑，指定為 s3://####/##。

對於 Hudi 資料存放區，Hudi 資料夾可能位於根資料夾的子資料夾中。爬蟲程式將掃描路徑下所有資料夾中的 Hudi 資料夾。

對於 JDBC 資料存放區

輸入 *<database>*/*<schema>*/*<table>* 或 *<database>*/*<table>*，視資料庫產品而定。Oracle 資料庫和 MySQL 不支援路徑中的結構描述。您可以用百分比 (%) 字元取代 *<schema>* 或 *<table>*。例如，對於系統識別碼 (SID) 為 `orcl` 的 Oracle 資料庫，請輸入 `orcl/%` 以匯入連線中命名使用者有權存取的所有資料表。

> **Important**
> 此欄位會區分大小寫。

對於 MongoDB、MongoDB Atlas 或 Amazon DocumentDB 資料存放區

輸入 ###/##。

對於 MongoDB、MongoDB Atlas 和 Amazon DocumentDB (with MongoDB compatibility)，語法是 `database/collection`。

對於 JDBC 資料存放區，語法為 `database-name/schema-name/table-name` 或 `database-name/table-name`。語法取決於資料庫引擎在資料庫中是否支援結構描述。例如，對於資料庫引擎 (如 MySQL 或 Oracle)，請勿在包含路徑中指定 `schema-name`。您可以在包含路徑中針對結構描述或資料表替代百分比符號 (%)，以代表資料庫中的所有結構描述或所有資料表。您不能替代包含路徑中資料庫的百分比符號 (%)。

橫向深度上限 (僅適用於 Iceberg 或 Hudi 資料存放區)

定義爬蟲程式可在 Amazon S3 路徑中周遊探索 Iceberg 或 Hudi 中繼資料資料夾的 Amazon S3 路徑深度上限。此參數的用途是限制爬蟲程式執行時間。預設值為 10，最大值為 20。

排除模式

這些模式可讓您排除抓取特定檔案或資料表。排除路徑是相對於包含路徑。例如，若要在您的 JDBC 資料存放區中排除資料表，請在排除路徑中輸入資料表的名稱。

爬蟲程式會使用 AWS Glue 連線 (其中包含 JDBC URI 連線字串) 連接到 JDBC 資料存放區。爬蟲程式只可以使用 AWS Glue 連線中的 JDBC 使用者名稱和密碼來存取資料庫引擎中的物件。爬蟲程式只能建立其可透過 JDBC 連線存取的表格。在爬蟲程式使用 JDBC URI 存取資料庫引擎後，您可使用包括路徑來判斷資料庫引擎中哪些資料表是在 Data Catalog 中所建立的。例如，若使用 MySQL，如果您指定 `MyDatabase/%` 的包括路徑，則 `MyDatabase` 中的所有資料表會在 Data Catalog 中建立。存取 Amazon Redshift 時，如果您指定 `MyDatabase/%` 的包括路徑，則資料庫 `MyDatabase` 所有結構描述中的所有資料表會在 Data Catalog 中建立。如果您指定 `MyDatabase/MySchema/%` 的包括路徑，則資料庫 `MyDatabase` 中的所有資料表和結構描述 `MySchema` 會加以建立。

在指定包含路徑後，您可以指定一個或多個 Unix 樣式的 `glob` 排除模式，從包含路徑包含的抓取中排除物件。這些模式會套用至您的包含路徑，以判斷哪些物件被排除。這些模式也

會在爬蟲程式所建立的表格中儲存為屬性。AWS Glue PySpark 副檔名 (例如) 會讀取表格屬性`create_dynamic_frame.from_catalog`，並排除排除由排除模式定義的物件。

AWS Glue 支援以下在排除模式中的 glob 模式。

排除模式	描述
`*.csv`	符合目前資料夾中代表以 `.csv` 結尾之物件名稱的 Amazon S3 路徑
`*.*`	符合所有包含點的物件名稱
`*.{csv,avro}`	符合以 `.csv` 或 `.avro` 結尾的物件名稱
`foo.?`	符合以 `foo.` 開頭並由單一字元副檔名接續的物件名稱
`myfolder/*`	符合位於 `myfolder` 下一層子資料夾中的物件，例如 `/myfolder/mysource`
`myfolder/*/*`	符合位於 `myfolder` 下兩層子資料夾中的物件，例如 `/myfolder/mysource/data`
`myfolder/**`	符合位於 `myfolder` 所有子資料中的物件，例如 `/myfolder/mysource/mydata` 和 `/myfolder/mysource/data`
`myfolder**`	符合資料夾 `myfolder` 以及 `myfolder` 下的檔案，例如 `/myfolder` 和 `/myfolder/mydata.txt`
`Market*`	符合 JDBC 資料庫中以 `Market` 開頭為名稱的資料表，例如 `Market_us` 和 `Market_fr`

AWS Glue 對 glob 排除模式的解譯如下：

- 斜線 (/) 字元為分隔符號，可將 Amazon S3 金鑰分隔至資料夾階層。
- 星號 (*) 字元符合不超過資料夾邊界之名稱元件的零個或多個字元。
- 雙星號 (**) 符合超過資料夾或結構描述邊界的零個或多個字元。

- 問號 (?) 字元符合名稱元件的單一字元。
- 反斜線 (\) 字元用於逸出可被解譯為特殊字元的字元。表達式 \\ 符合一個反斜線，而 \{ 符合左大括弧。
- 方括號 [ ] 會建立括號表達式，符合一組字元中一個名稱元件的單一字元。例如，[abc] 符合 a、b 或 c。連字號 (-) 可用於指定範圍，因此 [a-z] 指定的範圍符合從 a 到 z (包含)。這些形式可以混合，因此 [abce-g] 符合 a、b、c、e、f 或 g。如果方括號 ([) 後方的字元為驚嘆號 (!)，則此括號表達式為否定。例如，[!a-c] 符合 a、b 或 c 以外的字元。

  在括號表達式中，*、? 和 \ 字元符合本身。如果連字號 (-) 是方括號內的第一個字元，則其符合本身，如果連字號是 ! 後的第一個字元，則為否定。
- 大括弧 ({ }) 會括住一組子模式，如果群組中的任何子模式符合，則此群組也符合。逗號 (,) 字元用於分隔子模式。群組不能巢狀組合。
- 前置句點或點號字元在相符操作中會視為一般字元。例如，* 排除模式符合檔案名稱 .hidden。

Example Amazon S3 排除模式

每個排除模式都會以包含路徑來評估。例如，假設您有以下 Amazon S3 目錄結構：

```
/mybucket/myfolder/
   departments/
      finance.json
      market-us.json
      market-emea.json
      market-ap.json
   employees/
      hr.json
      john.csv
      jane.csv
      juan.txt
```

包含路徑為 s3://mybucket/myfolder/，而以下為排除模式的一些範例結果：

排除模式	結果
departments/**	排除 departments 以下所有檔案和資料夾，並包含 employees 資料夾及其檔案
departments/market*	排除 market-us.json 、market-emea.json 和 market-ap.json

排除模式	結果
`**.csv`	排除 `myfolder` 以下名稱以 `.csv` 結尾的所有物件
`employees/*.csv`	排除 `employees` 資料夾中所有的 `.csv` 檔案

Example 排除 Amazon S3 分割區的子集

假設您的資料以天為單位分割，那麼一年中的每一天都是一個獨立的 Amazon S3 分割區。2015 年 1 月有 31 個分割區。如果現在只要抓取 1 月第一週的資料，您就必須排除第 1 至第 7 天以外的所有分割區：

```
 2015/01/{[!0],0[8-9]}**, 2015/0[2-9]/**, 2015/1[0-2]/**
```

讓我們看看此全域模式的部分。第一部分的 `2015/01/{[!0],0[8-9]}**` 排除了 2015 年第 01 個月不以「0」開頭的所有天數 (除了第 08 天和第 09 天)。請注意，「**」的用途是天數模式的尾碼，會超過資料夾邊界至較低層級的資料夾。如果使用「*」，則較低的資料夾層級不會被排除。

第二部分的 `2015/0[2-9]/**` 會排除 2015 年第 02 至 09 個月裡的天數。

第三部分的 `2015/1[0-2]/**` 會排除 2015 年第 10、11、12 個月裡的天數。

Example JDBC 排除模式

假設您以下列的結構描述架構爬取 JDBC 資料庫：

```
MyDatabase/MySchema/
   HR_us
   HR_fr
   Employees_Table
   Finance
   Market_US_Table
   Market_EMEA_Table
   Market_AP_Table
```

包含路徑為 `MyDatabase/MySchema/%`，而以下為排除模式的一些範例結果：

排除模式	結果
HR*	排除名稱以 HR 開頭的資料表
Market_*	排除名稱以 Market_ 開頭的資料表
**_Table	排除所有名稱以 _Table 結尾的資料表

其他爬蟲程式來源參數

每個來源類型都需要一組不同的附加參數。以下提供非完整清單：

連線

選取或新增 AWS Glue 連線。如需連線的詳細資訊，請參閱 連線至資料。

其他中繼資料 – 選用 (適用於 JDBC 資料存放區)

選取要網路爬取的其他爬蟲程式中繼資料屬性。

- 註解：網路爬取相關的資料表層級和資料欄層級註解。
- 原始類型：將資料表資料欄的原始資料類型保存在其他中繼資料中。作為預設行為，爬蟲程式會將原始資料類型轉換為 Hive 相容的類型。

JDBC 驅動程式類別名稱：選用 (適用於 JDBC 資料存放區)

針對連線至資料來源的爬蟲程式，輸入自訂 JDBC 驅動程式類別名稱：

- Postgres：org.postgresql.Driver
- MySQL：com.mysql.jdbc.Driver、com.mysql.cj.jdbc.Driver
- Redshift：com.amazon.redshift.jdbc.Driver、com.amazon.redshift.jdbc42.Driver
- 甲骨文：甲骨文 .jdbc. 驅動程序。 OracleDriver
- SQL 服務器：微軟 SQL 服務器。ServerDriver

JDBC 驅動程式 S3 路徑：選用 (適用於 JDBC 資料存放區)

選擇 .jar 檔案的現有 Amazon S3 路徑。此為針對連線至資料來源的爬蟲程式使用自訂 JDBC 驅動程式時，用來存放 .jar 檔案的所在位置。

啟用資料取樣 (僅適用於 Amazon DynamoDB、MongoDB、MongoDB Atlas 和 Amazon DocumentDB 資料存放區)

選取是否只編目資料範例。如果未選取，則會編目整個資料表。當資料表不是高傳輸量資料表時，掃描所有記錄可能需要很長的時間。

建立用於查詢的資料表 (僅適用於 Delta Lake 資料存放區)

選取建立 Delta Lake 資料表的方式：

- 建立原生資料表：允許與支援直接查詢 Delta 交易日誌的查詢引擎整合。
- 建立符號連結資料夾：根據特定組態參數，使用由分割區索引鍵分割的資訊清單檔案建立符號連結清單檔案資料夾。

掃描速率 – 選用 (僅適用於 DynamoDB 資料存放區)

指定爬蟲程式要使用的 DynamoDB 資料表讀取容量單位百分比。讀取容量單位是 DynamoDB 定義的術語，此數值可作為每秒可在該資料表上執行的讀取次數速率限制符號。請輸入介於 0.1 到 1.5 之間的值。如未指定，則會將已佈建的資料表預設為 0.5%，並將隨需資料表預設為 1/4 的最大設定容量。請注意，只有佈建的容量模式應該與 AWS Glue 編目器搭配使用。

> **Note**
>
> 對於 DynamoDB 的資料儲存，請設定已佈建的容量模式，以便處理資料表的讀取和寫入。AWS Glue 爬行者程式不應與隨選容量模式搭配使用。

網路連線 – 選用 (僅適用於 Amazon S3 資料存放區)

選擇是否包含要用於此 Amazon S3 目標的網路連線。請注意，每個爬蟲程式僅限於一個網路連線，因此任何其他 Amazon S3 目標也會使用相同的連線 (如果保留空白，則無連線)。

如需連線的詳細資訊，請參閱 連線至資料。

僅對檔案的子集和樣本大小進行取樣 (僅適用於 Amazon S3 資料存放區)

指定在資料集中網路爬取範例檔案時，每個分葉資料夾中要編目的檔案數目。開啟此功能時，爬蟲程式會隨機選取每個分葉資料夾中要網路爬取的某些檔案，而不是網路爬取此資料集中的所有檔案。

取樣爬蟲程式最適合先前了解其資料格式，並且知道資料夾中的結構描述不會變更的客戶。開啟此功能會大幅減少爬蟲程式執行時間。

有效值是介於 1 到 249 之間的整數。如果未指定，則會網路爬取所有檔案。

後續爬蟲程式執行

此欄位是會影響所有 Amazon S3 資料來源的全域欄位。

- 網路爬取所有子資料夾：在每次後續網路爬取時，再次網路爬取所有資料夾。
- 僅網路爬取新的子資料夾：只會網路爬取自上次網路爬取以來新增的 Amazon S3 資料夾。如果結構描述相容，則會將新的分割區新增至現有資料表。如需詳細資訊，請參閱 the section called "用於新增分割區的增量編目"。
- 根據事件進行網路爬取：依賴 Amazon S3 事件來控制要網路爬取的資料夾。如需詳細資訊，請參閱 the section called "使用 Amazon S3 事件通知加速網路爬取"。

自訂分類器 – 選用

定義自訂分類器，再定義爬蟲程式。分類器會檢查指定的檔案是否採用爬蟲程式可以處理的格式。如果是，則分類器會以符合該資料格式的 `StructType` 物件形式，建立結構描述。

如需詳細資訊，請參閱 將分類器新增至 AWS Glue 中的爬蟲程式。

步驟 3：設定安全設定

IAM 角色

爬蟲程式會擔任此角色。它必須具有類似於 AWS 受管理策略的權限`AWSGlueServiceRole`。對於 Amazon S3 和 DynamoDB 來源，它也必須具有存取資料存放區的許可。如果爬蟲讀取使用 AWS Key Management Service (AWS KMS) 加密的 Amazon S3 資料，則該角色必須具有 AWS KMS 金鑰的解密許可。

對於 Amazon S3 資料存放區，連接到角色的其他許可將類似下列內容：

```
{
   "Version": "2012-10-17",
    "Statement": [
        {
          "Effect": "Allow",
          "Action": [
              "s3:GetObject",
              "s3:PutObject"
          ],
          "Resource": [
              "arn:aws:s3:::bucket/object*"
```

```
            ]
        }
    ]
}
```

對於 Amazon DynamoDB 資料存放區，連接到角色的其他許可將類似下列內容：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DescribeTable",
        "dynamodb:Scan"
      ],
      "Resource": [
        "arn:aws:dynamodb:region:account-id:table/table-name*"
      ]
    }
  ]
}
```

若要新增自己的 JDBC 驅動程式，需要新增其他許可。

- 授予下列任務動作的許可
  可：`CreateJob`、`DeleteJob`、`GetJob`、`GetJobRun`、`StartJobRun`。
- 授予 Amazon S3 動作的許可
  可：`s3:DeleteObjects`、`s3:GetObject`、`s3:ListBucket`、`s3:PutObject`。

> **Note**
>
> 如果 Amazon S3 儲存貯體政策已停用，則不需要 `s3:ListBucket`。

- 在 Amazon S3 政策中授予服務主體對儲存貯體/資料夾的存取權。

Amazon S3 政策範例：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name/driver-parent-folder/driver.jar",
                "arn:aws:s3:::bucket-name"
            ]
        }
    ]
}
```

AWS Glue 建立以下資料夾 (`_crawler` 和 `_glue_job_crawler` 位於與 Amazon S3 儲存貯體中 JDBC 驅動程式相同的層級)。例如，如果驅動程式路徑為 `<s3-path/driver_folder/driver.jar>`，則如果下列資料夾尚不存在，則會建立這些資料夾：

- <s3-path/driver_folder/_crawler>
- <s3-path/driver_folder/_glue_job_crawler>

或者，您可以新增安全組態至爬蟲程式以指定靜態加密選項。

如需詳細資訊，請參閱 步驟 2：為 AWS Glue 建立 IAM 角色 及 AWS Glue 的身分識別與存取管理。

Lake Formation 組態 – 選用

允許爬蟲程式使用 Lake Formation 憑證來網路爬取資料來源。

核取 Use Lake Formation credentials for crawling S3 data source (使用 Lake Formation 憑證網路爬取 S3 資料來源)，可讓爬蟲程式使用 Lake Formation 憑證來網路爬取資料來源。如果資料來源屬於其他帳戶，您必須提供註冊的帳戶 ID。否則，爬蟲程式只會網路爬取與帳戶相關聯的那些資料來源。僅適用於 Amazon S3 和 Data Catalog 資料來源。

安全組態 – 選用

設定包括安全組態。如需詳細資訊，請參閱下列內容：

- 對 AWS Glue 寫入的資料加密

> **Note**
>
> 在爬行者程式上設定安全性組態之後，您就可以進行變更，但無法將其移除。若要降低爬行者程式的安全層級，請在組態DISABLED中明確地將安全性功能設定為，或建立新的爬行者程式。

步驟 4：設定輸出和排程

輸出組態

這些選項包括當偵測到資料存放區的結構描述變更、刪除物件等情況，爬蟲程式應該如何處理。如需更多資訊，請參閱自訂爬行者程式行為

爬蟲程式排程

您可以在 AWS Glue中隨需執行爬蟲程式或為爬蟲程式和任務定義以時間為基礎的排程。這些排程的定義使用類似 Unix 的 cron 語法。如需詳細資訊，請參閱 排程 AWS Glue 爬蟲程式。

步驟 4：檢閱和建立

檢閱您設定的爬蟲程式設定，然後建立爬蟲程式。

將分類器新增至 AWS Glue 中的爬蟲程式

分類器讀取資料存放區中的資料。如果它能辨識資料的格式，將會產生結構描述。分類器也會傳回一個確定數字以表示所識別格式的確定程度。

AWS Glue 提供一組內建分類器，但您也可以建立自訂分類器。AWS Glue 會先依照您在爬蟲程式定義中指定的順序叫用自訂分類器。根據從自訂分類器傳回的結果，AWS Glue 也可能叫用內建分類器。如果分類器在處理期間傳回 `certainty=1.0`，表示它 100% 確定可以建立正確的結構描述。然後，AWS Glue 將使用此分類器的輸出。

如果沒有分類器傳回 `certainty=1.0`，AWS Glue 將使用確定程度最高的分類器的輸出。如果沒有分類器傳回大於 `0.0` 的確定程度，AWS Glue 將傳回預設的分類字串 `UNKNOWN`。

我應該在何時使用分類器？

當您在 AWS Glue Data Catalog 中探索資料存放區以定義中繼資料資料表時，可使用分類器。您可以用一組已排序的分類器來設定您的爬蟲程式。當爬蟲程式叫用分類器時，分類器會判斷資料是否能夠辨

識。如果分類器無法識別資料或無法 100% 確定，爬蟲程式將叫用清單中的下一個分類器，以判斷它是否能夠識別資料。

如需使用 AWS Glue 主控台建立分類器的詳細資訊，請參閱在 AWS Glue 主控台上使用分類器。

自訂分類器

分類器的輸出包含一個字串，它會指出檔案的分類或格式 (例如，json) 以及檔案結構描述。對於自訂分類器，您可以根據分類器的類型，定義建立結構描述的邏輯。分類器類型包括根據 grok 模式、XML 標籤以及 JSON 路徑定義結構描述。

如果您變更分類器定義，之前使用分類器探索的任何資料將不會重新分類。爬蟲程式會持續追蹤之前探索的資料。會使用更新分類器來分類新資料，這可能會更新結構描述。如果資料的結構描述擴張，則於爬蟲程式執行時將分類器更新到任何結構描述變更的帳戶。若要重新分類資料以更正不正確的分類器，請使用更新的分類器建立新爬蟲程式。

如需在 AWS Glue 中建立自訂分類器的詳細資訊，請參閱撰寫自訂分類器。

> **Note**
>
> 如果您的資料格式能夠由內建分類器之一加以辨識，您將無需建立自訂分類器。

AWS Glue 中的內建分類器

AWS Glue 提供各種格式的內建分類器，包括 JSON、CSV、Web 日誌，以及許多資料庫系統。

如果 AWS Glue 找不到能 100% 確定符合輸入資料格式的自訂分類器，將會依照下表所列順序叫用內建分類器。內建分類器傳回結果，指出格式是否符合 (`certainty=1.0`) 或不符合 (`certainty=0.0`)。第一個具有 `certainty=1.0` 的分類器將為 Data Catalog 的中繼資料資料表提供分類字串和結構描述。

分類器類型	分類字串	備註
Apache Avro	`avro`	讀取檔案開頭的結構描述以判斷格式。
Apache ORC	`orc`	讀取檔案中繼資料以判斷格式。
Apache Parquet	`parquet`	讀取檔案結尾的結構描述以判斷格式。
JSON	`json`	讀取檔案的開頭以判斷格式。

分類器類型	分類字串	備註
Binary JSON	`bson`	讀取檔案的開頭以判斷格式。
XML	`xml`	讀取檔案的開頭以判斷格式。AWS Glue 根據文件中的 XML 標籤來判斷資料表結構描述。 如需建立自訂 XML 分類器以指定文件中的列的詳細資訊，請參閱 撰寫 XML 自訂分類器。
Amazon Ion	`ion`	讀取檔案的開頭以判斷格式。
結合的 Apache 日誌	`combined_apache`	透過 grok 模式決定日誌格式。
Apache 日誌	`apache`	透過 grok 模式決定日誌格式。
Linux 核心日誌	`linux_kernel`	透過 grok 模式決定日誌格式。
Microsoft 日誌	`microsoft_log`	透過 grok 模式決定日誌格式。
Ruby 日誌	`ruby_logger`	讀取檔案的開頭以判斷格式。
Squid 3.x 日誌	`squid`	讀取檔案的開頭以判斷格式。
Redis 監控日誌	`redismonlog`	讀取檔案的開頭以判斷格式。
Redis 日誌	`redislog`	讀取檔案的開頭以判斷格式。
CSV	`csv`	檢查以下分隔符號：逗號 (,)、直立線符號 (|)、Tab (\t)、分號 (;) 和 Ctrl-A (\u0001)。Ctrl-A 為 `Start Of Heading` 的 Unicode 控制字元。
Amazon Redshift	`redshift`	使用 JDBC 連線來匯入中繼資料。
MySQL	`mysql`	使用 JDBC 連線來匯入中繼資料。
PostgreSQL	`postgresql`	使用 JDBC 連線來匯入中繼資料。
Oracle 資料庫	`oracle`	使用 JDBC 連線來匯入中繼資料。

分類器類型	分類字串	備註
Microsoft SQL Server	`sqlserver`	使用 JDBC 連線來匯入中繼資料。
Amazon DynamoDB	`dynamodb`	從 DynamoDB 資料表讀取資料。

可分類以下壓縮格式的檔案：

- ZIP (僅包含單一檔案的封存可支援)。請注意，Zip 在其他服務中並未完整支援 (因為封存的關係)。
- BZIP
- GZIP
- LZ4
- Snappy (支援標準和 Hadoop 原生 Snappy 格式)

內建 CSV 分類器

內建 CSV 分類器會剖析 CSV 檔案內容，以判斷 AWS Glue 資料表的結構描述。此分類器會檢查以下分隔符號：

- 逗號 (,)
- 管道 (|)
- Tab (\t)
- 分號 (;)
- Ctrl-A (\u0001)

  Ctrl-A 為 `Start Of Heading` 的 Unicode 控制字元。

若要被歸類為 CSV，資料表結構描述必須至少有兩個資料欄和兩個資料列。CSV 分類器使用多種啟發，以判斷在特定檔案中標頭是否存在。如果分類器無法判斷第一列資料的標頭，欄標頭會顯示為 `col1`、`col2`、`col3`，以此類推。內建 CSV 分類器會透過評估檔案的以下特點判斷是否要推斷標頭：

- 在潛在標頭中的每個欄位剖析為 STRING 資料類型。

- 除了最後一個欄位，每個在潛在標頭中內容少於 150 個字元的欄位。若要允許結尾為分隔符號，整個檔案的最後一個欄位可以是空的。
- 在潛在標頭中的每個欄必須符合欄位名稱的 AWS Glue `regex` 要求。
- 標頭資料列必須與資料列有足夠的差異。若要判斷此項目，一或多個資料列必須剖析為其他 STRING 類型。如果所有欄位的類型為 STRING，則第一列資料與用做為標頭的後續資料列的差異不足夠。

> **Note**
>
> 如果內建的 CSV 分類器無法如您想要地建立 AWS Glue 資料表，您可以使用以下其中一個替代選項：
>
> - 變更在 Data Catalog 中的欄名稱，將 `SchemaChangePolicy` 設定為 LOG，並將分區輸出配置設定為 `InheritFromTable` 以供未來爬蟲程式執行使用。
> - 建立自訂 grok 分類器來剖析資料並指派您想要的欄。
> - 內建 CSV 分類器會建立資料表，參考 `LazySimpleSerDe` 做為序列化的程式庫，這是類型推導的最佳選擇。不過，如果 CSV 資料包含引用字串，編輯資料表定義並將 SerDe 程式庫變更為 `OpenCSVSerDe`。將任何推導類型調整為 STRING，將 `SchemaChangePolicy` 設定為 LOG，並將分區輸出配置設定為 `InheritFromTable` 以供未來爬蟲程式執行使用。如需 SerDe 程式庫的更多資訊，請參閱《Amazon Athena 使用者指南》中的 SerDe 參考。

撰寫自訂分類器

您可以提供自訂分類器，分類在 AWS Glue 中的資料。您可以使用 grok 模式、XML 標籤、JavaScript Object 符號 (JSON) 或逗號分隔值 (CSV)，建立自訂分類器。AWS Glue 爬蟲程式呼叫自訂分類器。如果分類器能夠辨識資料，它會將資料的分類和結構描述傳回爬蟲程式。如果您的資料不符合任何內建的分類器，或者想要自訂由爬蟲程式建立的資料表，那麼您可能需要定義自訂分類器。

如需使用 AWS Glue 主控台建立分類器的詳細資訊，請參閱在 AWS Glue 主控台上使用分類器。

AWS Glue 會依照您指定的順序，在內建的分類器之前執行自訂分類器。當爬蟲程式找到符合資料的分類器時，分類字串和結構描述將用於資料表的定義中，這些資料表將寫入至您的 AWS Glue Data Catalog。

主題

- 撰寫 grok 自訂分類器

- 撰寫 XML 自訂分類器
- 撰寫 JSON 自訂分類器
- 撰寫 CSV 自訂分類器

撰寫 grok 自訂分類器

Grok 是一種工具，用於剖析指定相符模式的文字資料。grok 模式是一組命名規則表達式 (regex)，用於每次比對一行。AWS Glue 使用 grok 模式來推斷資料的結構描述。當 grok 模式符合您的資料時，AWS Glue 將使用此模式判斷您資料的結構，並將其映射到欄位。

AWS Glue 提供許多內建模式，或者您可以定義自己的模式。您可以在自訂分類器定義中使用內建模式和自訂模式來建立 grok 模式。您可以量身打造 grok 模式來分類自訂文字檔案格式。

> **Note**
>
> 針對 AWS Glue Data Catalog 中建立的資料表，AWS Glue Grok 自訂分類器會使用 `GrokSerDe` 序列化程式庫。因此，如果您正在使用 AWS Glue Data Catalog 搭配 Amazon Athena、Amazon EMR 或 Redshift Spectrum，則請參閱這些服務的文件，以取得 `GrokSerDe` 的支援資訊。目前，當您從 Amazon EMR 和 Redshift 查詢利用 `GrokSerDe` 建立的資料表時，可能會遇到問題。

以下是 grok 模式元件的基本語法：

```
%{PATTERN:field-name}
```

符合具名的 `PATTERN` 的資料會映射到結構描述中的 `field-name` 欄位符合，預設資料類型為 `string`。除此之外，您亦可選擇在產生的結構描述中，將欄位的資料類型轉換為 `byte`、`boolean`、`double`、`short`、`int`、`long` 或 `float`。

```
%{PATTERN:field-name:data-type}
```

例如，若要將 `num` 欄位轉換為 `int` 資料類型，您可以使用此模式：

```
%{NUMBER:num:int}
```

模式可由其他模式組成。舉例而言，您可使用月、日及時間的模式來定義 SYSLOG 時間戳記的模式 (例如 `Feb 1 06:25:43`)。您能夠使用下列模式來定義此資料：

```
SYSLOGTIMESTAMP %{MONTH} +%{MONTHDAY} %{TIME}
```

> **Note**
>
> Grok 模式每次只能處理一行。不支援多行模式。此外，不支援在模式中的換行。

AWS Glue 中的自訂分類器值

當您定義 grok 分類器時，您將以下值提供至 AWS Glue 以建立自訂分類器。

Name

分類器名稱。

分類

寫入的字串用於描述分類資料的格式；例如 `special-logs`。

Grok 模式

套用到資料存放區的一組模式可判斷是否符合。這些模式皆來自 AWS Glue 內建模式，以及您所定義的任何自訂模式。

以下為 grok 模式的範例：

```
%{TIMESTAMP_ISO8601:timestamp} \[%{MESSAGEPREFIX:message_prefix}\]
 %{CRAWLERLOGLEVEL:loglevel} : %{GREEDYDATA:message}
```

當資料符合 `TIMESTAMP_ISO8601`，將會建立結構描述欄位 `timestamp`。此行為類似範例中的其他具名模式。

自訂模式

您定義的選用自訂模式。分類您的資料的 grok 模式會參考這些模式。您可以在套用至您資料的 grok 模式中參考這些自訂模式。每個自訂元件模式都必須位於不同的行。規則表達式 (regex) 語法可用來定義模式。

以下是使用指定模式的範例：

```
CRAWLERLOGLEVEL (BENCHMARK|ERROR|WARN|INFO|TRACE)
MESSAGEPREFIX .*-.*-.*-.*-.*
```

當資料符合其中一個列舉字串時，第一個自訂具名模式 CRAWLERLOGLEVEL 即為相符。第二個自訂模式 MESSAGEPREFIX 嘗試符合訊息字首字串。

AWS Glue 追蹤建立時間、上次更新時間和您的分類器版本。

AWS Glue 內建模式

AWS Glue 提供許多常見的模式，您可用來建立自訂分類器。您將一個具名模式新增到分類器定義中的 grok pattern。

以下清單包含各個模式的一行。在每一行中，模式名稱之後是它的定義。規則表達式 (regex) 語法可用來定義模式。

```
#<noloc>&GLU;</noloc> Built-in patterns
 USERNAME [a-zA-Z0-9._-]+
 USER %{USERNAME:UNWANTED}
 INT (?:[+-]?(?:[0-9]+))
 BASE10NUM (?<![0-9.+-])(?>[+-]?(?:(?:[0-9]+(?:\.[0-9]+)?)|(?:\.[0-9]+)))
 NUMBER (?:%{BASE10NUM:UNWANTED})
 BASE16NUM (?<![0-9A-Fa-f])(?:[+-]?(?:0x)?(?:[0-9A-Fa-f]+))
 BASE16FLOAT \b(?<![0-9A-Fa-f.])(?:[+-]?(?:0x)?(?:(?:[0-9A-Fa-f]+(?:\.[0-9A-Fa-f]*)?)|
(?:\.[0-9A-Fa-f]+)))\b
 BOOLEAN (?i)(true|false)

 POSINT \b(?:[1-9][0-9]*)\b
 NONNEGINT \b(?:[0-9]+)\b
 WORD \b\w+\b
 NOTSPACE \S+
 SPACE \s*
 DATA .*?
 GREEDYDATA .*
 #QUOTEDSTRING (?:(?<!\\)(?:"(?:\\.|[^\\"])*"|(?:'(?:\\.|[^\\'])*')|(?:`(?:\\.|[^\
\`])*`)))
 QUOTEDSTRING (?>(?<!\\)(?>"(?>\\.|[^\\"]+)+"|""|(?>'(?>\\.|[^\\']+)+')|''|(?>`(?>\\.|
[^\\`]+)+`)|``))
 UUID [A-Fa-f0-9]{8}-(?:[A-Fa-f0-9]{4}-){3}[A-Fa-f0-9]{12}

 # Networking
 MAC (?:%{CISCOMAC:UNWANTED}|%{WINDOWSMAC:UNWANTED}|%{COMMONMAC:UNWANTED})
 CISCOMAC (?:(?:[A-Fa-f0-9]{4}\.){2}[A-Fa-f0-9]{4})
```

```
 WINDOWSMAC (?:(?:[A-Fa-f0-9]{2}-){5}[A-Fa-f0-9]{2})
 COMMONMAC (?:(?:[A-Fa-f0-9]{2}:){5}[A-Fa-f0-9]{2})
 IPV6 ((([0-9A-Fa-f]{1,4}:){7}([0-9A-Fa-f]{1,4}|:))|(([0-9A-Fa-f]{1,4}:){6}(:[0-9A-
Fa-f]{1,4}|((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d))
{3})|:))|(([0-9A-Fa-f]{1,4}:){5}(((:[0-9A-Fa-f]{1,4}){1,2})|:((25[0-5]|2[0-4]\d|1\d
\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|(([0-9A-Fa-f]{1,4}:){4}(((:
[0-9A-Fa-f]{1,4}){1,3})|((:[0-9A-Fa-f]{1,4})?:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.
(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(([0-9A-Fa-f]{1,4}:){3}(((:[0-9A-Fa-f]
{1,4}){1,4})|((:[0-9A-Fa-f]{1,4}){0,2}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|
2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(([0-9A-Fa-f]{1,4}:){2}(((:[0-9A-Fa-f]{1,4}){1,5})|
((:[0-9A-Fa-f]{1,4}){0,3}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d
\d|[1-9]?\d)){3}))|:))|(([0-9A-Fa-f]{1,4}:){1}(((:[0-9A-Fa-f]{1,4}){1,6})|((:[0-9A-Fa-
f]{1,4}){0,4}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d))
{3}))|:))|(:(((:[0-9A-Fa-f]{1,4}){1,7})|((:[0-9A-Fa-f]{1,4}){0,5}:((25[0-5]|2[0-4]\d|
1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))(%.+)?
 IPV4 (?<![0-9])(?:(?:25[0-5]|2[0-4][0-9]|[0-1]?[0-9]{1,2})[.](?:25[0-5]|2[0-4][0-9]|
[0-1]?[0-9]{1,2})[.](?:25[0-5]|2[0-4][0-9]|[0-1]?[0-9]{1,2})[.](?:25[0-5]|2[0-4][0-9]|
[0-1]?[0-9]{1,2}))(?![0-9])
 IP (?:%{IPV6:UNWANTED}|%{IPV4:UNWANTED})
 HOSTNAME \b(?:[0-9A-Za-z][0-9A-Za-z-_]{0,62})(?:\.(?:[0-9A-Za-z][0-9A-Za-z-_]
{0,62}))*(\.?|\b)
 HOST %{HOSTNAME:UNWANTED}
 IPORHOST (?:%{HOSTNAME:UNWANTED}|%{IP:UNWANTED})
 HOSTPORT (?:%{IPORHOST}:%{POSINT:PORT})

 # paths
 PATH (?:%{UNIXPATH}|%{WINPATH})
 UNIXPATH (?>/(?>[\w_%!$@:.,~-]+|\\.)*)+
 #UNIXPATH (?<![\w\/])(?:/[^\/\s?*]*)+
 TTY (?:/dev/(pts|tty([pq])?)(\w+)?/?(?:[0-9]+))
 WINPATH (?>[A-Za-z]+:|\\)(?:\\[^\\?*]*)+
 URIPROTO [A-Za-z]+(\+[A-Za-z+]+)?
 URIHOST %{IPORHOST}(?::%{POSINT:port})?
 # uripath comes loosely from RFC1738, but mostly from what Firefox
 # doesn't turn into %XX
 URIPATH (?:/[A-Za-z0-9$.+!*'(){},~:;=@#%_\-]*)+
 #URIPARAM \?(?:[A-Za-z0-9]+(?:=(?:[^&]*))?(?:&(?:[A-Za-z0-9]+(?:=(?:[^&]*))?)?)*)?
 URIPARAM \?[A-Za-z0-9$.+!*'|(){},~@#%&/=:;_?\-\[\]]*
 URIPATHPARAM %{URIPATH}(?:%{URIPARAM})?
 URI %{URIPROTO}://(?:%{USER}(?::[^@]*)?@)?(?:%{URIHOST})?(?:%{URIPATHPARAM})?

 # Months: January, Feb, 3, 03, 12, December
 MONTH \b(?:Jan(?:uary)?|Feb(?:ruary)?|Mar(?:ch)?|Apr(?:il)?|May|Jun(?:e)?|Jul(?:y)?|
Aug(?:ust)?|Sep(?:tember)?|Oct(?:ober)?|Nov(?:ember)?|Dec(?:ember)?)\b
```

```
 MONTHNUM (?:0?[1-9]|1[0-2])
 MONTHNUM2 (?:0[1-9]|1[0-2])
 MONTHDAY (?:(?:0[1-9])|(?:[12][0-9])|(?:3[01])|[1-9])

 # Days: Monday, Tue, Thu, etc...
 DAY (?:Mon(?:day)?|Tue(?:sday)?|Wed(?:nesday)?|Thu(?:rsday)?|Fri(?:day)?|
Sat(?:urday)?|Sun(?:day)?)

 # Years?
 YEAR (?>\d\d){1,2}
 # Time: HH:MM:SS
 #TIME \d{2}:\d{2}(?::\d{2}(?:\.\d+)?)?
 # TIME %{POSINT<24}:%{POSINT<60}(?::%{POSINT<60}(?:\.%{POSINT})?)?
 HOUR (?:2[0123]|[01]?[0-9])
 MINUTE (?:[0-5][0-9])
 # '60' is a leap second in most time standards and thus is valid.
 SECOND (?:(?:[0-5]?[0-9]|60)(?:[:.,][0-9]+)?)
 TIME (?!<[0-9])%{HOUR}:%{MINUTE}(?::%{SECOND})(?![0-9])
 # datestamp is YYYY/MM/DD-HH:MM:SS.UUUU (or something like it)
 DATE_US %{MONTHNUM}[/-]%{MONTHDAY}[/-]%{YEAR}
 DATE_EU %{MONTHDAY}[./-]%{MONTHNUM}[./-]%{YEAR}
 DATESTAMP_US %{DATE_US}[- ]%{TIME}
 DATESTAMP_EU %{DATE_EU}[- ]%{TIME}
 ISO8601_TIMEZONE (?:Z|[+-]%{HOUR}(?::?%{MINUTE}))
 ISO8601_SECOND (?:%{SECOND}|60)
 TIMESTAMP_ISO8601 %{YEAR}-%{MONTHNUM}-%{MONTHDAY}[T ]%{HOUR}:?%{MINUTE}(?::?
%{SECOND})?%{ISO8601_TIMEZONE}?
 TZ (?:[PMCE][SD]T|UTC)
 DATESTAMP_RFC822 %{DAY} %{MONTH} %{MONTHDAY} %{YEAR} %{TIME} %{TZ}
 DATESTAMP_RFC2822 %{DAY}, %{MONTHDAY} %{MONTH} %{YEAR} %{TIME} %{ISO8601_TIMEZONE}
 DATESTAMP_OTHER %{DAY} %{MONTH} %{MONTHDAY} %{TIME} %{TZ} %{YEAR}
 DATESTAMP_EVENTLOG %{YEAR}%{MONTHNUM2}%{MONTHDAY}%{HOUR}%{MINUTE}%{SECOND}
 CISCOTIMESTAMP %{MONTH} %{MONTHDAY} %{TIME}

 # Syslog Dates: Month Day HH:MM:SS
 SYSLOGTIMESTAMP %{MONTH} +%{MONTHDAY} %{TIME}
 PROG (?:[\w._/%-]+)
 SYSLOGPROG %{PROG:program}(?:\[%{POSINT:pid}\])?
 SYSLOGHOST %{IPORHOST}
 SYSLOGFACILITY <%{NONNEGINT:facility}.%{NONNEGINT:priority}>
 HTTPDATE %{MONTHDAY}/%{MONTH}/%{YEAR}:%{TIME} %{INT}

 # Shortcuts
 QS %{QUOTEDSTRING:UNWANTED}
```

```
 # Log formats
 SYSLOGBASE %{SYSLOGTIMESTAMP:timestamp} (?:%{SYSLOGFACILITY} )?%{SYSLOGHOST:logsource}
 %{SYSLOGPROG}:

 MESSAGESLOG %{SYSLOGBASE} %{DATA}

 COMMONAPACHELOG %{IPORHOST:clientip} %{USER:ident} %{USER:auth}
 \[%{HTTPDATE:timestamp}\] "(?:%{WORD:verb} %{NOTSPACE:request}(?: HTTP/
%{NUMBER:httpversion})?|%{DATA:rawrequest})" %{NUMBER:response} (?:%{Bytes:bytes=
%{NUMBER}|-})
 COMBINEDAPACHELOG %{COMMONAPACHELOG} %{QS:referrer} %{QS:agent}
 COMMONAPACHELOG_DATATYPED %{IPORHOST:clientip} %{USER:ident;boolean} %{USER:auth}
 \[%{HTTPDATE:timestamp;date;dd/MMM/yyyy:HH:mm:ss Z}\] "(?:%{WORD:verb;string}
 %{NOTSPACE:request}(?: HTTP/%{NUMBER:httpversion;float})?|%{DATA:rawrequest})"
 %{NUMBER:response;int} (?:%{NUMBER:bytes;long}|-)


 # Log Levels
 LOGLEVEL ([A|a]lert|ALERT|[T|t]race|TRACE|[D|d]ebug|DEBUG|[N|n]otice|NOTICE|[I|i]nfo|
INFO|[W|w]arn?(?:ing)?|WARN?(?:ING)?|[E|e]rr?(?:or)?|ERR?(?:OR)?|[C|c]rit?(?:ical)?|
CRIT?(?:ICAL)?|[F|f]atal|FATAL|[S|s]evere|SEVERE|EMERG(?:ENCY)?|[Ee]merg(?:ency)?)
```

撰寫 XML 自訂分類器

XML 會利用檔案中的標籤來定義文件的結構。使用 XML 自訂分類器，您可以指定用來定義一個列的標籤名稱。

AWS Glue 中的自訂分類器值

當您定義 XML 分類器時，您將以下值提供至 AWS Glue 以建立分類器。此分類器的分類欄位設定為 `xml`。

Name

分類器名稱。

Row 標籤

在 XML 文件中用於定義資料表列的 XML 標籤名稱，不含角括號 < >。此名稱必須符合 XML 的標籤規則。

> **Note**
>
> 包含列資料的元素不得為自我封閉的空元素。舉例而言，不會AWS Glue剖析此空元素：
>
> ```
> <row att1=”xx” att2=”yy” />
> ```
>
> 空元素可撰寫如下：
>
> ```
> <row att1=”xx” att2=”yy”> </row>
> ```

AWS Glue 追蹤建立時間、上次更新時間和您的分類器版本。

例如，假設您擁有以下 XML 檔案。若要建立僅含有作者和標題的 AWS Glue 資料表，請在 AWS Glue 主控台中以 Row 標籤做為 AnyCompany 來建立分類器。接著，新增並執行採用此自訂分類器的爬蟲程式即可。

```
<?xml version="1.0"?>
<catalog>
   <book id="bk101">
     <AnyCompany>
       <author>Rivera, Martha</author>
       <title>AnyCompany Developer Guide</title>
     </AnyCompany>
   </book>
   <book id="bk102">
     <AnyCompany>
       <author>Stiles, John</author>
       <title>Style Guide for AnyCompany</title>
     </AnyCompany>
   </book>
</catalog>
```

撰寫 JSON 自訂分類器

JSON 是資料交換格式。它定義資料結構與名稱值組或值的排序清單。使用 JSON 自訂分類器時，您可以指定資料結構的 JSON 路徑，以用於定義資料表的結構描述。

AWS Glue 中的自訂分類器值

當您定義 JSON 分類器時，您將以下值提供至 AWS Glue 以建立分類器。此分類器的分類欄位設定為 `json`。

Name

分類器名稱。

JSON 路徑

JSON 路徑會指向用來定義資料表結構描述的物件。您可以用點標記法或括號標記法來撰寫 JSON 路徑。以下是支援的運算子：

描述
JSON 物件的根元素。這會啟動所有路徑表達式
萬用字元。可在 JSON 路徑中用於需要名稱或數字的任何位置。
以點標記的子代。指定 JSON 物件中的子欄位。
以刮號標記的子代。指定 JSON 物件中的子欄位。您只能指定一個子欄位。
陣列索引。以索引指定陣列的值。

AWS Glue 追蹤建立時間、上次更新時間和您的分類器版本。

Example 使用 JSON 分類器從陣列中提取記錄

此處會假設您的 JSON 資料是一系列的記錄。舉例來說，檔案的前面幾行可能如下所示：

```
[
  {
    "type": "constituency",
    "id": "ocd-division\/country:us\/state:ak",
    "name": "Alaska"
  },
  {
    "type": "constituency",
    "id": "ocd-division\/country:us\/state:al\/cd:1",
    "name": "Alabama's 1st congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\/country:us\/state:al\/cd:2",
    "name": "Alabama's 2nd congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\/country:us\/state:al\/cd:3",
    "name": "Alabama's 3rd congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\/country:us\/state:al\/cd:4",
    "name": "Alabama's 4th congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\/country:us\/state:al\/cd:5",
    "name": "Alabama's 5th congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\/country:us\/state:al\/cd:6",
    "name": "Alabama's 6th congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\/country:us\/state:al\/cd:7",
    "name": "Alabama's 7th congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\/country:us\/state:ar\/cd:1",
```

```
    "name": "Arkansas's 1st congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\/country:us\/state:ar\/cd:2",
    "name": "Arkansas's 2nd congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\/country:us\/state:ar\/cd:3",
    "name": "Arkansas's 3rd congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\/country:us\/state:ar\/cd:4",
    "name": "Arkansas's 4th congressional district"
  }
]
```

透過內建 JSON 分類器執行爬蟲程式時，即可將整個檔案用來定義結構描述。因為您尚未 JSON 路徑，該爬蟲程式會將資料視為一個物件，亦即只是一個陣列。例如，結構描述看起來類似如下：

```
root
|-- record: array
```

然而，若要在 JSON 陣列中根據每個記錄建立結構描述，請建立自訂 JSON 分類器並指定 JSON 路徑為 `$[*]`。當您指定此 JSON 路徑時，分類器會詢問陣列中的 12 筆記錄以判斷結構描述。產生的結構描述將包含各個物件的獨立欄位，如以下範例所示：

```
root
|-- type: string
|-- id: string
|-- name: string
```

Example 使用 JSON 分類器僅檢查檔案的部分

此處假設 JSON 資料會遵循範例 JSON 檔案 `s3://awsglue-datasets/examples/us-legislators/all/areas.json` 的模式，該檔案取自 http://everypolitician.org/。JSON 檔案中的範例物件看起來會像下述內容：

```
{
  "type": "constituency",
  "id": "ocd-division\/country:us\/state:ak",
  "name": "Alaska"
}
{
  "type": "constituency",
  "identifiers": [
    {
      "scheme": "dmoz",
      "identifier": "Regional\/North_America\/United_States\/Alaska\/"
    },
    {
      "scheme": "freebase",
      "identifier": "\/m\/0hjy"
    },
    {
      "scheme": "fips",
      "identifier": "US02"
    },
    {
      "scheme": "quora",
      "identifier": "Alaska-state"
    },
    {
      "scheme": "britannica",
      "identifier": "place\/Alaska"
    },
    {
      "scheme": "wikidata",
      "identifier": "Q797"
    }
  ],
  "other_names": [
    {
      "lang": "en",
      "note": "multilingual",
      "name": "Alaska"
    },
    {
      "lang": "fr",
      "note": "multilingual",
```

```
      "name": "Alaska"
    },
    {
      "lang": "nov",
      "note": "multilingual",
      "name": "Alaska"
    }
  ],
  "id": "ocd-division\/country:us\/state:ak",
  "name": "Alaska"
}
```

透過內建 JSON 分類器執行爬蟲程式時，即可將整個檔案用來建立結構描述。您得到的結構描述可能類似：

```
root
|-- type: string
|-- id: string
|-- name: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
```

不過，您必須建立自訂 JSON 分類器並將 JSON 路徑指定為 `id`，才能僅使用「`$.id`」物件建立結構描述。如此一來，系統僅會依據「`id`」欄位來產生結構描述：

```
root
|-- record: string
```

使用此結構描述所擷取的前幾行資料如下所示：

```
{"record": "ocd-division/country:us/state:ak"}
{"record": "ocd-division/country:us/state:al/cd:1"}
{"record": "ocd-division/country:us/state:al/cd:2"}
{"record": "ocd-division/country:us/state:al/cd:3"}
{"record": "ocd-division/country:us/state:al/cd:4"}
{"record": "ocd-division/country:us/state:al/cd:5"}
{"record": "ocd-division/country:us/state:al/cd:6"}
{"record": "ocd-division/country:us/state:al/cd:7"}
{"record": "ocd-division/country:us/state:ar/cd:1"}
{"record": "ocd-division/country:us/state:ar/cd:2"}
{"record": "ocd-division/country:us/state:ar/cd:3"}
{"record": "ocd-division/country:us/state:ar/cd:4"}
{"record": "ocd-division/country:us/state:as"}
{"record": "ocd-division/country:us/state:az/cd:1"}
{"record": "ocd-division/country:us/state:az/cd:2"}
{"record": "ocd-division/country:us/state:az/cd:3"}
{"record": "ocd-division/country:us/state:az/cd:4"}
{"record": "ocd-division/country:us/state:az/cd:5"}
{"record": "ocd-division/country:us/state:az/cd:6"}
{"record": "ocd-division/country:us/state:az/cd:7"}
```

若要根據 JSON 檔案中的深度巢狀物件 (例如「identifier」) 來建立結構描述，則可建立自訂 JSON 分類器並將 JSON 路徑指定為 $.identifiers[*].identifier。儘管產生的結構描述與先前範例類似，但它是依據 JSON 檔案中不同的物件建立而成。

結構描述看起來類似如下：

```
root
|-- record: string
```

若列出資料表的前面幾行，即會顯示以「identifier」物件中資料為基礎的結構描述：

```
{"record": "Regional/North_America/United_States/Alaska/"}
{"record": "/m/0hjy"}
{"record": "US02"}
{"record": "5879092"}
{"record": "4001016-8"}
{"record": "destination/alaska"}
{"record": "1116270"}
{"record": "139487266"}
```

```
{"record": "n79018447"}
{"record": "01490999-8dec-4129-8254-eef6e80fadc3"}
{"record": "Alaska-state"}
{"record": "place/Alaska"}
{"record": "Q797"}
{"record": "Regional/North_America/United_States/Alabama/"}
{"record": "/m/0gyh"}
{"record": "US01"}
{"record": "4829764"}
{"record": "4084839-5"}
{"record": "161950"}
{"record": "131885589"}
```

您可以建立自訂 JSON 分類器並將 JSON 路徑指定為 `$.other_names[*].name`，藉此根據 JSON 檔案中其他深度巢狀物件 (例如「other_names」陣列中的「name」欄位) 來建立資料表。儘管產生的結構描述與先前範例類似，但它是依據 JSON 檔案中不同的物件建立而成。結構描述看起來類似如下：

```
root
|-- record: string
```

若列出資料表的前面幾行，即會顯示以「name」陣列中「other_names」物件資料為基礎的結構描述：

```
{"record": "Alaska"}
{"record": "Alaska"}
{"record": "Аляска"}
{"record": "Alaska"}
{"record": "Alaska"}
{"record": "Alaska"}
{"record": "Alaska"}
{"record": "Alaska"}
{"record": "Alaska"}
{"record": "######"}
{"record": "######"}
{"record": "######"}
{"record": "Alaska"}
{"record": "Alyaska"}
{"record": "Alaska"}
{"record": "Alaska"}
```

```
{"record": "Штат Аляска"}
{"record": "Аляска"}
{"record": "Alaska"}
{"record": "#######"}
```

撰寫 CSV 自訂分類器

自訂 CSV 分類器可讓您在自訂 csv 分類器欄位中為每一欄指定資料類型。您可以指定每欄的資料類型，用逗號分隔。透過指定資料類型，您可以覆寫爬蟲程式推斷的資料類型，並確保資料將適當地分類。

您可以在該分類器中設定用於處理 CSV 的 SerDe，並且將在資料型錄中套用該 SerDe。

當您建立自訂分類器時，您也可以針對不同的爬蟲程式重複使用分類器。

- 對於只有標題 (沒有資料) 的 csv 檔案，這些檔案將被分類為 UNKNOWN，因為沒有提供足夠的資訊。如果您在 Column headings (欄位標題) 中指定 CSV「具有標題」，並提供資料類型，我們可以正確地對這些檔案進行分類。

您可以使用自訂 CSV 分類器來推論各種 CSV 資料類型的結構描述。您可以為分類器提供的自訂屬性包含分隔符號、CSV SerDe 選項、標頭相關選項，以及是否要對資料執行特定驗證。

AWS Glue 中的自訂分類器值

當定義 CSV 分類器時，您要將下列值提供至 AWS Glue 以便建立分類器。此分類器的分類欄位設定為 `csv`。

分類器名稱

分類器名稱。

CSV SerDe

設定用於在分類器中處理 CSV 的 SerDe，並且將在資料型錄中套用該 Serde。選項有「開放式 CSV SerDe」、「延遲簡易 SerDe」和「無」。當您希望爬蟲程式執行偵測時，您可以指定「無」值。

欄位分隔符號

表示用於分隔資料列中每個欄位項目的自訂符號。提供 Unicode 字元。如果無法輸入分隔符號，您可以複製並貼上它。這個方式適用於可列印字元，包括系統不支援的字元 (通常顯示為 □)。

引號符號

用來表示將內容結合成單一欄位值的自訂符號。必須不同於欄位分隔符號。提供 Unicode 字元。如果無法輸入分隔符號，您可以複製並貼上它。這個方式適用於可列印字元，包括系統不支援的字元 (通常顯示為 □)。

欄位標題

指示在 CSV 檔案中應如何偵測出欄位標題的行為。如果您的自訂 CSV 檔案包含欄位標題，請輸入以逗號分隔的欄位標題清單。

處理選項：允許檔案包含單一欄位

啟用處理僅包含一個欄位的檔案。

處理選項：裁剪空格後再識別欄位值

指定在確認欄位值類型之前是否要裁剪值。

自訂資料類型 - 選用

輸入以逗號分隔的自訂資料類型。指定 CSV 檔案中的自訂資料類型。自訂資料類型必須是受支援的資料類型。支援的資料類型為："BINARY"、"BOOLEAN"、"DATE"、"DECIMAL"、"DOUBLE"、"FLOAT"、"INT"、"LONG"、"SHORT 不受支援的資料類型會顯示錯誤。

在 AWS Glue 主控台上使用分類器

分類器可判斷資料的結構描述。您可以編寫自訂分類器並從 AWS Glue 指向分類器。

檢視分類器

若要查看您已建立的所有分類器的清單，請開啟位於 https://console.aws.amazon.com/glue/ AWS Glue 主控台，然後選擇 Classifiers (分類器) 標籤。

清單顯示有關各分類器的下列屬性：

- 分類器 - 分類器名稱。建立分類器時，您必須提供其名稱。
- 分類 – 此分類器推斷的資料表分類類型。
- 上次更新 – 上一次更新此分類器的時間。

管理分類器

您可以在 主控台的 ClassifiersAWS Glue (分類器) 清單中新增、編輯和刪除分類器。若要查看分類器的詳細資訊，請在清單中選擇分類器的名稱。詳細資訊包含您在建立分類器時所定義的資訊。

建立分類器

若要在 AWS Glue 主控台新增分類器，請選擇 Add classifier (新增分類器)。定義分類器時，您提供以下值：

- 分類器名稱 – 提供分類器的唯一名稱。
- 分類器類型 – 此分類器推斷的資料表分類類型。
- 上次更新 – 上一次更新此分類器的時間。

分類器名稱

提供分類器的唯一名稱。

分類器類型

選擇要建立之分類器的類型。

根據您選擇的分類器類型，設定下列分類器的屬性：

Grok

- 分類

  描述分類資料的格式或類型，或提供自訂標籤。

- Grok 模式

  這是用於將資料剖析為結構化結構描述。grok 模式由描述資料存放區格式的具名模式組成。您使用 AWS Glue 提供的具名內建模式寫入此 grok 模式，自訂寫入的模式，並包含在 Custom patterns (自訂模式) 欄位內。雖然 grok 偵錯工具的結果可能不會完全符合 AWS Glue 的結果，我們建議您透過 grok 偵錯工具使用一些範例資料來嘗試模式。您可以從 Web 上找到 grok 偵錯工具。AWS Glue 提供的具名內建模式通常相容於 Web 上提供的 grok 模式。

  建置您的 grok 模式，反覆新增具名模式和在偵錯工具內檢查您的結果。此活動可讓您確信當 AWS Glue 爬蟲程式執行您的 grok 模式時，您的資料可以剖析。

- 自訂模式

  對於 grok 分類器，這些是您編寫的Grok 模式的選擇性建置區塊。內建的模式無法剖析您的資料時，您可能需要編寫自訂模式。這些自訂模式在此欄位中定義，且在 Grok 模式欄位中參考。每個自訂模式都必須在不同的行定義。就如同內建的模式，它包含具名模式定義，使用常規表達式 (regex) 的語法。

  例如，以下具有 MESSAGEPREFIX 名稱，接著是常規表達式定義，以套用到您的資料，判斷是否遵循模式。

  ```
  MESSAGEPREFIX .*-.*-.*-.*-.*
  ```

XML

- Row 標籤

  針對 XML 分類器，此為 XML 標籤名稱，定義 XML 文件中的資料表列。輸入類型的名稱，而不角括號 < >。此名稱必須符合 XML 的標籤規則。

  如需詳細資訊，請參閱 撰寫 XML 自訂分類器。

JSON

- JSON 路徑

  針對 JSON 分類器，此為連往物件、陣列或數值，定義所建立資料表之資料列的 JSON 物件。使用 AWS Glue 支援的運算子，以點或括號 JSON 語法輸入名稱。

  如需更多詳細資訊，請參閱 撰寫 JSON 自訂分類器 中的運算子清單。

CSV

- 欄位分隔符號

  表示用於分隔資料列中每個欄位項目的單一字元或符號。從清單中選擇分隔符號，或選擇 Other 以輸入自訂分隔符號。

- 引號符號

用來表示將內容結合成單一欄位值的單一字元或符號。必須不同於欄位分隔符號。從清單中選擇引號，或選擇 Other 以輸入自訂引號字元。

- 欄位標題

  指示在 CSV 檔案中應如何偵測出欄位標題的行為。您可以選擇 Has headings、No headings 或 Detect headings。如果您的自訂 CSV 檔案包含欄位標題，請輸入以逗號分隔的欄位標題清單。

- 允許具有單一欄的檔案

  若要被歸類為 CSV，資料必須至少有兩個資料欄和兩個資料列。使用此選項可允許處理僅包含一欄的檔案。

- 裁剪空格後再識別欄值

  此選項指定在確認欄位值類型之前是否要裁剪值。

- 自訂資料類型

  (選用) - 在逗號分隔清單中輸入自訂資料類型。支援的資料類型為："BINARY"、"BOOLEAN"、"DATE"、"DECIMAL"、"DOUBLE"、"FLOAT"、"INT"、"LONG"、"SHO

- CSV Serde

  (選用)：用於在分類器中處理 CSV 的 SerDe，並且將在資料型錄中套用該 Serde。從 Open CSV SerDe、Lazy Simple SerDe 或 None 中選擇。您可以指定希望爬蟲程式執行偵測時的 None 值。

如需詳細資訊，請參閱 撰寫自訂分類器。

排程 AWS Glue 爬蟲程式

您可以隨需或定期執行 AWS Glue 爬蟲程式。爬蟲程式排程可以使用 Cron 格式表示。如需詳細資訊，請參閱 Wikipedia 中的 Cron。

當您建立以排程為基礎的爬蟲程式時，您可以指定特定的限制條件，例如爬蟲程式執行的頻率、在一週中的哪幾天與什麼時間執行。這些限制條件以 Cron 為依據。設定爬蟲程式的排程時，應考慮 Cron 的功能和限制。例如，如果您選擇在每個月的 31 日執行您的爬蟲程式，請注意有些月份不到 31 天。

每個爬蟲程式的編目有效期最長為 12 個月

如需有關使用 Cron 排程工作和爬蟲程式的詳細資訊，請參閱 任務和爬蟲程式以時間為基礎的排程。

檢視爬蟲程式結果和詳細資訊

爬蟲程式成功執行後，它會在 Data Catalog 中建立資料表定義。在導覽窗格中選擇 Tables (資料表)，以查看爬蟲程式在您指定的資料庫中所建立的資料表。

您可以檢視爬蟲程式本身的相關資訊，如下所示：

- AWS Glue 主控台的「爬行者程式」頁面會顯示爬行者程式的下列特性：

屬性	描述
名稱	當您建立爬蟲程式時，必須指定唯一的名稱。
狀態	爬蟲程式可以處於就緒、啟動、停用、排程執行或排程暫停的狀態。爬蟲程式的執行會從啟動進展到停用。您可以恢復或暫停爬蟲程式所連接的排程。
排程	您可以選擇隨需執行，或選擇頻率來排程執行爬蟲程式。如需關於排程執行爬蟲程式的詳細資訊，請參閱排程爬蟲程式。
上次執行	爬蟲程式上次執行的日期和時間。
日誌	針對爬蟲程式最近一次的執行作業，連結到該項作業任何可用的日誌。
從上次執行以來的資料表變更	最近一次執行爬行者程式所更新的表格數目。AWS Glue Data Catalog

- 若要檢視爬蟲程式的歷史記錄，請選擇導覽窗格中的 Crawlers (爬蟲程式)，以查看您建立的爬蟲程式。從可用的爬蟲程式清單中選擇爬蟲程式。您可以在 Crawler runs (爬蟲程式執行) 索引標籤中檢視爬蟲程式屬性以及檢視爬蟲程式歷史記錄。

  「爬蟲程式執行」索引標籤會顯示爬蟲程式每次執行時的相關資訊，包括 Start time (UTC) (開始時間 (UTC))、End time (UTC) (結束時間 (UTC))、Duration (持續時間)、Status (狀態)、DPU hours (DPU 時數)，以及 Table changes (資料表變更)。

  爬蟲程式執行索引標籤僅顯示自爬蟲程式歷史記錄功能啟動日期以來發生的網路爬取，而且最多只會保留 12 個月的網路爬取。不會傳回較舊的網路爬取。

- 若要查看其他資訊，請選擇爬蟲程式詳細資料頁面中的索引標籤。每個索引標籤都會顯示與爬蟲程式相關的資訊。
  - Schedule (排程)：為爬蟲程式建立的任何排程都會顯示在此處。
  - Data sources (資料來源)：爬蟲程式掃描的所有資料來源都會顯示在此處。
  - Classifiers (分類器)：指派給爬蟲程式的所有分類器都會顯示在此處。
  - 標籤：任何建立並指派給 AWS 資源的標籤都會顯示在此處。

爬蟲程式在 Data Catalog 資料表上設定的參數

這些資料表屬性由 AWS Glue 爬蟲程式設定。我們希望使用者使用 `classification` 和 `compressionType` 屬性。其他屬性 (包括資料表大小估算) 用於內部計算，我們不能保證其準確性或適用於客戶使用案例。變更這些參數可能會改變爬蟲程式的行為，我們不支持此工作流程。

屬性索引鍵	屬性值
`UPDATED_BY_CRAWLER`	執行更新的爬蟲程式名稱。
`connectionName`	在 Data Catalog 中，用於連線至資料存放區的爬蟲程式連線名稱。
`recordCount`	根據檔案大小和標題估計資料表中的記錄數。
`skip.header.line.count`	跳過列以跳過標題。在分類為 CSV 的資料表上設定。
`CrawlerSchemaSerializerVersion`	供內部使用
`classification`	由爬蟲程式推斷的資料格式。如需 AWS Glue 爬蟲程式支援格式的詳細資訊，請參閱 the section called “AWS Glue 中的內建分類器”。
`CrawlerSchemaDeserializerVersion`	供內部使用
`sizeKey`	網路爬取的資料表中檔案的合併大小。

屬性索引鍵	屬性值
averageRecordSize	資料表中列的平均大小 (位元組)。
compressionType	資料表中資料所使用的壓縮類型。如需 AWS Glue 爬蟲程式支援之壓縮類型的詳細資訊，請參閱 the section called “AWS Glue 中的內建分類器”。
typeOfData	file、table 或 view。
objectCount	資料表的 Amazon S3 路徑下的物件數目。

這些額外的資料表屬性是由 Snowflake 資料存放區的 AWS Glue 爬蟲程式所設定。

屬性索引鍵	屬性值
aws:RawTableLastAltered	記錄 Snowflake 資料表的最後更改時間戳記。
ViewOriginalText	檢視 SQL 陳述式。
ViewExpandedText	檢視以 Base64 格式編碼的 SQL 陳述式。
ExternalTable:S3Location	Snowflake 外部資料表的 Amazon S3 位置。
ExternalTable:FileFormat	Snowflake 外部資料表的 Amazon S3 檔案格式。

這些額外的資料表屬性是由 JDBC 類型資料存放區 (例如 Amazon Redshift、Microsoft SQL Server、MySQL、PostgreSQL 和 Oracle) 的 AWS Glue 爬蟲程式所設定。

屬性索引鍵	屬性值
aws:RawType	在將資料存放在 Data Catalog 中時，爬蟲程式會將資料類型轉換為與 Hive 相容的類型，這會多次導致原生資料類型的資訊遺失。爬蟲程式會輸出 aws:RawType 參數以提供原生層級的資料類型。
aws:RawColumnComment	如果註解與資料庫中的資料欄相關聯，爬蟲程式會在目錄資料表中輸出對應的註解。註解字串被截斷為 255 個位元組。 Microsoft SQL Server 不支援註解。
aws:RawTableComment	如果註解與資料庫中的資料表相關聯，爬蟲程式會在目錄資料表中輸出對應的註解。註解字串被截斷為 255 個位元組。 Microsoft SQL Server 不支援註解。

自訂爬行者程式行為

當爬蟲程式執行時，它可能遇到您的資料存放區變更，導致結構描述或分割區不同於之前的編目。您可以使用 AWS Management Console 或 AWS Glue API 來設定爬行者程式處理特定類型變更的方式。

主題

- 用於新增分割區的增量編目
- 設定分割區索引爬蟲程式組態選項
- 使用 Amazon S3 事件通知加速網路爬取
- 如何防止爬蟲程式變更現有的結構描述
- 如何為每個 Amazon S3 包含路徑建立單一結構描述
- 如何指定資料表位置和分割層級
- 如何指定爬蟲程式可建立的資料表數目上限
- 如何為 Delta Lake 的資料儲存指定配置選項
- 如何設定爬蟲程式以使用 Lake Formation 憑證

Console

當您使用 AWS Glue 主控台定義爬蟲程式時，有幾個選項可設定爬蟲程式的行為。如需使用 AWS Glue 主控台新增爬蟲程式的詳細資訊，請參閱設定爬行者程式。

當爬蟲程式在之前編目過的資料存放區執行時，可能會發現結構描述已經變更，或是資料存放區中的部分物件已被刪除。爬蟲程式會記錄結構描述的變更。根據爬蟲程式的來源類型而定，無論結構描述變更政策為何，系統可能都會建立新的資料表和分割區。

若要指定爬蟲程式在發現結構描述變更時應執行何種動作，您可以在主控台選擇以下動作之一：

- Update the table definition in the Data Catalog (更新 Data Catalog 中的資料表定義) - 在 AWS Glue Data Catalog中新增欄位、移除遺漏欄位，以及修改現有欄位的定義。移除爬蟲程式未設定的任何中繼資料。這是預設設定。
- Add new columns only (僅新增欄位) - 對於映射至 Amazon S3 資料存放區的資料表，在它們被探索到時新增欄位，但不刪除或變更 Data Catalog 中現有欄位的類型。當 Data Catalog 中的目前欄位正確，而且您不希望爬蟲程式移除或變更現有欄位的類型時，選擇此選項。如果基本 Amazon S3 資料表屬性變更，例如分類、壓縮類型、或 CSV 分隔符號，請將該資料表標記為淘汰。維持輸入和輸出格式與它們在 Data Catalog 時相同。只有當 SerDe 參數是爬行者程式所設定的參數時，才更新參數。對於所有其他資料存放區，修改現有欄位的定義。
- Ignore the change and don't update the table in the Data Catalog (忽略變更且不更新 Data Catalog 中的資料表) - 僅建立新的資料表和分割區。

  這是增量網路爬取的預設設定。

爬蟲程式可能也會探索到新的或變更的分割區。依預設，新的分割區會新增，而現有分割區若有變更則會更新。此外，您可以在 主控台，將爬蟲程式的組態選項設定為 Update all new and existing partitions with metadata from the tableAWS Glue (以資料表的中繼資料更新所有新的和現有分割區)。設定此選項後，分割區會從其父表格繼承中繼資料屬性，例如其分類、輸入格式、輸出格式、SerDe資訊和結構描述。資料表中這些屬性的任何變更都會傳播至其分割區。現有的爬蟲程式以此組態選項設定後，下次爬蟲程式執行時，現有分割區就會更新以符合其父資料表的屬性。

若要指定當爬蟲程式在資料存放區發現刪除的物件，您可以選擇下列其中一個動作：

- 從 Data Catalog 刪除資料表和分割區
- 忽略變更且不更新 Data Catalog 中的資料表

  這是增量網路爬取的預設設定。
- Mark the table as deprecated in the Data Catalog (在 Data Catalog 中將該資料表標記為淘汰) - 這是預設設定。

AWS CLI

```
aws glue create-crawler \
--name "your-crawler-name" \
--role "your-iam-role-arn" \
--database-name "your-database-name" \
--targets 'S3Targets=[{Path="s3://your-bucket-name/path-to-data"}]' \
--configuration '{"Version": 1.0, "CrawlerOutput": {"Partitions":
 {"AddOrUpdateBehavior": "InheritFromTable"}, "Tables": {"AddOrUpdateBehavior":
 "MergeNewColumns"}}}'
```

API

使用 AWS Glue API 定義爬行者程式時，您可以從數個欄位中選擇以設定爬行者程式。爬蟲程式 API 中的 SchemaChangePolicy 可決定當爬蟲程式發現變更的結構描述或刪除的物件時，應執行何種動作。爬蟲程式在執行時會記錄結構描述的變更。

顯示爬蟲配置選項的示例 python 代碼

```
import boto3
import json

# Initialize a boto3 client for AWS Glue
glue_client = boto3.client('glue', region_name='us-east-1')  # Replace 'us-east-1'
 with your desired AWS region

# Define the crawler configuration
crawler_configuration = {
    "Version": 1.0,
    "CrawlerOutput": {
        "Partitions": {
            "AddOrUpdateBehavior": "InheritFromTable"
        },
        "Tables": {
            "AddOrUpdateBehavior": "MergeNewColumns"
        }
    }
}

configuration_json = json.dumps(crawler_configuration)
# Create the crawler with the specified configuration
response = glue_client.create_crawler(
```

```
    Name='your-crawler-name',  # Replace with your desired crawler name
    Role='crawler-test-role',  # Replace with the ARN of your IAM role for Glue
    DatabaseName='default',  # Replace with your target Glue database name
    Targets={
        'S3Targets': [
            {
                'Path': "s3://your-bucket-name/path/",  # Replace with your S3 path
 to the data
            },
        ],
        # Include other target types like 'JdbcTargets' if needed
    },
    Configuration=configuration_json,
    # Include other parameters like Schedule, Classifiers, TablePrefix,
 SchemaChangePolicy, etc., as needed
)

print(response)a
```

爬蟲程式執行時，無論結構描述變更政策為何，一律會建立新的資料表和分割區。您可以在 UpdateBehavior 結構的 SchemaChangePolicy 欄位中選擇以下動作之一，以決定當爬蟲程式發現變更的資料表結構描述時應採取何種動作：

- UPDATE_IN_DATABASE 在 更新資料表。 AWS Glue Data Catalog新增欄位、移除遺漏欄位，以及修改現有欄位的定義。移除爬蟲程式未設定的任何中繼資料。
- LOG – 忽略變更並且不更新 Data Catalog 中的資料表。

  這是增量網路爬取的預設設定。

您也可以使用爬蟲程式 API Configuration 欄位提供的 JSON 物件覆寫 SchemaChangePolicy 結構。此 JSON 物件可包含一個金鑰值對，將政策設定為不更新現有的欄位，而且僅新增欄位。例如，以字串提供下列 JSON 物件：

```
{
   "Version": 1.0,
   "CrawlerOutput": {
      "Tables": { "AddOrUpdateBehavior": "MergeNewColumns" }
   }
}
```

此選項對應至 主控台的 Add new columns onlyAWS Glue (僅新增欄位) 選項。它只會覆寫對 Amazon S3 資料存放區進行編目所產生資料表的 SchemaChangePolicy 結構。如果您想要維持中繼資料與它在 Data Catalog (真實來源) 時相同，請選擇此選項。在發現欄位時將會新增欄位，包括巢狀資料類型。但現有欄位不會移除，其類型也不會變更。如果 Amazon S3 資料表屬性大幅變更，請將資料表標記為淘汰，並記錄不相容的屬性需要解析的警告。此選項不適用於增量爬蟲程式。

爬蟲程式在之前抓取過的資料存放區執行時，可能會發現新的或變更過的分割區。依預設，新的分割區會新增，而現有分割區若有變更則會更新。此外，您可以將爬蟲程式的組態選項設定為 InheritFromTable [對應至 主控台的 Update all new and existing partitions with metadata from the tableAWS Glue (以資料表的中繼資料更新所有新的和現有分割區) 選項]。設定此選項後，分割區會從其父表格繼承中繼資料屬性，例如其分類、輸入格式、輸出格式、 SerDe 資訊和結構描述。父資料表的任何屬性對變更都會傳播至其分割區。

現有的爬蟲程式以此組態選項設定後，下次爬蟲程式執行時，現有分割區就會更新以符合其父資料表的屬性。此行為是設定爬蟲程式 API Configuration 欄位。例如，以字串提供下列 JSON 物件：

```
{
   "Version": 1.0,
   "CrawlerOutput": {
      "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" }
   }
}
```

爬蟲程式 API Configuration 欄位可設定多個組態選項。例如，若要設定爬蟲程式輸出至分割區和資料表，您可以提供一個字串呈現下列 JSON 物件：

```
{
    "Version": 1.0,
    "CrawlerOutput": {
       "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" },
       "Tables": {"AddOrUpdateBehavior": "MergeNewColumns" }
    }
}
```

您可以選擇下列其中一個動作，以決定當爬蟲程式在資料存放區發現刪除的物件時，應採取何種動作。爬蟲程式 API 的 SchemaChangePolicy 結構中的 DeleteBehavior 欄位可設定當爬蟲程式探索到刪除的物件時的行為。

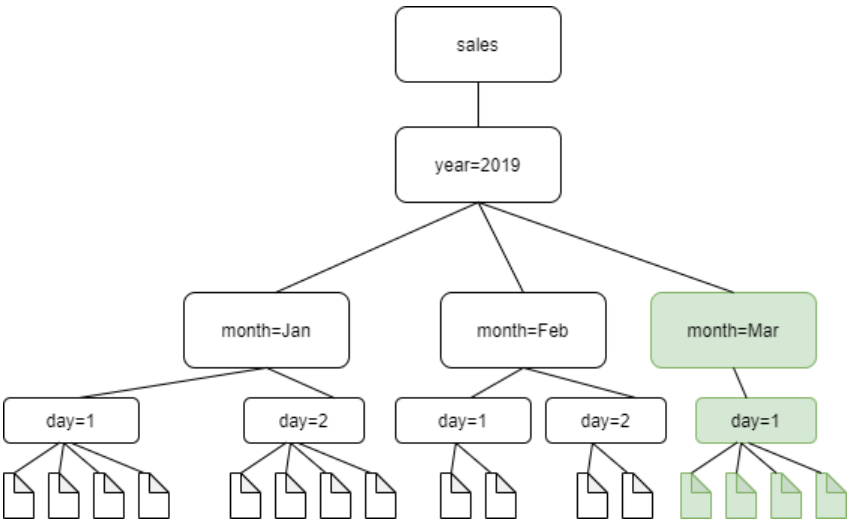
- DELETE_FROM_DATABASE – 從 Data Catalog 刪除資料表和分割區。
- LOG - 忽略變更。不更新 Data Catalog 。改為寫入日誌訊息。
- DEPRECATE_IN_DATABASE – 在 Data Catalog 將資料表標記為棄用。這是預設設定。

用於新增分割區的增量編目

爬蟲程式會提供新增新分割區的選項，讓具有穩定資料表結構描述的增量資料集能更快進行網路爬取。典型的使用案例是排程的爬蟲程式，每次網路爬取期間都會新增新的分割區。在開啟此選項時，其會先在目標資料集上執行完整網路爬取，讓爬蟲程式能記錄初始結構描述和分割區結構。在重新進行網路爬取期間，僅在結構描述相容時，新的分割區才會被新增至現有資料表。在第一次編目執行之後，不會進行任何結構描述變更，也不會將任何新表格加入至資料目錄。

您可以在設定 Amazon S3 資料來源時使用此選項。您可以在 CreateCrawler API 中使用 RecrawlBehavior 將 RecrawlPolicy 設定為 "Crawl_New_Folders"，或是在主控台中將後續爬蟲程式設定為僅網路爬取新子資料夾。

繼續使用the section called “爬蟲程式如何決定何時建立分割區？”中的範例，下圖顯示已新增三月的檔案。



如果您將 RecrawlBehavior 設定為 "Crawl_New_Folders" 選項，則只會網路爬取新的資料夾 month=Mar。

備註和限制

開啟此選項時，您無法在編輯爬蟲程式程式時變更 Amazon S3 目標資料存放區。此選項會影響特定爬蟲程式組態設定。開啟時，它會強制爬蟲程式的更新行為和刪除行為至 LOG。這表示：

- 如果它發現結構描述不相容的物件，爬行者程式就不會在「資料目錄」中新增物件，而會將此詳細資訊新增為記錄檔中 CloudWatch 的記錄。
- 其將不會在 Data Catalog 中更新已刪除的物件。

如需更多詳細資訊，請參閱 the section called “自訂爬行者程式行為”。

設定分割區索引爬蟲程式組態選項

資料型錄支援分割區索引，以提供高效的特定分割區查閱。如需詳細資訊，請參閱在 AWS Glue中使用分割區索引。 AWS Glue 爬行者程式會根據不必要的方式為 Amazon S3 和達美湖目標建立分割區索引。

當您定義 cralwer 時，依預設會在 [設定輸出和排程] 頁面的 [進階選項] 下啟用 [自動建立分割區索引] 選項。

要禁用此選項，您可以取消選中複選框在控制台中自動創建分區索引。您也可以使用搜尋器 API 來停用此選項，`CreatePartitionIndex` 在中設定。`Configuration`預設值為 true。

分割區索引的使用說明

- 依預設，爬蟲程式建立的資料表沒有變數 `partition_filtering.enabled`。如需詳細資訊，請參閱 AWS Glue 分割區索引和篩選。
- 不支援為加密分割區建立分割區索引。

使用 Amazon S3 事件通知加速網路爬取

您可以將爬蟲程式設定為使用 Amazon S3 事件來尋找任何變更，而不是從 Amazon S3 或 Data Catalog 目標列出物件。此功能可使用 Amazon S3 事件識別兩個編目之間的變更，方法是列出觸發事件的子資料夾中的所有檔案，而不是列出完整的 Amazon S3 或 Data Catalog 目標，藉此改善重新編目時間。

第一個編目會列出目標中的所有 Amazon S3 物件。第一次成功編目之後，您可以選擇手動或按設定的排程重新編目。爬蟲程式將僅列出這些事件中的物件，而不是列出所有物件。

移動到基於 Amazon S3 事件的爬蟲程式的優點是：

- 因為不需要列出目標中的所有物件，而是在新增或刪除物件的位置完成特定資料夾清單，從而更快速地重新編目。
- 減少整體編目成本，因為會在新增或刪除物件的位置完成特定資料夾清單。

Amazon S3 事件編目會根據爬蟲程式排程從 SQS 佇列中使用 Amazon S3 事件來執行。如果佇列中沒有事件，則不會產生任何費用。您可以將 Amazon S3 事件設定為直接轉至 SQS 佇列，或在多個取用者需要相同事件 (即 SNS 和 SQS 的組合) 的情況下設定事件。如需詳細資訊，請參閱 the section called “為 Amazon S3 事件通知設定您的帳戶”。

在事件模式下建立並設定爬蟲程式之後，第一次編目會執行完整的 Amazon S3 或 Data Catalog 目標清單，藉此以清單模式執行。下列日誌會在第一次成功編目之後使用 Amazon S3 事件來確認編目的操作：「透過使用 Amazon S3 事件來執行編目。」

在建立 Amazon S3 事件編目並更新可能會影響編目的爬蟲程式屬性之後，編目會以清單模式運作，並新增下列日誌：「編目未在 S3 事件模式下執行」。

> **Note**
>
> 每個編目可使用的訊息數目上限為 10,000 封郵件。

Catalog 目標

當目標為 Data Catalog 時，爬蟲程式會使用變更來更新 Data Catalog 中的現有資料表 (例如，資料表中的額外分割區)。

主題

- 為 Amazon S3 事件通知設定您的帳戶
- 將加密與 Amazon S3 事件爬蟲程式搭配使用

為 Amazon S3 事件通知設定您的帳戶

本節說明如何為 Amazon S3 事件通知設定帳戶，並提供使用指令碼或 AWS Glue 主控台執行此操作的指示。

必要條件

完成下列設定任務。請注意括號中的值會參考指令碼中的可設定項。

1. 建立 Amazon S3 儲存貯體 (`s3_bucket_name`)
2. 識別爬蟲程式目標 (`folder_name`，例如 "test1")，它是已識別儲存貯體中的路徑。
3. 準備爬蟲程式名稱 (`crawler_name`)
4. 準備 SNS 主題名稱 (`sns_topic_name`)，它可以與爬蟲程式名稱相同。

5. 準備要執行爬行程式且 S3 儲存貯體存在的 AWS 區域 (`region`)。
6. 如果使用電子郵件來取得 Amazon S3 事件 (`subscribing_email`)，則可選擇性地準備電子郵件地址。

您也可以使用 CloudFormation 堆疊來建立資源。請完成下列步驟：

1. 在美國東部 (維吉尼亞北部) 啟動您的 CloudFormation 堆疊：
2. 在「參數」下，輸入 Amazon S3 儲存貯體的名稱 (包括帳號)。
3. 選取 `I acknowledge that AWS CloudFormation might create IAM resources with custom names`。
4. 選擇 `Create stack`。

限制:

- 無論是 Amazon S3 目標還是 Data Catalog 目標，爬蟲程式只支援單一目標。
- 不支援私有 VPC 上的 SQS。
- 不支援 Amazon S3 取樣。
- 爬蟲程式目標應該是 Amazon S3 目標的資料夾，或是 Data Catalog 目標的一個或多個 AWS Glue Data Catalog 資料表。
- 不支援「所有」路徑萬用字元：s3://%
- 對於 Data Catalog 目標，所有目錄資料表都應指向 Amazon S3 事件模式的相同 Amazon S3 儲存貯體。
- 對於 Data Catalog 目標，目錄資料表不應指向 Delta Lake 格式的 Amazon S3 位置 (包含 _symlink 資料夾或檢查目錄資料表的 `InputFormat`)。

若要使用以 Amazon S3 事件為基礎的爬蟲程式，您應該在 S3 儲存貯體上啟用事件通知，在其中根據字首篩選出事件，這些字首與 S3 目標相同並存放在 SQS 中。您可以透過主控台設定 SQS 和事件通知，方法是遵循演練：設定儲存貯體的通知中的步驟或使用 the section called “從目標產生 SQS 和設定 Amazon S3 事件的指令碼”。

SQS 政策

新增下列必須連接至爬蟲程式所使用之角色的 SQS 政策。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "sqs:DeleteMessage",
                "sqs:GetQueueUrl",
                "sqs:ListDeadLetterSourceQueues",
                "sqs:ReceiveMessage",
                "sqs:GetQueueAttributes",
                "sqs:ListQueueTags",
                "sqs:SetQueueAttributes",
                "sqs:PurgeQueue"
            ],
            "Resource": "arn:aws:sqs:{region}:{accountID}:cfn-sqs-queue"
        }
    ]
}
```

從目標產生 SQS 和設定 Amazon S3 事件的指令碼

確保滿足先決條件後，您可以執行以下 Python 指令碼來建立 SQS。以根據先決條件準備的名稱取代「可設定項」。

> **Note**
>
> 執行指令碼之後，請登入 SQS 主控台，尋找建立之 SQS 的 ARN。

Amazon SQS 設定可見性逾時，這是 Amazon SQS 防止其他取用者接收和處理訊息的期間。設定可見性逾時大致等於編目執行時間。

```
#!venv/bin/python
import boto3
import botocore

#---------Start : READ ME FIRST ----------------------#
# 1. Purpose of this script is to create the SQS, SNS and enable S3 bucket
 notification.
#     The following are the operations performed by the scripts:
```

```
#      a. Enable S3 bucket notification to trigger 's3:ObjectCreated:' and
 's3:ObjectRemoved:' events.
#      b. Create SNS topic for fan out.
#      c. Create SQS queue for saving events which will be consumed by the crawler.
#          SQS Event Queue ARN will be used to create the crawler after running the
 script.
# 2. This script does not create the crawler.
# 3. SNS topic is created to support FAN out of S3 events. If S3 event is also used by
 another
#    purpose, SNS topic created by the script can be used.
# 1. Creation of bucket is an optional step.
#    To create a bucket set create_bucket variable to true.
# 2. The purpose of crawler_name is to easily locate the SQS/SNS.
#     crawler_name is used to create SQS and SNS with the same name as crawler.
# 3. 'folder_name' is the target of crawl inside the specified bucket 's3_bucket_name'
#
#---------End : READ ME FIRST ------------------------#


#--------------------------------#
# Start : Configurable settings  #
#--------------------------------#

#Create
region = 'us-west-2'
s3_bucket_name = 's3eventtestuswest2'
folder_name = "test"
crawler_name = "test33S3Event"
sns_topic_name = crawler_name
sqs_queue_name = sns_topic_name
create_bucket = False

#--------------------------------#
# End : Configurable settings   #
#--------------------------------#

# Define aws clients
dev = boto3.session.Session(profile_name='myprofile')
boto3.setup_default_session(profile_name='myprofile')
s3 = boto3.resource('s3', region_name=region)
sns = boto3.client('sns', region_name=region)
sqs = boto3.client('sqs', region_name=region)
client = boto3.client("sts")
account_id = client.get_caller_identity()["Account"]
```

```
queue_arn = ""


def print_error(e):
    print(e.message + ' RequestId: ' + e.response['ResponseMetadata']['RequestId'])

def create_s3_bucket(bucket_name, client):
    bucket = client.Bucket(bucket_name)
    try:
        if not create_bucket:
            return True
        response = bucket.create(
            ACL='private',
            CreateBucketConfiguration={
                'LocationConstraint': region
            },
        )
        return True
    except botocore.exceptions.ClientError as e:
        print_error(e)
        if 'BucketAlreadyOwnedByYou' in e.message:  # we own this bucket so continue
            print('We own the bucket already. Lets continue...')
            return True
    return False

def create_s3_bucket_folder(bucket_name, client, directory_name):
    s3.put_object(Bucket=bucket_name, Key=(directory_name + '/'))

def set_s3_notification_sns(bucket_name, client, topic_arn):
    bucket_notification = client.BucketNotification(bucket_name)
    try:

        response = bucket_notification.put(
            NotificationConfiguration={
                'TopicConfigurations': [
                    {
                        'Id' : crawler_name,
                        'TopicArn': topic_arn,
                        'Events': [
                            's3:ObjectCreated:*',
                            's3:ObjectRemoved:*',

                        ],
```

```
                        'Filter' :  {'Key': {'FilterRules': [{'Name': 'prefix',
 'Value': folder_name}]}}
                    },
                ]
            }
        )
        return True
    except botocore.exceptions.ClientError as e:
        print_error(e)
    return False


def create_sns_topic(topic_name, client):
    try:
        response = client.create_topic(
            Name=topic_name
        )
        return response['TopicArn']
    except botocore.exceptions.ClientError as e:
        print_error(e)
    return None

def set_sns_topic_policy(topic_arn, client, bucket_name):
    try:
        response = client.set_topic_attributes(
            TopicArn=topic_arn,
            AttributeName='Policy',
            AttributeValue='''{
              "Version": "2008-10-17",
              "Id": "s3-publish-to-sns",
              "Statement": [{
                  "Effect": "Allow",
                  "Principal": { "AWS" : "*" },
                  "Action": [ "SNS:Publish" ],
                  "Resource": "%s",
                  "Condition": {
                    "StringEquals": {
                        "AWS:SourceAccount": "%s"
                    },
                    "ArnLike": {
                        "aws:SourceArn": "arn:aws:s3:*:*:%s"
                    }
                }
              }]
```

```
  }''' % (topic_arn, account_id, bucket_name)
        )
        return True
    except botocore.exceptions.ClientError as e:
        print_error(e)

    return False


def subscribe_to_sns_topic(topic_arn, client, protocol, endpoint):
    try:
        response = client.subscribe(
            TopicArn=topic_arn,
            Protocol=protocol,
            Endpoint=endpoint
        )
        return response['SubscriptionArn']
    except botocore.exceptions.ClientError as e:
        print_error(e)
    return None


def create_sqs_queue(queue_name, client):
    try:
        response = client.create_queue(
            QueueName=queue_name,
        )
        return response['QueueUrl']
    except botocore.exceptions.ClientError as e:
        print_error(e)
    return None


def get_sqs_queue_arn(queue_url, client):
    try:
        response = client.get_queue_attributes(
            QueueUrl=queue_url,
            AttributeNames=[
                'QueueArn',
            ]
        )
        return response['Attributes']['QueueArn']
    except botocore.exceptions.ClientError as e:
        print_error(e)
```

```
    return None

def set_sqs_policy(queue_url, queue_arn, client, topic_arn):
    try:
        response = client.set_queue_attributes(
            QueueUrl=queue_url,
            Attributes={
                'Policy': '''{
                  "Version": "2012-10-17",
                  "Id": "AllowSNSPublish",
                  "Statement": [
                    {
                      "Sid": "AllowSNSPublish01",
                      "Effect": "Allow",
                      "Principal": "*",
                      "Action": "SQS:SendMessage",
                      "Resource": "%s",
                      "Condition": {
                        "ArnEquals": {
                          "aws:SourceArn": "%s"
                        }
                      }
                    }
                  ]
}''' % (queue_arn, topic_arn)
            }
        )
        return True
    except botocore.exceptions.ClientError as e:
        print_error(e)
    return False


if __name__ == "__main__":
    print('Creating S3 bucket %s.' % s3_bucket_name)
    if create_s3_bucket(s3_bucket_name, s3):
        print('\nCreating SNS topic %s.' % sns_topic_name)
        topic_arn = create_sns_topic(sns_topic_name, sns)
        if topic_arn:
            print('SNS topic created successfully: %s' % topic_arn)

            print('Creating SQS queue %s' % sqs_queue_name)
            queue_url = create_sqs_queue(sqs_queue_name, sqs)
            if queue_url is not None:
```

```
                print('Subscribing sqs queue with sns.')
                queue_arn = get_sqs_queue_arn(queue_url, sqs)
                if queue_arn is not None:
                    if set_sqs_policy(queue_url, queue_arn, sqs, topic_arn):
                        print('Successfully configured queue policy.')
                        subscription_arn = subscribe_to_sns_topic(topic_arn, sns,
'sqs', queue_arn)
                        if subscription_arn is not None:
                            if 'pending confirmation' in subscription_arn:
                                print('Please confirm SNS subscription by visiting the
subscribe URL.')
                            else:
                                print('Successfully subscribed SQS queue: ' +
queue_arn)
                        else:
                            print('Failed to subscribe SNS')
                    else:
                        print('Failed to set queue policy.')
                else:
                    print("Failed to get queue arn for %s" % queue_url)
            # ------------ End subscriptions to SNS topic -----------------

            print('\nSetting topic policy to allow s3 bucket %s to publish.' %
s3_bucket_name)
            if set_sns_topic_policy(topic_arn, sns, s3_bucket_name):
                print('SNS topic policy added successfully.')
                if set_s3_notification_sns(s3_bucket_name, s3, topic_arn):
                    print('Successfully configured event for S3 bucket %s' %
s3_bucket_name)
                    print('Create S3 Event Crawler using SQS ARN %s' % queue_arn)
                else:
                    print('Failed to configure S3 bucket notification.')
            else:
                print('Failed to add SNS topic policy.')
        else:
            print('Failed to create SNS topic.')
```

使用主控台為 Amazon S3 事件通知設定爬蟲程式 (Amazon S3 目標)

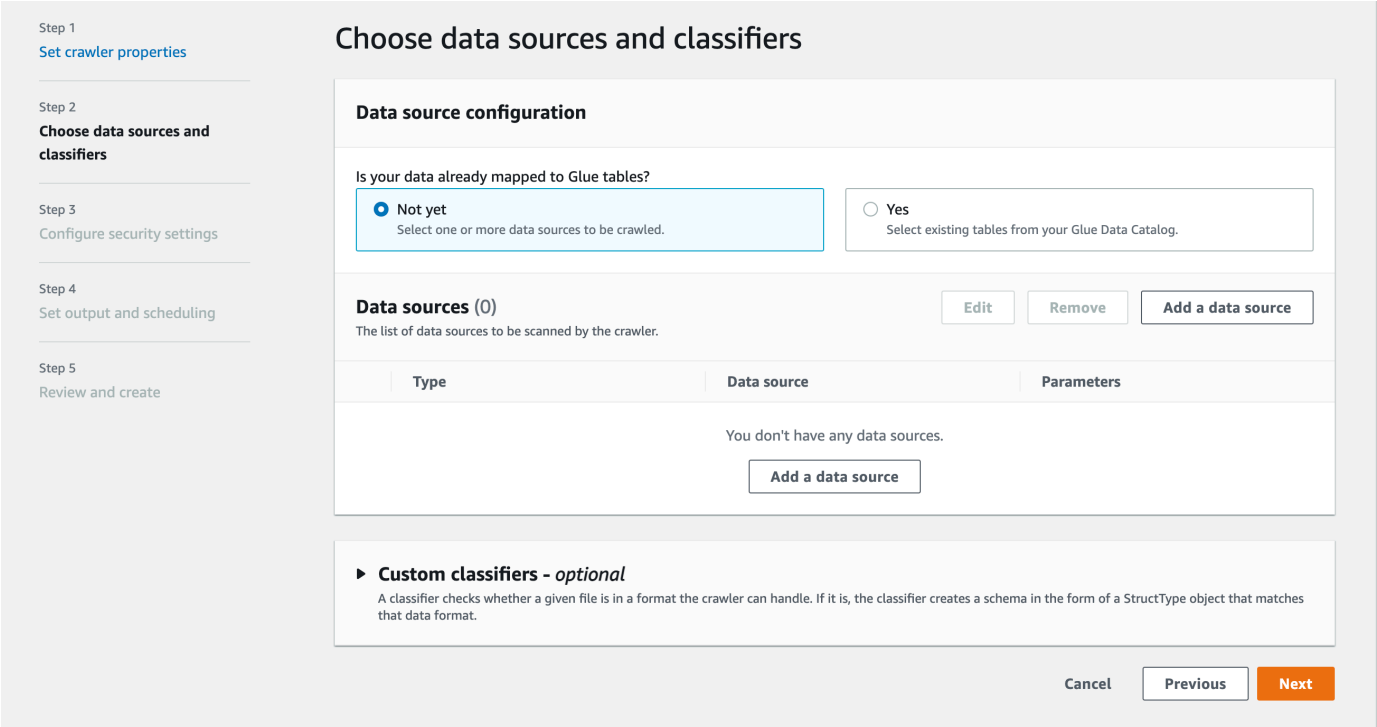
若要使用 AWS Glue 主控台為 Amazon S3 目標的 Amazon S3 事件通知設定爬蟲程式：

1. 設定爬蟲程式屬性。如需詳細資訊，請參閱在 AWS Glue 主控台上設定爬蟲程式組態選項。

2. 在 Data source configuration (資料來源組態) 區段中，您被問到 Is your data already mapped to AWS Glue tables? (您的資料是否已對應至 GLU 資料表？)

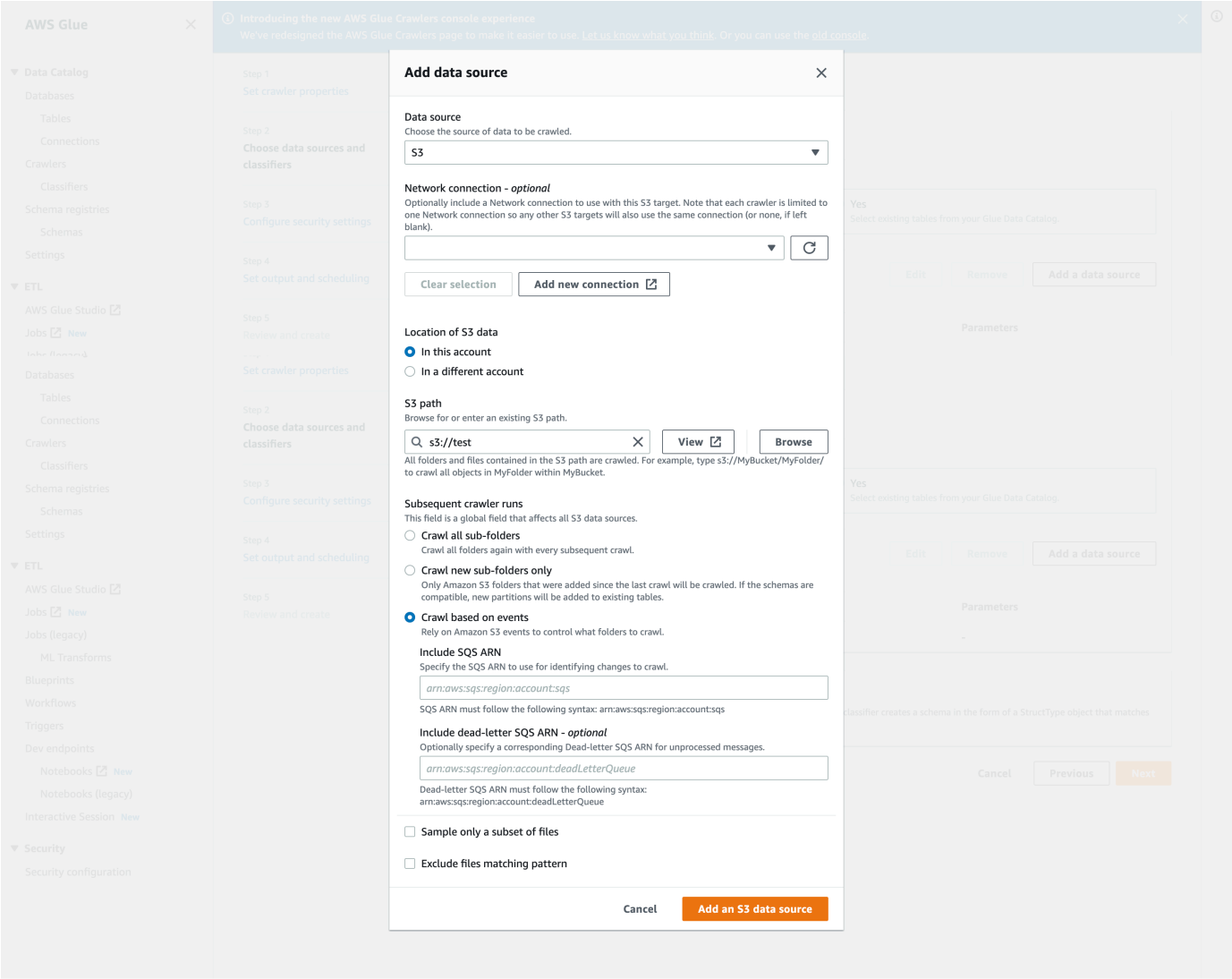
   依預設已選取 Not yet (尚未)。將其保留為預設值，因為您使用的是 Amazon S3 資料來源，且資料尚未映射至 AWS Glue 資料表。

3. 在 Data sources (資料來源) 區段中，選擇 Add a data source (新增資料來源)。



4. 在 Add data source (新增資料來源) 強制回應視窗中，設定 Amazon S3 資料來源：

   - Data source (資料來源)：依預設，已選取 Amazon S3。
   - Network connection (網路連線) (選用)：選擇 Add new connection (新增連線)。
   - Location of Amazon S3 data (Amazon S3 資料的位置)：依預設，已選取 In this account (在此帳戶中)。
   - Amazon S3 path (Amazon S3 路徑)：指定在其中網路爬取資料夾和檔案的 Amazon S3 路徑。
   - Subsequent crawler runs (後續爬蟲程式執行)：選擇 Crawl based on events (根據事件進行網路爬取) 以針對爬蟲程式使用 Amazon S3 事件通知。
   - Include SQS ARN (包含 SQS ARN)：指定包括有效 SQS ARN 的資料存放區參數。(例如 `arn:aws:sqs:region:account:sqs`)。
   - Include dead-letter SQS ARN (包含無效字母 SQS ARN) (選用)：指定有效的 Amazon 無效字母 SQS ARN。(例如 `arn:aws:sqs:region:account:deadLetterQueue`)。
   - 選擇 Add an Amazon S3 data source (新增 Amazon S3 資料來源)。



使用設定 Amazon S3 事件通知的爬蟲程式 AWS CLI

以下是在 Amazon S3 目標儲存貯體上建立 SQS 佇列和設定事件通知的 Amazon S3 AWS CLI 呼叫範例。

```
S3 Event AWS CLI
aws sqs create-queue --queue-name MyQueue --attributes file://create-queue.json
create-queue.json
```
```
{
    "Policy": {
```

```
        "Version": "2012-10-17",
        "Id": "example-ID",
        "Statement": [
            {
                "Sid": "example-statement-ID",
                "Effect": "Allow",
                "Principal": {
                    "Service": "s3.amazonaws.com"
                },
                "Action": [
                    "SQS:SendMessage"
                ],
                "Resource": "SQS-queue-ARN",
                "Condition": {
                    "ArnLike": {
                        "aws:SourceArn": "arn:aws:s3:*:*:awsexamplebucket1"
                    },
                    "StringEquals": {
                        "aws:SourceAccount": "bucket-owner-account-id"
                    }
                }
            }
        ]
    }
}
```

```
aws s3api put-bucket-notification-configuration --bucket customer-data-pdx --
notification-configuration file://s3-event-config.json
s3-event-config.json
```

```
{
    "QueueConfigurations": [
        {
          "Id": "s3event-sqs-queue",
          "QueueArn": "arn:aws:sqs:{region}:{account}:queuename",
          "Events": [
                "s3:ObjectCreated:*",
                "s3:ObjectRemoved:*"
          ],
          "Filter": {
              "Key": {
                 "FilterRules": [
                     {
                         "Name": "Prefix",
```

```
                        "Value": "/json"
                    }
                ]
            }
        }
    }
]
}
```

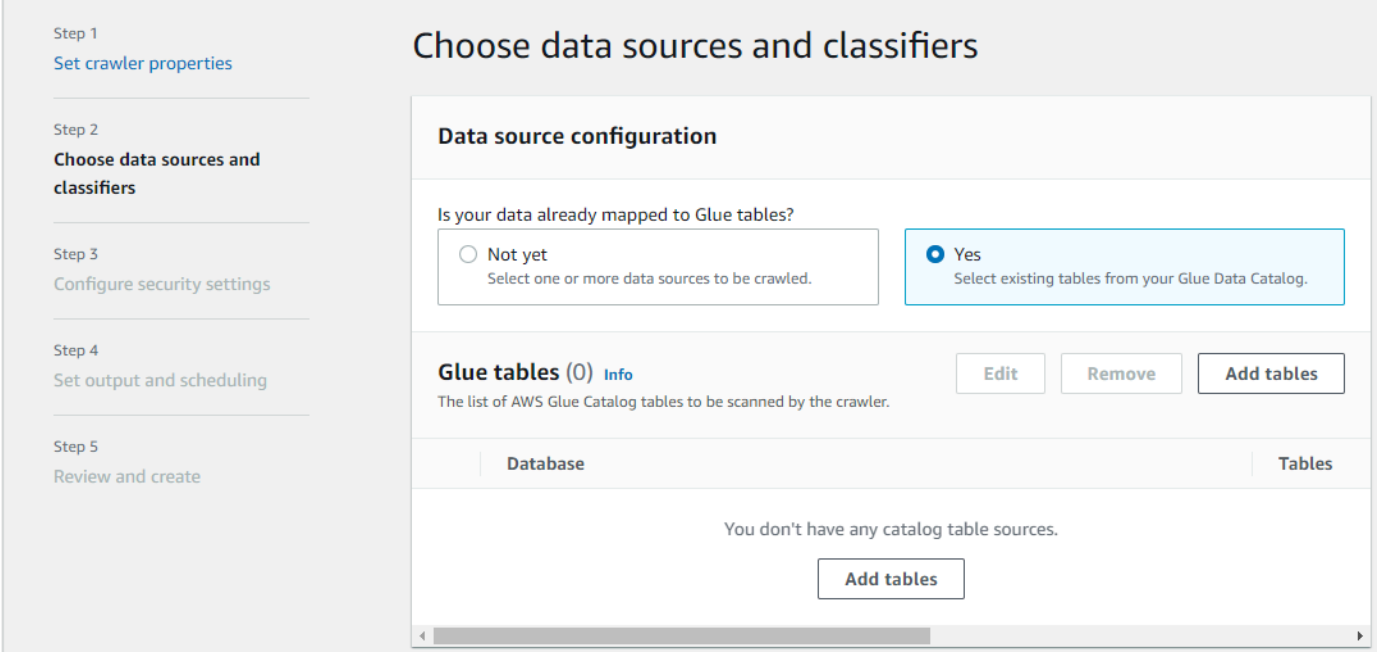
Create Crawler:

使用主控台為 Amazon S3 事件通知設定爬蟲程式 (Data Catalog 目標)

當您擁有目錄目標時，使用 AWS Glue 主控台對 Amazon S3 事件通知設定爬蟲程式：

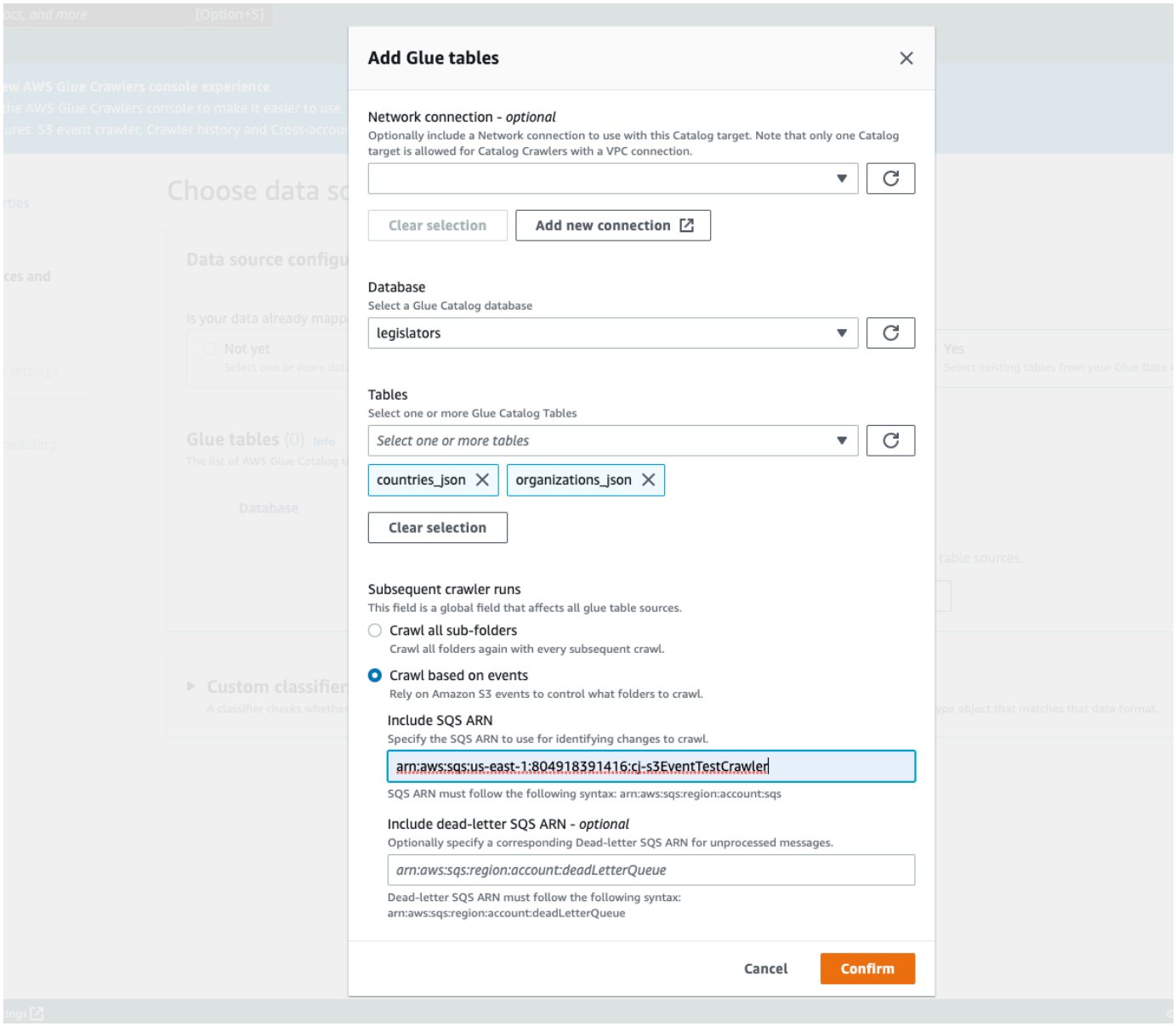
1. 設定爬蟲程式屬性。如需詳細資訊，請參閱在 AWS Glue 主控台上設定爬蟲程式組態選項。
2. 在 Data source configuration (資料來源組態) 區段中，您被問到 Is your data already mapped to AWS Glue tables? (您的資料是否已對應至 GLU 資料表？)

   選取 Yes (是)，從 Data Catalog 中選取現有資料表作為資料來源。
3. 在 Glue tables (Glue 資料表) 區段中，選擇 Add tables (新增資料表)。



4. 在 Add table (新增資料表) 模式中，設定資料庫和資料表：

- Network connection (網路連線) (選用)：選擇 Add new connection (新增連線)。
- Database (資料庫)：在 Data Catalog 中選取資料庫。
- Tables (資料表)：從 Data Catalog 的資料庫中選取一個或多個資料表。
- Subsequent crawler runs (後續爬蟲程式執行)：選擇 Crawl based on events (根據事件進行網路爬取) 以針對爬蟲程式使用 Amazon S3 事件通知。
- Include SQS ARN (包含 SQS ARN)：指定包括有效 SQS ARN 的資料存放區參數。(例如 `arn:aws:sqs:region:account:sqs`)。
- Include dead-letter SQS ARN (包含無效字母 SQS ARN) (選用)：指定有效的 Amazon 無效字母 SQS ARN。(例如 `arn:aws:sqs:region:account:deadLetterQueue`)。
- 選擇確認。



將加密與 Amazon S3 事件爬蟲程式搭配使用

本節說明僅在 SQS 上使用加密，或在 SQS 和 Amazon S3 上使用加密。

主題

- 僅在 SQS 上啟用加密
- 在 SQS 和 Amazon S3 上啟用加密
- 常見問答集

僅在 SQS 上啟用加密

Amazon SQS 預設會在傳輸過程中提供加密。若要新增選用的伺服器端加密 (SSE) 到佇列中，您可以在編輯面板中連接客戶主金鑰 (CMK)。這表示 SQS 會將 SQS 伺服器上的所有靜態客戶資料加密。

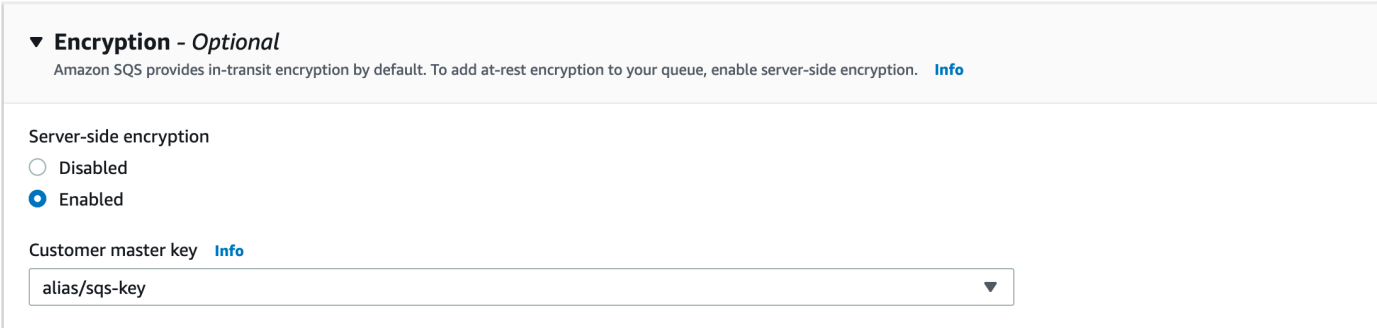
建立客戶主金鑰 (CMK)

1. 選擇 Key Management Service (KMS) (金鑰管理服務 (KMS)) > Customer Managed Keys (客戶受管金鑰) > Create key (建立金鑰)。
2. 依照以下步驟新增您自己的別名和說明。
3. 新增您希望能夠使用此金鑰的個別 IAM 角色。
4. 在金鑰政策中，將另一個陳述式新增至「陳述式」清單，讓您的自訂金鑰政策為 Amazon SNS 提供足夠的金鑰使用許可。

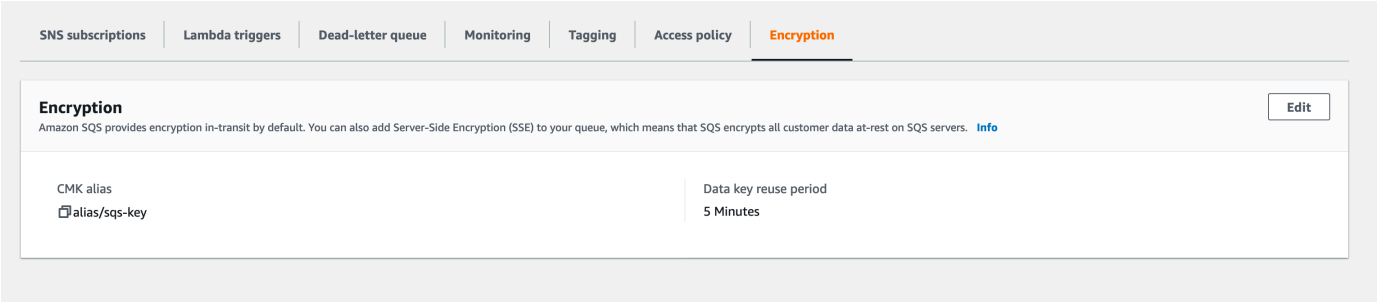
```
"Statement": [
       {
           "Effect": "Allow",
           "Principal": {
               "Service": "sns.amazonaws.com"
           },
           "Action": [
               "kms:GenerateDataKey",
               "kms:Decrypt"
           ],
           "Resource": "*"
       }
]
```

在佇列上啟用伺服器端加密 (SSE)

1. 選擇 Amazon SQS > Queues (佇列) > sqs_queue_name > Encryption (加密) 索引標籤。
2. 選擇 Edit (編輯)，然後向下捲動到 Encryption (加密) 下拉式功能表。
3. 選取 Enabled (啟用) 以新增 SSE。
4. 選取您之前建立的 CMK，而不是名稱為 `alias/aws/sqs` 的預設金鑰。



新增此金鑰後，您的「Encryption (加密)」索引標籤會以新增的金鑰更新。



> **Note**
>
> Amazon SQS 會自動刪除在佇列上存在超過訊息保留期間上限的訊息。預設的訊息保留期間為 4 天。若要避免遺失事件，請變更 SQS MessageRetentionPeriod 到最大的 14 天。

在 SQS 和 Amazon S3 上啟用加密

在 SQS 上啟用伺服器端加密 (SSE)

1. 請遵循 the section called “僅在 SQS 上啟用加密” 中的步驟。
2. 在 CMK 設定的最後一個步驟中，給予 Amazon S3 足夠的金鑰使用許可。

   將下列項目貼到「陳述式」清單中：

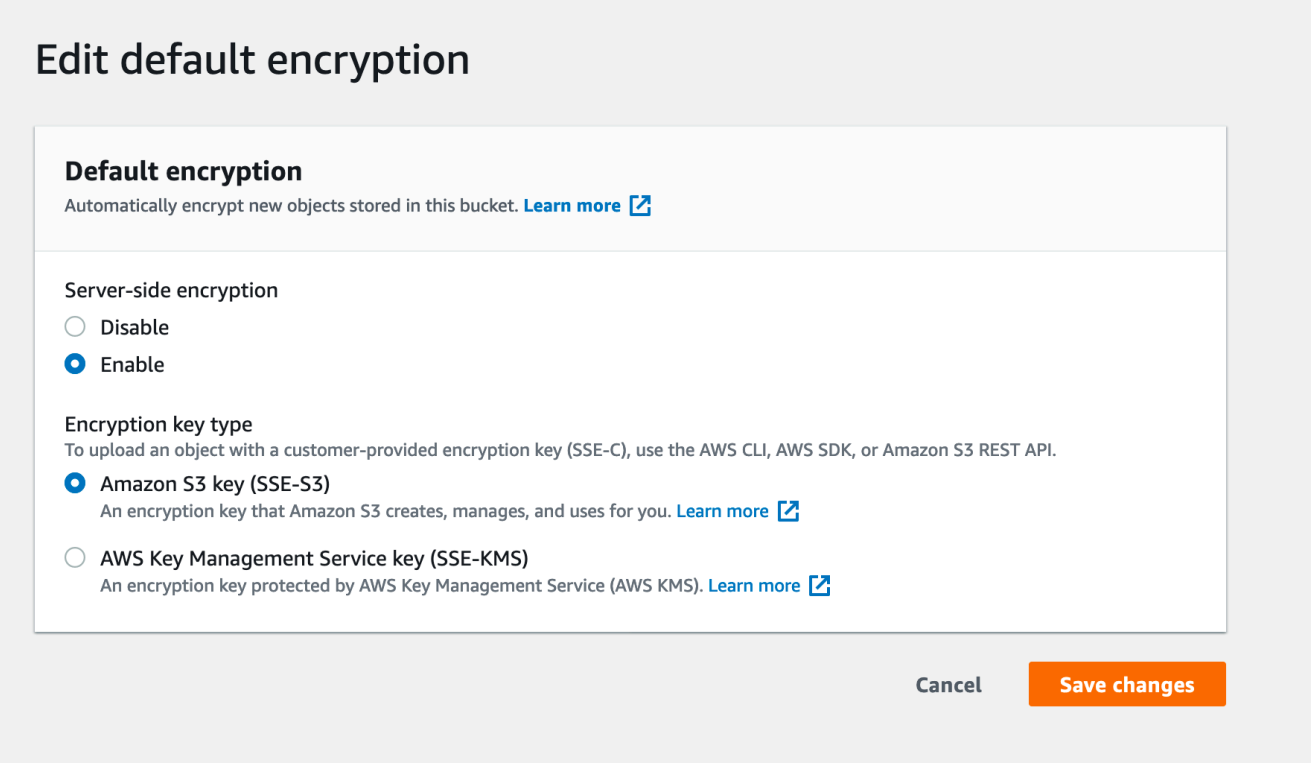
```
"Statement": [
      {
          "Effect": "Allow",
          "Principal": {
              "Service": "s3.amazonaws.com"
          },
```

```
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "*"
        }
]
```

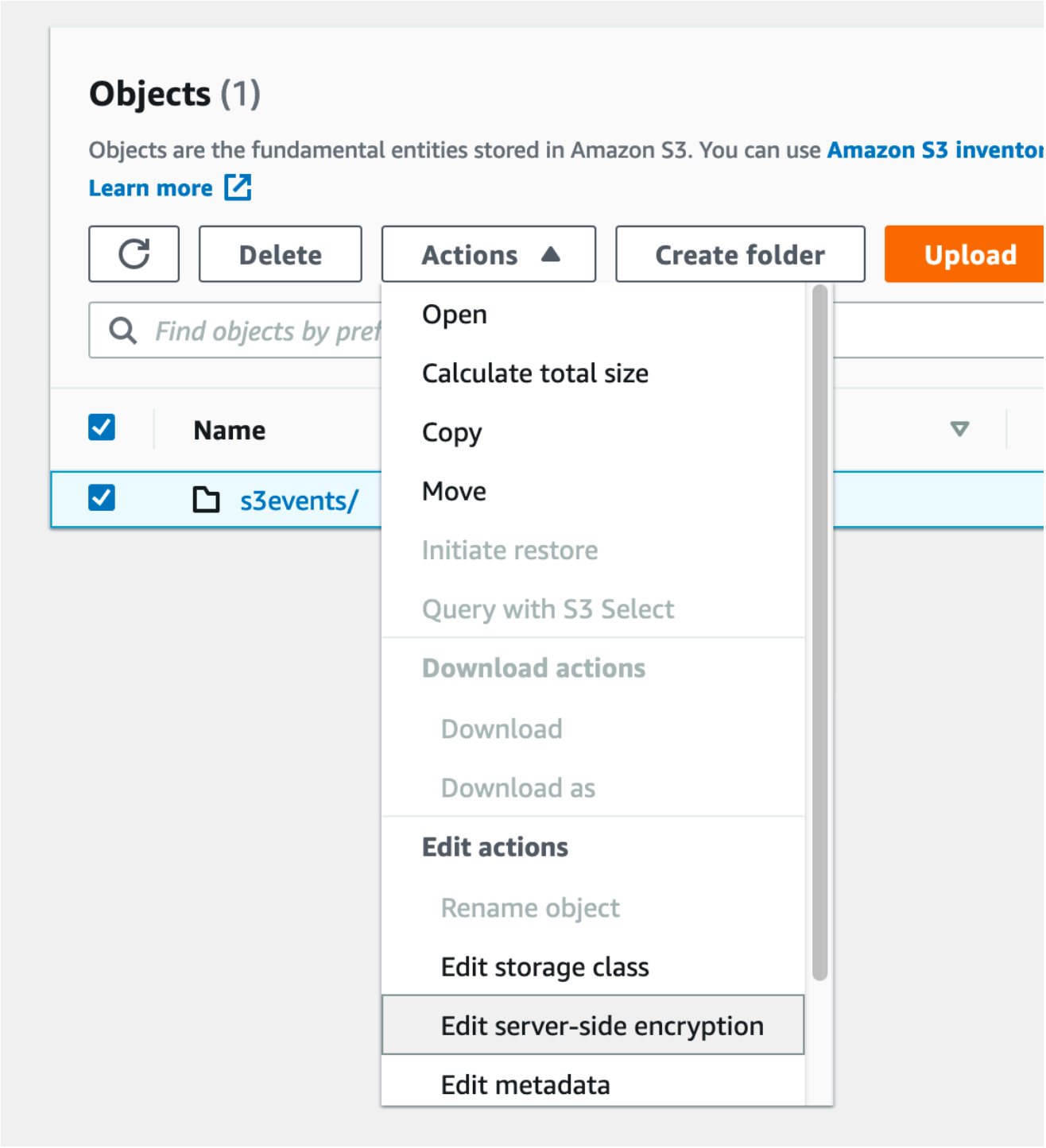
在 Amazon S3 儲存貯體上啟用伺服器端加密 (SSE)

1. 請遵循 the section called “僅在 SQS 上啟用加密” 中的步驟。
2. 執行下列任意一項：
   - 若要為整個 S3 儲存貯體啟用 SSE，請瀏覽至 Properties (屬性) 索引標籤。

     您可以在其中啟用 SSE 並選擇要使用的加密類型。Amazon S3 提供 Amazon S3 為您建立、管理和使用的加密金鑰，或者可以從 KMS 中選擇金鑰。



   - 若要在特定資料夾上啟用 SSE，請按一下目標資料夾旁的核取方塊，然後選擇 Actions (動作) 下拉式功能表下的 Edit server-side encryption (編輯伺服器端加密)。



常見問答集

為什麼發佈到 Amazon SNS 主題的訊息不會傳遞到已啟用伺服器端加密 (SSE) 的訂閱 Amazon SQS 佇列？

仔細檢查您的 Amazon SQS 佇列是否正在使用：

1. 由客戶管理的客戶主金鑰 (CMK)。不是 SQS 提供的預設金鑰。
2. (1) 中的 CMK 包括一個自訂金鑰政策，其為 Amazon SNS 提供足夠的金鑰使用許可。

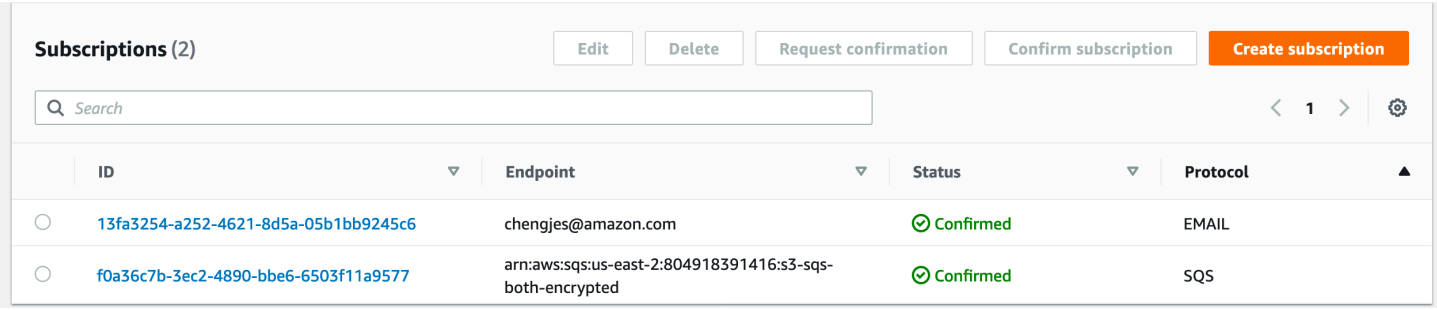
如需詳細資訊，請參閱知識中心的這篇文章。

我已訂閱電子郵件通知，但是當我編輯 Amazon S3 儲存貯體時，沒有收到任何電子郵件更新。

請按一下電子郵件中的「Confirm Subscription (確認訂閱)」連結，確定您已確認電子郵件地址。您可以透過檢查 SNS 主題下的 Subscriptions (訂閱) 資料表來驗證確認狀態。

選擇 Amazon SNS > Topics (主題) > `sns_topic_name` > Subscriptions table (訂閱資料表)。

如果您遵循我們的必要條件指令碼，您會發現 `sns_topic_name` 等於您的 `sqs_queue_name`。其看起來與下列類似：



在我的 SQS 佇列上啟用伺服器端加密後，只有我新增的一些資料夾顯示在資料表中。為什麼我遺失了一些 Parquet？

如果在 SQS 佇列上啟用 SSE 之前進行了 Amazon S3 儲存貯體變更，爬蟲程式可能不會拾取這些變更。若要確保您已編目 S3 儲存貯體的所有更新，請以清單模式 (「編目所有資料夾」) 再次執行爬蟲程式。另一種選擇是透過建立啟用 S3 事件的新爬蟲程式來重新開始編目。

如何防止爬蟲程式變更現有的結構描述

如果您不希望爬蟲程式覆寫您在 Amazon S3 資料表定義中對現有欄位的更新，請在主控台選擇選項 Add new columns only (僅新增欄位) 或設定組態選項 `MergeNewColumns`。這適用於資料表和分割區，除非 `Partitions.AddOrUpdateBehavior` 被覆寫為 `InheritFromTable`。

如果您不希望爬蟲程式執行時變更資料表結構描述，可將結構描述變更政策設為 `LOG`。您也可以設定組態選項，設定從資料表繼承分割區結構描述。

如果您在主控台設定爬蟲程式，您可以選擇下列動作：

- 忽略變更且不更新 Data Catalog 中的資料表

- Update all new and existing partitions with metadata from the table (以表格的中繼資料更新所有新的和現有的分割區)

當您使用 API 設定爬蟲程式時，請設定下列參數：

- 將 SchemaChangePolicy 結構中的 UpdateBehavior 欄位設為 LOG。
- 在爬蟲程式 API 中，以呈現以下 JSON 物件的字串設定 Configuration 欄位，例如：

```
{
   "Version": 1.0,
   "CrawlerOutput": {
      "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" }
   }
}
```

如何為每個 Amazon S3 包含路徑建立單一結構描述

依預設，爬蟲程式為 Amazon S3 中存放的資料定義資料表時，其會同時考量資料是否相容與結構描述是否類似。考量的資料相容性因素包括資料是否具有相同的格式類型 (例如 JSON)、相同的壓縮類型 (例如 GZIP)、Amazon S3 路徑結構及其他資料屬性。結構描述相似性是測量個別 Amazon S3 物件的結構描述相似程度。

當可能時，您可以設定爬蟲程式為 CombineCompatibleSchemas 到常用資料表定義。使用此選項，爬蟲程式仍會考慮資料相容性，但在評估特定包含路徑的 Amazon S3 物件時，會忽略特定結構描述的相似性。

如果您正在主控台設定爬蟲程式，若要結合結構描述，請選擇爬蟲程式選項 Create a single schema for each S3 path (為每個 S3 路徑建立一個單一結構描述)。

當您使用 API 設定爬蟲程式時，請設定下列組態選項：

- 在爬蟲程式 API 中，以呈現以下 JSON 物件的字串設定 Configuration 欄位，例如：

```
{
   "Version": 1.0,
   "Grouping": {
      "TableGroupingPolicy": "CombineCompatibleSchemas" }
}
```

為協助說明此選項，我們假設您使用了包含路徑 s3://bucket/table1/ 來定義爬蟲程式。當爬蟲程式執行時，它會尋找兩個具備以下特點的 JSON 檔案：

- 檔案 1 - S3://bucket/table1/year=2017/data1.json
- 檔案內容 - {“A”: 1, “B”: 2}
- 結構描述 - A:int, B:int

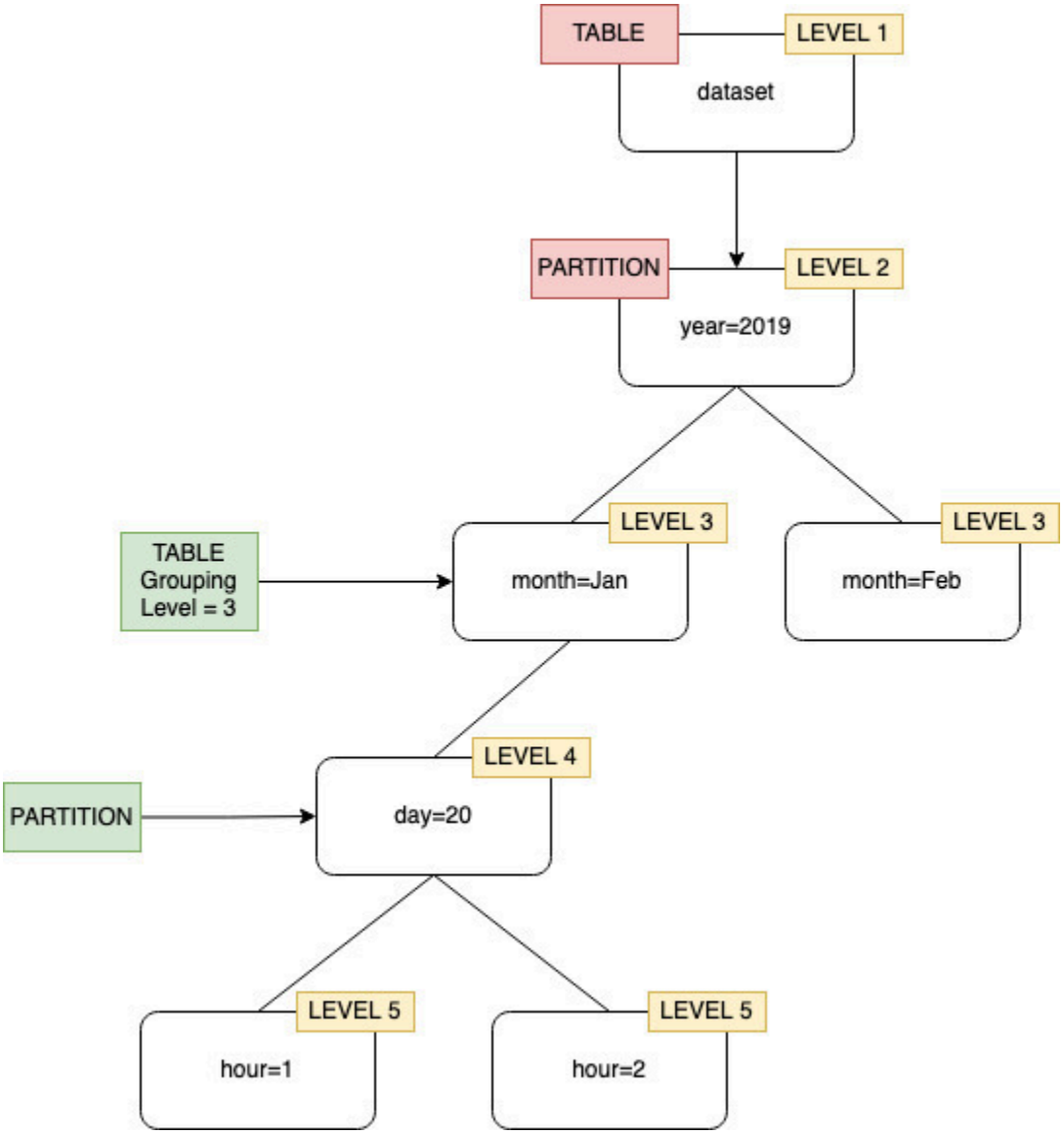
- 檔案 2 - S3://bucket/table1/year=2018/data2.json
- 檔案內容 - {“C”: 3, “D”: 4}
- 結構描述 - C: int, D: int

依預設，由於結構描述的沒有足夠相似性，故爬蟲程式會建立兩個資料表，名為 year_2017 和 year_2018。不過，如果已選擇 Create a single schema for each S3 path (為每個 S3 路徑建立單一結構描述)，且如果該資料相容，爬蟲程式會建立一個資料表。該資料表包含結構描述 A:int,B:int,C:int,D:int 和 partitionKey year:string。

如何指定資料表位置和分割層級

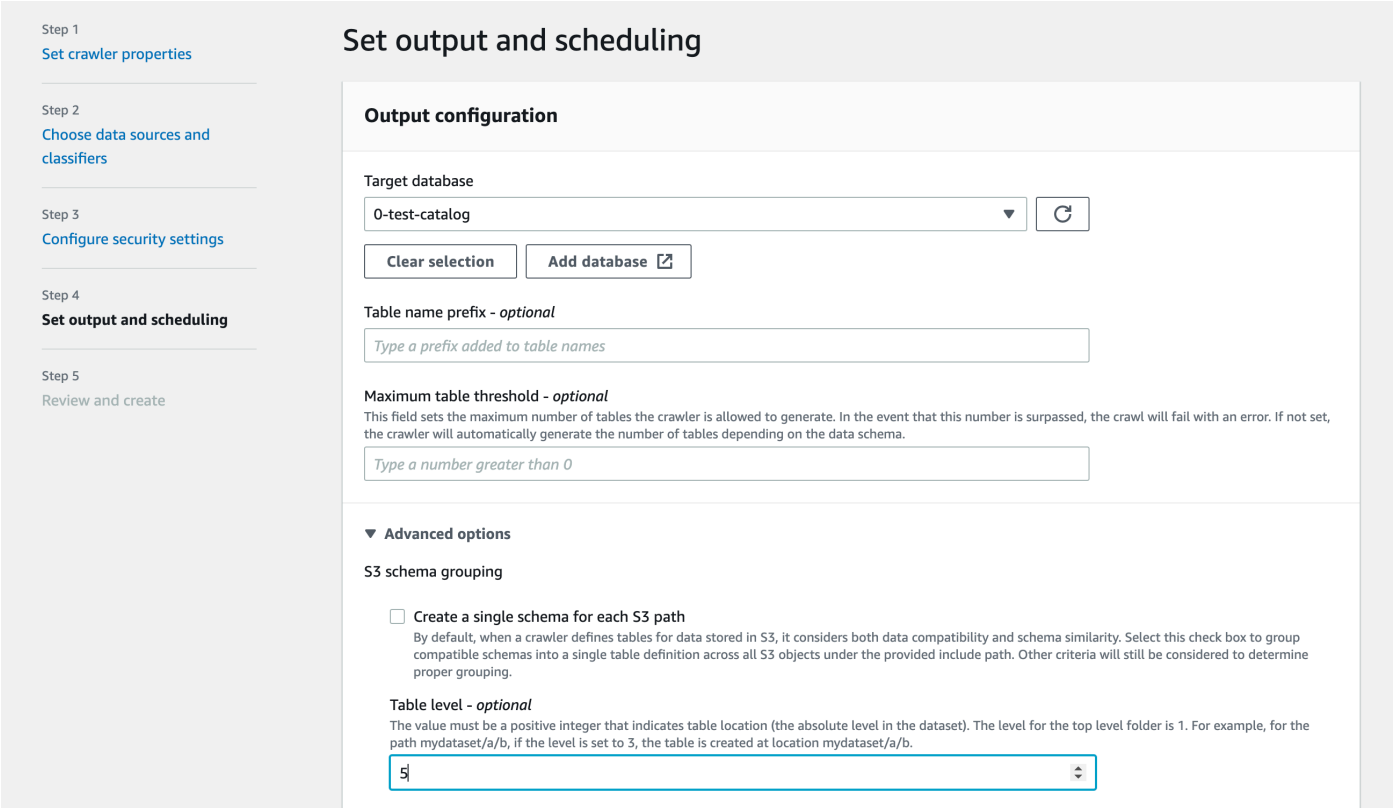
依預設，爬蟲程式為 Amazon S3 中存放的資料定義資料表時，爬蟲程式會嘗試將結構描述合併在一起並建立頂層資料表 (year=2019)。在某些情況下，您可能會預期爬蟲程式為資料夾 month=Jan 建立資料表，然而由於同級資料夾 (month=Mar) 已合併到相同的資料表中，因此爬蟲程式是建立分割區。

資料表層級爬蟲程式選項讓您可以靈活地告訴爬蟲程式資料表的位置，以及建立分割區的方式。當您指定 Table level (資料表層級)，資料表會從 Amazon S3 儲存貯體在該絕對層級建立。



在主控台設定爬蟲程式時，您可指定 Table level (資料表層級) 爬蟲程式選項的值。值必須是正整數，表示資料表位置 (資料集中的絕對層級)。頂層資料夾的層級為 1。例如，對於路徑 `mydataset/year/month/day/hour`，如果層級設定為 3，則資料表會在位置 `mydataset/year/month`。

Console



API

使用 API 設定爬蟲程式時，以呈現以下 JSON 物件的字串設定 `Configuration` 欄位；例如：

```
configuration = jsonencode(

{
   "Version": 1.0,
   "Grouping": {
            TableLevelConfiguration = 2
        }
})
```

CloudFormation

在此範例中，您可以在範 CloudFormation 本中設定主控台中可用的 [資料表層級] 選項：

```
"Configuration": "{
    \"Version\":1.0,
    \"Grouping\":{\"TableLevelConfiguration\":2}
}"
```

如何指定爬蟲程式可建立的資料表數目上限

您可以TableThreshold透過 AWS Glue 主控台或 CLI 指定，選擇性地指定允許爬行者程式建立的表格數目上限。如果爬蟲程式在其網路爬取期間偵測到的資料表大於此輸入值，網路爬取就會失敗，且不會將任何資料寫入資料型錄。

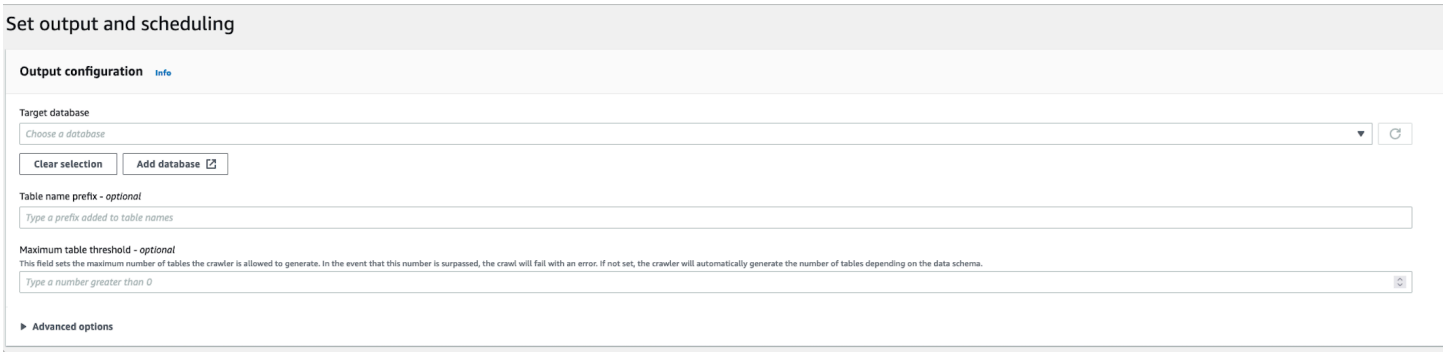
當爬蟲程式偵測並建立的資料表比您預期的要大得多時，此參數非常有用。這可能有多種原因，例如：

- 使用 AWS Glue 任務填入 Amazon S3 位置時，最終可能會出現與資料夾相同層級的空檔案。在這種情況下，當您在此 Amazon S3 位置執行爬蟲程式時，由於檔案和資料夾存在於相同層級，爬蟲程式會建立多個資料表。
- 如果未設定 `"TableGroupingPolicy": "CombineCompatibleSchemas"`，您最終可能會得到比預期更多的資料表。

您可以指定 `TableThreshold` 作為大於 0 的整數值。此值的設定是以每個爬蟲程式為基礎。也就是說，對於每個網路爬取，都會考慮此值。例如：爬蟲程式具有設定為 5 的 `TableThreshold` 值。在每個編目中，會 AWS Glue 比較偵測到的表格數目與此資料表臨界值 (5)，如果偵測到的資料表數目少於 5，則會將資料表 AWS Glue 寫入「資料目錄」，如果沒有，編目就會失敗，而不寫入「資料目錄」。

主控台

若要TableThreshold使用 AWS 主控台進行設定：



CLI

若要TableThreshold使用 AWS CLI 進行設定：

```
"{"Version":1.0,
"CrawlerOutput":
{"Tables":{"AddOrUpdateBehavior":"MergeNewColumns",
```

```
"TableThreshold":5}}}";
```

錯誤訊息會記錄下來，以協助您識別資料表路徑並清理資料。在爬蟲程式因為資料表計數大於提供的資料表閾值而失敗的情況下，登入帳戶的範例：

```
Table Threshold value = 28, Tables detected - 29
```

在中 CloudWatch，我們記錄檢測為 INFO 消息的所有表位置。將錯誤記錄為失敗原因。

```
ERROR com.amazonaws.services.glue.customerLogs.CustomerLogService - CustomerLogService
 received CustomerFacingException with message
The number of tables detected by crawler: 29 is greater than the table threshold value
 provided: 28. Failing crawler without writing to Data Catalog.
com.amazonaws.services.glue.exceptions.CustomerFacingInternalException: The number of
 tables detected by crawler: 29 is greater than the table threshold value provided:
 28.
Failing crawler without writing to Data Catalog.
```

如何為 Delta Lake 的資料儲存指定配置選項

為 Delta Lake 資料儲存設定爬蟲程序時，請指定以下設定參數：

連線

選擇性地選取或新增要用於此 Amazon S3 目標的網路連線。如需連線的詳細資訊，請參閱 連線至資料。

建立要查詢的資料表

選取建立 Delta Lake 資料表的方式：

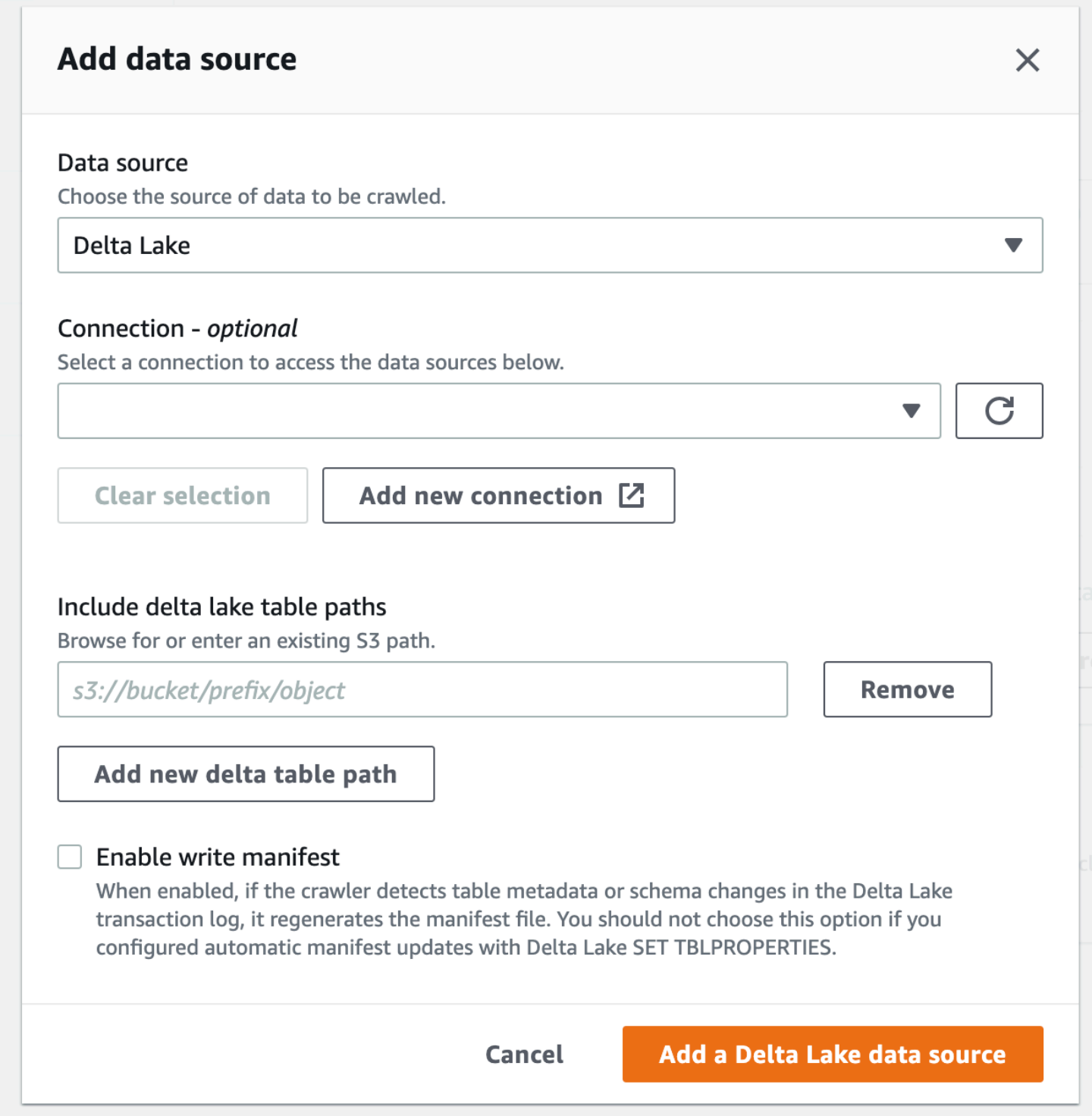
- 建立原生資料表：允許與支援直接查詢 Delta 交易日誌的查詢引擎整合。
- 建立符號連結資料夾：根據特定組態參數，使用由分割區索引鍵分割的資訊清單檔案建立符號連結清單檔案資料夾。

啟用寫入清單檔案 (僅在您已針對 Delta Lake 來源選取建立 Symlink 資料表時才可設定)

選擇是否偵測 Delta Lake 交易記錄中的資料表中繼資料或結構描述變更；這會重新產生資訊清單檔案。如果您使用 Delta Lake SET TBLPROPERTIES 設定了自動更新資訊清單，則不應選擇此選項。

包括 Delta Lake 資料表路徑

將一個或多個指向 Delta 資料表的 Amazon S3 路徑指定為：s3://*bucket*/*prefix*/*object*。



如何設定爬蟲程式以使用 Lake Formation 憑證

您可以將爬蟲設定為使用 AWS Lake Formation 登入資料存取 Amazon S3 資料存放區或資料目錄表格，其中的基礎 Amazon S3 位置位於相同 AWS 帳戶 或另一個 AWS 帳戶位置。如果爬蟲程式和資料

型錄資料表位於同一帳戶中，您可以將現有的資料型錄資料表設定為爬蟲程式的目標。目前，使用資料型錄資料表作為爬蟲程式的目標時，僅允許具有單一型錄資料表的單一型錄目標。

> **Note**
>
> 將資料型錄資料表定義為爬蟲程式目標時，請確定資料型錄資料表的基礎位置是 Amazon S3 位置。使用 Lake Formation 憑證的爬蟲程式僅支援具有基礎 Amazon S3 位置的資料型錄目標。

當爬蟲程式和已註冊的 Amazon S3 位置或資料型錄資料表位於同一帳戶 (帳戶內網路爬取) 時，需要進行設定

若要允許爬蟲程式使用 Lake Formation 憑證存取資料存放區或資料型錄資料表，您需要向 Lake Formation 註冊資料位置。此外，爬蟲程式的 IAM 角色必須具備從 Amazon S3 儲存貯體註冊所在的目的地讀取資料的許可。

您可以使用 AWS Management Console 或 AWS Command Line Interface (AWS CLI) 完成下列組態步驟。

AWS Management Console

1. 在設定爬蟲程式以存取爬蟲程式來源之前，請先向 Lake Formation 註冊資料存放區或資料型錄的資料位置。在 Lake Formation 主控台 (https://console.aws.amazon.com/lakeformation/) 中，將 Amazon S3 位置註冊為定義爬蟲程式所 AWS 帳戶 在資料湖的根位置。如需詳細資訊，請參閱 Registering an Amazon S3 location (註冊 Amazon S3 位置)。
2. 向用於爬蟲程式執行的 IAM 角色授與 Data location (資料位置) 許可，以便爬蟲程式可以從 Lake Formation 中的目的地讀取資料。如需詳細資訊，請參閱 Granting data location permissions (same account) (授與資料位置許可 (相同帳戶))。
3. 將爬蟲程式角色存取許可 (`Create`) 授與指定作為輸出資料庫的資料庫。如需詳細資訊，請參閱 Granting database permissions using the Lake Formation console and the named resource method (使用 Lake Formation 主控台和具名資源方法授與資料庫許可)。
4. 在 IAM 主控台 (https://console.aws.amazon.com/iam/) 中，建立爬蟲程式的 IAM 角色。將 `lakeformation:GetDataAccess` 政策新增至該角色。
5. 在 AWS Glue 主控台 (https://console.aws.amazon.com/glue/) 中，設定爬蟲程式時，選取「使用 Lake Formation 登入資料來爬取 Amazon S3 資料來源」選項。

> **Note**
>
> accountId 欄位對於帳戶內網路爬取而言是選填的。

AWS CLI

```
aws glue --profile demo create-crawler --debug --cli-input-json '{
    "Name": "prod-test-crawler",
    "Role": "arn:aws:iam::111122223333:role/service-role/AWSGlueServiceRole-prod-
test-run-role",
    "DatabaseName": "prod-run-db",
    "Description": "",
    "Targets": {
    "S3Targets":[
                {
                 "Path": "s3://crawl-testbucket"
                }
                ]
                },
   "SchemaChangePolicy": {
      "UpdateBehavior": "LOG",
      "DeleteBehavior": "LOG"
  },
  "RecrawlPolicy": {
    "RecrawlBehavior": "CRAWL_EVERYTHING"
  },
  "LineageConfiguration": {
    "CrawlerLineageSettings": "DISABLE"
  },
  "LakeFormationConfiguration": {
    "UseLakeFormationCredentials": true,
    "AccountId": "111122223333"
  },
  "Configuration": {
           "Version": 1.0,
           "CrawlerOutput": {
             "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" },
             "Tables": {"AddOrUpdateBehavior": "MergeNewColumns" }
           },
           "Grouping": { "TableGroupingPolicy": "CombineCompatibleSchemas" }
         },
```

```
  "CrawlerSecurityConfiguration": "",
  "Tags": {
    "KeyName": ""
  }
}'
```

爬蟲程式和註冊的 Amazon S3 位置位於不同帳戶 (跨帳戶網路爬取) 時需要進行設定

若要允許爬蟲程式使用 Lake Formation 憑證存取其他帳戶中的資料存放區，您必須先向 Lake Formation 註冊 Amazon S3 資料位置。接著，您可以透過執行下列步驟，將資料位置許可授與爬蟲程式的帳戶。

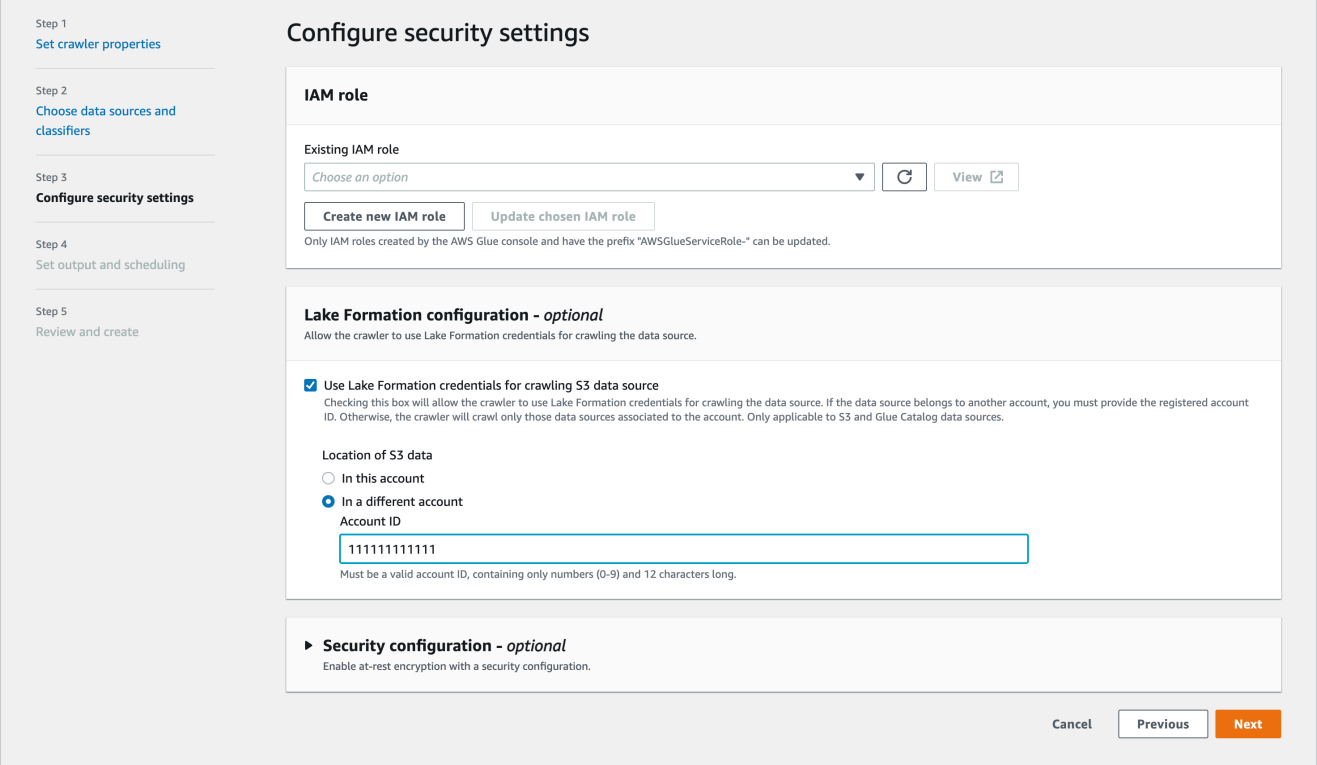
您可以使用 AWS Management Console 或來完成下列步驟 AWS CLI。

AWS Management Console

1. 在註冊 Amazon S3 位置的帳戶中 (帳戶 B)：
   a. 向 Lake Formation 註冊 Amazon S3 路徑。如需詳細資訊，請參閱 Registering Amazon S3 location (註冊 Amazon S3 位置)。
   b. 將 Data location (資料位置) 許可授與爬蟲程式執行所在的帳戶 (帳戶 A)。如需詳細資訊，請參閱 Grant data location permissions (授與資料位置許可)。
   c. 在 Lake Formation 中建立空的資料庫，並將基礎位置作為目標 Amazon S3 位置。如需詳細資訊，請參閱建立資料庫。
   d. 授與帳戶 A (爬蟲程式執行所在的帳戶) 您在上一個步驟中建立的資料庫存取權。如需詳細資訊，請參閱授與資料庫許可。
2. 在爬蟲程式建立並將執行所在的帳戶中 (帳戶 A)：
   a. 使用 AWS RAM 主控台，接受從外部帳戶 (帳戶 B) 共用的資料庫。如需詳細資訊，請參閱接受來源共用邀請 AWS Resource Access Manager。
   b. 建立爬蟲程式的 IAM 角色。將 `lakeformation:GetDataAccess` 政策新增至該角色。
   c. 在 Lake Formation 主控台 (https://console.aws.amazon.com/lakeformation/) 中，將目標 Amazon S3 位置的 Data location (資料位置) 許可授與用於爬蟲程式執行的 IAM 角色，以便爬蟲程式可以從 Lake Formation 中的目的地讀取資料。如需詳細資訊，請參閱 Granting data location permissions (授與資料位置許可)。
   d. 在共用資料庫上建立資源連結。如需詳細資訊，請參閱建立資源連結。
   e. 授與爬蟲程式角色共用資料庫和 (`Describe`) 資源連結的存取許可 (`Create`)。資源連結在爬蟲程式的輸出中指定。

f. 在 AWS Glue 主控台 (https://console.aws.amazon.com/glue/) 中，設定爬行者程式時，選取「使用 Lake Formation 登入資料來爬取 Amazon S3 資料來源」選項。

對於跨帳戶探索，請指定目標 Amazon S3 位置向 Lake Formation 註冊的 AWS 帳戶 ID。對於帳戶內網路爬取，accountId 欄位是選填的。



AWS CLI

```
aws glue --profile demo create-crawler --debug --cli-input-json '{
    "Name": "prod-test-crawler",
    "Role": "arn:aws:iam::111122223333:role/service-role/AWSGlueServiceRole-prod-
test-run-role",
    "DatabaseName": "prod-run-db",
    "Description": "",
    "Targets": {
    "S3Targets":[
                {
                 "Path": "s3://crawl-testbucket"
                }
                ]
                },
   "SchemaChangePolicy": {
```

```
      "UpdateBehavior": "LOG",
      "DeleteBehavior": "LOG"
  },
  "RecrawlPolicy": {
    "RecrawlBehavior": "CRAWL_EVERYTHING"
  },
  "LineageConfiguration": {
    "CrawlerLineageSettings": "DISABLE"
  },
  "LakeFormationConfiguration": {
    "UseLakeFormationCredentials": true,
    "AccountId": "111111111111"
  },
  "Configuration": {
           "Version": 1.0,
           "CrawlerOutput": {
             "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" },
             "Tables": {"AddOrUpdateBehavior": "MergeNewColumns" }
           },
           "Grouping": { "TableGroupingPolicy": "CombineCompatibleSchemas" }
         },
  "CrawlerSecurityConfiguration": "",
  "Tags": {
    "KeyName": ""
  }
}'
```

Note

- 只有 Amazon S3 和資料型錄目標才支援使用 Lake Formation 憑證的爬蟲程式。
- 對於使用 Lake Formation 憑證販售的目標，基礎 Amazon S3 位置必須屬於同一個儲存貯體。例如，只要所有目標位置都位於同一個儲存貯體 (bucket1) 下，客戶就可以使用多個目標 (s3://bucket1/folder1、s3://bucket1/folder2)。不允許指定不同的儲存貯體 (s3://bucket1/folder1、s3://bucket2/folder2)。
- 目前對於資料型錄目標爬蟲程式而言，僅允許具有單一型錄資料表的單一型錄目標。

教學課程：新增 AWS Glue 爬蟲程式

對於這個 AWS Glue 案例，您需要分析主要航空公司的抵達資料，以計算出發機場每月的受歡迎程度。您有以 CSV 格式存放在 Amazon S3 中的 2016 年航班資料。在轉換和分析資料之前，您先將其中繼資料編目至 AWS Glue Data Catalog。

在本教學課程中，讓我們在 Amazon S3 中新增可推斷這些航班記錄中繼資料的爬蟲程式，並在 Data Catalog 中建立資料表。

主題

先決條件

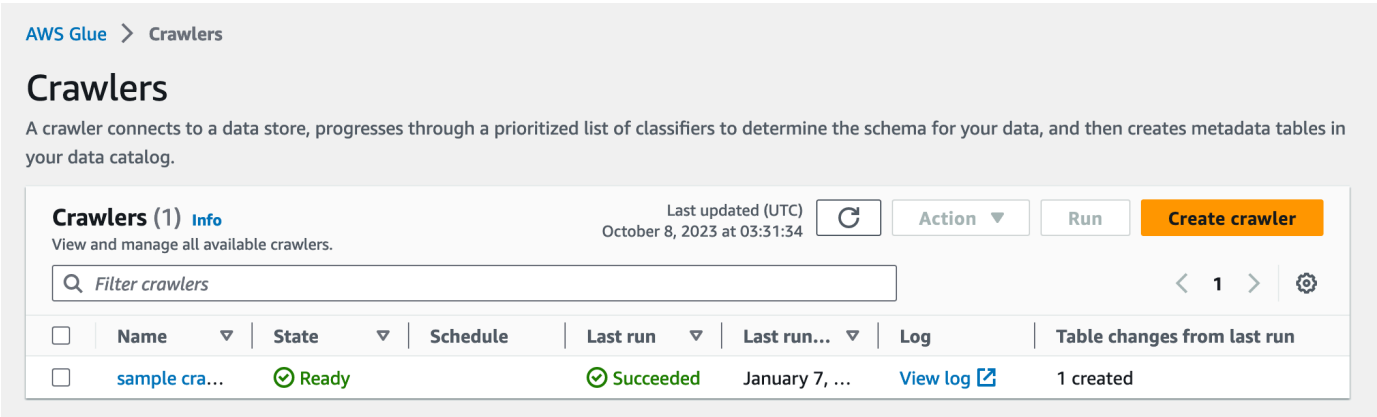
本教學課程假設您擁有 AWS 帳戶並能存取 AWS Glue。

步驟 1：新增爬蟲程式

使用這些步驟來設定和執行爬蟲程式，該爬蟲程式會從存放在 Amazon S3 的 CSV 檔案中擷取中繼資料。

建立讀取存放在 Amazon S3 上之檔案的爬蟲程式

1. 在 AWS Glue 服務主控台的左側選單上，選擇 Crawlers (爬蟲程式)。
2. 在爬蟲程式頁面上，選擇建立爬蟲程式。這會啟動一系列頁面，提示您輸入爬蟲程式詳細資訊。



3. 在爬蟲程式名稱欄位中，輸入 **`Flights Data Crawler`**，然後選擇 Next (下一步)。

   爬蟲程式會呼叫分類器，以推斷資料的結構描述。本教學課程預設使用 CSV 的內建分類器。

4. 對於爬蟲程式來源類型，選擇 Data stores (資料存放區)，然後選擇 Next (下一步)。

5. 現在讓我們將爬蟲程式指向您的資料。在 Add a data store (新增資料存放區) 頁面上，選擇 Amazon S3 資料存放區。本教學課程不使用連線，因此請將 Connection (連線) 欄位保留空白 (如果可見)。

   對於 Crawl data in (抓取資料位置) 選項，選擇 Specified path in another account (指定其他帳戶內的路徑)。然後，對於 Include path (包含路徑)，輸入爬蟲程式可以找到航班資料的路徑，也就是 **`s3://crawler-public-us-east-1/flight/2016/csv`**。輸入路徑後，此欄位的標題會變更為 Include path (包含路徑)。選擇 Next (下一步)。

6. 您可以使用單一爬蟲程式來編目多個資料存放區。但是，在本教學課程中，我們只使用一個資料存放區，因此選擇 No (否)，然後選擇 Next (下一步)。

7. 爬蟲程式需要許可才能存取資料存放區，並在 AWS Glue Data Catalog 中建立物件。若要設定這些許可，請選擇 Create an IAM role (建立 IAM 角色)。IAM 角色名稱的開頭為 `AWSGlueServiceRole-`，然後在欄位中輸入角色名稱的最後一部分。輸入 **`CrawlerTutorial`**，然後選擇 Next (下一步)。

   > **Note**
   >
   > 若要建立 IAM 角色，您的 AWS 使用者必須擁有 `CreateRole`、`CreatePolicy` 以及 `AttachRolePolicy` 許可。

   精靈會建立名為 `AWSGlueServiceRole-CrawlerTutorial` 的 IAM 角色、連接 AWS 受管政策 `AWSGlueServiceRole` 至此角色，並新增允許對 Amazon S3 位置 `s3://crawler-public-us-east-1/flight/2016/csv` 的讀取存取權限的內嵌政策。

8. 建立爬蟲程式的排程。對於 Frequency (頻率) 選擇 Run on demand (隨需執行)，然後選擇 Next (下一步)。

9. 爬蟲程式會在您的 Data Catalog 中建立資料表。資料表包含在 Data Catalog 的資料庫中。首先，選擇 Add database (新增資料庫) 建立資料庫。在彈出式視窗中，輸入 **`test-flights-db`** 做為資料庫名稱，然後選擇 Create (建立)。

   接下來，針對 Prefix added to tables (新增至資料表的字首) 輸入 **`flights`**。針對其餘的選項，使用預設值並選擇 Next (下一步)。

10. 驗證您在 Add crawler (新增爬蟲程式) 精靈中做出的選擇。如果您看到任何錯誤，您可以選擇 Back (上一步) 返回上一頁並進行變更。

    檢閱完資訊後，請選擇 Finish (完成) 建立爬蟲程式。

步驟 2：執行爬蟲程式

建立爬蟲程式之後，精靈會將您傳送至爬蟲程式檢視頁面。因為您建立具有隨需排程的爬蟲程式，所以您可以選擇執行爬蟲程式。

執行爬蟲程式

1. 此頁面頂端附近的橫幅可讓您知道已建立爬蟲程式，並詢問您是否要立即執行它。選擇 Run it now? (現在執行？) 執行爬蟲程式。

   橫幅會變更為針對您的爬蟲程式顯示「正在嘗試執行」和「正在執行」訊息。爬蟲程式開始執行後，橫幅就會消失，而且爬蟲程式顯示會更新，以顯示爬蟲程式的 Starting (啟動中) 狀態。一分鐘後，您可以按一下 [重新整理] 圖示來更新資料表中顯示之爬蟲程式的狀態。
2. 爬蟲程式完成時，會出現一個新的橫幅，說明爬蟲程式所做的變更。您可以選擇 test-flights-db 連結以檢視 Data Catalog 物件。

步驟 3：檢視 AWS Glue Data Catalog 物件

爬蟲程式會讀取來源位置的資料，並在 Data Catalog 中建立資料表。資料表是呈現您資料的中繼資料定義，包括其結構描述。 Data Catalog 中的資料表不包含資料。相反地，您可以使用這些資料表做為任務定義中的來源或目標。

檢視爬蟲程式建立的 Data Catalog 物件

1. 在左側導覽的 Data catalog (Data Catalog) 下，選擇 Databases (資料庫)。在這裡，您可以查看爬蟲程式所建立的 `flights-db` 資料庫。
2. 在左側導覽中的 Data catalog (Data Catalog) 及 Databases (資料庫) 下方，選擇 Tables (資料表)。在這裡，您可以檢視爬蟲程式所建立的 `flightscsv` 資料表。如果您選擇資料表名稱，則可以檢視資料表設定、參數和屬性。在此檢視中向下捲動，您可以檢視結構描述，也就是資料表的欄和資料類型的相關資訊。
3. 如果選擇資料表檢視頁面上的 View partitions (檢視分割區)，您可以看到為資料建立的分割區。第一欄是資料分割區索引鍵。

手動定義元資料

資 AWS Glue 料目錄是儲存有關資料來源和資料集的中繼資料的中央儲存庫。雖然爬行者程式可以自動編目並填入支援資料來源的中繼資料，但在某些情況下，您可能需要在「資料目錄」中手動定義中繼資料：

- 不支援的資料格式 — 如果您有爬行者程式不支援的資料來源，則需要在「資料目錄」中手動定義這些資料來源的中繼資料。
- 自訂中繼資料需求 — 根據預先定義的規則和慣例 AWS Glue 編目程式 推斷中繼資料。如果您的特定中繼資料需求未涵蓋於 AWS Glue 編目程式 推斷的中繼資料中繼資料，您可以手動定義中繼資料以符合您的需求
- 資料控管和標準化 — 在某些情況下，基於資料控管、合規性或安全性原因，您可能希望對中繼資料定義有更多控制權。手動定義中繼資料可讓您確保中繼資料符合組織的標準和原則。
- future 資料擷取的預留位置 — 如果您有無法立即使用或無法存取的資料來源，您可以建立空的結構定義表格作為預留位置。資料來源可供使用之後，您可以在表格中填入實際資料，同時保留預先定義的結構。

若要手動定義中繼資料，您可以使用 AWS Glue 主控台、Lake Formation 主控台、 AWS Glue API 或 AWS Command Line Interface (AWS CLI)。您可以建立資料庫、表格和分割區，並指定中繼資料屬性，例如欄名稱、資料類型、說明和其他屬性。

建立資料庫

資料庫用於在 AWS Glue中組織中繼資料資料表。當您在中定義表格時 AWS Glue Data Catalog，您可以將其加入至資料庫。資料表一次只能隸屬於一個資料庫。

您的資料庫可包含多個資料表，用來定義來自多種不同資料存放區的資料。此資料可包含 Amazon Simple Storage Service (Amazon S3) 中物件，以及 Amazon Relational Database Service 中的關聯資料表。

> **Note**
>
> 當您從 AWS Glue Data Catalog 刪除資料庫，也會刪除資料庫中內含的所有資料表。

若要檢視資料庫清單，請登入 AWS Management Console 並開啟AWS Glue主控台，網址為 https://console.aws.amazon.com/glue/。選擇 Databases (資料庫)，接著在清單中選擇資料庫名稱，以檢視詳細資訊。

您可以在 主控台的 DatabasesAWS Glue 標籤新增、編輯、刪除資料庫：

- 要建立新資料庫，請選擇 Add database (新增資料庫) 並提供名稱和描述。為了相容於其他中繼資料存放 (例如 Apache Hive)，名稱會變為小寫字元。

  > **Note**
  >
  > 如果您預計從 Amazon Athena 存取資料庫，則應提供只有英數字元和底線字元的名稱。如需詳細資訊，請參閱 Athena 名稱。

- 若要編輯資料庫的描述，請選取資料庫名稱旁的核取方塊，然後選擇 Edit (編輯)。
- 若要刪除資料庫，請選取資料庫名稱旁的核取方塊，然後選擇 Remove (移除)。
- 若要顯示資料庫中包含的資料表清單，請選擇資料庫名稱，資料庫屬性就會顯示資料庫中的所有資料表。

要變更爬蟲程式寫入的資料庫，您必須先變更爬蟲程式定義。如需詳細資訊，請參閱 使用編目器填入資料目錄 。

資料庫資源連結

AWS Glue 主控台最近已更新。目前版本的主控台不支援資料庫資源連結。

Data Catalog 也可以包含資料庫的資源連結。資料庫資源連結是本機或共用資料庫的連結。您目前只可以在 AWS Lake Formation建立資源連結。建立資料庫的資源連結後，您可以在任何使用資料庫名稱的地方使用資源連結名稱。連同您擁有或與您共用的資料庫，資料表資源連結會由 `glue:GetDatabases()` 傳回，並顯示為 AWS Glue 主控台中 Databases (資料庫) 頁面的項目。

Data Catalog 也可以包含資料表資源連結。

如需資源連結的詳細資訊，請參閱 AWS Lake Formation 開發人員指南中的建立資源連結。

建立資料表

即使執行爬行者程式是清查資料存放區中資料的建議方法，您仍可以 AWS Glue Data Catalog 手動將中繼資料表新增至。這種方法可讓您更好地控制中繼資料定義，並根據您的特定需求對其進行自訂。

您也可以使用以下方式手動將表格加入至「資料目錄」：

- 使用 AWS Glue 主控台，在 AWS Glue Data Catalog中手動建立資料表。如需詳細資訊，請參閱 在 AWS Glue 主控台上使用資料表。
- 在 AWS Glue API 中使用 `CreateTable` 操作，在 AWS Glue Data Catalog中建立資料表。如需詳細資訊，請參閱 CreateTable 動作（Python：創建表）。
- 使用 AWS CloudFormation 範本。如需詳細資訊，請參閱 適用於 AWS Glue 的 AWS CloudFormation。

當您使用主控台或 API 手動定義資料表時，需指定資料表結構描述和分類欄位的值，以表示資料來源中的資料類型和格式。如果是爬蟲程式來建立資料表，資料格式和結構描述就會由內建分類器或自訂分類器決定。如需使用 AWS Glue 主控台建立資料表的詳細資訊，請參閱 在 AWS Glue 主控台上使用資料表。

主題

- 資料表分割區
- 資料表資源連結
- 使用爬蟲程式更新手動建立的資料目錄資料表
- Data Catalog 資料表屬性
- 在 AWS Glue 主控台上使用資料表
- 使用在 AWS Glue 中的分割區索引

資料表分割區

Amazon Simple Storage Service (Amazon S3) 資料夾的 AWS Glue 資料表定義可描述已分割的資料表。例如，為了改善查詢效能，分割的資料表可能使用月份名稱為金鑰，以將每月資料分隔成不同的檔案。在 AWS Glue 中，資料表定義包含資料表的分割金鑰。AWS Glue 在評估 Amazon S3 資料夾中的資料以編目資料表時，會決定要新增個別資料表或分割資料表。

您可以在資料表上建立分割區索引來擷取分割區的子集，而不是載入資料表中的所有分割區。如需使用分割區索引的詳細資訊，請參閱使用在 AWS Glue 中的分割區索引。

要讓 AWS Glue 為 Amazon S3 資料夾建立分割資料表，以下所有條件都必須為 true：

- 檔案的結構描述類似，由 AWS Glue 決定。
- 檔案的日期格式相同。
- 檔案的壓縮格式相同。

例如，您可能擁有一個名稱為 my-app-bucket 的 Amazon S3 儲存貯體，用於存放 iOS 和 Android 應用程式的銷售資料。資料按照年、月、日分割。iOS 和 Android 銷售的資料檔案有相同的結構描述、日期格式和壓縮格式。在中 AWS Glue Data Catalog，AWS Glue爬行者程式會使用年、月和日的分割索引鍵建立一個資料表定義。

以下的 Amazon S3 my-app-bucket 清單顯示了一些分割區。= 符號用於指派分割區金鑰值。

```
my-app-bucket/Sales/year=2010/month=feb/day=1/iOS.csv
my-app-bucket/Sales/year=2010/month=feb/day=1/Android.csv
my-app-bucket/Sales/year=2010/month=feb/day=2/iOS.csv
my-app-bucket/Sales/year=2010/month=feb/day=2/Android.csv
...
my-app-bucket/Sales/year=2017/month=feb/day=4/iOS.csv
my-app-bucket/Sales/year=2017/month=feb/day=4/Android.csv
```

資料表資源連結

AWS Glue 主控台最近已更新。目前版本的主控台不支援資料表資源連結。

Data Catalog 也可以包含資料表的資源連結。資料表資源連結是本機或共用資料表的連結。您目前只可以在 AWS Lake Formation建立資源連結。建立資料表的資源連結後，您可以在任何使用資料表名稱的地方使用資源連結名稱。連同您擁有或與您共用的資料表，資料表資源連結會由 glue:GetTables() 傳回，並顯示為 AWS Glue 主控台中 Tables (資料表) 的項目。

Data Catalog 也可以包含資料庫資源連結。

如需資源連結的詳細資訊，請參閱 AWS Lake Formation 開發人員指南中的建立資源連結。

使用爬蟲程式更新手動建立的資料目錄資料表

您可能想要手動建立 AWS Glue Data Catalog 資料表，然後使用AWS Glue檢索器保持更新資料表。按排程執行的爬蟲程式可以新增新的分割區，並更新含有任何結構描述變更的資料表。這也適用於從 Apache Hive 中繼存放區遷移來的資料表。

若要執行此作業，當您定義爬蟲程式時，請指定一或多個現有的 Data Catalog 資料表，而不是指定一或多個資料存放區做為抓取來源。爬蟲程式即會抓取目錄資料表指定的資料存放區。在這種情況下，系統就不會建立任何新的資料表，而會更新您手動建立的資料表。

以下是建議您手動建立目錄資料表，並指定目錄資料表做為爬蟲程式來源的其他原因：

- 您想選擇目錄資料表名稱，而不倚賴目錄資料表命名演算法。
- 在檔案格式可能會干擾分割區偵測，但資料來源路徑中誤存該檔案的情況下，您想防止建立新的資料表。

如需詳細資訊，請參閱 步驟 2：選擇資料來源和分類器。

Data Catalog 資料表屬性

AWS CLI 中已知的資料表屬性或參數是未驗證的索引鍵和值字串。您可以在資料表中設定自己的屬性，以支援在 AWS Glue外部使用 Data Catalog。使用資料目錄的其他服務也可能會這樣做。 AWS Glue 在執行工作或爬行程式時設定一些表格屬性。除非另有說明，否則這些屬性僅供內部使用，我們不支援其以當前形式繼續存在，也不支援手動變更這些屬性時的產品行為。

如需 AWS Glue 爬行者程式設定之表格屬性的詳細資訊，請參閱the section called “爬蟲程式在 Data Catalog 資料表上設定的參數”。

在 AWS Glue 主控台上使用資料表

中的資料表 AWS Glue Data Catalog 是表示資料倉庫中資料的中繼資料定義。您可在執行爬蟲程式時建立資料表，或者也可以在 AWS Glue 主控台中手動建立資料表。在 主控台中的 TablesAWS Glue (資料表) 清單，會顯示您資料表中繼資料的值。在建立 ETL (擷取、轉換和載入) 任務時，您可以使用資料表的定義來指定來源與目標。

> **Note**
>
> AWS 管理主控台最近變更後，您可能需要修改現有的 IAM 角色才能取`SearchTables`得權限。針對新角色建立，`SearchTables` API 許可已新增為預設值。

若要開始使用，請登入 AWS Management Console 並開啟AWS Glue主控台，網址為 https://console.aws.amazon.com/glue/。選擇 Tables (資料表) 索引標籤，然後使用 Add tables (新增資料表) 按鈕來建立資料表 (藉由使用爬蟲程式或手動輸入屬性)。

在主控台上新增資料表

若要使用爬蟲程式來新增資料表，請依序選擇 Add tables (新增資料表)、Add tables using a crawler (使用爬蟲程式來新增資料表)。然後，依照 Add crawler (新增爬蟲程式) 精靈中的說明來進行操作。當

爬蟲程式執行時，資料表就會新增至 AWS Glue Data Catalog。如需詳細資訊，請參閱 使用編目器填入資料目錄 。

如果您知道在 Data Catalog 中建立 Amazon Simple Storage Service (Amazon S3) 資料表定義所需的屬性，則可以使用資料表精靈來建立。依序選擇 Add tables (新增資料表)、Add table manually (手動新增資料表)，然後依照 Add table (新增資料表) 精靈中的說明來進行操作。

透過主控台來手動新增資料表時，請注意下列事項：

- 如果您預計從 Amazon Athena 存取資料表，則應提供只有英數字元和底線字元的名稱。如需詳細資訊，請參閱 Athena 名稱。
- 您的來源資料的位置必須是 Amazon S3 路徑。
- 資料的格式必須符合精靈中所列出的其中一種格式。系統會根據選擇的格式自動填入對應的分類 SerDe、和其他表格性質。您可以使用下列格式來定義資料表：

  Avro

  Apache Avro JSON 二進位格式。

  CSV

  字元分隔值。您也可以指定分隔符號 (逗號、管線、分號、Tab 或 Ctrl-A)。

  JSON

  JavaScript 對象符號。

  XML

  可擴展標記語言的格式。指定 XML 標籤，以定義列的資料。欄會在列標籤中定義。

  Parquet

  Apache Parquet 單欄式儲存。

  ORC

  最佳化資料列單欄式 (ORC) 檔案格式。旨在高效存放 Hive 資料的格式。
- 您可以定義資料表的分區金鑰。
- 目前，透過主控台所建立的分區資料表，不能在 ETL 任務中使用。

資料表屬性

下列是一些重要的資料表屬性：

名稱

此名稱在資料表建立已決定，您無法變更。在許多的 AWS Glue 操作中，都會參照資料表的名稱。

資料庫

您的資料表所在的容器物件。此物件包含表格的組織，該組織存在於資料倉庫中， AWS Glue Data Catalog 且可能與資料倉庫中的組織不同。當您刪除資料庫，也會從 Data Catalog 刪除資料庫中內含的所有資料表。

描述

資料表的說明。您可以撰寫說明，來協助您了解資料表的內容。

資料表格式

指定創建一個標準 AWS Glue 表，或 Apache 冰山格式的表。

啟用壓縮功能

選擇啟用壓縮，將資料表中的小型 Amazon S3 物件壓縮為較大的物件。

IAM 角色

若要執行壓縮，服務會代表您擔任 IAM 角色。您可以使用下拉式選單選擇 IAM 角色。請確認角色具有啟用壓縮的必要權限。

若要進一步了解 IAM 角色的必要權限，請參閱 資料表最佳化先決條件 。

位置

指標，指向此資料表定義所代表資料存放區中資料的位置。

分類

建立資料表時所提供的分類值。這通常是在爬蟲程式執行時寫入，用來指定來源資料的格式。

上次更新

此資料表在 Data Catalog 中更新的時間和日期 (UTC)。

加入日期

此資料表新增至 Data Catalog 的時間和日期 (UTC)。

已棄用

如果 AWS Glue 發現 Data Catalog 中的資料表不再存在於其原始資料存放區，會在 Data Catalog 中將此資料表標記為已作廢。如果您執行的任務參照了已作廢的資料表，則任務可能會失敗。針對

參照已作廢資料表的任務來進行編輯，將這些做為來源和目標的資料表移除。我們建議您刪除不再需要的已作廢資料表。

連線

如果 AWS Glue 需要連線至您的資料存放區，連線的名稱會與資料表相關。

檢視與編輯資料表的詳細資訊

若要檢視現有資料表的詳細資訊，請在清單中選擇資料表名稱，然後選擇 Action, View details (動作，檢視詳細資訊)。

資料表的詳細資訊包括資料表的屬性及其結構描述。此檢視會顯示資料表的結構描述，包括欄名稱 (依資料表定義的順序排列)、資料類型和分區金鑰欄位。如果欄為複雜的類型，您可以選擇 View properties (檢視屬性) 來顯示該欄位結構的詳細資訊，如下列範例所示：

```
{
"StorageDescriptor":
    {
      "cols": {
         "FieldSchema": [
           {
             "name": "primary-1",
             "type": "CHAR",
             "comment": ""
           },
           {
             "name": "second ",
             "type": "STRING",
             "comment": ""
           }
         ]
      },
      "location": "s3://aws-logs-111122223333-us-east-1",
      "inputFormat": "",
      "outputFormat": "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
      "compressed": "false",
      "numBuckets": "0",
      "SerDeInfo": {
          "name": "",
          "serializationLib": "org.apache.hadoop.hive.serde2.OpenCSVSerde",
          "parameters": {
              "separatorChar": "|"
```

```
            }
      },
      "bucketCols": [],
      "sortCols": [],
      "parameters": {},
      "SkewedInfo": {},
      "storedAsSubDirectories": "false"
    },
    "parameters": {
       "classification": "csv"
    }
}
```

如需關於資料表的屬性 (例如 StorageDescriptor) 的詳細資訊，請參閱 StorageDescriptor 結構。

若要變更資料表的結構描述，請選擇 Edit schema (編輯結構描述) 來新增和移除欄、變更欄名稱和變更資料類型。

若要比較不同版本的表格 (包括其綱要)，請選擇 side-by-side 比較版本，查看表格之兩個結構描述版本的比較。如需詳細資訊，請參閱 比較資料表結構描述版本 。

若要顯示 Amazon S3 分區中所包含的檔案，請選擇 View partition (檢視分割區)。對於 Amazon S3 資料表，Key (金鑰) 欄會顯示分區金鑰，用來將來源資料存放區中的資料表分區。分區 (Partitioning) 是一種方法，用來根據金鑰欄位的值 (例如日期、位置或部門)，將資料表劃分為相關的部分。如需關於分區的詳細資訊，請搜尋網際網路，取得「Hive 分區」的相關資訊。

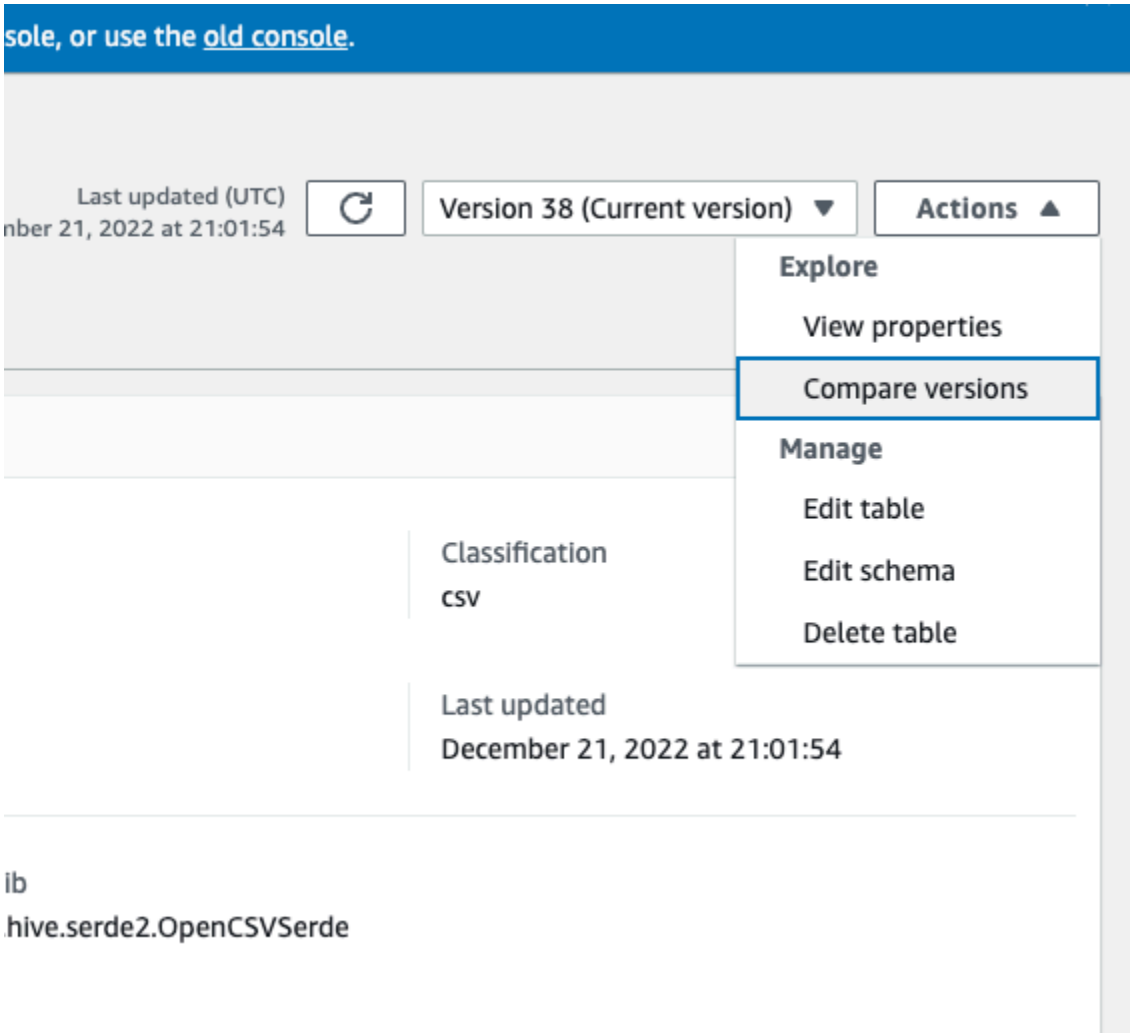
> **Note**
>
> 若要取得檢視表格詳細資料的 step-by-step 指引，請參閱主控台中的 Ex plore (Explore) 表格教學課程。

比較資料表結構描述版本

當您比較兩個版本的資料表結構定義時，您可以展開和收合巢狀資料列、比較兩個版本的結構定義，以及檢視資料表屬性 side-by-side side-by-side，來比較巢狀資料列變更。

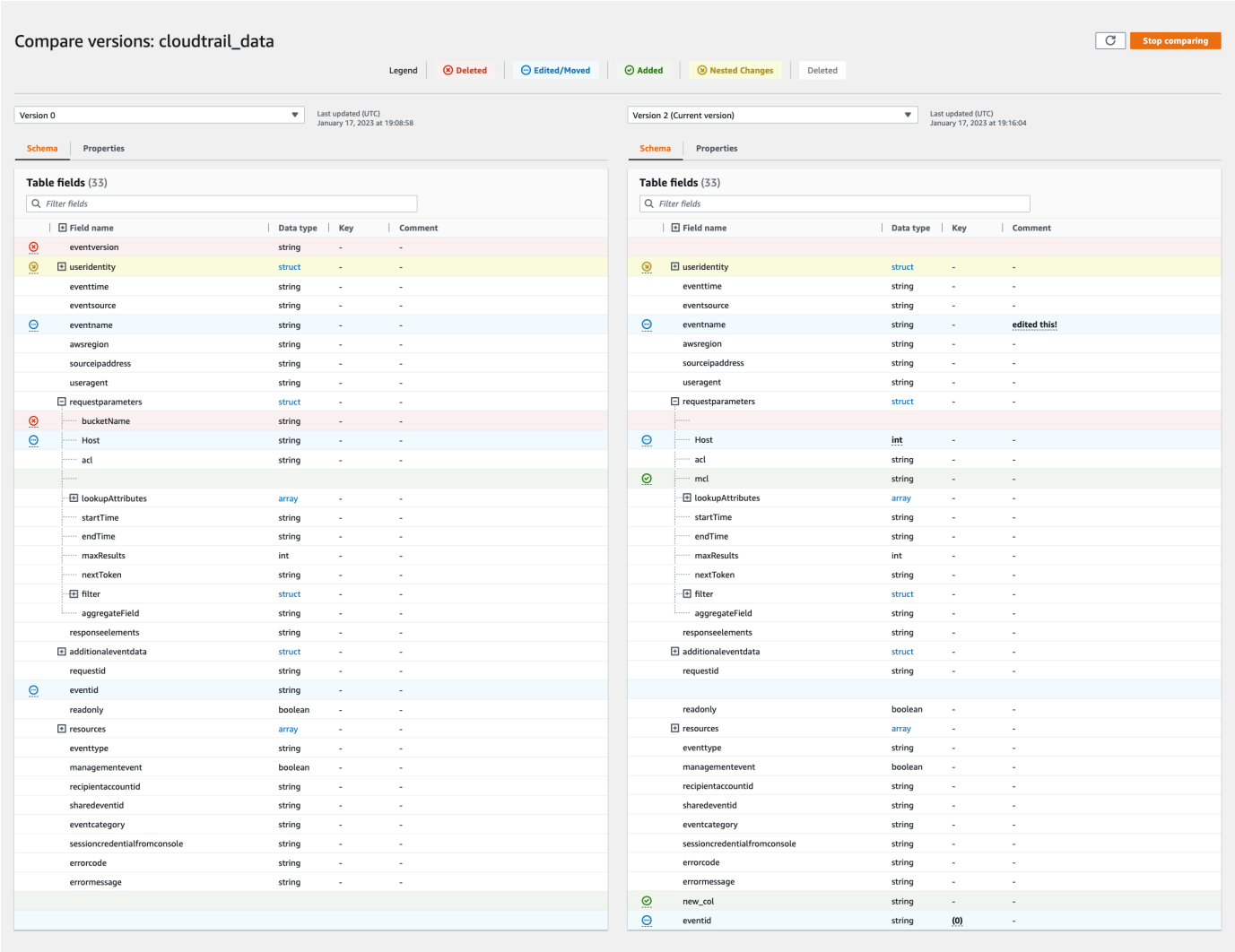
比較版本

1. 從 AWS Glue 主控台中，依序選擇資料表、動作和比較版本。



2. 選擇版本下拉式選單，以選擇要比較的版本。比較結構描述時，「結構描述」索引標籤會以橙色反白顯示。
3. 當您比較兩個版本之間的資料表時，資料表結構描述會顯示在畫面的左側和右側。這可讓您透過比較「欄名稱」、「資料類型」、「索引鍵」和「註解」欄位，以視覺方式判斷變更 side-by-side。發生變更時，彩色圖示會顯示所做變更的類型。

   - 已刪除 – 以紅色圖示顯示，指示資料欄已從舊版資料表結構描述中移除的位置。
   - 已編輯或已移動 – 以藍色圖示顯示，指示資料欄在較新版資料表結構描述中修改或移動的位置。
   - 已新增 – 以綠色圖示顯示，指示資料欄新增至較新版資料表結構描述的位置。
   - 巢狀變更 – 以黃色圖示顯示，指示巢狀資料欄包含變更的位置。選擇要展開的資料欄，並檢視已刪除、已編輯、已移動或已新增的資料欄。



4. 使用篩選欄位搜尋列，根據您在此輸入的字元顯示欄位。如果您在任一資料表版本中輸入資料欄名稱，則篩選的欄位會顯示在兩個資料表版本中，以顯示發生變更的位置。
5. 若要比較屬性，請選擇屬性索引標籤。
6. 若要停止比較版本，請選擇停止比較，以返回資料表清單。

使用在 AWS Glue 中的分割區索引

隨著時間的推移，數十萬個分割區被新增到資料表中。該 GetPartitions API 用於獲取表中的分區。API 傳回符合請求中所提供表達式的分割區。

讓我們以 sales_data 表作為由鍵國家，類別，年，月和 creationDate 進行分區的例子。如果您想獲取 2020 年 08 月 15 日之後 2020 年「書籍」類別中所有出售物品的銷售數據，則必須向「數據目錄」提出`GetPartitions`請求，其中包含「類別 = '書籍'，creationDate > '2020-08-15'」一詞。

如果資料表上沒有分割區索引，AWS Glue 會載入資料表的所有分割區，然後使用使用者在 GetPartitions 請求中提供的查詢表達式來篩選載入的分割區。查詢需要更多的時間來執行，因為沒有索引的資料表上的分割區數目會增加。使用索引，GetPartitions 查詢可嘗試擷取分割區的子集，而不是載入資料表中的所有分割區。

主題

關於分割區索引

在您建立分割區索引時，指定已存在於指定資料表上的分割區索引鍵清單。分割區索引是在資料表中定義的分割區索引鍵的子清單。分割區索引可以在資料表上定義的分割區索引鍵的任何排列上建立。對於上述 sales_data 表，可能的索引包括（國家/地區、類別、creationDate）、（國家、類別、年）、（國家/地區、類別）、（國家/地區）、（類別、國家/地區、年份、月份）等。

Data Catalog 會依建立索引時提供的順序連接分割區值。當分割區新增至資料表時，索引會一致地建立。可以為字符串（字符串，字符和 varchar），數字（整數，大整型，長，小型和小型）和日期（YYYY-MM-DD）列類型創建索引。

支援的資料類型

- 日期 — ISO 格式的日期，例如YYYY-MM-DD。例如，日期2020-08-15。格式使用連字號 (-) 來分隔年、月和日。索引日期的允許範圍從到0000-01-01跨越。9999-12-31
- String — 以單引號或雙引號括住的字串常值。
- 字元 — 固定長度字元資料，指定長度介於 1 到 255 之間，例如字元 (10)。
- 瓦爾查爾 — 可變長度字元資料，其指定長度介於 1 到 65535 之間，例如 varchar (10)。
- 數字-整型，大整數，長，小型和小型

「數字」、「字串」和「日期」資料類型的索引支援 =、>、>=、<、<= 以及運算子之間的索引。索引解決方案目前只支援 AND 邏輯運算子。使用索引進行篩選的表達式會忽略具有運算子「LIKE」、

「IN」、「OR」和「NOT」的子表達式。篩選忽略的子表達式會在套用索引篩選之後擷取的分割區上完成。

對於新增到資料表中的每個分割區，會有一個相應的索引項目建立。對於具有 'n' 個分割區的資料表，1 個分割區索引會產生 'n' 個分割區索引項目。同一個資料表上的 'm' 個分割區索引會產生 'm*n' 個分割區索引項目。每個分割區索引項目將根據目前的 AWS Glue Data Catalog 儲存體的定價政策。如需儲存物件定價的詳細資訊，請參閱 AWS Glue 定價。

建立具有分割區索引的資料表

您可以在建立資料表期間建立分割區索引。CreateTable 要求會取得 PartitionIndex 物件的清單作為輸入。指定資料表上最多可以建立 3 個分割區索引。每個分割區索引都需要一個名稱和為資料表定義的 partitionKeys 清單。在資料表上建立的索引可以使用 GetPartitionIndexes API 擷取

將分割區索引新增至現有資料表

若要將分割區索引新增至現有資料表，請使用 CreatePartitionIndex 操作。每個 CreatePartitionIndex 操作只能建立一個 PartitionIndex。新增索引不會影響資料表的可用性，因為資料表會在建立索引時繼續可用。

新增分割區的索引狀態設定為 CREATING，並啟動索引資料的建立。如果建立索引的處理程序成功，indexStatus 會更新為 ACTIVE，並且對於不成功的處理程序，索引狀態會更新為 FAILED。索引建立可能會因多種原因而失敗，您可以使用 GetPartitionIndexes 作業擷取失敗詳細資訊。可能失敗如下：

- ENCRYPTED_PARTITION_ERROR — 不支援在具有加密分割區的資料表上建立索引。
- INVALID_PARTITION_TYPE_DATA_ERROR — 當 partitionKey 值不是對應 partitionKey 資料類型的有效值時觀察到。例如：具有 'int' 資料類型的 partitionKey 有一個 'foo' 值。
- MISSING_PARTITION_VALUE_ERROR — 當 indexedKey 的 partitionValue 不存在時觀察到。當資料表沒有一致地進行分割時，可能會發生這種情況。
- UNSUPPORTED_PARTITION_CHARACTER_ERROR — 當索引分割區索引鍵的值包含字元 \u0000、\u0001 或 \u0002 時觀察到。
- INTERAL_ERROR — 建立索引時發生內部錯誤。

描述資料表上的分割區索引

若要擷取資料表上建立的分割區索引，請使用 GetPartitionIndexes 操作。回應會傳回資料表上的所有索引，以及每個索引的目前狀態 (IndexStatus)。

分割區索引的 IndexStatus 會是下列其中一項：

- CREATING – 正在建立索引，尚無法使用。
- ACTIVE— 索引已準備就緒可供使用。請求可以使用索引來執行最佳化的查詢。
- DELETING – 正在刪除索引，無法再使用。處於活動狀態的索引可以使用 DeletePartitionIndex 請求，該請求將狀態從 ACTIVE 移動到 DELETING。
- FAILED — 在現有資料表上建立索引失敗。每個資料表儲存最後 10 個失敗的索引。

在現有資料表上建立的索引的可能狀態轉換是：

- CREATING → ACTIVE → DELETING
- CREATING → FAILED

使用分割區索引的限制

建立分割區索引之後，請注意下列資料表和分割區功能變更：

建立新的分割區 (在新增索引之後)

在資料表上建立分割區索引之後，新增至資料表的所有新分割區都會驗證索引鍵的資料類型檢查。索引索引鍵的分割區值將針對資料類型格式進行驗證。如果資料類型檢查失敗，建立分割區作業將會失敗。對於 sales_data，如果針對 (category, year) 索引鍵建立索引，其中 category 為 string 類型且 year 為 int 類型，則建立具有 YEAR 值為「foo」的新分割區將失敗。

啟用索引之後，新增具有索引索引鍵值的分割區，其字元為 U+0000、U+00001 和 U+0002 將會開始失敗。

表格更新

在資料表上建立分割區索引之後，您就無法修改現有分割區索引鍵的分割區索引鍵名稱，也無法變更與索引註冊之索引鍵的類型或順序。

使用索引進行優化的 GetPartitions 調用

當您在具有索引的資料表上呼叫 GetPartitions，您可以包含表達式，如果適用， Data Catalog 將使用索引 (如果可能)。索引的第一個索引鍵應該在用於篩選索引的表達式中傳遞。篩選中的索引最佳化會盡最大努力應用。 Data Catalog 會嘗試盡可能使用索引最佳化，但如果遺失索引或不受支援的運算子，它會回退到載入所有分割區的現有實作。

於上述 sales_data 資料表，讓我們新增索引 [Country, Category, Year]。如果表達式中未傳遞 "Country"，註冊的索引將無法使用索引篩選分割區。您最多可以新增 3 個索引，以支援各種查詢模式。

讓我們舉一些範例表達式，看看索引如何在它們上運作：

表達式	如何使用索引
Country = 'US'	索引將用於篩選分割區。
Country = 'US' and Category = 'Shoes'	索引將用於篩選分割區。
Category = 'Shoes'	表達式中沒有提供 "country"，所以不會使用索引。所有分割區將被載入以傳回回應。
Country = 'US' and Category = 'Shoes' and Year > '2018'	索引將用於篩選分割區。
Country = 'US' and Category = 'Shoes' and Year > '2018' and month = 2	索引將用於擷取 country = "US" 和 category = "shoes" 和 year > 2018 的所有分割區。然後，將執行月份表達式上的篩選。
Country = 'US' AND Category = 'Shoes' OR Year > '2018'	由於 OR 運算子存在於表達式中，因此不會使用索引。
Country = 'US' AND Category = 'Shoes' AND (Year = 2017 OR Year = '2018')	索引將用於擷取 country = "US" 和 category = "shoes" 的所有分割區然後執行對年份表達式的篩選。
Country in ('US', 'UK') AND Category = 'Shoes'	由於目前不支援 IN 運算子，因此不會使用索引進行篩選。
Country = 'US' AND Category in ('Shoes', 'Books')	索引將用於擷取具有 country = "US" 的所有分割區，然後執行對「類別」表達式的篩選。
國家 =「美國」，類別在（「鞋子」，「書籍」）和（creationDate >「2023-9-01」	索引將用於獲取國家/地區 =「US」的所有分區，creationDate > '2023-9-01'，然後對類別表達式進行過濾將被執行。

與引擎整合

Redshift 頻譜，Amazon EMR 和 AWS Glue ETL Spark 能夠 DataFrames 在索引處於活動狀態後利用索引來獲取分區。AWS GlueAthena 和 AWS Glue ETL 動態框架會要求您遵循額外的步驟以利用索引進行查詢改進。

啟用分割區篩選

若要在 Athena 中啟用分割區篩選，您需要更新資料表屬性，如下所示：

1. 在AWS Glue主控台的「資料目錄」下，選擇「表格」。
2. 選擇 表格。
3. 在「動作」 下，選擇「編輯表」。
4. 在 [表格屬性] 下，新增下列項目：
   - 關鍵 — `partition_filtering.enabled`
   - 價值 — `true`
5. 選擇套用。

或者，您也可以在 Athena 中執行 ALTER 資料表集屬性查詢來設定此參數。

```
ALTER TABLE partition_index.table_with_index
SET TBLPROPERTIES ('partition_filtering.enabled' = 'true')
```

與其他 AWS 服務整合

雖然您可以使用 AWS Glue 編目程式 s 來填入 AWS Glue Data Catalog，但有數個 AWS 服務可以自動整合並為您填入目錄。下列各節提供有關可填入資料目錄之 AWS 服務所支援之特定使用案例的詳細資訊。

主題

- AWS Lake Formation
- Amazon Athena

AWS Lake Formation

AWS Lake Formation 是一項可讓您更輕鬆地在中設定安全資料湖的服務 AWS。Lake Formation 是建立在上面的 AWS Glue，和 Lake Formation 和 AWS Glue 共享相同的 AWS Glue Data Catalog。您可以向 Lake Formation 註冊 Amazon S3 資料位置，並使用 Lake Formation 主控台在 AWS Glue 資料目錄中建立資料庫和表格、定義資料存取政策，以及從中央位置稽核跨資料湖的資料存取。您可以使用 Lake Formation 精細的存取控制來管理現有的資料目錄資源和 Amazon S3 資料位置。

透過向 Lake Formation 註冊的資料，您可以在 IAM 主體、 AWS 帳戶、組織和組織單位之間安全地共用資料目錄資源。 AWS

如需有關使用 Lake Formation 建立資料目錄資源的詳細資訊，請參閱 AWS Lake Formation 開發人員指南中的建立資料目錄表格和資料庫。

Amazon Athena

Amazon Athena 使用資料目錄存放和擷取 AWS 帳戶中 Amazon S3 資料的表格中繼資料。資料表中繼資料可讓 Athena 查詢引擎知道如何尋找、讀取和處理您想要查詢的資料。

您可以直接使 AWS Glue Data Catalog 用 Athena `CREATE TABLE` 陳述式填入。您可以在「資料目錄」中手動定義和填入結構描述資料和分割區中繼資料，而不需要執行爬行者程式。

1. 在 Athena 主控台中，建立將資料表中繼資料儲存在資料目錄中的資料庫。
2. 使用`CREATE EXTERNAL TABLE`陳述式來定義資料來源的結構描述。
3. 如果您的資料已分割，請使用`PARTITIONED BY`子句來定義任何分割索引鍵。
4. 使用該`LOCATION`子句指定存放實際資料檔案的 Amazon S3 路徑。
5. 執行 `CREATE TABLE` 陳述式。

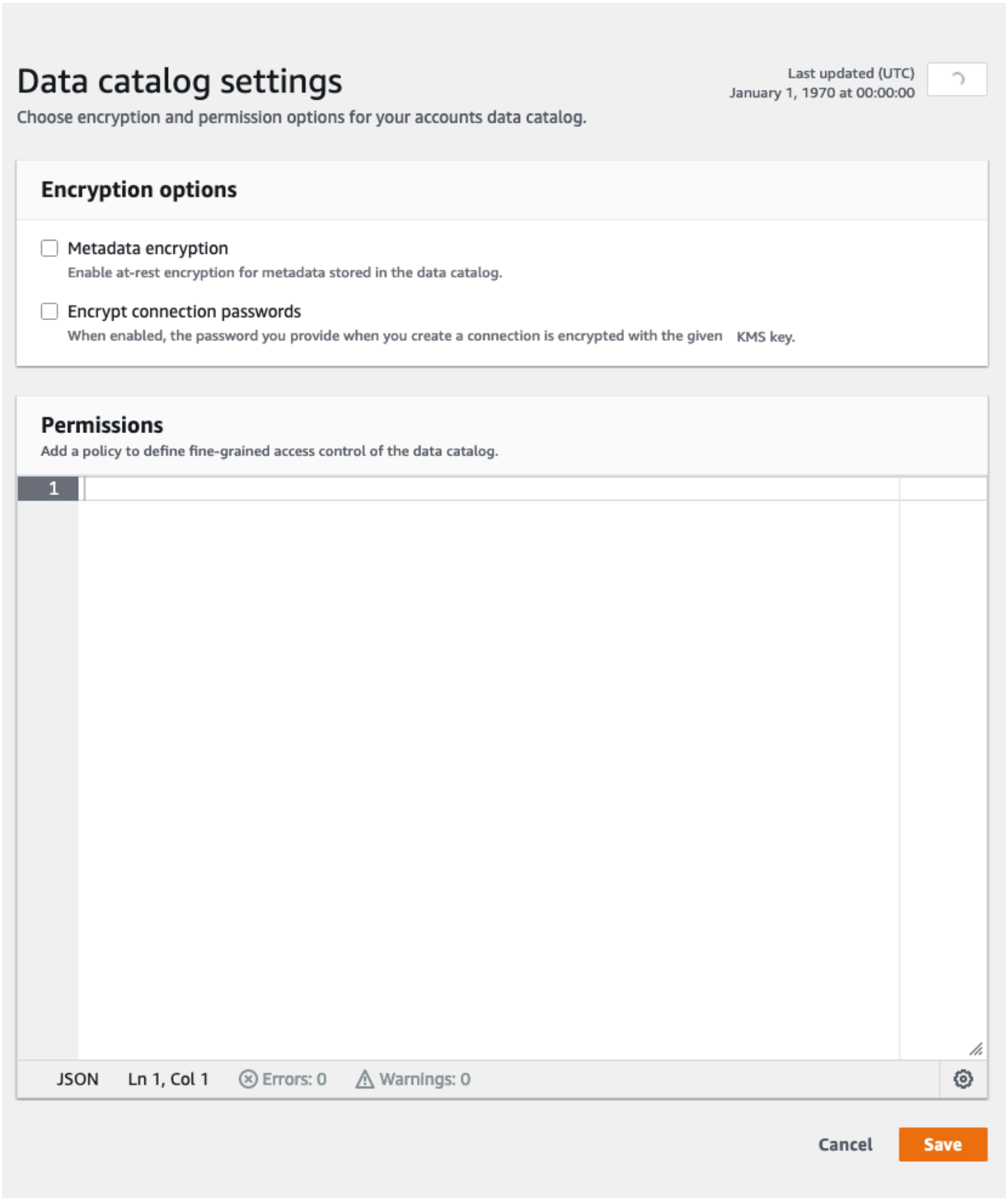
   此查詢會根據您定義的結構描述和分割區，在「資料目錄」中建立資料表中繼資料，而不會實際探索資料。

您可以在 Athena 中查詢資料表，它會使用資料目錄中的中繼資料存取和查詢 Amazon S3 中的資料檔案。

如需詳細資訊，請參閱 Amazon Athena 使用者指南中的建立資料庫和表格。

資料目錄設定

資料目錄設定包含用於設定帳戶中資料目錄的加密和權限選項的選項。


Data catalog settings
Choose encryption and permission options for your accounts data catalog.
Last updated (UTC)
January 1, 1970 at 00:00:00
Encryption options
Metadata encryption
Enable at-rest encryption for metadata stored in the data catalog.
Encrypt connection passwords
When enabled, the password you provide when you create a connection is encrypted with the given KMS key.
Permissions
Add a policy to define fine-grained access control of the data catalog.
1
JSON
Ln 1, Col 1
Errors: 0
Warnings: 0
Cancel
Save

變更資料目錄的精細存取控制

1. 請登入 AWS Management Console 並開啟AWS Glue主控台，網址為 https://console.aws.amazon.com/glue/。
2. 選擇加密選項。

   - 中繼資料加密 – 勾選此核取方塊，以加密 Data Catalog 中的中繼資料。中繼資料會使用您指定的 AWS Key Management Service (AWS KMS) 金鑰在靜態時加密。如需詳細資訊，請參閱 加密您的資料目錄。
   - 加密連線密碼 – 勾選此核取方塊，以在建立或更新連線時為 AWS Glue 連線物件加密密碼。密碼會使用您指定的 AWS KMS 金鑰加密。密碼皆以加密形式傳回。此選項套用至資料目錄中的所有 AWS Glue 連線。如果您清除此核取方塊，則先前已加密的密碼會繼續使用在建立或更新時使用的金鑰加密。如需有關 AWS Glue 連線的詳細資訊，請參閱 連線至資料。

     啟用此選項時，請選擇 AWS KMS 金鑰，或選擇輸入金鑰 ARN 並提供金鑰的 Amazon 資源名稱 (ARN)。以 `arn:aws:kms:region:account-id:key/key-id` 的形式輸入 ARN。您也可以提供 ARN 做為金鑰別名，例如 `arn:aws:kms:region:account-id:alias/alias-name`。

     > **Important**
     >
     > 若選擇此選項，則建立或更新連線的使用者或角色必須擁有指定 KMS 金鑰的 `kms:Encrypt` 許可。

     如需詳細資訊，請參閱 加密連線密碼。
3. 選擇 Settings (設定)，然後在 Permissions (許可) 編輯器中新增政策陳述式，以變更您帳戶之資料目錄的精細存取控制。一次只有一個政策可以連接至資料目錄。您可以將 JSON 資源政策貼入此控制項。如需詳細資訊，請參閱 Glue 中 AWS 基於資源的政策。
4. 選擇 Save (儲存)，以使用您進行的任何變更來更新資料目錄。

您也可以使用 AWS Glue API 操作來放置、取得和刪除資源政策。如需更多詳細資訊，請參閱 中的安全性 API AWS Glue。

填入及管理交易資料表

阿帕奇冰山、阿帕奇胡迪和 Linux 基金會三角洲湖是專為處理 Apache Spark 中的大規模資料分析和資料湖工作負載而設計的開放原始碼資料表格式。

您可以 AWS Glue Data Catalog 使用下列方法在中填入「冰山」、「胡迪湖」和「三角洲湖」表格：

- AWS Glue 編目程式; — AWS Glue 編目程式 s 可以在數據目錄中自動發現和填充冰山，Hudi 和 Delta 湖表中繼資料。如需詳細資訊，請參閱 使用編目器填入資料目錄 。
- AWS Glue ETL 工作 — 您可以建立 ETL 任務，將資料寫入冰山、Hudi 和三角洲湖資料表，並在資料目錄中填入其中繼資料。如需詳細資訊，請參閱搭配 AWS Glue ETL 工作使用資料湖架構。
- AWS Glue 主控台、 AWS Lake Formation 主控台 AWS CLI 或 API — 您可以使用 AWS Glue 主控台、Lake Formation 主控台或 API 在「資料目錄」中建立和管理 Iceberg 表格定義。

主題

- 創建阿帕奇冰山表
- 最佳化處理 Iceberg 資料表

創建阿帕奇冰山表

您可以創建使用 Apache 鑲木地板數據格式 AWS Glue Data Catalog 與駐留在 Amazon S3 的數據格式阿帕奇冰山表。「資料目錄」中的表格是表示資料倉庫中資料的中繼資料定義。默認情況下， AWS Glue 創建冰山 v2 表。有關 v1 和 v2 資料表之間的區別，請參閱 Apache Iceberg 文件中的格式版本變更。

Apache Iceberg 是開放式的資料表格式，專用於非常大型的分析資料集。Iceberg 允許輕鬆更改模式，也稱為模式演進，這意味著用戶可以在不中斷基礎數據的情況下從數據表中添加，重命名或刪除列。Iceberg 還為數據版本控制提供支持，允許用戶跟踪數據超時的更改。這會啟用時間行程功能，讓使用者能夠存取和查詢資料的歷史版本，並分析更新和刪除之間的資料變更。

您可以使用 AWS Glue 或 Lake Formation 控制台或 AWS Glue API 中的`CreateTable`操作在「數據目錄」中創建冰山表。如需詳細資訊，請參閱CreateTable 動作 (Python: 建立表格)。

在資料目錄中建立 Iceberg 表格時，必須在 Amazon S3 中指定表格式和中繼資料檔案路徑，才能執行讀取和寫入。

當您向 Amazon S3 資料位置註冊時，您可以使用 Lake Formation 使用精細的存取控制許可來保護您的 Iceberg 資料表。 AWS Lake Formation對於 Amazon S3 中的來源資料和未向 Lake Formation 註

冊的中繼資料，存取權由 Amazon S3 的 IAM 許可政策和 AWS Glue 動作決定。如需詳細資訊，請參閱管理權限。

> **Note**
>
> 資料目錄不支援建立磁碟分割和新增 Iceberg 資料表屬性。

必要條件

若要在資料目錄中建立 Iceberg 表格，並設定 Lake Formation 資料存取權限，您需要完成下列需求：

1. 創建冰山表所需的權限，而無需在 Lake Formation 註冊的數據。

   除了在「資料目錄」中建立資料表所需的權限外，資料表建立者還需要下列權限：

   - `s3:PutObject`在資源陣列上:aw:s3::: {儲存 bucketName}
   - `s3:GetObject`在資源陣列上:aw:s3::: {儲存 bucketName}
   - `s3:DeleteObject`在資源陣列上:aw:s3::: {儲存 bucketName}
2. 使用 Lake Formation 註冊的數據創建冰山表所需的權限：

   若要使用 Lake Formation 管理和保護資料湖中的資料，請使用 Lake Formation 註冊具有表格資料的 Amazon S3 位置。這樣一來，Lake Formation 可以將憑據 AWS 分析服務（例如 Athena，Redshift 頻譜和 Amazon EMR）出售以訪問數據。如需註冊 Amazon S3 位置的詳細資訊，請參閱將 Amazon S3 位置新增至資料湖。

   讀取和寫入在 Lake Formation 註冊的基礎資料的主體需要下列權限：

   - `lakeformation:GetDataAccess`
   - `DATA_LOCATION_ACCESS`

     對某個位置具有資料位置權限的主參與者也具有所有子位置的位置權限。

     有關資料位置權限的詳細資訊，請參閱基礎資料存取控制 ulink。

若要啟用壓縮，服務必須採用具有更新資料目錄中資料表之權限的 IAM 角色。如需詳細資訊，請參閱資料表最佳化先決條件

創建一個冰山表

您可以使用 AWS Glue 或 Lake Formation 控制台或 AWS Command Line Interface 如本頁所述創建冰山 v1 和 v2 表。您也可以使用建立冰山表格。 AWS Glue 編目程式如需詳細資訊，請參閱 AWS Glue 開發人員指南中的資料目錄和檢索器。

若要建立冰山表

Console

1. 請登入 AWS Management Console 並開啟 AWS Glue 主控台，網址為 https://console.aws.amazon.com/glue/。
2. 在「資料目錄」下，選擇「表格」，然後使用「建立表格」按鈕指定下列屬性：
   - 表格名稱 — 輸入表格的名稱。如果您使用 Athena 存取表格，請參閱 Amazon Athena 使用者指南中的這些命名提示。
   - 資料庫 — 選擇現有資料庫或建立新資料庫。
   - 「描述」— 表格的描述。您可以撰寫說明，來協助您了解資料表的內容。
   - 表格格式 — 對於表格格式，請選擇 Apache 冰山。
   - 啟用壓縮 — 選擇「啟用壓縮」，將表格中的小型 Amazon S3 物件壓縮為更大的物件。
   - IAM 角色 — 若要執行壓縮，服務會代表您擔任 IAM 角色。您可以使用下拉式選單選擇 IAM 角色。請確認角色具有啟用壓縮的必要權限。

     若要進一步瞭解所需權限，請參閱 資料表最佳化先決條件 。
   - 位置 — 指定 Amazon S3 中存放中繼資料表的資料夾路徑。Iceberg 需要資料目錄中的中繼資料檔案和位置，才能執行讀取和寫入。
   - 結構描述 — 選擇新增資料欄以新增資料欄的資料欄和資料類型。您可以選擇建立空白資料表並稍後更新結構定義。數據目錄支持蜂巢數據類型。如需詳細資訊，請參閱 Hive 資料類型。

     Iceberg 允許您在創建表後進化模式和分區。您可以使用 Athena 查詢來更新資料表結構描述和 Spark 查詢以更新分割區。

AWS CLI

```
aws glue create-table \
    --database-name iceberg-db \
    --region us-west-2 \
    --open-table-format-input '{
```

```
    "IcebergInput": {
         "MetadataOperation": "CREATE",
         "Version": "2"
       }
    }' \
  --table-input '{"Name":"test-iceberg-input-demo",
          "TableType": "EXTERNAL_TABLE",
          "StorageDescriptor":{
             "Columns":[
                 {"Name":"col1", "Type":"int"},
                 {"Name":"col2", "Type":"int"},
                 {"Name":"col3", "Type":"string"}
              ],
             "Location":"s3://DOC_EXAMPLE_BUCKET_ICEBERG/"
          }
      }'
```

最佳化處理 Iceberg 資料表

使用開放資料表格式 (例如，Apache Iceberg) 的 Amazon S3 資料湖會以 Amazon S3 物件形式儲存資料。如果資料湖資料表中具有數千個小型 Amazon S3 物件，會增加 Iceberg 資料表中的中繼資料額外負荷，並影響讀取效能。為了透過 Amazon EMR Amazon Athena 和 AWS Glue ETL 任務等 AWS 分析服務提供更好的讀取效能，請在資料目錄中為 Iceberg 表格 AWS Glue Data Catalog 提供受管壓縮 (將小型 Amazon S3 物件壓縮為較大物件的程序)。您可以使用 Lake Formation 主控 AWS Glue 台 AWS CLI、主控台或 AWS API 來啟用或停用資料目錄中個別 Iceberg 資料表的壓縮功能。

資料表最佳化工具會持續監視資料表分割區，並在檔案數量和檔案大小超過閾值時啟動壓縮程序。一個冰山表有資格壓縮，如果在寫入指定的文件大小。 target-file-size-bytes 屬性在 128MB 至 512 MB 範圍內。在「資料目錄」中，如果資料表有五個以上的檔案 (每個檔案都小於寫入的 75%)，則壓縮程序就會開始。 target-file-size-bytes 財產。

例如，您在寫入時將檔案大小臨界值設定為 512MB 的資料表。 target-file-size-bytes 屬性（在 128MB 到 512MB 的規定範圍內），並且該表包含 10 個文件。如果 10 個文件中的 6 個小於 384MB（.75* 512），則數據目錄觸發壓縮。

Data Catalog 會在不干擾並行查詢的情況下執行壓縮程序。Data Catalog 僅支援 Parquet 格式資料表的資料壓縮。

如需支援的資料類型、壓縮格式和限制，請參閱受管理資料壓縮的支援格式和限制 。

主題

資料表最佳化先決條件

資料表最佳化工具會假設您在啟用資料表壓縮時指定的 AWS Identity and Access Management (IAM) 角色許可。IAM 角色必須具有讀取資料和更新 Data Catalog 中繼資料的權限。您可以建立 IAM 角色，並連接下列內嵌政策：

- 新增下列內嵌政策，針對未向 Lake Formation 註冊的資料的位置授予 Amazon S3 讀取/寫入權限。此原則也包含更新資料目錄中的表格，以及允許在記錄檔中 Amazon CloudWatch 新 AWS Glue 增記錄檔和發佈指標的權限。針對 Amazon S3 中未向 Lake Formation 註冊的來源資料，存取權將由 Amazon S3 和 AWS Glue 動作的 IAM 權限政策決定。

  在下列內嵌政策中，請將 `bucket-name` 取代為 Amazon S3 儲存貯體名稱，將 `aws-account-id` 和 `region` 取代為有效的 AWS 帳號和 Data Catalog 的區域，將 `database_name` 取代為資料庫的名稱，以及將 `table_name` 取代為資料表的名稱。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::<bucket-name>/*"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::<bucket-name>"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:UpdateTable",
                "glue:GetTable"
            ],
            "Resource": [
                "arn:aws:glue:<region>:<aws-account-id>:table/<database-name>/<table-
name>",
                "arn:aws:glue:<region>:<aws-account-id>:database/<database-name>",
                "arn:aws:glue:<region>:<aws-account-id>:catalog"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:<region>:<aws-account-id>:log-group:/aws-glue/
iceberg-compaction/logs:*"
        }
    ]
}
```

- 使用下列政策針對向 Lake Formation 註冊的資料啟用壓縮功能。

  如果壓縮角色沒有授與資料表的IAM_ALLOWED_PRINCIPALS群組權限，則該角色需要 Lake Formation ALTER INSERT 和資料表的DELETE權限。DESCRIBE

  如需透過 Lake Formation 註冊 Amazon S3 儲存貯體的詳細資訊，請參閱將 Amazon S3 位置新增至資料湖。

```
{
 "Version": "2012-10-17",
 "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "lakeformation:GetDataAccess"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
        "glue:UpdateTable",
        "glue:GetTable"
        ],

        "Resource": [
            "arn:aws:glue:<region>:<aws-account-
id>:table/<databaseName>/<tableName>",
            "arn:aws:glue:<region>:<aws-account-id>:database/<database-name>",
            "arn:aws:glue:<region>:<aws-account-id>:catalog"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
        ],
        "Resource": "arn:aws:logs:<region>:<aws-account-id>:log-group:/aws-glue/
iceberg-compaction/logs:*"
        }
    ]
 }
```

- (選用) 若要使用伺服器端加密加密之 Amazon S3 儲存貯體中的資料壓縮 Iceberg 資料表，則壓縮角色需要解密 Amazon S3 物件的權限，並產生新的資料金鑰以將物件寫入加密的儲存貯體。將下列原則新增至所需的 AWS KMS 金鑰。我們僅支援儲存貯體層級加密。

```
{
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::<aws-account-id>:role/<compaction-role-name>"
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*"
}
```

- (選用) 針對向 Lake Formation 註冊的資料位置，用於註冊該位置的角色需要解密 Amazon S3 物件並產生新的資料金鑰以將物件寫入加密儲存貯體的權限。如需詳細資訊，請參閱註冊加密的 Amazon S3 位置。
- (選擇性) 如果 AWS KMS 金鑰儲存在不同的 AWS 帳戶中，您必須包含壓縮角色的下列權限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": ["arn:aws:kms:<REGION>:<KEY_OWNER_ACCOUNT_ID>:key/<KEY_ID>" ]
        }
    ]
}
```

- 您用來執行壓縮程序的角色必須具有該角色的 `iam:PassRole` 權限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
```

```
            ],
            "Resource": [
                "arn:aws:iam::<account-id>:role/<compaction-role-name>"
            ]
        }
    ]
}
```

- 將下列信任政策新增至 AWS Glue 服務角色，以承擔 IAM 角色以執行壓縮程序。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
 }
```

啟用壓縮功能

您可以使用 Lake Formation 主控 AWS Glue 台 AWS CLI、主控台或 AWS API，為資料目錄中的 Apache 冰山資料表啟用壓縮功能。針對新的資料表，您可以選擇 Apache Iceberg 作為資料表格式，並在您建立資料表時啟用壓縮功能。新資料表依預設會停用壓縮功能。

Console

啟用壓縮功能

1. 在 https://console.aws.amazon.com/glue/ 開啟 AWS Glue 主控台，然後以資料湖管理員、表格建立者或已獲得資料表`glue:UpdateTable`和`lakeformation:GetDataAccess`權限的使用者身分登入。
2. 在導覽面板的 Data Catalog 下方，選擇資料表。
3. 在資料表頁面中，選擇您想要啟用壓縮功能之開放資料表格式的資料表，然後在動作功能表下選擇啟用壓縮。

4. 您也可以透過選取資料表並開啟資料表詳細資料頁面，來啟用壓縮功能。選擇頁面下半區段的資料表最佳化索引標籤，然後選擇啟用壓縮。
5. 接下來，從下拉式選單中選取現有的 IAM 角色，其權限會在 資料表最佳化先決條件 區段中顯示。

   當您選擇建立新 IAM 角色選項時，服務會建立具有執行壓縮程序之必要權限的自訂角色。

   請依照以下步驟更新現有 IAM 角色：

   a. 若要更新 IAM 角色的權限政策，請在 IAM 主控台中，前往用於執行壓縮程序的 IAM 角色。
   b. 在新增權限區段中，選擇建立政策。在新開啟的瀏覽器視窗中，建立要搭配您角色使用的新政策。
   c. 在建立政策頁面上，選擇 JSON 標籤。將先決條件中顯示的 JSON 代碼複製到策略編輯器字段中。

AWS CLI

下列範例顯示如何啟用壓縮功能。將帳號 ID 取代為有效的 AWS 帳號 ID。將資料庫名稱和資料表名稱取代為實際的 Iceberg 資料表名稱和資料庫名稱。將其取`roleArn`代為 IAM 角色的 AWS 資源名稱 (ARN)，以及具有執行壓縮所需權限的 IAM 角色名稱。

```
aws glue create-table-optimizer \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
  --table-optimizer-configuration
 '{"roleArn":"arn:aws:iam::123456789012:role/compaction_role", "enabled":'true'}' \
  --type compaction
```

AWS API

呼叫 `CreateTableOptimizer` 操作以啟用資料表的壓縮功能。

啟用壓縮功能後，資料表最佳化索引標籤會顯示下列壓縮詳細資料 (大約 15-20 分鐘後)：

開始時間

在 Lake Formation 內開始壓實過程的時間。該值為以 UTC 時間為單位的時間戳記。

結束時間

壓實程序在資料目錄中結束的時間。該值為以 UTC 時間為單位的時間戳記。

Status

壓實執行的狀態。值會是 success 或 fail。

壓縮的檔案

壓縮的檔案總數。

字節壓縮

壓縮的字節總數。

停用壓縮功能

您可以禁用自動壓縮使用 AWS Glue 控制台或特定的 Apache 冰山表。 AWS CLI

Console

1. 選擇 Data Catalog，然後選擇資料表。從資料表清單中，選擇您想要停用壓縮功能之開放資料表格式的資料表。
2. 您可以選擇 Iceberg 資料表，然後選擇動作下方的停用壓縮。

   您也可以透過選擇資料表詳細資料頁面下半區段的停用壓縮，來停用資料表的壓縮功能。
3. 在確認訊息中，選擇停用壓縮。您可以在稍後重新啟用壓縮功能。

   當您確認後，壓縮功能會停用，而資料表的壓縮狀態會變回 `Off`。

AWS CLI

在下列範例中，將帳戶 ID 取代為有效的 AWS 帳戶 ID。將資料庫名稱和資料表名稱取代為實際的 Iceberg 資料表名稱和資料庫名稱。將其取`roleArn`代為 IAM 角色的 AWS 資源名稱 (ARN)，以及具有執行壓縮所需權限的 IAM 角色的實際名稱。

```
aws glue update-table-optimizer \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
  --table-optimizer-configuration
 '{"roleArn":"arn:aws:iam::123456789012:role/compaction_role", "enabled":'false'}'\
```

```
  --type compaction
```

AWS API

調用 UpdateTableOptimizer 操作以禁用特定表的壓縮。

檢視壓縮詳細資料

您可以在 Lake Formation 控制台中查看 Apache 冰山的壓縮狀態 AWS CLI，或使用 AWS API 操作。

Console

若要檢視冰山表的壓縮狀態 (主控台)

- 您可以選擇「資料目錄」下的「表格」，在 Lake Formation 主控台上檢視冰山表的壓縮狀態。壓縮狀態欄位會顯示壓縮執行的狀態。您可以使用資料表偏好設定，來顯示資料表格式和壓縮狀態。
- 若要檢視特定表格的壓縮執行歷程記錄，請選擇 [下的表格] AWS Glue Data Catalog，然後選擇表格來檢視表格詳細資訊。資料表最佳化索引標籤會顯示資料表的壓縮歷史記錄。

AWS CLI

您可以使用檢視壓實詳細資訊 AWS CLI。

在下列範例中，請以有效 AWS 的帳戶 ID、資料庫名稱和資料表名稱取代為實際的 Iceberg 資料表名稱。

- 取得資料表的上次壓縮執行詳細資料

```
aws get-table-optimizer \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
  --type compaction
```

- 使用下列範例擷取特定資料表的最佳化工具歷史記錄。

```
aws list-table-optimizer-runs \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
```

```
  --type compaction
```

- 下列範例顯示如何擷取多個最佳化工具的壓縮執行和組態詳細資料。您最多可以指定 20 個最佳化工具。

```
aws glue batch-get-table-optimizer \
--entries '[{"catalogId":"123456789012", "databaseName":"iceberg_db",
 "tableName":"iceberg_table", "type":"compaction"}]'
```

AWS API

- 使用 GetTableOptimizer 操作來擷取最佳化工具的上次執行詳細資料。
- 使用 ListTableOptimizerRuns 操作來擷取特定資料表中特定最佳化工具的歷史記錄。您可以在單一 API 呼叫中指定 20 個最佳化工具。
- 使用 BatchGetTableOptimizer 操作來擷取帳戶中多個最佳化工具的組態詳細資料。此操作不支援跨帳戶呼叫。

檢視 Amazon CloudWatch 量度

成功執行壓縮之後，服務會建立壓縮工作效能的 Amazon CloudWatch 指標。您可以轉到CloudWatch 指標並選擇度量標準，所有度量標準。您可以依特定命名空間 (例如 AWS Glue)、表格名稱或資料庫名稱來篩選測量結果。

如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的檢視可用指標。

- 壓縮的位元組數
- 壓縮的檔案數
- 配置給工作的 DPU 數目
- 任務持續時間 (小時)

刪除最佳化工具

您可以使用 AWS CLI 或 AWS API 操作刪除表的優化器和關聯的元數據。

執行下列 AWS CLI 命令以刪除表格的壓縮歷程記錄。

```
aws glue delete-table-optimizer \
```

```
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
  --type compaction
```

使用 DeleteTableOptimizer 操作刪除資料表的最佳化工具。

受管理資料壓縮的支援格式和限制

若要透過 Amazon Athena、Amazon EMR 和 AWS Glue ETL 任務等 AWS 分析服務提供更佳的讀取效能，請為資料型錄中的冰山資料表 AWS Glue Data Catalog 提供受管壓縮 (將小型 Amazon S3 物件壓縮為較大物件的程序)。

數據壓縮支持多種數據類型和壓縮格式，用於讀取和寫入數據，包括從加密表中讀取數據。

資料壓縮支援：

- 檔案類型 — 實木地板
- 資料類型 — 布林值、整數、長、浮點數、雙精度、字串、十進位、日期、時間、時間戳記、字串、UUID、二進位
- 壓縮-壓縮，壓縮，活潑，未壓縮
- 加密 — 資料壓縮僅支援預設的 Amazon S3 加密 (SSE-S3) 和伺服器端 KMS 加密 (SSE-KMS)。
- BinPack 壓縮
- 結構描述演進
- 具有目標檔案大小的表格 (寫入。 target-file-size-bytes 在冰山配置中的屬性) 包含範圍內 128MB 至 512 MB。
- 區域
  - 亞太區域 (東京)
  - 亞太區域 (首爾)
  - 亞太區域 (孟買)
  - 亞太區域 (新加坡)
  - 歐洲 (愛爾蘭)
  - 歐洲 (倫敦)
  - 歐洲 (法蘭克福)
  - 美國東部 (維吉尼亞北部)
  - 美國東部 (俄亥俄)

  - 美國西部 (加利佛尼亞北部)
  - 南美洲 (聖保羅)
- 當儲存基礎資料的 Amazon S3 儲存貯體位於其他帳戶中時，您可以從 Data Catalog 所在的帳戶中執行壓縮程序。若要執行此程序，壓縮角色需要 Amazon S3 儲存貯體的存取權。

資料壓縮目前不支援：

- 檔案類型 — 阿夫羅、ORC
- 數據類型-固定
- 壓縮 brotli，LZ4
- 在分區規格發展時壓縮文件。
- 一般排序或堆疊順序排序
- 合併或刪除檔案 — 壓縮程序會略過具有刪除與檔案相關聯之檔案的資料檔案。
- 跨帳戶資料表壓縮 — 您無法在跨帳戶資料表上執行壓縮。
- 跨區域資料表壓縮 — 您無法在跨區域資料表上執行壓縮。
- 在資源連結上啟用壓縮功能
- Amazon S3 儲存貯體的 VPC 端點
- DynamoDB 鎖定管理員 — 使用資料壓縮時，不應將其他資料載入作業當做組織使用`lock-impl`。DynamoDbLockManager。

管理資料目錄

這 AWS Glue Data Catalog 是一個中央中繼資料儲存庫，可存放 Amazon S3 資料集的結構化和操作中繼資料。有效管理資料型錄對於維護資料品質、效能、安全性和控管至關重要。

透過瞭解並套用這些資料目錄管理實務，您可以確保中繼資料隨著資料環境的發展保持正確、高效能、安全且受到妥善管理。

本節涵蓋資料目錄管理的下列層面：

- 更新資料表結構定義和資料分割隨著資料的發展，您可能需要更新資料目錄中定義的資料表結構定義或資料分割結構。如需如何使用 AWS Glue ETL 以程式設計方式進行這些更新的詳細資訊，請參閱使用 AWS Glue ETL 工作更新結構描述，並在「資料目錄」中新增分割區。
- 管理資料欄統計資料：精確的資料欄統計資料有助於最佳化查詢計畫並 如需如何產生、更新和管理資料行統計資料的詳細資訊，請參閱使用資料欄統計資料來最佳化。

- 加密資料目錄若要保護敏感的中繼資料，您可以使用 AWS Key Management Service (AWS KMS) 加密資料目錄。本節說明如何啟用和管理資料目錄的加密。
- 使用 AWS Lake Formation Lake Formation 保護資料目錄可提供資料湖安全性和存取控制的全方位方法。您可以使用 Lake Formation 來保護和管理對資料目錄和基礎資料的存取。

主題

使用 AWS Glue ETL 工作更新結構描述，並在「資料目錄」中新增分割區

您的擷取、轉換和載入 (ETL) 任務可能會在目標資料存放區中建立新的資料表分割區。您的資料集結構描述可能會隨著時間演進，而與 AWS Glue 資料目錄結構描述有所不同。AWS GlueETL 任務目前提供數個功能，您可以在 ETL 指令碼中使用這些功能來更新資料目錄中的 ETL 結構描述和分割區。這些功能可讓您在資料目錄中查看 ETL 運作的結果，無需重新執行爬蟲程式。

新分割區

如果您想要檢視中的新分割區 AWS Glue Data Catalog，您可以執行下列其中一個動作：

- 任務完成後，重新執行爬蟲程式，並在爬蟲程式完成後在主控台上檢視新的分割區。
- 任務完成後，立即在主控台上檢視新的分割區，而不重新執行爬蟲程式。您可以在 ETL 指令碼中加入幾行程式碼，以啟用此功能，如下列範例所示。程式碼會使用 `enableUpdateCatalog` 引數來指出在資料目錄建立新分割區時，要在任務執行期間更新。

方法 1

以選項引數傳遞 `enableUpdateCatalog` 和 `partitionKeys`。

Python

```
additionalOptions = {"enableUpdateCatalog": True}
additionalOptions["partitionKeys"] = ["region", "year", "month", "day"]
```

```
sink = glueContext.write_dynamic_frame_from_catalog(frame=last_transform,
 database=<target_db_name>,

 table_name=<target_table_name>, transformation_ctx="write_sink",

 additional_options=additionalOptions)
```

Scala

```
val options = JsonOptions(Map(
    "path" -> <S3_output_path>,
    "partitionKeys" -> Seq("region", "year", "month", "day"),
    "enableUpdateCatalog" -> true))
val sink = glueContext.getCatalogSink(
    database = <target_db_name>,
    tableName = <target_table_name>,
    additionalOptions = options)sink.writeDynamicFrame(df)
```

方法 2

以 getSink() 傳遞 enableUpdateCatalog 和 partitionKeys，並呼叫 DataSink 物件上的 setCatalogInfo()。

Python

```
sink = glueContext.getSink(
    connection_type="s3",
    path="<S3_output_path>",
    enableUpdateCatalog=True,
    partitionKeys=["region", "year", "month", "day"])
sink.setFormat("json")
sink.setCatalogInfo(catalogDatabase=<target_db_name>,
 catalogTableName=<target_table_name>)
sink.writeFrame(last_transform)
```

Scala

```
val options = JsonOptions(
   Map("path" -> <S3_output_path>,
       "partitionKeys" -> Seq("region", "year", "month", "day"),
       "enableUpdateCatalog" -> true))
val sink = glueContext.getSink("s3", options).withFormat("json")
sink.setCatalogInfo(<target_db_name>, <target_table_name>)
```

```
sink.writeDynamicFrame(df)
```

現在，您可以建立新的目錄資料表、使用修改的結構描述更新現有資料表，並使用 AWS Glue ETL 任務在資料目錄新增新的資料表分割區，而無需重新執行爬蟲程式。

更新資料表結構描述

如果您想覆寫資料目錄資料表的結構描述，可以執行以下任一作業：

- 任務完成後，重新執行爬蟲程式，並確定您的爬蟲程式已設定為更新資料表定義。當爬蟲程式完成時，檢視主控台上的新分割區以及任何結構描述更新。如需詳細資訊，請參閱使用 API 設定爬蟲程式。
- 任務完成後，立即在主控台上檢視修改的結構描述，無需重新執行爬蟲程式。您可以在 ETL 指令碼中加入幾行程式碼，以啟用此功能，如下列範例所示。程式碼使用設為 true 的 `enableUpdateCatalog`，並同時將 `updateBehavior` 設為 `UPDATE_IN_DATABASE`，表示在任務執行期間，在資料目錄中覆寫結構描述並新增新的分割區。

Python

```
additionalOptions = {
    "enableUpdateCatalog": True,
    "updateBehavior": "UPDATE_IN_DATABASE"}
additionalOptions["partitionKeys"] = ["partition_key0", "partition_key1"]

sink = glueContext.write_dynamic_frame_from_catalog(frame=last_transform,
 database=<dst_db_name>,
    table_name=<dst_tbl_name>, transformation_ctx="write_sink",
    additional_options=additionalOptions)
job.commit()
```

Scala

```
val options = JsonOptions(Map(
    "path" -> outputPath,
    "partitionKeys" -> Seq("partition_0", "partition_1"),
    "enableUpdateCatalog" -> true))
val sink = glueContext.getCatalogSink(database = nameSpace, tableName = tableName,
 additionalOptions = options)
sink.writeDynamicFrame(df)
```

如果您想防止資料表結構描述遭到覆寫，但仍想要新增新的分割區，您也可以將 updateBehavior 值設為 LOG。updateBehavior 的預設值為 UPDATE_IN_DATABASE，因此，如果您未明確定該值，則會覆寫資料表結構描述。

如果 enableUpdateCatalog 未設為 true，無論針對 updateBehavior 選取哪個選項，ETL 任務都不會在資料目錄中更新資料表。

建立新的資料表

您也可以使用相同選項，在資料目錄中建立新資料表。您可以使用 setCatalogInfo 指定資料庫和新的資料表名稱。

Python

```
sink = glueContext.getSink(connection_type="s3", path="s3://path/to/data",
    enableUpdateCatalog=True, updateBehavior="UPDATE_IN_DATABASE",
    partitionKeys=["partition_key0", "partition_key1"])
sink.setFormat("<format>")
sink.setCatalogInfo(catalogDatabase=<dst_db_name>, catalogTableName=<dst_tbl_name>)
sink.writeFrame(last_transform)
```

Scala

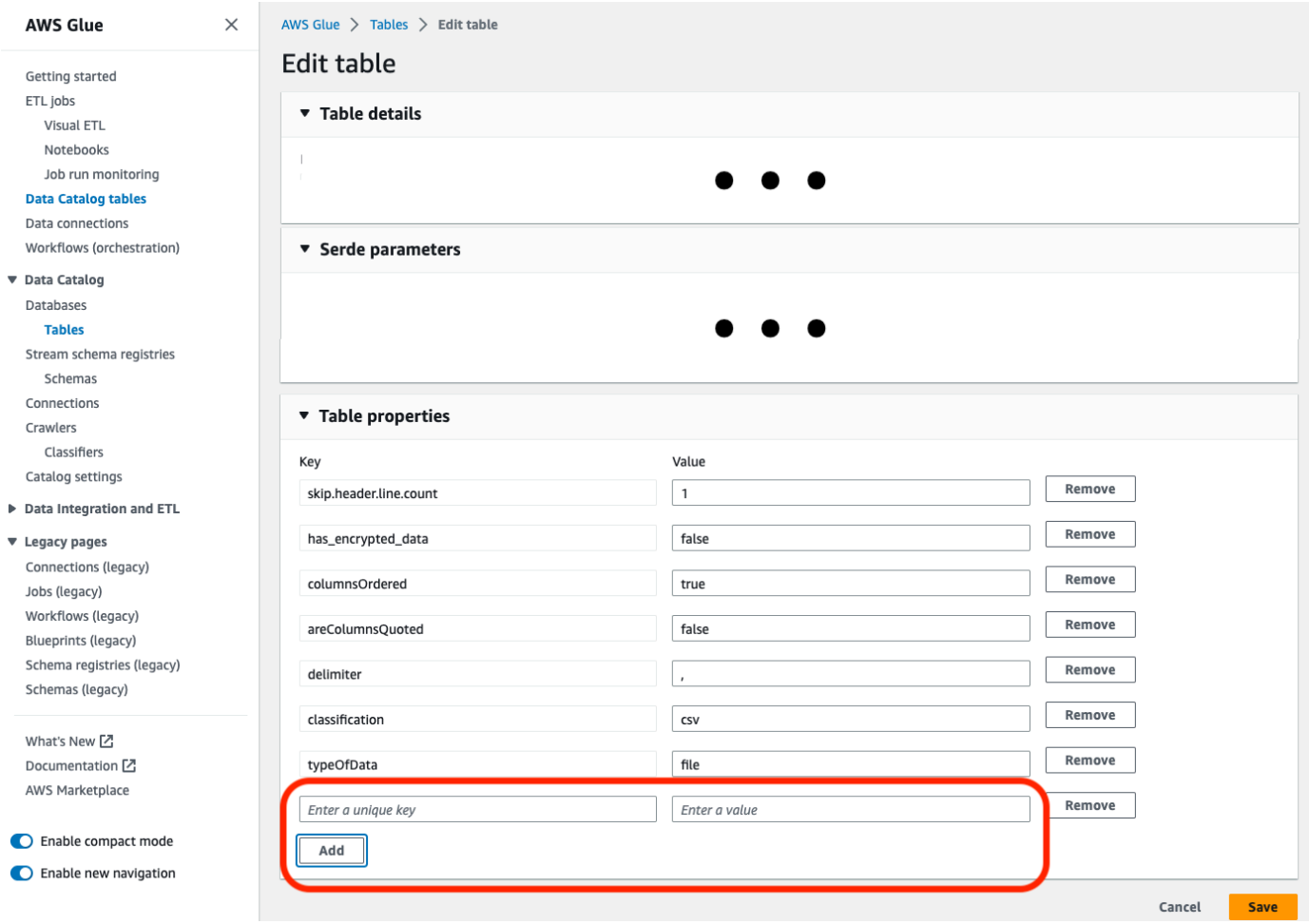
```
val options = JsonOptions(Map(
    "path" -> outputPath,
    "partitionKeys" -> Seq("<partition_1>", "<partition_2>"),
    "enableUpdateCatalog" -> true,
    "updateBehavior" -> "UPDATE_IN_DATABASE"))
val sink = glueContext.getSink(connectionType = "s3", connectionOptions =
 options).withFormat("<format>")
sink.setCatalogInfo(catalogDatabase = “<dst_db_name>”, catalogTableName =
 “<dst_tbl_name>”)
sink.writeDynamicFrame(df)
```

限制

請注意下列限制：

- 僅支援 Amazon Simple Storage Service (Amazon S3) 的目標。
- 控管的資料表不支援 enableUpdateCatalog 功能。

- 僅支援下列格式：json、csv、avro 和 parquet。
- 若要使用parquet分類建立或更新表格，您必須使用AWS Glue最佳化的實木複合地板寫入器 DynamicFrames。這可以透過以下方法之一來達成：
  - 如果要使用 parquet 分類來更新型錄中的現有資料表，則在更新之前，資料表的 "useGlueParquetWriter" 資料表屬性必須設定為 true。您可以透過 AWS Glue API/SDK、主控台或透過 Athena DDL 陳述式來設定此屬性。



設定型錄資料表屬性後，您便可以使用以下程式碼片段，以新資料更新型錄資料表：

```
glueContext.write_dynamic_frame.from_catalog(
    frame=frameToWrite,
    database="dbName",
    table_name="tableName",
    additional_options={
        "enableUpdateCatalog": True,
        "updateBehavior": "UPDATE_IN_DATABASE"
    }
```

```
)
```

- 如果型錄中尚不存在資料表，您可以在指令碼中使用 `getSink()` 方法和 `connection_type="s3"` 將資料表及其分割區新增至型錄中，並將資料寫入 Amazon S3。為您的工作流程提供適當的 `partitionKeys` 和 `compression`。

```
s3sink = glueContext.getSink(
    path="s3://bucket/folder/",
    connection_type="s3",
    updateBehavior="UPDATE_IN_DATABASE",
    partitionKeys=[],
    compression="snappy",
    enableUpdateCatalog=True
)

s3sink.setCatalogInfo(
    catalogDatabase="dbName", catalogTableName="tableName"
)

s3sink.setFormat("parquet", useGlueParquetWriter=true)
s3sink.writeFrame(frameToWrite)
```

  - `glueparquet`格式值是啟用實 AWS Glue 木複合地板寫入器的傳統方法。
- 將 `updateBehavior` 設為 `LOG` 後，只有在 DynamicFrame 結構描述等同於或包含資料目錄資料表結構描述中定義之欄的子集時，才會新增新的分割區。
- 未分割資料表不支援結構描述更新 (不使用 "partitionKeys" 選項)。
- 在 ETL 指令碼中傳遞的參數和資料目錄資料表結構描述中的 partitionKeys 之間，您的 partitionKeys 必須等效，而且順序相同。
- 此功能目前尚不支援更新/建立更新巢狀結構描述的資料表 (例如，結構內的陣列)。

如需更多詳細資訊，請參閱 the section called “AWS Glue for Spark”。

在 ETL 任務中使用 MongoDB 連線

您可以為 MongoDB 建立一個連線，然後在 AWS Glue 任務中使用該連線。如需詳細資訊，請參閱《AWS Glue 程式設計指南》中的 the section called “MongoDB 連線”。連線 `url`、`username` 和 `password` 儲存在 MongoDB 連線中。其他選項可以在 ETL 任務指令碼中使用 `glueContext.getCatalogSource` 的 `additionalOptions` 參數指定。其他選項包括：

- `database`：(必要) 讀取的 MongoDB 資料庫。

- `collection`：(必要) 讀取的 MongoDB 集合。

藉由將 `database` 和 `collection` 資訊放在 ETL 任務指令碼中，您可以在多個任務中使用相同的連線。

1. 為 MongoDB 資料來源建立 AWS Glue Data Catalog 連線。請參閱 "connectionType": "mongodb" 以取得此連線參數的描述。您可以使用主控台、API 或 CLI 來建立連線。
2. 在 AWS Glue Data Catalog 建立資料庫來儲存您的 MongoDB 資料的資料表定義。如需詳細資訊，請參閱建立資料庫。
3. 建立爬蟲程式，使用連接到 MongoDB 的連線中的資訊，網路爬取 MongoDB 中的資料。爬蟲程式在 AWS Glue Data Catalog 建立資料表，它描述了您在任務中使用的 MongoDB 資料庫中的資料表。如需詳細資訊，請參閱使用編目器填入資料目錄 。
4. 使用自訂指令碼來建立任務。您可以使用主控台、API 或 CLI 來建立它們。如需詳細資訊，請參閱在 AWS Glue 新增任務。
5. 選擇任務的資料目標。代表資料目標的資料表可在您的資料目錄中定義，或者您的任務可在執行時建立目標資料表。選擇編寫任務時的目標位置。如果目標需要連線，您的任務也會參照此連線。如果您的任務需要多個資料目標，可在之後編輯指令碼以新增來源。
6. 為任務及產生的指令碼提供引數，以自訂任務處理環境。

   這裡提供了從基於資料目錄中定義之資料表結構的 MongoDB 資料庫建立 `DynamicFrame` 的範例。程式碼使用 `additionalOptions` 以提供其他資料來源資訊：

   Scala

   ```
   val resultFrame: DynamicFrame = glueContext.getCatalogSource(
           database = catalogDB,
           tableName = catalogTable,
           additionalOptions = JsonOptions(Map("database" -> DATABASE_NAME,
                   "collection" -> COLLECTION_NAME))
         ).getDynamicFrame()
   ```

   Python

   ```
   glue_context.create_dynamic_frame_from_catalog(
           database = catalogDB,
           table_name = catalogTable,
           additional_options = {"database":"database_name",
   ```

```
                "collection":"collection_name"})
```

7. 隨需或透過觸發執行任務。

使用資料欄統計資料來最佳化

您可以計算資料格式 (例如實木複合地板、ORC、JSON、ION、CSV 和 XML) 資料 AWS Glue Data Catalog 表的資料行層級統計資料，而無需設定其他資料管線。資料欄統計資料可協助您透過深入了解資料欄內的值，了解資料設定檔。Data Catalog 支援產生資料欄值的統計資料 (例如，最小值、最大值、總 Null 值、總相異值、值平均長度及 true 值出現總數)。

AWS 分析服務如 Amazon Redshift， Amazon Athena 可以使用這些列統計信息來生成查詢執行計劃，並選擇提高查詢性能的最佳計劃。

您可以設定為使用 AWS Glue 主控台或執行資料行統計資料產生工作 AWS CLI。當您啟動程序時，會在背景 AWS Glue 啟動 Spark 工作，並更新「資料目錄」中的 AWS Glue 表格中繼資料。您可以使用 AWS Glue 主控台 AWS CLI 或呼叫資料GetColumnStatisticsFor表 API 作業來檢視資料行統計資料。

> **Note**
>
> 如果您正在使用 Lake Formation 權限控制資料表的存取權，則資料欄統計資料任務所擔任的角色將需要完整的資料表存取權，才可產生統計資料。

主題

產生資料欄統計資料的先決條件

若要產生或更新資料欄統計資料，統計資料產生任務會代表您擔任 AWS Identity and Access Management (IAM) 角色。根據授與角色的權限，資料欄統計資料產生任務可以從 Amazon S3 資料存放區讀取資料。

> **Note**
>
> 若要針對由 Lake Formation 管理的資料表產生統計資料，則用於產生統計資料的 IAM 角色需要取得完整資料表存取權。

若要使用角色行存取控制，您必須建立具有以下政策所列之權限的 IAM 角色，並將該角色新增至資料欄統計資料產生任務。

建立產生資料欄統計資料的 IAM 角色

1. 若要建立 IAM 角色，請參閱建立 AWS GlueIAM 角色。
2. 若要更新現有角色，請在 IAM 主控台中，前往產生資料欄統計資料程序正在使用的 IAM 角色。
3. 在新增權限區段，選擇連接政策。在新開啟的瀏覽器視窗中，選擇`AWSGlueServiceRole` AWS 受管理的策略。
4. 您也需要納入從 Amazon S3 資料位置讀取資料的權限。

   在新增權限區段中，選擇建立政策。在新開啟的瀏覽器視窗中，建立要搭配您角色使用的新政策。
5. 在建立政策頁面中，選擇 JSON 索引標籤。將下列 `JSON` 程式碼複製到政策編輯器欄位。

   > **Note**
   >
   > 在下列政策中，請將帳戶 ID 取代為有效的帳戶 ID AWS 帳戶，並以表格的區域和 `bucket-name` Amazon S3 儲存貯體名稱取`region`代。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3BucketAccess",
            "Effect": "Allow",
            "Action": [
```

```
                "s3:ListBucket",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::<bucket-name>/*",
                "arn:aws:s3:::<bucket-name>"
            ]
        }
    ]
}
```

6. (選用) 如果您正在使用 Lake Formation 權限向資料提供存取權，則 IAM 角色需要 `lakeformation:GetDataAccess` 權限。

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
            "Sid": "LakeFormationDataAccess",
            "Effect": "Allow",
            "Action": "lakeformation:GetDataAccess",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

如果 Amazon S3 資料位置已向 Lake Formation 註冊，且資料欄統計資料產生任務擔任的 IAM 角色沒有授予資料表的 `IAM_ALLOWED_PRINCIPALS` 群組權限，則該角色需要資料表中的 Lake Formation `ALTER` 和 `DESCRIBE` 權限。用於註冊 Amazon S3 儲存貯體的角色需要資料表中的 Lake Formation `INSERT` 和 `DELETE` 權限。

如果 Amazon S3 資料位置已向 Lake Formation 註冊，且 IAM 角色沒有授予資料表的 `IAM_ALLOWED_PRINCIPALS` 群組權限，則該角色需要資料表中的 Lake Formation `ALTER`、`DESCRIBE`、`INSERT` 及 `DELETE` 權限。

7. (選用) 寫入加密 Amazon CloudWatch Logs 的資料欄統計資料產生任務，需要金鑰政策中的下列權限。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "CWLogsKmsPermissions",
        "Effect":  "Allow",
        "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:AssociateKmsKey"
        ],
        "Resource": [
            "arn:aws:logs:<region>:111122223333:log-group:/aws-glue:*"
        ]
    },
    {
        "Sid": "KmsPermissions",
        "Effect": "Allow",
        "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt",
            "kms:Encrypt"
        ],
        "Resource": [
        "arn:aws:kms:<region>:111122223333:key/"arn of key used for ETL cloudwatch
 encryption"
        ],
        "Condition": {
            "StringEquals": {
                "kms:ViaService": ["glue.<region>.amazonaws.com"]
            }
        }
    }
    ]
}
```

8. 您用來執行資料行統計資料的角色必須具有角色的iam:PassRole權限。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "iam:PassRole"
```

```
        ],
        "Resource": [
            "arn:aws:iam::111122223333:role/<columnstats-role-name>"
        ]
    }]
}
```

9. 建立用於產生資料欄統計資料的 IAM 角色時，該角色也必須具有下列信任政策，使服務可以擔任該角色。

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Sid": "TrustPolicy",
        "Effect": "Allow",
        "Principal": {
          "Service": "glue.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
      }
   ]
}
```

產生資料欄統計資料

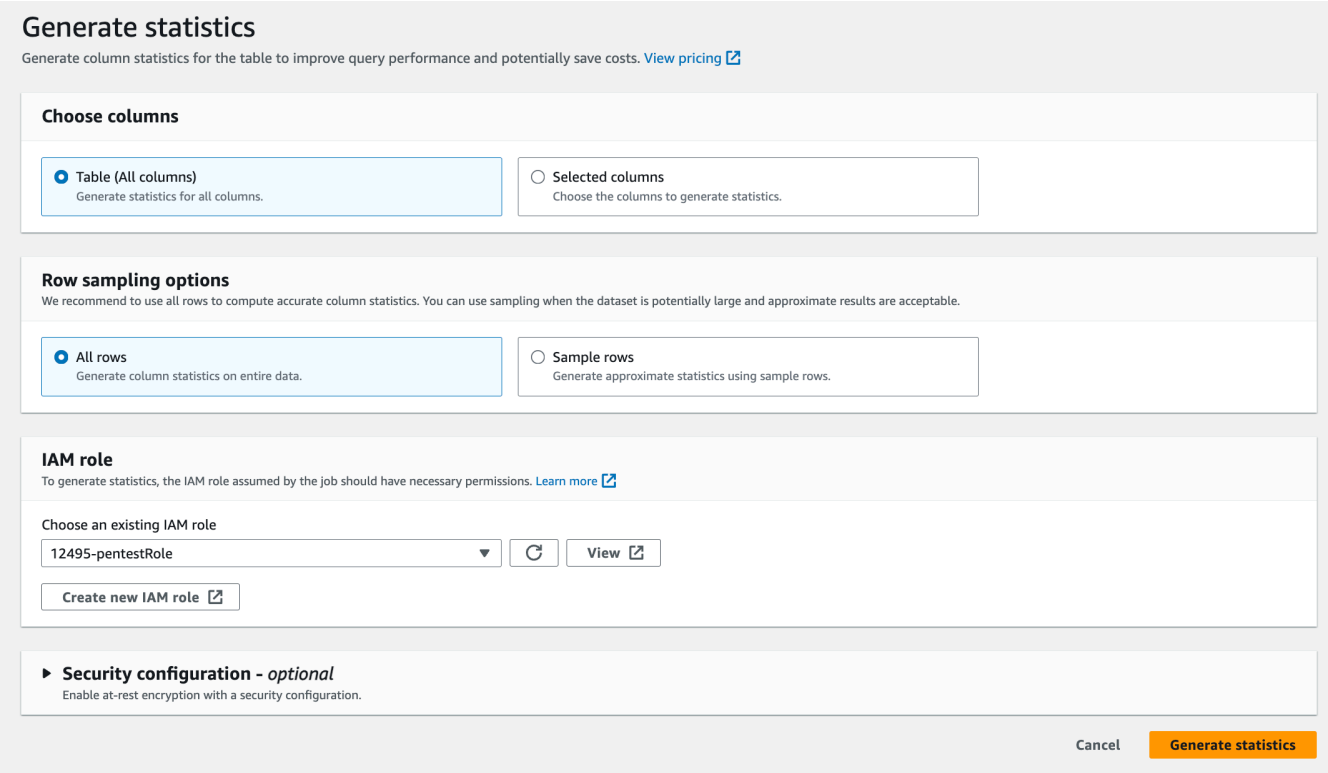
請依照下列步驟，使用 AWS Glue 主控台或 AWS CLI 管理 Data Catalog 中的統計資料產生。

Console

使用主控台產生資料欄統計資料

1. 登入 AWS Glue 主控台，請前往：https://console.aws.amazon.com/glue/。
2. 選擇 Data Catalog 資料表。
3. 從清單中選擇資料表。
4. 選擇動作功能表下方的產生統計資料。

   您也可選擇資料表頁面下半區段之資料欄統計資料索引標籤下的產生統計資料按鈕。
5. 在產生統計資料頁面中，指定下列選項：



- 資料表 (所有資料欄)：選擇此選項可產生資料表中所有資料欄的統計資料。
- 選取的資料欄：選擇此選項可產生特定資料欄的統計資料。您可以從下拉式清單中選取資料欄。
- 所有資料列：從資料表中選擇所有資料列，以產生準確的統計資料。
- 範例資料列：僅從資料表中選擇特定的資料列百分比以產生統計資料。預設值為所有資料列。使用向上和向下箭頭以增加或減少百分比值。

> **Note**
>
> 我們建議在資料表中包含所有資料列，以計算準確的統計資料。只有在接受近似值時，才使用範例資料列產生資料欄統計資料。

6. (選用) 接著，選擇安全組態，針對日誌啟用靜態加密。
7. 選擇產生統計資料以執行程序。

AWS CLI

在下列範例中，將 DatabaseName、TableName 及 ColumnNameList 的值取代為實際的資料庫、資料表及資料欄名稱。將帳戶 ID 取代為有效的 AWS 帳戶，並以您用於產生統計資料的 IAM 角色名稱取代角色名稱。

```
aws glue start-column-statistics-task-run --input-cli-json file://input.json
{
    "DatabaseName": "<test-db>",
    "TableName": "<test-table>",
    "ColumnNameList": [
        "<column1>",
        "<column2>",
    ],
    "Role": "arn:aws:iam::<123456789012>:role/<Stats-Role>",
    "SampleSize": 10.0
}
```

您也可透過呼叫 StartColumnStatisticsTaskRun 操作，以產生資料欄統計資料。

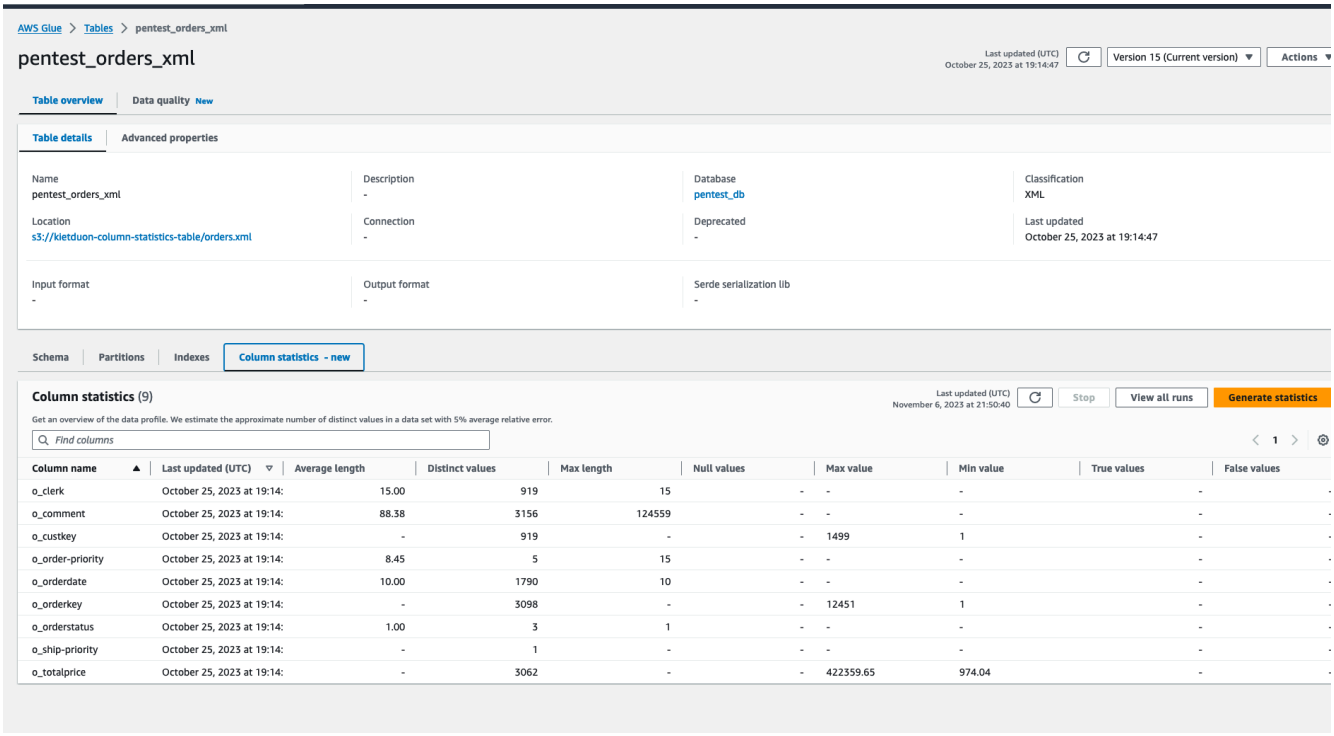
檢視資料欄統計資料

成功產生統計資料後，Data Catalog 會儲存此位於 Amazon Athena 和 Amazon Redshift 之成本型最佳化工具的資訊，以在執行查詢時進行最佳選擇。統計資料會根據資料欄類型而異。

AWS Management Console

檢視資料表的資料欄統計資料

- 執行資料欄統計資料任務後，資料表詳細資料頁面中的資料欄統計資料索引標籤會顯示資料表的統計資料。



下列為可用的統計資料：

- 資料欄名稱：用來產生統計資料的資料欄名稱
- 上次更新：統計資料產生時的資料和時間
- 平均長度：資料欄中值的平均長度
- 相異值：資料欄中的相異值總數。我們會以 5% 相對誤差率預估資料欄中相異值數。
- 最大值：資料欄中的最大值。
- 最小值：資料欄中的最小值。
- 最大長度：資料欄中的最高值長度。
- Null 值：資料欄中的 Null 值總數。
- True 值：資料欄中的 true 值總數。
- False 值：資料欄中的 false 值總數。

AWS CLI

下列範例顯示如何使用 AWS CLI 擷取資料欄統計資料。

```
aws glue get-column-statistics-for-table \
    --database-name <test_db> \
```

```
    --table-name <test_tble> \
    --column-names <col1>
```

您可以使用 GetColumnStatisticsForTable API 操作，來檢視資料欄統計資料。

更新資料欄統計資料

將統計資料保持在最新狀態，可讓查詢規劃工具選擇最佳計劃，進而改善查詢效能。您需要從 AWS Glue 主控台明確執行產生統計資料任務，以重新整理資料欄統計資料。Data Catalog 不會自動重新整理統計資料。

如果您沒有在主控台中使用 AWS Glue 的統計資料產生功能，可以使用 UpdateColumnStatisticsForTable API 操作或 AWS CLI 手動更新資料欄統計資料。下列範例顯示如何使用 AWS CLI 更新資料欄統計資料。

```
aws glue update-column-statistics-for-table --cli-input-json:

{
    "CatalogId": "111122223333",
    "DatabaseName": "test_db",
    "TableName": "test_table",
    "ColumnStatisticsList": [
        {
            "ColumnName": "col1",
            "ColumnType": "Boolean",
            "AnalyzedTime": "1970-01-01T00:00:00",
            "StatisticsData": {
                "Type": "BOOLEAN",
                "BooleanColumnStatisticsData": {
                    "NumberOfTrues": 5,
                    "NumberOfFalses": 5,
                    "NumberOfNulls": 0
                }
            }
        }
    ]
}
```

刪除資料欄統計資料

您可以使用 DeleteColumnStatisticsForTable API 操作或 AWS CLI 來刪除資料欄統計資料。下列範例顯示如何使用 AWS Command Line Interface (AWS CLI) 刪除資料欄統計資料。

```
aws glue delete-column-statistics-for-table \
    --database-name test_db \
    --table-name test_table \
    --column-name col1
```

檢視資料欄統計資料任務執行

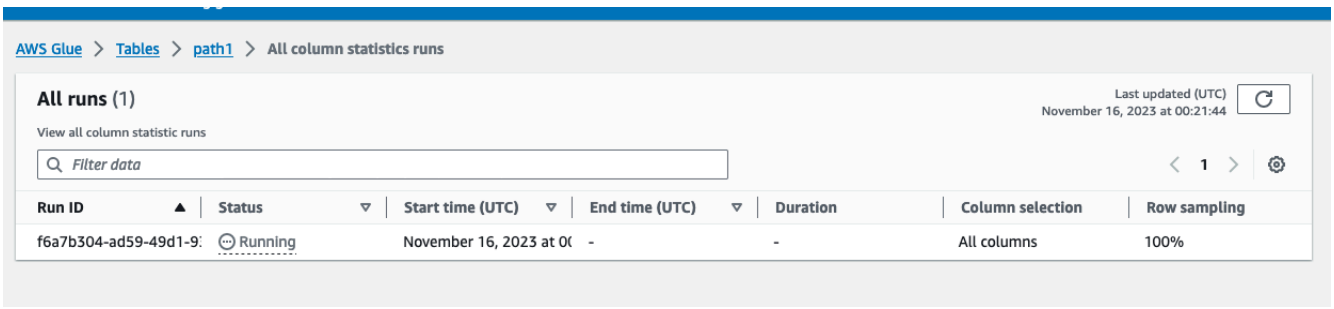
執行資料欄統計資料任務之後，您可以使用 AWS Glue 主控台、AWS CLI 或使用 GetColumnStatisticsTaskRuns 操作，來探索資料表的任務執行詳細資料。

Console

檢視資料欄統計資料任務執行詳細資料

1. 在 AWS Glue 主控台中，選擇 Data Catalog 下方的資料表。
2. 選取包含資料欄統計資料的資料表。
3. 在資料表詳細資料頁面中，選擇資料資料欄統計資料。
4. 選擇檢視執行。

   您可以查看與指定資料表相關聯之所有執行的相關資訊。



AWS CLI

在下列範例中，將 DatabaseName 和 TableName 值取代為實際的資料庫和資料表名稱。

```
aws glue get-column-statistics-task-runs --input-cli-json file://input.json
{
```

```
    "DatabaseName": "<test-db>",
    "TableName": "<test-table>"
}
```

停止資料欄統計資料任務執行

您可以針對使用 AWS Glue 主控台、AWS CLI 或使用 StopColumnStatisticsTaskRun 操作的資料表停止資料欄統計資料任務執行。

Console

停止資料行統計資料工作執行

1. 在 AWS Glue 主控台中，選擇 Data Catalog 下方的資料表。
2. 選取正在執行資料欄統計資料任務的資料表。
3. 在資料表詳細資料頁面中，選擇資料資料欄統計資料。
4. 選擇 Stop (停止)。

   如果您在執行完成前停止任務，系統將不會針對該資料表產生資料欄統計資料。

AWS CLI

在下列範例中，將 `DatabaseName` 和 `TableName` 值取代為實際的資料庫和資料表名稱。

```
aws glue stop-column-statistics-task-run --input-cli-json file://input.json
{
    "DatabaseName": "<test-db>",
    "TableName": "<test-table>"
}
```

考量與限制

下列考量與限制適用於產生資料欄統計資料。

考量事項

- 使用取樣產生統計資料可縮短執行時間，但可能產生不準確的統計資料。

- 每個資料欄統計資料執行都需要處理整個資料集。
- Data Catalog 不會儲存不同版本的統計資料。
- 每個資料表一次僅能執行一個統計資料產生任務。
- 如果使用向 Data Catalog 註冊的客戶 AWS KMS 金鑰加密資料表，則 AWS Glue 會使用相同的金鑰來加密統計資料。

資料欄統計資料任務支援產生統計資料：

- 當 IAM 角色具有完整資料表權限 (IAM 或 Lake Formation) 時。
- 當 IAM 角色具有使用 Lake Formation 混合存取模式的資料表權限時。

資料欄統計資料任務不支援產生統計資料：

- 具有以 Lake Formation 儲存格為基礎之存取控制的資料表。
- 交易資料湖：Linux Foundation Delta Lake、Apache Iceberg、Apache Hudi。
- 聯合資料庫中的資料表：Hive 中繼存放區、Amazon Redshift 資料共用
- 巢狀資料欄、陣列及結構資料類型。
- 從其他帳戶與您共用的資料表。

加密您的資料目錄

您可以使用 () 管理的加密金鑰來保護儲存在靜態中的中繼資 AWS Key Management Service AWS KMS料。 AWS Glue Data Catalog 您可以使用資料目錄設定為新資料目錄啟用資料目錄加密。您可以視需要啟用或停用現有資料目錄的加密。啟用時，會 AWS Glue 加密寫入目錄的所有新中繼資料，而現有的中繼資料則保持未加密狀態。

如需加密資料目錄的詳細資訊，請參閱加密您的資料目錄。

使用 Lake Formation 保護您的資料目錄

AWS Lake Formation 是一項可讓您更輕鬆地在中設定安全資料湖的服務 AWS。它透過定義精細的存取控制權限，提供建立和安全管理資料湖的集中位置。Lake Formation 使用「資料目錄」來儲存和擷取有關資料湖的中繼資料，例如資料表定義、結構描述資訊和資料存取控制設定。

您可以向 Lake Formation 註冊中繼資料表或資料庫的 Amazon S3 資料位置，並使用它來定義資料目錄資源的中繼資料層級許可。您也可以使用 Lake Formation 代表整合式分析引擎，針對存放在 Amazon S3 中的基礎資料管理儲存存取權限。

如需詳細資訊，請參閱什麼是 AWS Lake Formation？ 。

存取資料目錄

您可以使用 AWS Glue Data Catalog 來探索和瞭解您的資料。資料目錄提供一致的方式來維護結構描述定義、資料類型、位置和其他中繼資料。您可以使用下列方法存取「資料目錄」：

- AWS Glue 控制台 — 您可以通過基於 Web 的用戶界面 AWS Glue 控制台訪問和管理數據目錄。主控台可讓您瀏覽和搜尋資料庫、表格及其關聯的中繼資料，以及建立、更新和刪除中繼資料定義。
- AWS Glue 編目程式 — 爬蟲是自動掃描您的資料來源並使用中繼資料填入資料目錄的程式。您可以建立並執行爬取器，以探索和編目來自不同來源的資料，例如 Amazon S3、Amazon RDS、亞馬遜 DynamoDB 和與 JDB 相容的關聯式資料庫 (例如 MySQL 和 PostgreSQL)，以及數個非來源 (例如雪花和 Google)。 Amazon CloudWatchAWS BigQuery
- AWS Glue API — 您可以使用 AWS Glue API 以程式設計方式存取資料目錄。這些 API 可讓您透過程式設計方式與資料目錄互動，以實現與其他應用程式和服務的自動化和整合。
- AWS Command Line Interface (AWS CLI) — 您可以使用 AWS CLI 從指令行存取和管理「資料目錄」。CLI 提供用於建立、更新和刪除中繼資料定義，以及查詢和擷取中繼資料資訊的命令。
- 與其他 AWS 服務整合 — 資料型錄與其他各種 AWS 服務整合，可讓您存取和使用目錄中儲存的中繼資料。例如，您可以使用 Amazon Athena 使用資料目錄中的中繼資料查詢資料來源，並用 AWS Lake Formation 於管理資料目錄資源的資料存取和控管。

AWS Glue 資料目錄最佳做法

本節涵蓋有效管理和使用 AWS Glue Data Catalog. 它強調實踐，例如有效的爬蟲使用，元數據組織，安全性，性能優化，自動化，數據控管以及與其他 AWS 服務的集成。

- 有效使用檢索器 — 定期執行檢索器，讓資料目錄 up-to-date 隨著資料來源的變更保持在一起。針對頻繁變更的資料來源使用增量編目，以提升效能。設定爬行者程式，以便在偵測到變更時自動新增分割區或更新結構描述。
- 組織和命名中繼資料表 — 為「資料目錄」中的資料庫和表格建立一致的命名慣例。將相關資料來源分組成邏輯資料庫或資料夾，以便更好地組織 使用描述性名稱來傳達每個表的目的和內容。

- 有效管理結構描述 — 利用 AWS Glue 編目器的結構描述推論功能。在套用結構描述變更之前，請先檢閱和更新結構描述變更，以避免 使用結構描述演進功能優雅地處理結構描述變更。
- 保護資料目錄 — 為資料目錄啟用靜態和傳輸中的資料加密。實作精細的存取控制原則，以限制對敏感資料的存取。定期稽核和檢閱資料目錄權限和活動記錄。
- 與其他 AWS 服務整合資料型錄使用資料目錄做為 Amazon Athena、Redshift 頻譜和 AWS Lake Formation服務的集中中繼資料層。利用 AWS Glue ETL 工作將資料轉換並載入到各種資料存放區，同時維護資料目錄中的中繼資料。
- 監視和最佳化效能資料目錄使 Amazon CloudWatch 用指標監視編目器和 ETL 工作的效能。對資料目錄中的大型資料集進行分割，以改善查詢效能。為經常存取的中繼資料實作效能最佳化。
- 使用 AWS Glue 文件和最佳做法資料型錄隨時更新定期查看 AWS Glue 文件和 AWS Glue 資源，以取得最新的更新、最佳做法和建議。參加 AWS Glue 網路研討會、研討會和其他活動，向專家學習並隨時瞭解新功能。

AWS Glue 結構描述登錄檔

> **Note**
>
> AWS Glue 主控台中的下列區域不支援 AWS Glue 結構描述登錄檔：亞太區域 (雅加達) 和中東 (阿拉伯聯合大公國)。

AWS Glue 結構描述登錄檔是一項新功能，可讓您集中探索、控制和發展資料串流結構描述。結構描述定義資料記錄的結構和格式。透過AWS Glue結構描述登錄，您可以使用與 Apache Kafka、Amazon Kinesis 資料串流、適用於 Apache Flink 的亞馬遜管理服務和整合 Amazon Managed Streaming for Apache Kafka，在資料串流應用程式上管理和強制執行結構描述。AWS Lambda

AWS Glue 結構描述登錄檔支援 AVRO (v1.10.2) 資料格式、JSON 資料格式 (具有 JSON 結構描述格式 (適用於結構描述 (規格 Draft-04、Draft-06 和 Draft-07)) 和使用 Everit 程式庫進行 JSON 結構描述驗證)、不支援 `extensions` 或 `groups` 的協定緩衝區 (Protobuf) 版本 proto2 和 proto3，以及 Java 語言，對其他資料格式和語言的支援即將推出。支援的功能包括相容性、透過中繼資料取得結構描述、自動註冊結構描述、IAM 相容性，以及選用的 ZLIB 壓縮，以減少儲存和資料傳輸。AWS Glue結構描述登錄檔是無伺服器且可免費使用。

使用結構描述作為生產者與消費者之間的資料格式合約，可改善資料控管、更高品質的資料，並讓資料消費者能夠彈性相容上游變更。

結構描述登錄檔允許不同的系統共用序列化和還原序列化的結構描述。例如，假設您擁有資料的生產者和消費者。生產者在發佈資料時知道結構描述。結構描述登錄檔提供某些系統，如 Amazon MSK 或 Apache Kafka 的序列化程式和還原序列化程式。

如需詳細資訊，請參閱 結構描述登錄檔的運作方式。

主題

結構描述

結構描述定義資料記錄的結構和格式。結構描述是可靠的資料發佈、耗用或儲存的版本化規格。

在 Avro 的這個範例結構描述中，格式和結構由配置和欄位名稱定義，欄位名稱的格式由資料類型定義 (例如，`string`、`int`)。

```
{
    "type": "record",
    "namespace": "ABC_Organization",
    "name": "Employee",
    "fields": [
        {
            "name": "Name",
            "type": "string"
        },
        {
            "name": "Age",
            "type": "int"
        },
```

```
        {
            "name": "address",
            "type": {
                "type": "record",
                "name": "addressRecord",
                "fields": [
                    {
                        "name": "street",
                        "type": "string"
                    },
                    {
                        "name": "zipcode",
                        "type": "int"
                    }
                ]
            }
        }
    ]
}
```

在此範例 JSON Schema Draft-07 for JSON 中，格式是由 JSON 結構描述組織所定義。

```
{
 "$id": "https://example.com/person.schema.json",
 "$schema": "http://json-schema.org/draft-07/schema#",
 "title": "Person",
 "type": "object",
 "properties": {
  "firstName": {
   "type": "string",
   "description": "The person's first name."
  },
  "lastName": {
   "type": "string",
   "description": "The person's last name."
  },
  "age": {
   "description": "Age in years which must be equal to or greater than zero.",
   "type": "integer",
   "minimum": 0
  }
 }
}
```

在這個 Protobuf 範例中，格式由協定緩衝區語言的版本 2 (proto2) 定義。

```
syntax = "proto2";

package tutorial;

option java_multiple_files = true;
option java_package = "com.example.tutorial.protos";
option java_outer_classname = "AddressBookProtos";

message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    optional string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phones = 4;
}

message AddressBook {
  repeated Person people = 1;
}
```

登錄檔

登錄檔是結構描述的邏輯容器。登錄檔允許您組織您的結構描述，以及管理應用程式的存取控制。登錄檔具有 Amazon Resource Name (ARN)，可讓您組織和設定登錄檔中結構描述操作的不同存取許可。

您可以使用預設登錄檔，或視需要建立許多新的登錄檔。

AWS Glue 結構描述登錄檔階層

- RegistryName：[字符串]

  - RegistryArn：[AWS ARN]
  - CreatedTime：[時間戳記]
  - UpdatedTime：[時間戳記]

- SchemaName：[字符串]
  - SchemaArn：[AWS ARN]
  - DataFormat：[阿夫羅，Json 或原生物]
  - Compatibility: [eg. BACKWARD, BACKWARD_ALL, FORWARD, FORWARD_ALL, FULL, FULL_ALL, NONE, DISABLED]
  - Status: [eg. PENDING, AVAILABLE, DELETING]
  - SchemaCheckpoint：[整數]
  - CreatedTime：[時間戳記]
  - UpdatedTime：[時間戳記]
- SchemaVersion：[字符串]
  - SchemaVersionNumber：[整數]
  - Status: [eg. PENDING, AVAILABLE, DELETING, FAILURE]
  - SchemaDefinition：[字符串，值：JSON]
  - CreatedTime：[時間戳記]
- SchemaVersionMetadata：[列表]
  - MetadataKey：[字符串]
  - MetadataInfo
  - MetadataValue：[字符串]
  - CreatedTime：[時間戳記]

結構描述版本控制和相容性

每個結構描述可以有多個版本。版本控制是由套用在結構描述上的相容性規則所控制。結構描述登錄檔會根據此規則檢查登錄新結構描述版本的要求，然後才能成功。

標示為檢查點的結構描述版本是用來判斷註冊新結構描述版本的相容性。第一次建立結構描述時，預設檢查點將是第一個版本。當結構描述隨著更多版本發展，您可以使用 CLI/SDK 將檢查點變更為使用

UpdateSchema API，它遵循一組約束。在主控台中，編輯結構描述定義或相容性模式預設會將檢查點變更為最新版本。

相容性模式可讓您控制結構描述可以或不可以隨著時間變化的方式。這些模式形成應用程式產生和使用資料之間的合約。當新版的結構描述提交至登錄檔時，會使用套用至結構描述名稱的相容性規則來判斷是否可以接受新版本。共有 8 種相容性模式：NONE、DISABLED、BACKWARD、BACKWARD_ALL、FORWARD、FORWARD_ALL、FULL、FULL_ALL

在 Avro 資料格式中，欄位可以是選用的或必要的。選用欄位是其中 Type 包含 null。必要欄位的 Type 沒有 null。

在 Protobuf 資料格式中，proto2 語法中的欄位可以是選用 (包括重複) 或必要欄位，而 proto3 語法中的所有欄位均是選用 (包括重複) 欄位。所有相容性規則均是根據對協定緩衝區規範的理解以及 Google 協定緩衝區文件中的指導所決定。

- NONE：未套用相容性模式。您可以在開發案例中使用此選項，或者如果您不知道要套用至結構描述的相容性模式。任何新增的版本都會被接受，而不經過相容性檢查。
- DISABLED：此相容性選項可防止特定結構描述的版本控制。無法新增新版本。
- BACKWARD：建議使用此相容性選項，因為它允許消費者讀取目前和先前的結構描述版本。當您刪除欄位或新增選用欄位時，您可以使用此選項來檢查與先前結構描述版本的相容性。一個典型的 BACKWARD 使用案例是，當您的應用程式為最新的結構描述而建立時。

  AVRO

  例如，假設您有一個由名字 (必要)、姓氏 (必要)、電子郵件 (必要) 和電話號碼 (選用) 所定義的結構描述。

  如果您的下一個結構描述版本移除必要的電子郵件欄位，將可成功註冊。BACKWARD 相容性要求消費者能夠讀取目前和先前的結構描述版本。您的消費者將能夠讀取新的結構描述，因為舊訊息中的額外電子郵件欄位被忽略。

  如果您有新增必要欄位 (例如郵遞區號) 的提議新結構描述版本，則不會成功註冊 BACKWARD 相容性。新版本的消費者將無法在結構描述變更之前讀取舊訊息，因為他們缺少必要的郵遞區號欄位。但是，如果在新的結構描述中將郵遞區號欄位設定為選用，則提議的版本將成功註冊，因為消費者可以在沒有選用郵遞區碼欄位的情況下讀取舊的結構描述。

  JSON

  例如，假設您有一個由名字 (選用)、姓氏 (選用)、電子郵件 (選用) 和電話號碼 (選用) 所定義的結構描述版本。

如果您的下一個結構描述版本新增了選用的電話號碼屬性，只要原始結構描述版本不允許透過將 `additionalProperties` 欄位設定為 false 的任何其他屬性，這就會成功註冊。BACKWARD 相容性要求消費者能夠讀取目前和先前的結構描述版本。您的消費者將能夠讀取電話號碼屬性不存在的原始結構描述產生的資料。

如果您有新增選用電話號碼屬性的提議新結構描述版本，當原始結構描述版本將 `additionalProperties` 欄位設定為 true 時 (即允許任何其他屬性)，這將無法成功註冊 BACKWARD 相容性。新版本的消費者將無法在結構描述變更之前讀取舊訊息，因為他們無法讀取具有不同類型 (例如是字串而不是數字) 之電話號碼屬性的資料。

PROTOBUF

例如，假設您有一個由訊息 `Person` 與 proto2 語法下的 `first name` (必要)、`last name` (必要)、`email` (必要) 和 `phone number` (選用) 等欄位所定義的結構描述版本。

與 AVRO 案例類似，如果您的下一個結構描述版本移除必要的 `email` 欄位，該版本可成功註冊。BACKWARD 相容性要求消費者能夠讀取目前和先前的結構描述版本。舊訊息中的額外 `email` 欄位會被忽略，您的消費者將能夠讀取新的結構描述。

如果您有新增了必要欄位 (例如 `zip code`) 的提議新結構描述版本，根據 BACKWARD 相容性規則，該版本無法成功註冊。新版本的消費者將無法讀取結構描述變更之前的舊訊息，因為它們缺少必要的 `zip code` 欄位。但是，如果在新的結構描述中將 `zip code` 欄位設定為選用，則提議的版本可成功註冊，因為消費者可以讀取不含 `zip code` 選用欄位的結構描述。

在 GrPC 使用案例中，新增新 RPC 服務或 RPC 方法是向後相容的變更。例如，假設您有一個由 RPC 服務 `MyService` 使用 `Foo` 和 `Bar` 兩種 RPC 方法所定義的結構描述版本。

如果您的下一個結構描述版本新增了名為 `Baz` 的新 RPC 方法，該版本可成功註冊。因為新加入的 RPC 方法 `Baz` 為可選方法，根據 BACKWARD 相容性規則，您的消費者將能夠讀取原始結構描述產生的資料。

如果您有移除了現有 RPC 方法 `Foo` 的提議新結構描述版本，根據 BACKWARD 相容性規則，該版本無法成功註冊。新版本的消費者將無法讀取結構描述變更之前的舊訊息，因為他們無法使用 gRPC 應用程式中不存在的 RPC 方法 `Foo` 來理解和讀取資料。

- BACKWARD_ALL：此相容性選項可讓消費者同時讀取目前和所有先前的結構描述版本。當您刪除欄位或新增選用欄位時，您可以使用此選項來檢查所有先前結構描述版本的相容性。
- FORWARD：此相容性選項可讓消費者同時讀取目前和後續的結構描述版本，但不一定是較新的版本。當您新增欄位或刪除選用欄位時，您可以使用此選項來檢查與上一個結構描述版本的相容

性。FORWARD 的典型使用案例是您的應用程式已經為之前的結構描述建立，並且應該能夠處理更新的結構描述。

AVRO

例如，假設您有一個由名字 (必要)、姓氏 (必要)、電子郵件 (選用) 定義的結構描述版本。

如果您有一個新的結構描述版本新增了必要欄位，例如電話號碼，該版本可成功註冊。FORWARD 相容性要求消費者能夠使用舊版本來讀取以新結構描述產生的資料。

如果您有刪除了必要名字欄位的提議結構描述版本，根據 FORWARD 相容性規則，該版本無法成功註冊。您在以前版本上的消費者將無法讀取提議的結構描述，因為他們缺少必要的名字欄位。但是，如果名字欄位最初是選用的，那麼提議的新結構描述將成功註冊，因為消費者可以根據沒有選用名字欄位的新結構描述讀取資料。

JSON

例如，假設您有一個由名字 (選用)、姓氏 (選用)、電子郵件 (選用) 和電話號碼 (選用) 所定義的結構描述版本。

如果您有移除選用電話號碼屬性的新結構描述版本，只要新結構描述版本不允許透過將 `additionalProperties` 欄位設定為 false 的任何其他屬性，這就會成功註冊。FORWARD 相容性要求消費者能夠使用舊版本來讀取以新結構描述產生的資料。

如果您有刪除選用電話號碼屬性的提議結構描述版本，則當新的結構描述版本將 `additionalProperties` 欄位設定為 true，即允許任何額外的屬性，這不會成功註冊 FORWARD 相容性。舊版本的消費者將無法讀取提議的結構描述，因為他們可能具有不同類型的電話號碼屬性，例如是字串而不是數字。

PROTOBUF

例如，假設您有一個由訊息 Person 與 proto2 語法下的 `first name` (必要)、`last name` (必要) 和 `email` (選用) 等欄位所定義的結構描述版本。

與 AVRO 案例類似，如果您有一個新的結構描述版本新增了必要欄位，例如 `phone number`，該版本可成功註冊。FORWARD 相容性要求消費者能夠使用舊版本來讀取以新結構描述產生的資料。

如果您有刪除了必要 `first name` 欄位的提議結構描述版本，根據 FORWARD 相容性規則，該版本無法成功註冊。您在以前版本上的消費者將無法讀取提議的結構描述，因為它們缺少必要的

`first name` 欄位。但是，如果 `first name` 欄位最初是選用欄位，則提議的新結構描述可成功註冊，因為消費者可以讀取沒有 `first name` 選用欄位的新結構描述資料。

在 gRPC 使用案例中，移除 RPC 服務或 RPC 方法是向前相容的變更。例如，假設您有一個由 RPC 服務 `MyService` 使用 `Foo` 和 `Bar` 兩種 RPC 方法所定義的結構描述版本。

如果您的下一個結構描述版本刪除了名為 `Foo` 的現有 RPC 方法，根據 FORWARD 相容性規則，該版本可成功註冊，因為消費者可以使用舊版本來讀取新結構描述產生的資料。如果您有新增了 RPC 方法 `Baz` 的提議新結構描述版本，根據 FORWARD 相容性規則，該版本無法成功註冊。您以前版本上的消費者將無法讀取提議的結構描述，因為它們缺少 RPC 方法 `Baz`。

- FORWARD_ALL：此相容性選項可讓消費者讀取任何新註冊結構描述的生產者所寫入的資料。當您需要新增欄位或刪除選用欄位，並檢查與所有先前結構描述版本的相容性時，您可以使用此選項。
- FULL：此相容性選項可讓消費者讀取使用前一個或下一個版本的結構描述，但不能讀取更舊或更新版本的生產者所寫入的資料。當您新增或移除選用欄位時，您可以使用此選項來檢查與上一個結構描述版本的相容性。
- FULL_ALL：此相容性選項可讓消費者讀取生產者使用所有先前的結構描述版本所寫入的資料。當您新增或移除選用欄位時，您可以使用此選項來檢查所有先前結構描述版本的相容性。

開源 Serde 程式庫

AWS 提供開源 Serde 庫作為序列化和反序列化數據的框架。這些程式庫的開源設計允許通用的開源應用程式和架構在他們的專案中支援這些程式庫。

如需 Serde 函式庫如何運作的詳細資訊，請參閱結構描述登錄檔的運作方式。

結構描述登錄檔的配額

配額 (在中 AWS也稱為限制) 是您 AWS 帳戶中資源、動作和項目的最大值。下列是 AWS Glue 中結構描述登錄檔的軟性限制。

結構描述版本中繼資料索引鍵-值對

每個 AWS 區域最多可以有 10 SchemaVersion 個鍵值對。

您可以使用 QuerySchemaVersionMetadata 行動（Python：查詢模式版本的元數據）或 PutSchemaVersionMetadata 行動（Python：放置版本的元數據）API 檢視或設定索引鍵值中繼資料配對。

下列是 AWS Glue 中結構描述登錄檔的硬性限制。

登錄檔

此帳戶的每個 AWS 區域最多可有 100 個登錄。

SchemaVersion

此帳戶的每個 AWS 區域最多可以有 10000 個結構描述版本。

每個新模式都會創建一個新的模式版本，因此理論上，如果每個模式只有一個版本，則每個區域每個帳戶最多可以有 10000 個模式。

結構描述承載

結構描述裝載的大小限制為 170 KB。

結構描述登錄檔的運作方式

本節說明結構描述登錄檔中的序列化和還原序列化處理程序的運作方式。

1. 註冊結構描述：如果結構描述不存在於登錄檔中，則可以使用等於目的地名稱的結構描述名稱來註冊結構描述 (例如，test_topic、test_stream、prod_firehose)，或者生產者可以提供結構描述的自訂名稱。生產者也可以將索引鍵-值對做為中繼資料新增至結構描述，例如 source: msk_kafka_topic_A，或在建立結構描述時套用 AWS 標籤至結構描述。結構描述註冊後，結構描述登錄檔會將結構描述版本 ID 傳回至序列化程式。如果結構描述存在，但序列化程式正在使用不存在的新版本，結構描述登錄檔會檢查結構描述參考相容性規則，以確保新版本是相容性的，然後再將它註冊為新版本。

   註冊結構描述的方法有兩種：手動註冊和自動註冊。您可以透過 AWS Glue 主控台或 CLI/SDK 手動註冊結構描述。

   當序列化程式設定中開啟自動註冊時，將執行結構描述的自動註冊。如果在生產者組態中沒有提供 `REGISTRY_NAME`，則自動註冊將在預設登錄檔 (default-registry) 下註冊新的結構描述版本。請參閱安裝 SerDe 程式庫，以取得有關指定自動註冊屬性的資訊。
2. 序列化程式會針對結構描述驗證資料記錄：當產生資料的應用程式已註冊其結構描述時，結構描述登錄檔序列化程式會驗證應用程式所產生的記錄是以符合已註冊的結構描述的欄位和資料類型構成。如果記錄的結構描述不符合註冊的結構描述，序列化程式將傳回例外狀況，應用程式將無法將記錄傳遞到目的地。

   如果沒有結構描述存在，且結構描述名稱未透過生產者組態提供，則結構描述會以與主題名稱 (如果是 Apache Kafka 或 Amazon MSK) 或串流名稱 (如果是 Kinesis Data Streams) 相同的名稱建立結構描述。

每個記錄都有結構描述定義和資料。結構描述定義會根據結構描述登錄檔中的現有結構描述和版本進行查詢。

依預設，生產者快取已註冊結構描述的結構描述定義和結構描述版本 ID。如果記錄的結構描述版本定義不符合快取中的可用內容，生產者將嘗試使用結構描述登錄檔來驗證結構描述。如果結構描述版本有效，則其版本 ID 和定義將在生產者本機快取。

您可以在安裝 SerDe 程式庫的步驟 #3 的選用生產者屬性中調整預設快取期間 (24 小時)。

3. 序列化和傳遞記錄：如果記錄符合結構描述，序列化程式會使用結構描述版本 ID 來裝飾每個記錄，根據選取的資料格式序列化記錄 (AVRO、JSON、Protobuf 或即將推出的其他格式) 壓縮記錄 (選用生產者組態)，並將其傳遞給目的地。
4. 消費者還原序列化資料：讀取此資料的消費者使用結構描述登錄檔還原序列化程式庫，從記錄承載剖析結構描述版本 ID。
5. 還原序列化程式可能會從結構描述登錄檔要求結構描述：如果這是還原序列化程式第一次看到具有特定結構描述版本 ID 的記錄，則使用結構描述版本 ID，還原序列化程式會從結構描述登錄檔要求結構描述，並在本機快取消費者。如果結構描述登錄檔無法還原序列化記錄，消費者可以從記錄記錄中記錄資料，然後繼續，或停止應用程式。
6. 還原序列化程式使用結構描述來還原序列化記錄：當還原序列化程式從結構描述登錄檔擷取結構描述版本 ID 時，還原序列化程式會解壓縮記錄 (如果產生者傳送的記錄已壓縮)，並使用結構描述來還原序列化記錄。應用程式現在會處理記錄。

> **Note**
>
> 加密：您的用戶端透過 API 呼叫與結構描述登錄檔進行通訊，這些呼叫會使用透過 HTTPS 的 TLS 加密來加密傳輸中的資料。儲存在結構描述登錄檔中的結構描述一律使用服務管理的 AWS Key Management Service (AWS KMS) 金鑰來靜態加密。

> **Note**
>
> 使用者授權：結構描述登錄檔支援身分型 IAM 政策。

結構描述登錄檔入門

以下章節將提供概觀，並逐步引導您設定和使用結構描述登錄檔。如需結構描述登錄檔概念和要素的詳細資訊，請參閱AWS Glue 結構描述登錄檔。

主題

安裝 SerDe 程式庫

> **Note**
>
> 必要條件：在完成下列步驟前，您必須擁有執行中的 Amazon Managed Streaming for Apache Kafka (Amazon MSK) 或 Apache Kafka 叢集。您的生產者和消費者需要在 Java 8 或更高版本上執行。

這些 SerDe 庫提供了一個用於序列化和反序列化數據的框架。

您將為產生資料的應用程式安裝開源序列化程式 (統稱為「序列化程式」)。序列化程式會處理序列化、壓縮以及與結構描述登錄檔的互動。序列化程式會自動從寫入結構描述登錄檔相容目的地的記錄擷取結構描述，例如 Amazon MSK。同樣地，您將在使用資料的應用程式上安裝開源還原序列化程式。

若要在生產者和消費者上安裝程式庫：

1. 在生產者和消費者的 pom.xml 檔案中，透過下面的程式碼新增此相依性：

```
<dependency>
    <groupId>software.amazon.glue</groupId>
    <artifactId>schema-registry-serde</artifactId>
    <version>1.1.5</version>
</dependency>
```

   或者，您也可以複製 AWS Glue 結構描述登錄檔 Github 儲存庫。

2. 使用這些必要屬性設定您的生產者：

```
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
 StringSerializer.class.getName()); // Can replace StringSerializer.class.getName())
 with any other key serializer that you may use
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
 GlueSchemaRegistryKafkaSerializer.class.getName());
props.put(AWSSchemaRegistryConstants.AWS_REGION, "us-east-2");
properties.put(AWSSchemaRegistryConstants.DATA_FORMAT, "JSON"); // OR "AVRO"
```

   如果沒有現有的結構描述，則需要開啟自動註冊 (下一個步驟)。如果您確實有一個想套用的結構描述，那麼請用您的結構描述名稱替換「my-schema」。如果結構描述自動註冊關閉，則也必須提供「registry-name」。如果結構描述是在「default-registry」下建立的，則登錄檔名稱可以省略。

3. (選用) 設定這些選用生產者屬性中的任何一個。如需詳細的性質描述，請參閱 ReadMe 檔案。

```
props.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, "true"); // If
 not passed, uses "false"
props.put(AWSSchemaRegistryConstants.SCHEMA_NAME, "my-schema"); // If not passed,
 uses transport name (topic name in case of Kafka, or stream name in case of Kinesis
 Data Streams)
props.put(AWSSchemaRegistryConstants.REGISTRY_NAME, "my-registry"); // If not passed,
 uses "default-registry"
props.put(AWSSchemaRegistryConstants.CACHE_TIME_TO_LIVE_MILLIS, "86400000"); // If
 not passed, uses 86400000 (24 Hours)
props.put(AWSSchemaRegistryConstants.CACHE_SIZE, "10"); // default value is 200
props.put(AWSSchemaRegistryConstants.COMPATIBILITY_SETTING, Compatibility.FULL); //
 Pass a compatibility mode. If not passed, uses Compatibility.BACKWARD
props.put(AWSSchemaRegistryConstants.DESCRIPTION, "This registry is used for several
 purposes."); // If not passed, constructs a description
```

```
props.put(AWSSchemaRegistryConstants.COMPRESSION_TYPE,
 AWSSchemaRegistryConstants.COMPRESSION.ZLIB); // If not passed, records are sent
 uncompressed
```

自動註冊會在預設登錄檔（「default-registry」）下註冊結構描述版本。如果在上一個步驟中未指定 SCHEMA_NAME，則主題名稱會被推斷為 SCHEMA_NAME。

如需相容性模式的詳細資訊，請參閱結構描述版本控制和相容性。

4. 使用下列必要屬性設定您的消費者：

```
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
 StringDeserializer.class.getName());
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
 GlueSchemaRegistryKafkaDeserializer.class.getName());
props.put(AWSSchemaRegistryConstants.AWS_REGION, "us-east-2"); // Pass an AWS ##
props.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
 AvroRecordType.GENERIC_RECORD.getName()); // Only required for AVRO data format
```

5. (選用) 設定這些選用的消費者屬性。如需詳細的性質描述，請參閱 ReadMe 檔案。

```
properties.put(AWSSchemaRegistryConstants.CACHE_TIME_TO_LIVE_MILLIS, "86400000"); //
 If not passed, uses 86400000
props.put(AWSSchemaRegistryConstants.CACHE_SIZE, "10"); // default value is 200
props.put(AWSSchemaRegistryConstants.SECONDARY_DESERIALIZER,
 "com.amazonaws.services.schemaregistry.deserializers.external.ThirdPartyDeserializer"); //
 For migration fall back scenario
```

針對 AWS Glue 結構描述登錄檔 API 使用 AWS CLI

若要針對 AWS Glue 結構描述登錄檔 API 使用 AWS CLI，請務必將您的 AWS CLI 更新至最新版本。

建立登錄檔

您可以使用預設登錄檔，或使用 AWS Glue API 或 AWS Glue 主控台建立任意數量的新登錄檔。

AWS Glue API

您可以使用這些步驟，用 AWS Glue API 執行此任務。

若要新增登錄檔，請使用 CreateRegistry 動作（Python：創建註冊表）API。指定 RegistryName 作為要建立的登錄檔的名稱，最大長度為 255，只能包含字母、數字、連字號、底線、金額符號或井號。

將 a 指定`Description`為長度不超過 2048 個位元組的字串，符合 URI 位址多行字串樣式。

(選用) 為登錄檔指定一個或多個 `Tags`，作為索引鍵-值對的映射陣列。

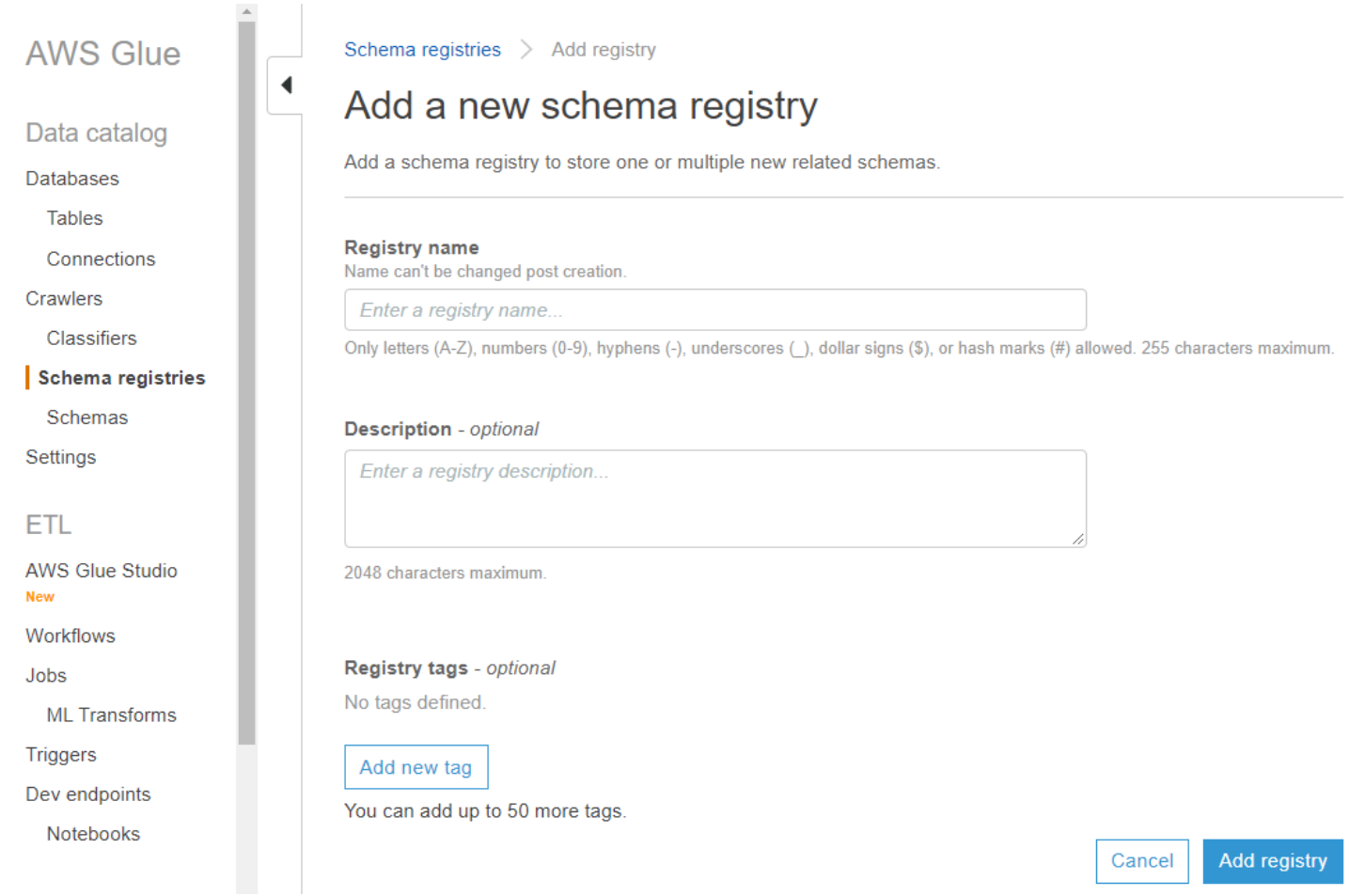
```
aws glue create-registry --registry-name registryName1 --description description
```

建立登錄檔後，系統會指派 Amazon Resource Name (ARN)，您可以在 API 回應的 `RegistryArn` 中檢視。現在您已建立登錄檔，請為該登錄檔建立一或多個結構描述。

AWS Glue 主控台

新增新的登錄檔AWS Glue主控台：

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格的 Data catalog 下，選擇 Schema registries (結構描述登錄檔)。
3. 選擇 Add registry (新增登錄檔)。
4. 輸入登錄名稱，由字母、數字、連字號或底線組成。無法變更此名稱。
5. 輸入登錄檔的 Description (描述) (選用)。
6. 或者，將一或多個標籤套用到您的登錄檔。選擇 Add new tag (新增標籤)，然後指定 Tag key (標籤鍵) 以及選用的 Tag value (標籤值)。
7. 選擇 Add registry (新增登錄檔)。



建立登錄檔後，系統會指派 Amazon Resource Name (ARN)，您可以在 Schema registries (結構描述登錄檔) 的清單中選擇登錄檔來查看。現在您已建立登錄檔，請為該登錄檔建立一或多個結構描述。

處理 JSON 的特定記錄 (JAVA POJO)

您可以使用純舊 Java 物件 (POJO) 並將該物件作為記錄傳遞。這類似於 AVRO 中的特定記錄的概念。mbknor-jackson-jsonschema可以為傳遞的 POJO 生成一個 JSON 模式。此程式庫也可以在 JSON 結構描述中注入其他資訊。

AWS Glue 結構描述登錄檔程式庫會使用注入的「className」欄位，在結構描述中提供完全分類的類別名稱。「className」欄位由還原序列化程式用於還原序列化為該類別的物件。

```
 Example class :

@JsonSchemaDescription("This is a car")
@JsonSchemaTitle("Simple Car Schema")
@Builder
@AllArgsConstructor
```

```
@EqualsAndHashCode
// Fully qualified class name to be added to an additionally injected property
// called className for deserializer to determine which class to deserialize
// the bytes into
@JsonSchemaInject(
        strings = {@JsonSchemaString(path = "className",
                value =
 "com.amazonaws.services.schemaregistry.integrationtests.generators.Car")}
)
// List of annotations to help infer JSON Schema are defined by https://github.com/
mbknor/mbknor-jackson-jsonSchema
public class Car {
    @JsonProperty(required = true)
    private String make;

    @JsonProperty(required = true)
    private String model;

    @JsonSchemaDefault("true")
    @JsonProperty
    public boolean used;

    @JsonSchemaInject(ints = {@JsonSchemaInt(path = "multipleOf", value = 1000)})
    @Max(200000)
    @JsonProperty
    private int miles;

    @Min(2000)
    @JsonProperty
    private int year;

    @JsonProperty
    private Date purchaseDate;

    @JsonProperty
    @JsonFormat(shape = JsonFormat.Shape.NUMBER)
    private Date listedDate;

    @JsonProperty
    private String[] owners;

    @JsonProperty
    private Collection<Float> serviceChecks;
```

```
    // Empty constructor is required by Jackson to deserialize bytes
    // into an Object of this class
    public Car() {}
}
```

建立結構描述

您可以使用 AWS Glue API 或 AWS Glue 主控台建立結構描述。

AWS Glue API

您可以使用這些步驟，用 AWS Glue API 執行此任務。

若要新增結構描述，請使用 CreateSchema 動作 (Python: 建立結構描述) API。

指定 `RegistryId` 結構來指出結構描述的登錄檔。或者，省略 `RegistryId` 以使用預設登錄檔。

指定由字母、數字、連字號或底線組成的 `SchemaName`，以及指定 `DataFormat` 為 **`AVRO`** 或 **`JSON`**。在結構描述上設定 `DataFormat` 後就不可改變。

指定 `Compatibility` 模式：

- 向後 (建議使用) — 消費者可以同時讀取目前版本和先前版本。
- 全部向後 — 消費者可以讀取目前版本和所有先前版本。
- 向前 — 消費者可以讀取目前和後續版本。
- 全部向前 — 消費者可以讀取目前版本和所有後續版本。
- 完整 — 向後和向前的組合。
- 完整全部 — 全部向後與全部向前的組合。
- 無 — 不會執行相容性檢查。
- 已停用 — 防止此結構描述的任何版本控制。

選用地指定結構描述的 `Tags`。

指定 `SchemaDefinition` 可定義 Avro、JSON 或 Protobuf 資料格式的結構描述。請參閱範例。

對於 Avro 資料格式：

```
aws glue create-schema --registry-id RegistryName="registryName1" --schema-name
 testschema --compatibility NONE --data-format AVRO --schema-definition "{\"type\":
```

```
 \"record\", \"name\": \"r1\", \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"},
 {\"name\": \"f2\", \"type\": \"string\"} ]}"
```

```
aws glue create-schema --registry-id RegistryArn="arn:aws:glue:us-
east-2:901234567890:registry/registryName1" --schema-name testschema --compatibility
 NONE --data-format AVRO  --schema-definition "{\"type\": \"record\", \"name\": \"r1\",
 \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"}, {\"name\": \"f2\", \"type\":
 \"string\"} ]}"
```

對於 JSON 資料格式：

```
aws glue create-schema --registry-id RegistryName="registryName" --schema-name
 testSchemaJson --compatibility NONE --data-format JSON --schema-definition "{\"$schema
\": \"http://json-schema.org/draft-07/schema#\",\"type\":\"object\",\"properties\":
{\"f1\":{\"type\":\"string\"}}}"
```

```
aws glue create-schema --registry-id RegistryArn="arn:aws:glue:us-
east-2:901234567890:registry/registryName" --schema-name testSchemaJson --compatibility
 NONE --data-format JSON --schema-definition "{\"$schema\": \"http://json-schema.org/
draft-07/schema#\",\"type\":\"object\",\"properties\":{\"f1\":{\"type\":\"string\"}}}"
```

對於 Protobuf 資料格式：

```
aws glue create-schema --registry-id RegistryName="registryName" --schema-name
 testSchemaProtobuf --compatibility NONE --data-format PROTOBUF --schema-definition
 "syntax = \"proto2\";package org.test;message Basic { optional int32 basic = 1;}"
```

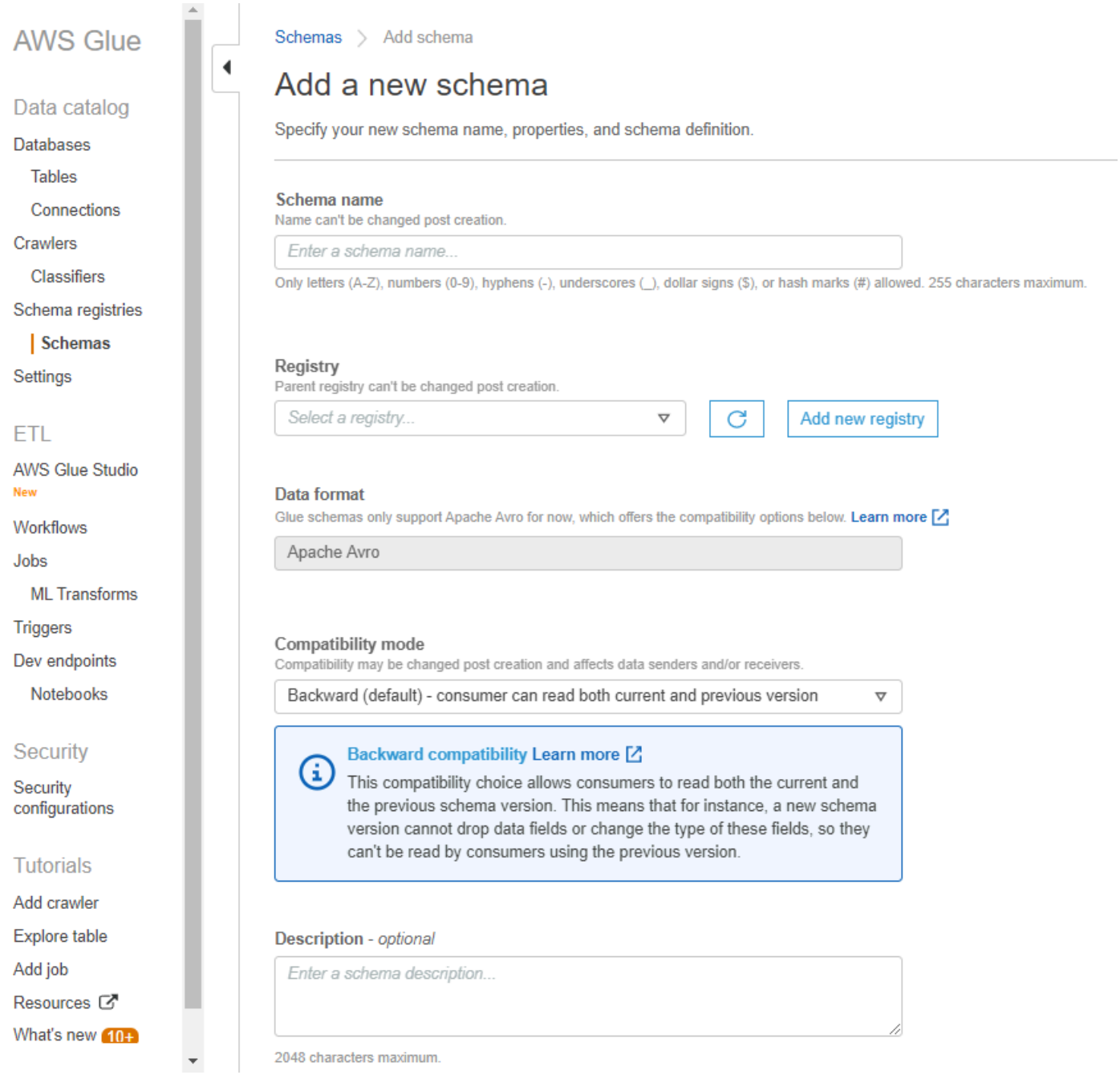
```
aws glue create-schema --registry-id RegistryArn="arn:aws:glue:us-
east-2:901234567890:registry/registryName" --schema-name testSchemaProtobuf
 --compatibility NONE --data-format PROTOBUF --schema-definition "syntax =
 \"proto2\";package org.test;message Basic { optional int32 basic = 1;}"
```

AWS Glue 主控台

若要使用 AWS Glue 主控台來新增結構描述：

1. 登入 AWS 管理主控台並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格中，於 Data catalog 下選擇 Schemas (Data Catalog) (結構描述 Data Catalog)。
3. 選擇 Add schema (新增結構描述)。

4. 輸入 Schema name (結構描述名稱)，由字母、數字、連字號、底線、金額符號或井號組成。無法變更此名稱。
5. 選擇 Registry (登錄檔)，其中結構描述會從下拉式選單儲存。建立後，無法變更父登錄檔。
6. 將 Data format (資料格式) 保留為 Apache Avro 或 JSON。此格式會套用至此結構描述的所有版本。
7. 選擇 Compatibility mode (相容性模式)。
   - 向後 (建議使用) — 接收者可以讀取目前和以前的版本。
   - 全部向後 — 接收者可以讀取目前和所有以前的版本。
   - 向前 — 傳送者可以寫入目前和先前的版本。
   - 全部向前 — 傳送者可以寫入目前和所有先前的版本。
   - 完整 — 向後和向前的組合。
   - 完整全部 — 全部向後與全部向前的組合。
   - 無 — 不執行相容性檢查。
   - 已停用 — 防止此結構描述的任何版本控制。
8. 為登錄檔輸入選用 Description (描述)，最多 250 個字元。

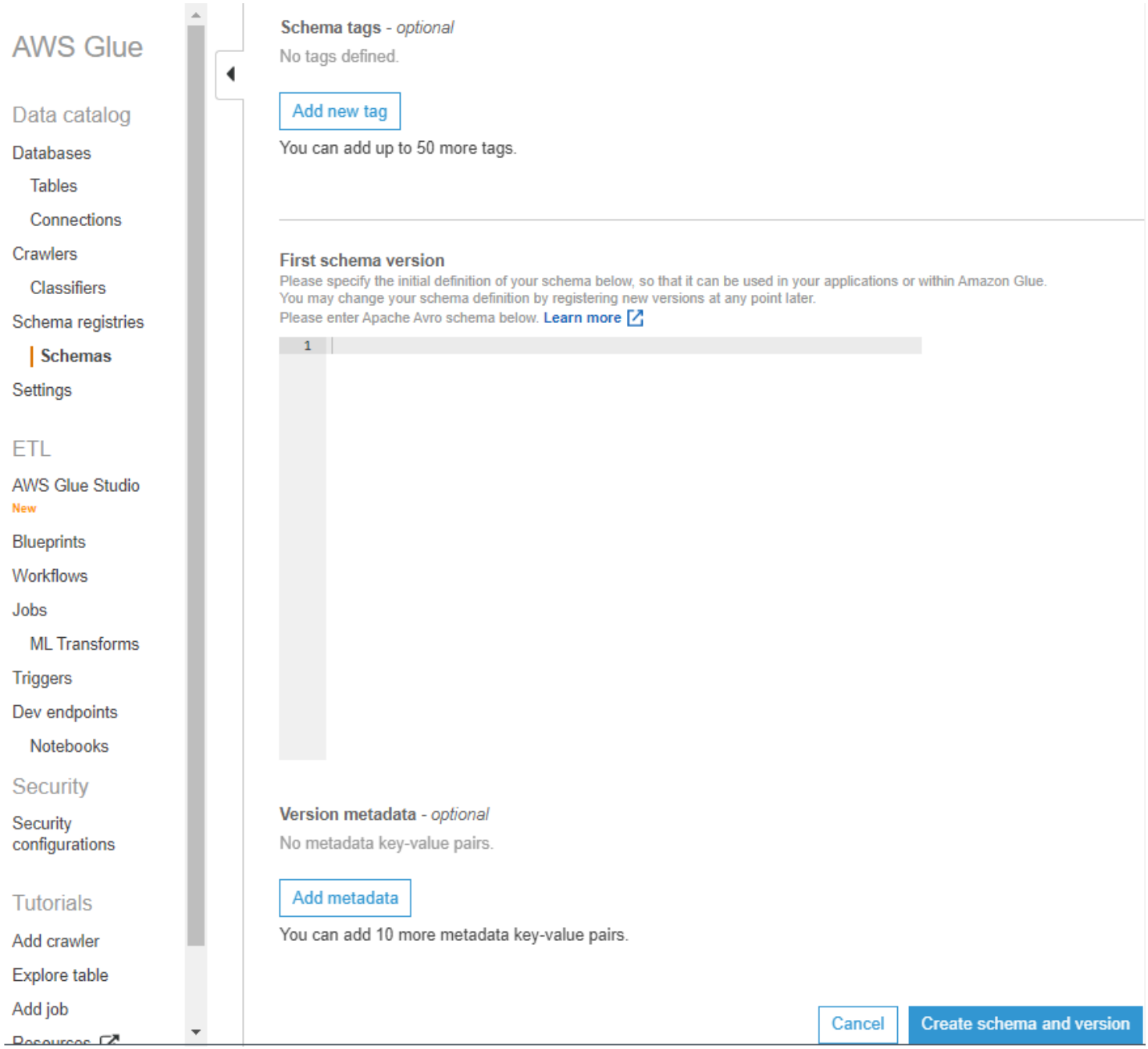


9. 或者，將一或多個標籤套用到您的結構描述。選擇 Add new tag (新增標籤)，然後指定 Tag key (標籤鍵) 以及選用的 Tag value (標籤值)。
10. 在 First schema version (第一個結構描述版本) 方塊中，輸入或貼上您的初始結構描述。

    如需 Avro 格式，請參閱使用 Avro 資料格式

    如需 JSON 格式的資訊，請參閱使用 JSON 資料格式
11. 選用地選擇 Add metadata (新增中繼資料) 以新增版本中繼資料來標註或分類您的結構描述版本。

12選擇 Create schema and version (建立結構描述和版本)。



結構描述隨即建立，並顯示在 Schemas (結構描述) 清單下。

使用 Avro 資料格式

Avro 提供資料序列化和資料交換服務。Avro 儲存在 JSON 格式的資料定義，使得它易於閱讀和解釋。資料本身會以二進位格式儲存。

如需定義 Apache Avro 結構描述的相關資訊，請參閱 Apache Avro 規格。

使用 JSON 資料格式

資料可以用 JSON 格式來序列化。JSON 結構描述格式定義了 JSON 結構描述格式的標準。

更新結構描述或登錄檔

建立後，您可以編輯結構描述、結構描述版本或登錄檔。

更新登錄檔

您可以使用 AWS Glue API 或 AWS Glue 主控台更新登錄檔。無法編輯現有登錄檔的名稱。您可以編輯登錄檔的描述。

AWS Glue API

若要更新現有的登錄檔，請使用 UpdateRegistry 行動（Python：更新註冊表）API。

指定 `RegistryId` 結構，以指出您要更新的登錄檔。傳遞 `Description` 以變更登錄檔的描述。

```
aws glue update-registry --description updatedDescription --registry-id
 RegistryArn="arn:aws:glue:us-east-2:901234567890:registry/registryName1"
```

AWS Glue 主控台

使用 AWS Glue 主控台更新登錄檔：

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格的 Data catalog 下，選擇 Schema registries (結構描述登錄檔)。
3. 從登錄檔清單中選擇登錄檔，方法是勾選其方塊。
4. 在 Actions (動作) 選單上，選擇 Edit registry (編輯登錄檔)。

更新結構描述

您可以更新結構描述的描述或相容性設定。

若要更新現有的結構描述，請使用 UpdateSchema 行動（Python：更新模式）API。

指定 `SchemaId` 結構描述，以指出您要更新的結構描述。必須提供其中一個 `VersionNumber` 或 `Compatibility`。

程式碼範例 11：

```
aws glue update-schema --description testDescription --schema-id
 SchemaName="testSchema1",RegistryName="registryName1" --schema-version-number
 LatestVersion=true --compatibility NONE
```

```
aws glue update-schema --description testDescription --schema-id
 SchemaArn="arn:aws:glue:us-east-2:901234567890:schema/registryName1/testSchema1" --
schema-version-number LatestVersion=true --compatibility NONE
```

新增結構描述版本

當您新增結構描述版本時，您需要比較版本以確保新結構描述可被接受。

若要將新版本新增到現有結構描述，請使用 RegisterSchemaVersion 行動（Python：註冊模式版本）API。

指定 SchemaId 結構來指示您想要新增版本的結構描述，以及 SchemaDefinition 來定義結構描述。

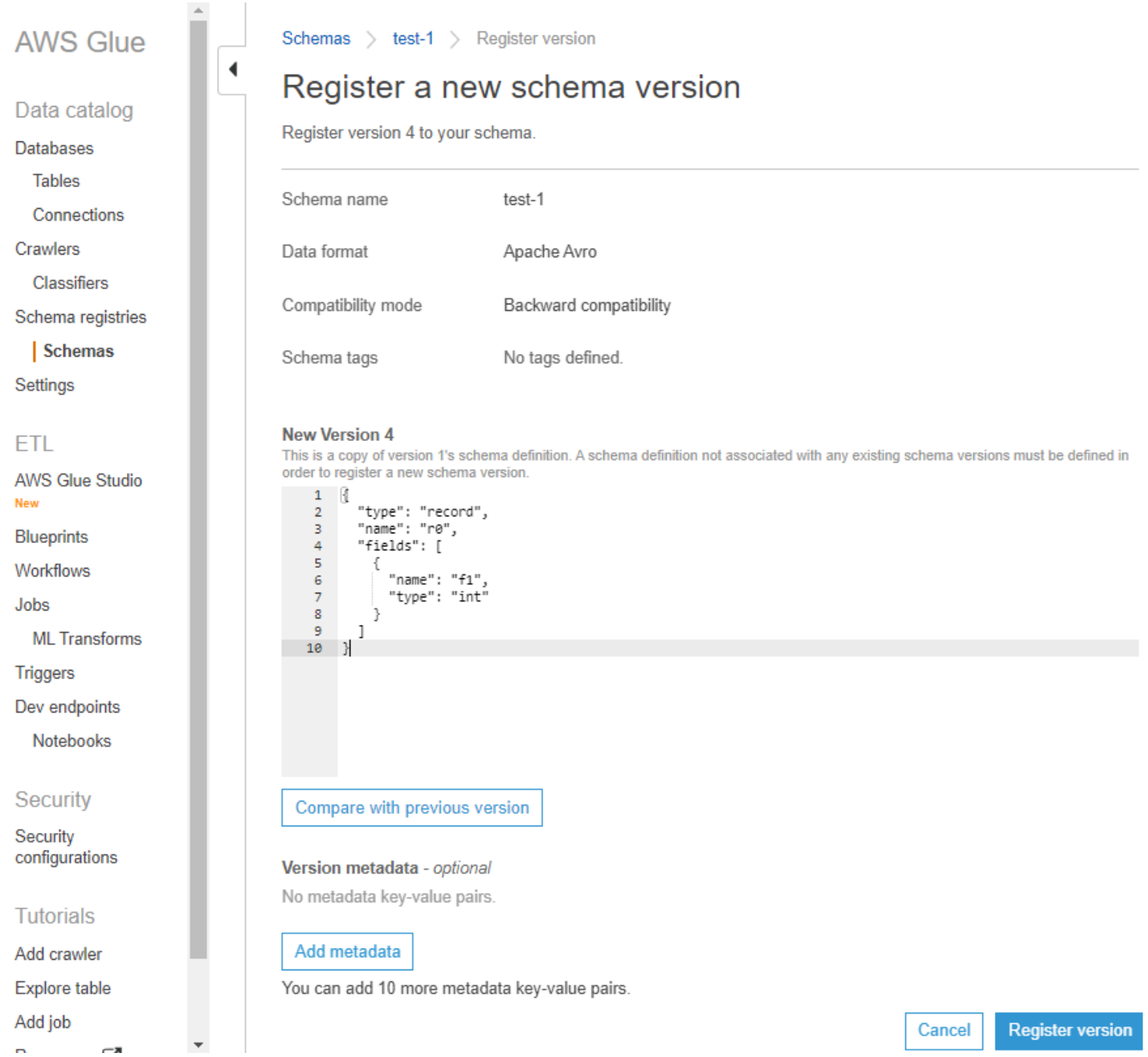
程式碼範例 12：

```
aws glue register-schema-version --schema-definition "{\"type\": \"record\", \"name\":
 \"r1\", \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"}, {\"name\": \"f2\", \"type
\": \"string\"} ]}" --schema-id SchemaArn="arn:aws:glue:us-east-1:901234567890:schema/
registryName/testschema"
```

```
aws glue register-schema-version --schema-definition "{\"type\": \"record\", \"name\":
 \"r1\", \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"}, {\"name\": \"f2\", \"type
\": \"string\"} ]}" --schema-id SchemaName="testschema",RegistryName="testregistry"
```

1. 登入 AWS Management Console，並前往 https://console.aws.amazon.com/glue/ 開啟 AWS Glue 主控台。
2. 在導覽窗格中，於 Data catalog 下選擇 Schemas (Data Catalog) (結構描述 Data Catalog)。
3. 從結構描述清單中選擇結構描述，方法是勾選其方塊。
4. 從清單中選擇一或多個結構描述，方法是勾選其方塊。
5. 在 Action (動作) 選單中，選擇 Register new version (註冊新版本)。

6. 在 New version (新版本) 方塊中，輸入或貼上新的結構描述。
7. 選擇 Compare with previous version (與舊版本比較) 以查看與以前的結構描述版本的差異。
8. 選用地選擇 Add metadata (新增中繼資料) 以新增版本中繼資料來標註或分類您的結構描述版本。輸入 Key (索引鍵) 和選用的 Value (值)。
9. 選擇 Register version (註冊版本)。



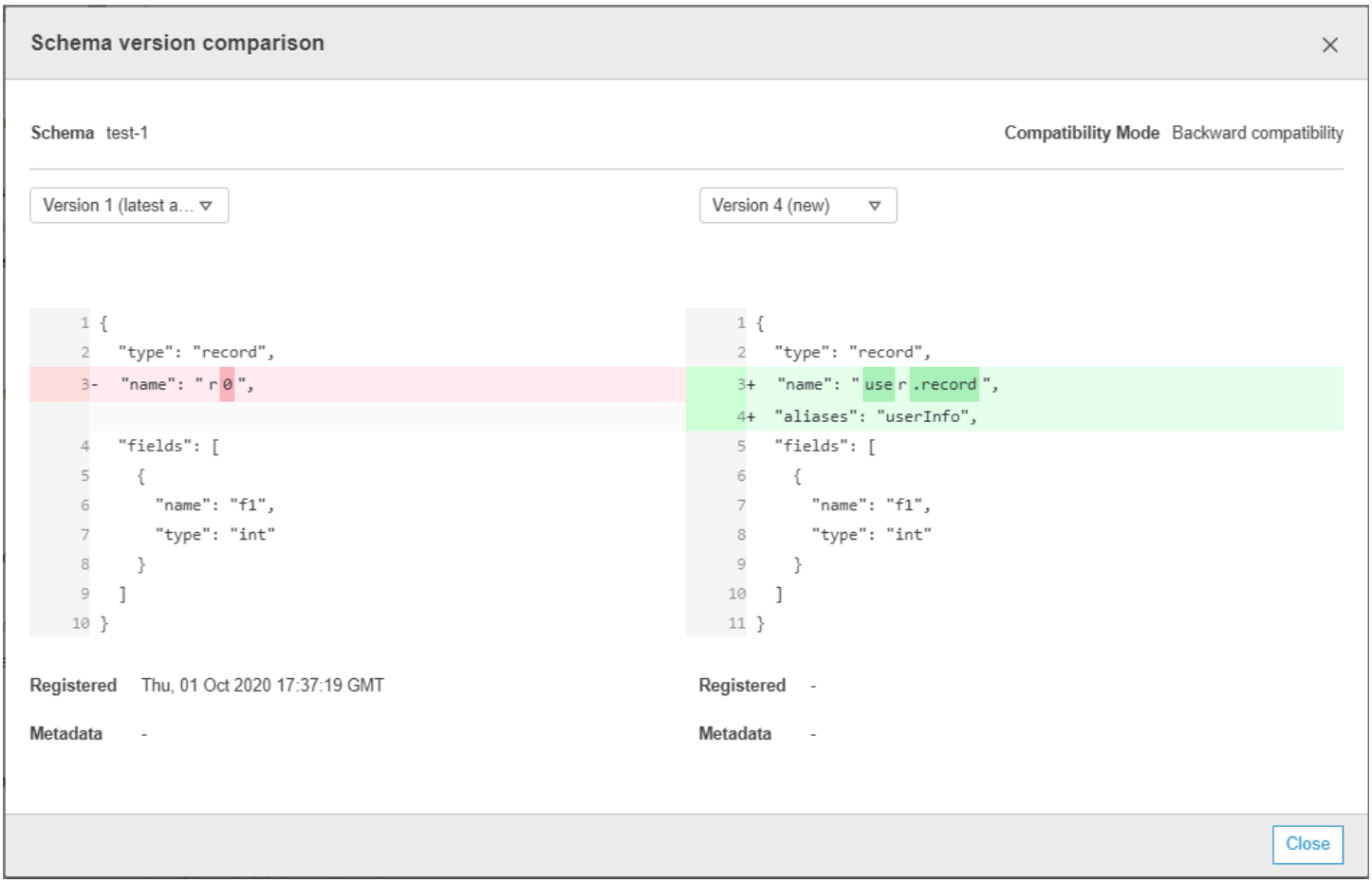
結構描述版本會顯示在版本清單中。如果版本變更了相容性模式，版本將被標記為檢查點。

結構描述版本比較的範例

當您選擇 Compare with previous version (與舊版本比較)，您會看到上一個版本和新版本一起顯示。變更的資訊將會反白顯示如下：

- 黃色：表示已變更的資訊。
- 綠色：表示在最新版本中新增的內容。
- 紅色：表示在最新版本中移除的內容。

您也可以與更舊版本進行比較。



刪除結構描述或登錄檔

刪除結構描述、結構描述版本或登錄檔是無法復原的永久動作。

刪除結構描述

當結構描述不再用於登錄檔中時，您可以使用AWS Management Console或 DeleteSchema 操作（Python：刪除模式） API 來刪除結構描述。

刪除一或多個結構描述是無法復原的永久動作。請確定已不再需要結構描述。

若要從登錄檔刪除結構描述，請呼叫 DeleteSchema 操作（Python：刪除模式）API, 指定 SchemaId 結構來識別結構描述。

例如：

```
aws glue delete-schema --schema-id SchemaArn="arn:aws:glue:us-
east-2:901234567890:schema/registryName1/schemaname"
```

```
aws glue delete-schema --schema-id SchemaName="TestSchema6-
deleteschemabyname",RegistryName="default-registry"
```

AWS Glue 主控台

從 AWS Glue 主控台刪除結構描述：

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格的 Data catalog 下，選擇 Schema registries (結構描述登錄檔)。
3. 從登錄檔清單中選擇包含結構描述的登錄檔。
4. 從清單中選擇一或多個結構描述，方法是勾選其方塊。
5. 在 Action (動作) 選單中，選擇 Delete schema (刪除結構描述)。
6. 在欄位中輸入文字 **`Delete`** 以確認刪除。
7. 選擇刪除。

您指定的結構描述會從登錄檔中刪除。

刪除結構描述版本

由於結構描述累積在登錄檔中，您可能想要使用AWS Management Console或 DeleteSchemaVersions 行動（Python：刪除模式版本）API 刪除不需要的結構描述版本。刪除一或多個結構描述版本是無法復原的永久動作。請確定不再需要結構描述版本。

刪除結構描述版本時，請注意以下限制：

- 您無法刪除檢查點的版本。
- 連續版本的範圍不能超過 25。

- 最新結構描述版本不可處於待定狀態。

指定 SchemaId 結構來識別結構描述，並指定 Versions 作為要刪除的版本範圍。如需指定版本或版本範圍的詳細資訊，請參閱DeleteRegistry 行動（Python：刪除註冊表）。您指定的結構描述版本會從登錄檔中刪除。

在此呼叫後呼叫 ListSchemaVersions 行動（Python：列表模式版本） API 將列出已刪除版本的狀態。

例如：

```
aws glue delete-schema-versions --schema-id
 SchemaName="TestSchema6",RegistryName="default-registry" --versions "1-1"
```

```
aws glue delete-schema-versions --schema-id SchemaArn="arn:aws:glue:us-
east-2:901234567890:schema/default-registry/TestSchema6-NON-Existent" --versions "1-1"
```

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格的 Data catalog 下，選擇 Schema registries (結構描述登錄檔)。
3. 從登錄檔清單中選擇包含結構描述的登錄檔。
4. 從清單中選擇一或多個結構描述，方法是勾選其方塊。
5. 在 Action (動作) 選單中，選擇 Delete schema (刪除結構描述)。
6. 在欄位中輸入文字 **`Delete`** 以確認刪除。
7. 選擇刪除。

您指定的結構描述版本會從登錄檔中刪除。

刪除登錄檔

當登錄檔包含的結構描述不應再組織在該登錄檔下時，您可能會想要刪除登錄檔。您將需要將這些結構描述重新指派到另一個登錄檔。

刪除一或多個登錄檔是無法復原的永久動作。請確定登錄檔不再需要。

預設登錄檔可以使用 AWS CLI 來刪除。

AWS Glue API

若要刪除整個登錄檔，包括結構描述及其所有版本，請呼叫 DeleteRegistry 行動（Python：刪除註冊表）API。指定 RegistryId 結構描述以識別登錄檔。

例如：

```
aws glue delete-registry --registry-id RegistryArn="arn:aws:glue:us-
east-2:901234567890:registry/registryName1"
```

```
aws glue delete-registry --registry-id RegistryName="TestRegistry-deletebyname"
```

若要取得刪除操作的狀態，您可以在非同步呼叫之後呼叫 GetRegistry API。

AWS Glue 主控台

若要從 AWS Glue 主控台刪除登錄檔：

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格的 Data catalog 下，選擇 Schema registries (結構描述登錄檔)。
3. 從清單中選擇登錄檔，方法是勾選其方塊。
4. 在 Actions (動作) 選單中，選擇 Delete registry (刪除登錄檔)。
5. 在欄位中輸入文字 **`Delete`** 以確認刪除。
6. 選擇刪除。

您選取的登錄檔會從 AWS Glue 刪除。

序列化程式的 IAM 範例

> **Note**
>
> AWS 受管政策授予常用案例所需的許可。如需使用受管政策來管理結構描述登錄檔的相關資訊，請參閱AWS 的管理 (預先定義) 策略 AWS Glue。

對於序列化程式，您應該建立類似於下面的最小政策，以便您能夠找到特定結構描述定義的 schemaVersionId。請注意，您應該有登錄檔的讀取許可，才能讀取登錄檔中的結構描述。您可以使用 Resource 子句，限制可以讀取的登錄檔。

程式碼範例 13：

```
{
    "Sid" : "GetSchemaByDefinition",
    "Effect" : "Allow",
    "Action" :
 [
        "glue:GetSchemaByDefinition"
    ],
        "Resource" : ["arn:aws:glue:us-east-2:012345678:registry/registryname-1",
                      "arn:aws:glue:us-east-2:012345678:schema/registryname-1/
schemaname-1",
                      "arn:aws:glue:us-east-2:012345678:schema/registryname-1/
schemaname-2"
                     ]
}
```

此外，您還可以允許生產者透過包括以下額外的方法來建立新的結構描述和版本。請注意，您應該能夠檢查登錄檔以新增/刪除/演變其中的結構描述。您可以使用 Resource 子句，限制可以檢查的登錄檔。

程式碼範例 14：

```
{
    "Sid" : "RegisterSchemaWithMetadata",
    "Effect" : "Allow",
    "Action" :
 [
        "glue:GetSchemaByDefinition",
        "glue:CreateSchema",
        "glue:RegisterSchemaVersion",
        "glue:PutSchemaVersionMetadata",
    ],
    "Resource" : ["arn:aws:glue:aws-region:123456789012:registry/registryname-1",
                  "arn:aws:glue:aws-region:123456789012:schema/registryname-1/
schemaname-1",
                  "arn:aws:glue:aws-region:123456789012:schema/registryname-1/
schemaname-2"
                 ]
}
```

還原序列化程式的 IAM 範例

對於還原序列化程式 (消費者方面)，您應該建立類似下面的政策，以允許還原序列化程式從結構描述登錄檔中擷取結構描述以進行還原序列化。請注意，您應該能夠檢查登錄檔以擷取其中的結構描述。

程式碼範例 15：

```
{
    "Sid" : "GetSchemaVersion",
    "Effect" : "Allow",
    "Action" :
 [
        "glue:GetSchemaVersion"
    ],
    "Resource" : ["*"]
}
```

使用 AWS PrivateLink 的私有連線

您可以使用 AWS PrivateLink 將資料生產者的 VPC 連接到 AWS Glue，方法是定義 AWS Glue 的界面 VPC 端點。使用 VPC 介面端點時，VPC 和 AWS Glue 之間的通訊完全在 AWS 網路中執行。如需詳細資訊，請參閱使用 AWS Glue 搭配 VPC 端點。

存取 Amazon CloudWatch 指標

Amazon CloudWatch 指標可作為免費方案 CloudWatch的一部分提供。您可以在「 CloudWatch 主控台」中存取這些指標。API 層級量度包括 CreateSchema (成功和延遲)、(成功和延遲) GetSchemaByDefinition、(成功和延遲)、 GetSchemaVersion (成功和延遲)、RegisterSchemaVersion (成功和延遲)、 PutSchemaVersionMetadata (成功和延遲)。資源層級度量包括登錄檔。 ThrottledByLimit， SchemaVersion。 ThrottledByLimit、 SchemaVersion。大小。

結構描述登錄檔的範例 AWS CloudFormation 範本

以下是在 AWS CloudFormation 中建立結構描述登錄檔資源的範例範本。若要在您的帳戶中建立此堆疊，請將上述範本複製到檔案 `SampleTemplate.yaml`，然後執行下列命令：

```
aws cloudformation create-stack --stack-name ABCSchemaRegistryStack --template-body
 "'cat SampleTemplate.yaml'"
```

此範例使用 AWS::Glue::Registry 建立登錄檔、使用 AWS::Glue::Schema 建立結構描述、使用 AWS::Glue::SchemaVersion 建立結構描述版本，以及使用 AWS::Glue::SchemaVersionMetadata 填入結構描述版本中繼資料。

```
Description: "A sample CloudFormation template for creating Schema Registry resources."
Resources:
  ABCRegistry:
    Type: "AWS::Glue::Registry"
    Properties:
      Name: "ABCSchemaRegistry"
      Description: "ABC Corp. Schema Registry"
      Tags:
        - Key: "Project"
          Value: "Foo"
  ABCSchema:
    Type: "AWS::Glue::Schema"
    Properties:
      Registry:
        Arn: !Ref ABCRegistry
      Name: "TestSchema"
      Compatibility: "NONE"
      DataFormat: "AVRO"
      SchemaDefinition: >
        {"namespace":"foo.avro","type":"record","name":"user","fields":
[{"name":"name","type":"string"},{"name":"favorite_number","type":"int"}]}
      Tags:
        - Key: "Project"
          Value: "Foo"
  SecondSchemaVersion:
    Type: "AWS::Glue::SchemaVersion"
    Properties:
      Schema:
        SchemaArn: !Ref ABCSchema
      SchemaDefinition: >
        {"namespace":"foo.avro","type":"record","name":"user","fields":
[{"name":"status","type":"string", "default":"ON"}, {"name":"name","type":"string"},
{"name":"favorite_number","type":"int"}]}
  FirstSchemaVersionMetadata:
    Type: "AWS::Glue::SchemaVersionMetadata"
    Properties:
      SchemaVersionId: !GetAtt ABCSchema.InitialSchemaVersionId
      Key: "Application"
      Value: "Kinesis"
```

```
  SecondSchemaVersionMetadata:
    Type: "AWS::Glue::SchemaVersionMetadata"
    Properties:
      SchemaVersionId: !Ref SecondSchemaVersion
      Key: "Application"
      Value: "Kinesis"
```

與 AWS Glue 結構描述登錄檔整合

這些章節說明與 AWS Glue 結構描述登錄檔的整合。這些章節中的範例顯示了 AVRO 資料格式的結構描述。如需更多範例，包括 JSON 資料格式的結構描述，請參閱結AWS Glue構描述登錄開放原始碼儲存庫中的整合測試和 ReadMe 資訊。

主題

- 使用案例：將結構描述登錄檔連線到 Amazon MSK 或 Apache Kafka
- 使用案例：將 Amazon Kinesis Data Streams 與 AWS Glue 結構描述登錄檔整合
- 使用案例：適用於阿帕奇 Flink 的 Amazon 託管服務
- 使用案例：與 AWS Lambda 整合
- 使用案例：AWS Glue Data Catalog
- 使用案例：AWS Glue 串流
- 使用案例：Apache Kafka Streams
- 使用案例：Apache Kafka Connect

使用案例：將結構描述登錄檔連線到 Amazon MSK 或 Apache Kafka

假設您正在將資料寫入 Apache Kafka 主題，並且您可以按照下列步驟開始使用。

1. 建立具有至少一個主題的 Amazon Managed Streaming for Apache Kafka (Amazon MSK) 或 Apache Kafka 叢集。如果是建立 Amazon MSK 叢集，您可以使用AWS Management Console。請遵循以下說明：Amazon Managed Streaming for Apache Kafka 開發人員指南中的 Amazon MSK 入門。
2. 遵循上述的安裝 SerDe 程式庫步驟。
3. 若要建立結構描述登錄檔、結構描述或結構描述版本，請依照這份文件的結構描述登錄檔入門一節指示進行。

4. 啟動您的生產者和消費者以使用結構描述登錄檔來從/向 Amazon MSK 或 Apache Kafka 主題寫入和讀取記錄。示例生產者和消費者代碼可以在 Serde 庫的 ReadMe 文件中找到。生產者上的結構描述登錄檔程式庫會自動序列化記錄，並使用結構描述版本 ID 裝飾記錄。
5. 如果已輸入此記錄的結構描述，或者如果開啟自動註冊，則結構描述將已在結構描述登錄檔中註冊。
6. 消費者使用 AWS Glue 結構描述登錄檔程式庫從 Amazon MSK 或 Apache Kafka 主題讀取，會自動從結構描述登錄檔查詢結構描述。

使用案例：將 Amazon Kinesis Data Streams 與 AWS Glue 結構描述登錄檔整合

此整合要求您擁有現有的 Amazon Kinesis 資料串流。如需詳細資訊，請參閱 Amazon Kinesis Data Streams 開發人員指南中的 Amazon Kinesis Data Streams 入門。

您可以透過兩種方式與 Kinesis 資料串流中的資料互動。

- 透過 Java 中的 Kinesis Producer Library (KPL) 和 Kinesis Client Library (KCL) 程式庫。不提供多語言支援。
- 透過 AWS SDK for Java 中提供的 `PutRecords`、`PutRecord` 以及 `GetRecords` Kinesis Data Streams API。

如果您目前使用 KPL/KCL 程式庫，我們建議您繼續使用該方法。有更新的 KCL 和 KPL 版本與結構描述登錄檔整合，如範例所示。否則，您可以使用範例程式碼來利用 AWS Glue 結構描述登錄檔 (如果直接使用 KDS API)。

結構描述登錄檔整合只適用於 KPL v0.14.2 或更高版本和 KCL v2.3 或更高版本。結構描述登錄檔與 JSON 資料格式整合適用於 KPL v0.14.8 或更高版本和 KCL v2.3.6 或更高版本。

使用 Kinesis SDK V2 與資料互動

本節說明如何使用 Kinesis SDK V2 與 Kinesis 互動

```
// Example JSON Record, you can construct a AVRO record also
private static final JsonDataWithSchema record =
 JsonDataWithSchema.builder(schemaString, payloadString);
private static final DataFormat dataFormat = DataFormat.JSON;

//Configurations for Schema Registry
GlueSchemaRegistryConfiguration gsrConfig = new GlueSchemaRegistryConfiguration("us-
east-1");
```

```
GlueSchemaRegistrySerializer glueSchemaRegistrySerializer =
        new GlueSchemaRegistrySerializerImpl(awsCredentialsProvider, gsrConfig);
GlueSchemaRegistryDataFormatSerializer dataFormatSerializer =
        new GlueSchemaRegistrySerializerFactory().getInstance(dataFormat, gsrConfig);

Schema gsrSchema =
        new Schema(dataFormatSerializer.getSchemaDefinition(record), dataFormat.name(),
 "MySchema");

byte[] serializedBytes = dataFormatSerializer.serialize(record);

byte[] gsrEncodedBytes = glueSchemaRegistrySerializer.encode(streamName, gsrSchema,
 serializedBytes);

PutRecordRequest putRecordRequest = PutRecordRequest.builder()
        .streamName(streamName)
        .partitionKey("partitionKey")
        .data(SdkBytes.fromByteArray(gsrEncodedBytes))
        .build();
shardId = kinesisClient.putRecord(putRecordRequest)
        .get()
        .shardId();

GlueSchemaRegistryDeserializer glueSchemaRegistryDeserializer = new
 GlueSchemaRegistryDeserializerImpl(awsCredentialsProvider, gsrConfig);

GlueSchemaRegistryDataFormatDeserializer gsrDataFormatDeserializer =
        glueSchemaRegistryDeserializerFactory.getInstance(dataFormat, gsrConfig);

GetShardIteratorRequest getShardIteratorRequest = GetShardIteratorRequest.builder()
        .streamName(streamName)
        .shardId(shardId)
        .shardIteratorType(ShardIteratorType.TRIM_HORIZON)
        .build();

String shardIterator = kinesisClient.getShardIterator(getShardIteratorRequest)
        .get()
        .shardIterator();

GetRecordsRequest getRecordRequest = GetRecordsRequest.builder()
        .shardIterator(shardIterator)
        .build();
GetRecordsResponse recordsResponse = kinesisClient.getRecords(getRecordRequest)
```

```
        .get();

List<Object> consumerRecords = new ArrayList<>();
List<Record> recordsFromKinesis = recordsResponse.records();

for (int i = 0; i < recordsFromKinesis.size(); i++) {
    byte[] consumedBytes = recordsFromKinesis.get(i)
            .data()
            .asByteArray();

    Schema gsrSchema = glueSchemaRegistryDeserializer.getSchema(consumedBytes);
    Object decodedRecord =
 gsrDataFormatDeserializer.deserialize(ByteBuffer.wrap(consumedBytes),

 gsrSchema.getSchemaDefinition());
    consumerRecords.add(decodedRecord);
}
```

使用 KPL/KCL 程式庫與資料互動

本節說明使用 KPL/KCL 程式庫將 Kinesis Data Streams 與結構描述登錄檔整合。如需使用 KPL/KCL 的詳細資訊，請參閱 Amazon Kinesis Data Streams 開發人員指南中的使用 Amazon Kinesis Producer Library 開發生產者。

在 KPL 中設定結構描述登錄檔

1. 定義 AWS Glue 結構描述登錄檔中撰寫的資料的結構描述定義、資料格式和結構描述名稱。
2. 選用地設定 `GlueSchemaRegistryConfiguration` 物件。
3. 將結構描述物件傳遞給 `addUserRecord` API。

```
private static final String SCHEMA_DEFINITION = "{"namespace": "example.avro",\n"
+ " "type": "record",\n"
+ " "name": "User",\n"
+ " "fields": [\n"
+ " {"name": "name", "type": "string"},\n"
+ " {"name": "favorite_number", "type": ["int", "null"]},\n"
+ " {"name": "favorite_color", "type": ["string", "null"]}\n"
+ " ]\n"
+ "}";

KinesisProducerConfiguration config = new KinesisProducerConfiguration();
config.setRegion("us-west-1")
```

```
//[Optional] configuration for Schema Registry.

GlueSchemaRegistryConfiguration schemaRegistryConfig =
new GlueSchemaRegistryConfiguration("us-west-1");

schemaRegistryConfig.setCompression(true);

config.setGlueSchemaRegistryConfiguration(schemaRegistryConfig);

///Optional configuration ends.

final KinesisProducer producer =
      new KinesisProducer(config);

final ByteBuffer data = getDataToSend();

com.amazonaws.services.schemaregistry.common.Schema gsrSchema =
    new Schema(SCHEMA_DEFINITION, DataFormat.AVRO.toString(), "demoSchema");

ListenableFuture<UserRecordResult> f = producer.addUserRecord(
config.getStreamName(), TIMESTAMP, Utils.randomExplicitHashKey(), data, gsrSchema);

private static ByteBuffer getDataToSend() {
      org.apache.avro.Schema avroSchema =
        new org.apache.avro.Schema.Parser().parse(SCHEMA_DEFINITION);

      GenericRecord user = new GenericData.Record(avroSchema);
      user.put("name", "Emily");
      user.put("favorite_number", 32);
      user.put("favorite_color", "green");

      ByteArrayOutputStream outBytes = new ByteArrayOutputStream();
      Encoder encoder = EncoderFactory.get().directBinaryEncoder(outBytes, null);
      new GenericDatumWriter<>(avroSchema).write(user, encoder);
      encoder.flush();
      return ByteBuffer.wrap(outBytes.toByteArray());
 }
```

設定 Kinesis client library

您將用 Java 開發 Kinesis Client Library 消費者。如需詳細資訊，請參閱 Amazon Kinesis Data Streams 開發人員指南中的以 Java 開發 Kinesis Client Library 消費者。

1. 傳遞 `GlueSchemaRegistryConfiguration` 物件，建立 `GlueSchemaRegistryDeserializer` 的執行個體。
2. 傳遞 `GlueSchemaRegistryDeserializer` 至 `retrievalConfig.glueSchemaRegistryDeserializer`。
3. 透過呼叫 `kinesisClientRecord.getSchema()` 存取內送訊息的結構描述。

```
GlueSchemaRegistryConfiguration schemaRegistryConfig =
    new GlueSchemaRegistryConfiguration(this.region.toString());

 GlueSchemaRegistryDeserializer glueSchemaRegistryDeserializer =
    new
 GlueSchemaRegistryDeserializerImpl(DefaultCredentialsProvider.builder().build(),
 schemaRegistryConfig);

 RetrievalConfig retrievalConfig =
 configsBuilder.retrievalConfig().retrievalSpecificConfig(new
 PollingConfig(streamName, kinesisClient));
 retrievalConfig.glueSchemaRegistryDeserializer(glueSchemaRegistryDeserializer);

  Scheduler scheduler = new Scheduler(
            configsBuilder.checkpointConfig(),
            configsBuilder.coordinatorConfig(),
            configsBuilder.leaseManagementConfig(),
            configsBuilder.lifecycleConfig(),
            configsBuilder.metricsConfig(),
            configsBuilder.processorConfig(),
            retrievalConfig
        );

 public void processRecords(ProcessRecordsInput processRecordsInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Processing {} record(s)",
                processRecordsInput.records().size());
                processRecordsInput.records()
                .forEach(
                    r ->
```

```
                        log.info("Processed record pk: {} -- Seq: {} : data {} with
schema: {}",
                        r.partitionKey(),
r.sequenceNumber(), recordToAvroObj(r).toString(), r.getSchema()));
            } catch (Throwable t) {
                log.error("Caught throwable while processing records. Aborting.");
                Runtime.getRuntime().halt(1);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
}

private GenericRecord recordToAvroObj(KinesisClientRecord r) {
   byte[] data = new byte[r.data().remaining()];
   r.data().get(data, 0, data.length);
   org.apache.avro.Schema schema = new
org.apache.avro.Schema.Parser().parse(r.schema().getSchemaDefinition());
   DatumReader datumReader = new GenericDatumReader<>(schema);

   BinaryDecoder binaryDecoder = DecoderFactory.get().binaryDecoder(data, 0,
data.length, null);
   return (GenericRecord) datumReader.read(null, binaryDecoder);
}
```

使用 Kinesis Data Streams API 與資料互動

本節說明使用 Kinesis Data Streams API 將 Kinesis Data Streams 與結構描述登錄檔整合。

1. 更新這些 Maven 相依性：

```
<dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>com.amazonaws</groupId>
                <artifactId>aws-java-sdk-bom</artifactId>
                <version>1.11.884</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
```

```
    <dependencies>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-kinesis</artifactId>
        </dependency>

        <dependency>
            <groupId>software.amazon.glue</groupId>
            <artifactId>schema-registry-serde</artifactId>
            <version>1.1.5</version>
        </dependency>

        <dependency>
            <groupId>com.fasterxml.jackson.dataformat</groupId>
            <artifactId>jackson-dataformat-cbor</artifactId>
            <version>2.11.3</version>
        </dependency>
    </dependencies>
```

2. 在生產者中，使用 Kinesis Data Streams 中的 PutRecords 或 PutRecord API 新增結構描述標頭資訊。

```
//The following lines add a Schema Header to the record
        com.amazonaws.services.schemaregistry.common.Schema awsSchema =
            new com.amazonaws.services.schemaregistry.common.Schema(schemaDefinition,
 DataFormat.AVRO.name(),
                schemaName);
        GlueSchemaRegistrySerializerImpl glueSchemaRegistrySerializer =
            new
 GlueSchemaRegistrySerializerImpl(DefaultCredentialsProvider.builder().build(), new
 GlueSchemaRegistryConfiguration(getConfigs()));
        byte[] recordWithSchemaHeader =
            glueSchemaRegistrySerializer.encode(streamName, awsSchema,
 recordAsBytes);
```

3. 在生產者中，使用 PutRecords 或 PutRecord API，將記錄放入資料串流中。
4. 在消費者中，從標頭移除結構描述記錄，並序列化 Avro 結構描述記錄。

```
//The following lines remove Schema Header from record
        GlueSchemaRegistryDeserializerImpl glueSchemaRegistryDeserializer =
            new
 GlueSchemaRegistryDeserializerImpl(DefaultCredentialsProvider.builder().build(),
 getConfigs());
```

```
        byte[] recordWithSchemaHeaderBytes = new
byte[recordWithSchemaHeader.remaining()];
        recordWithSchemaHeader.get(recordWithSchemaHeaderBytes, 0,
recordWithSchemaHeaderBytes.length);
        com.amazonaws.services.schemaregistry.common.Schema awsSchema =
            glueSchemaRegistryDeserializer.getSchema(recordWithSchemaHeaderBytes);
        byte[] record =
glueSchemaRegistryDeserializer.getData(recordWithSchemaHeaderBytes);

        //The following lines serialize an AVRO schema record
        if (DataFormat.AVRO.name().equals(awsSchema.getDataFormat())) {
            Schema avroSchema = new
org.apache.avro.Schema.Parser().parse(awsSchema.getSchemaDefinition());
            Object genericRecord = convertBytesToRecord(avroSchema, record);
            System.out.println(genericRecord);
        }
```

使用 Kinesis Data Streams API 與資料互動

以下是使用 PutRecords 和 GetRecords API 的範例程式碼。

```
//Full sample code
import
 com.amazonaws.services.schemaregistry.deserializers.GlueSchemaRegistryDeserializerImpl;
import
 com.amazonaws.services.schemaregistry.serializers.GlueSchemaRegistrySerializerImpl;
import com.amazonaws.services.schemaregistry.utils.AVROUtils;
import com.amazonaws.services.schemaregistry.utils.AWSSchemaRegistryConstants;
import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericDatumReader;
import org.apache.avro.generic.GenericDatumWriter;
import org.apache.avro.generic.GenericRecord;
import org.apache.avro.io.Decoder;
import org.apache.avro.io.DecoderFactory;
import org.apache.avro.io.Encoder;
import org.apache.avro.io.EncoderFactory;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.services.glue.model.DataFormat;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
```

```
import java.nio.ByteBuffer;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;


public class PutAndGetExampleWithEncodedData {
    static final String regionName = "us-east-2";
    static final String streamName = "testStream1";
    static final String schemaName = "User-Topic";
    static final String AVRO_USER_SCHEMA_FILE = "src/main/resources/user.avsc";
    KinesisApi kinesisApi = new KinesisApi();

    void runSampleForPutRecord() throws IOException {
        Object testRecord = getTestRecord();
        byte[] recordAsBytes = convertRecordToBytes(testRecord);
        String schemaDefinition =
 AVROUtils.getInstance().getSchemaDefinition(testRecord);

        //The following lines add a Schema Header to a record
        com.amazonaws.services.schemaregistry.common.Schema awsSchema =
            new com.amazonaws.services.schemaregistry.common.Schema(schemaDefinition,
 DataFormat.AVRO.name(),
                schemaName);
        GlueSchemaRegistrySerializerImpl glueSchemaRegistrySerializer =
            new
 GlueSchemaRegistrySerializerImpl(DefaultCredentialsProvider.builder().build(), new
 GlueSchemaRegistryConfiguration(regionName));
        byte[] recordWithSchemaHeader =
            glueSchemaRegistrySerializer.encode(streamName, awsSchema, recordAsBytes);

        //Use PutRecords api to pass a list of records
        kinesisApi.putRecords(Collections.singletonList(recordWithSchemaHeader),
 streamName, regionName);

        //OR
        //Use PutRecord api to pass single record
        //kinesisApi.putRecord(recordWithSchemaHeader, streamName, regionName);
    }

    byte[] runSampleForGetRecord() throws IOException {
        ByteBuffer recordWithSchemaHeader = kinesisApi.getRecords(streamName,
 regionName);
```

```
        //The following lines remove the schema registry header
        GlueSchemaRegistryDeserializerImpl glueSchemaRegistryDeserializer =
            new
GlueSchemaRegistryDeserializerImpl(DefaultCredentialsProvider.builder().build(), new
GlueSchemaRegistryConfiguration(regionName));
        byte[] recordWithSchemaHeaderBytes = new
byte[recordWithSchemaHeader.remaining()];
        recordWithSchemaHeader.get(recordWithSchemaHeaderBytes, 0,
recordWithSchemaHeaderBytes.length);

        com.amazonaws.services.schemaregistry.common.Schema awsSchema =
            glueSchemaRegistryDeserializer.getSchema(recordWithSchemaHeaderBytes);

        byte[] record =
glueSchemaRegistryDeserializer.getData(recordWithSchemaHeaderBytes);

        //The following lines serialize an AVRO schema record
        if (DataFormat.AVRO.name().equals(awsSchema.getDataFormat())) {
            Schema avroSchema = new
org.apache.avro.Schema.Parser().parse(awsSchema.getSchemaDefinition());
            Object genericRecord = convertBytesToRecord(avroSchema, record);
            System.out.println(genericRecord);
        }

        return record;
    }

    private byte[] convertRecordToBytes(final Object record) throws IOException {
        ByteArrayOutputStream recordAsBytes = new ByteArrayOutputStream();
        Encoder encoder = EncoderFactory.get().directBinaryEncoder(recordAsBytes,
null);
        GenericDatumWriter datumWriter = new
GenericDatumWriter<>(AVROUtils.getInstance().getSchema(record));
        datumWriter.write(record, encoder);
        encoder.flush();
        return recordAsBytes.toByteArray();
    }

    private GenericRecord convertBytesToRecord(Schema avroSchema, byte[] record) throws
IOException {
        final GenericDatumReader<GenericRecord> datumReader = new
GenericDatumReader<>(avroSchema);
        Decoder decoder = DecoderFactory.get().binaryDecoder(record, null);
        GenericRecord genericRecord = datumReader.read(null, decoder);
```

```
        return genericRecord;
    }

    private Map<String, String> getMetadata() {
        Map<String, String> metadata = new HashMap<>();
        metadata.put("event-source-1", "topic1");
        metadata.put("event-source-2", "topic2");
        metadata.put("event-source-3", "topic3");
        metadata.put("event-source-4", "topic4");
        metadata.put("event-source-5", "topic5");
        return metadata;
    }

    private GlueSchemaRegistryConfiguration getConfigs() {
        GlueSchemaRegistryConfiguration configs = new
 GlueSchemaRegistryConfiguration(regionName);
        configs.setSchemaName(schemaName);
        configs.setAutoRegistration(true);
        configs.setMetadata(getMetadata());
        return configs;
    }

    private Object getTestRecord() throws IOException {
        GenericRecord genericRecord;
        Schema.Parser parser = new Schema.Parser();
        Schema avroSchema = parser.parse(new File(AVRO_USER_SCHEMA_FILE));

        genericRecord = new GenericData.Record(avroSchema);
        genericRecord.put("name", "testName");
        genericRecord.put("favorite_number", 99);
        genericRecord.put("favorite_color", "red");

        return genericRecord;
    }
}
```

使用案例：適用於阿帕奇 Flink 的 Amazon 託管服務

Apache Flink 是一個流行的開源架構和分散式處理引擎，用於對未限制和有限制資料串流進行狀態計算。適用於 Apache Flink 的 Amazon 受管服務是一項全受管AWS服務，可讓您建立和管理 Apache Flink 應用程式以處理串流資料。

開源 Apache Flink 提供一些來源和接收器。例如，預先定義的資料來源包括從檔案、目錄和通訊端讀取，以及從集合和迭代器擷取資料。阿帕奇 Flink DataStream 連接器提供的代碼 Apache Flink 與各種第三方系統，如阿帕奇卡夫卡或 Kinesis 作為源和/或接收器接口。

如需詳細資訊，請參閱 Amazon Kinesis Data Analytics 開發人員指南。

Apache Flink Kafka 連接器

Apache Flink 提供 Apache Kafka 資料串流連接器，用於使用恰好一次的保證從 Kafka 主題讀取資料以及寫入資料至 Kafka 主題。Flink 的 Kafka 消費者 `FlinkKafkaConsumer` 提供從一個或多個 Kafka 主題的讀取存取權。Apache Flink 的 Kafka 生產者 `FlinkKafkaProducer` 允許寫入記錄串流到一個或多個 Kafka 主題。如需詳細資訊，請參閱 Apache Kafka 連接器。

Apache Flink Kinesis 串流連接器

Kinesis Data Streams 連接器可存取 Amazon Kinesis Data Streams。`FlinkKinesisConsumer` 是恰好一次的平行串流資料來源，可訂閱相同 AWS 服務區域中的多個 Kinesis 串流，並且可以在任務執行時透明地處理串流的重新分片。消費者的每個子工作都負責從多個 Kinesis 分片擷取資料記錄。每個子工作擷取的碎片數量將隨著碎片關閉並由 Kinesis 建立而改變。`FlinkKinesisProducer` 使用 Kinesis Producer Library (KPL) 將 Apache Flink 串流中的資料放入 Kinesis 串流中。如需詳細資訊，請參閱 Amazon Kinesis Streams Connector。

如需詳細資訊，請參閱 AWS Glue 結構描述 GitHub 儲存庫。

與 Apache Flink 整合

與架構註冊表一起提供的 SerDes 庫與 Apache Flink 集成在一起。若要使用 Apache Flink，您需要實作稱為 `GlueSchemaRegistryAvroSerializationSchema` 和 `GlueSchemaRegistryAvroDeserializationSchema` 的 `SerializationSchema` 和 `DeserializationSchema` 介面，然後將介面插入 Apache Flink 連接器。

新增 AWS Glue 結構描述登錄檔相依性到 Apache Flink 應用程式

在 Apache Flink 應用程式中將整合相依性設定為 AWS Glue 結構描述登錄檔：

1. 將相依性新增到 `pom.xml` 檔案。

```
<dependency>
    <groupId>software.amazon.glue</groupId>
    <artifactId>schema-registry-flink-serde</artifactId>
    <version>1.0.0</version>
```

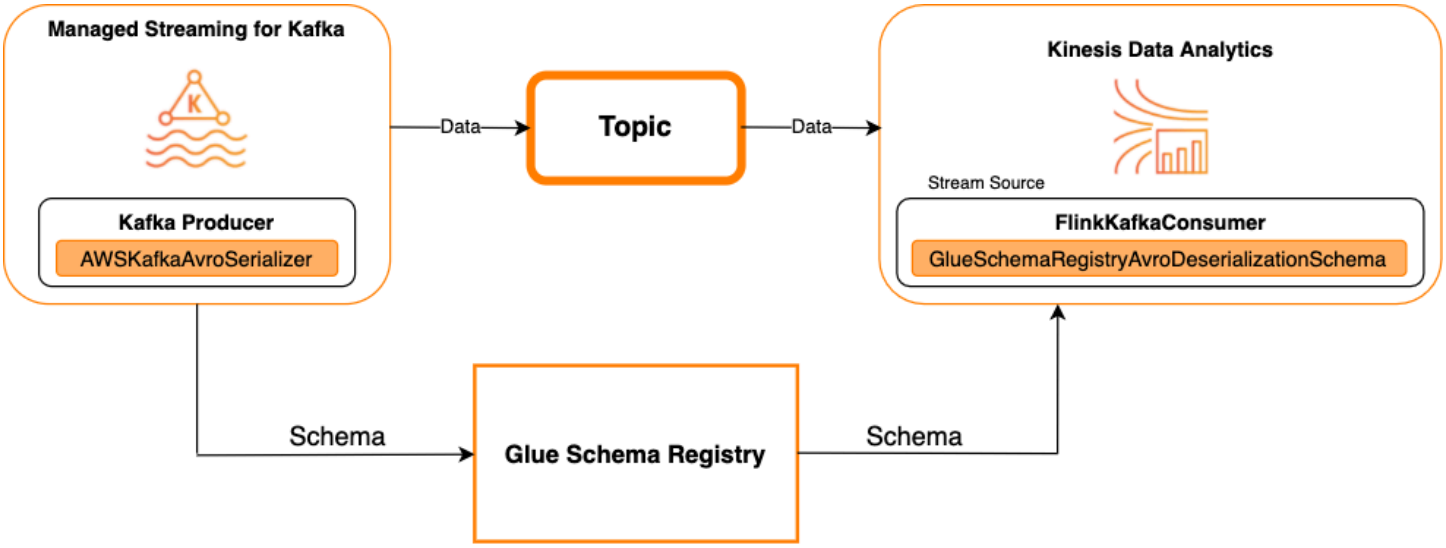
```
</dependency>
```

將 Kafka 或 Amazon MSK 與 Apache Flink 整合

您可以使用 Managed Service for Apache Flink，搭配 Kafka 作為來源或搭配 Kafka 作為接收器。

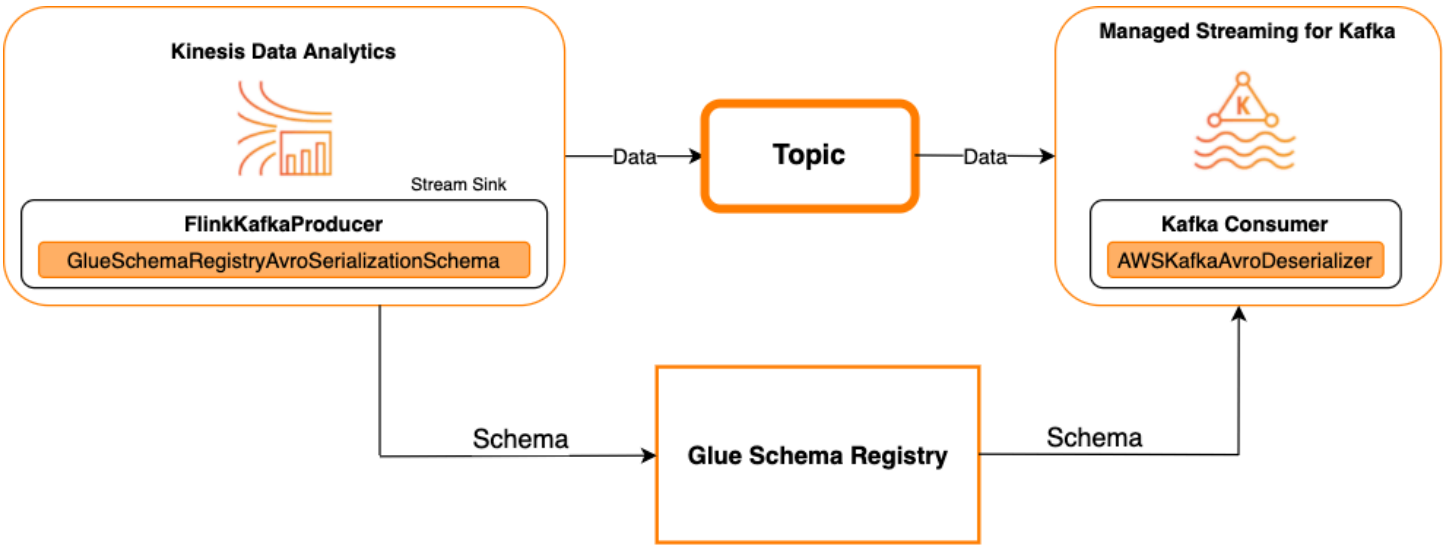
Kafka 作為來源

下圖顯示將 Kinesis Data Streams 與 Managed Service for Apache Flink 整合，並將 Kafka 作為來源。



Kafka 做為接收器

下圖顯示將 Kinesis Data Streams 與 Managed Service for Apache Flink 整合，並將 Kafka 作為接收器。



若要整合 Kafka (或 Amazon MSK) 與 Managed Service for Apache Flink，搭配 Kafka 作為來源或搭配 Kafka 作為接收器，請對程式碼進行下列變更。將粗體程式碼區塊新增到類似區段中的各自程式碼中。

如果 Kafka 是來源，請使用還原序列化程式程式碼 (區塊 2)。如果 Kafka 是接收器，請使用序列化程式程式碼 (區塊 3)。

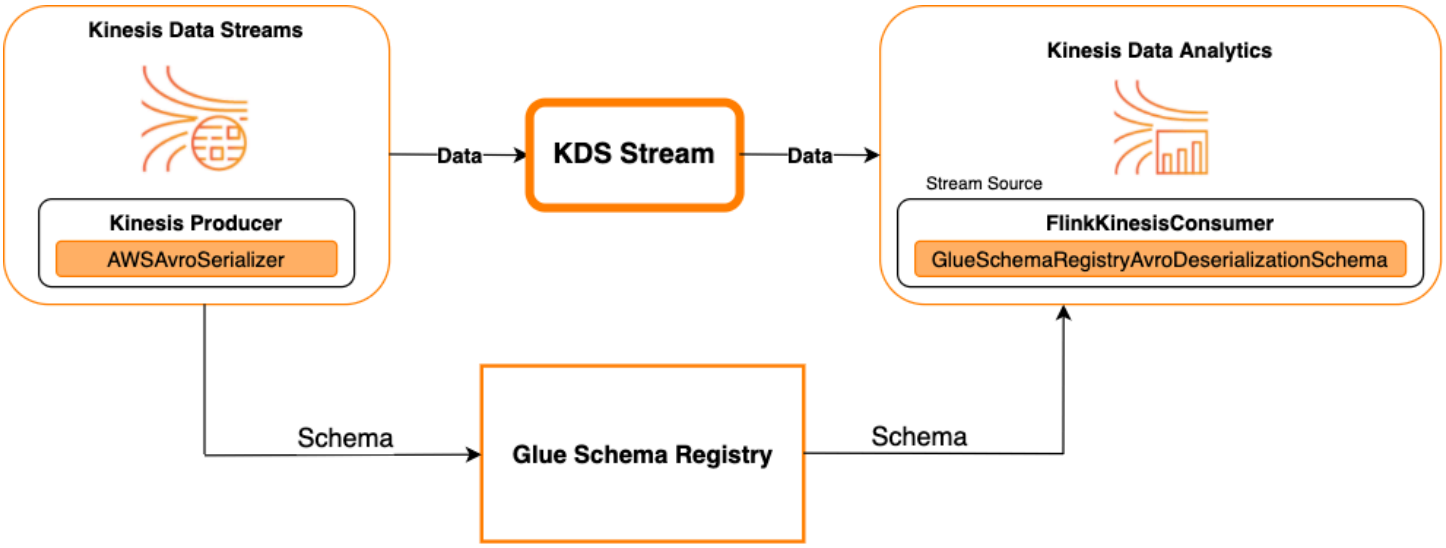
```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

String topic = "topic";
Properties properties = new Properties();
properties.setProperty("bootstrap.servers", "localhost:9092");
properties.setProperty("group.id", "test");

// block 1
Map<String, Object> configs = new HashMap<>();
configs.put(AWSSchemaRegistryConstants.AWS_REGION, "aws-region");
configs.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, true);
configs.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
 AvroRecordType.GENERIC_RECORD.getName());

FlinkKafkaConsumer<GenericRecord> consumer = new FlinkKafkaConsumer<>(
    topic,
    // block 2
    GlueSchemaRegistryAvroDeserializationSchema.forGeneric(schema, configs),
    properties);

FlinkKafkaProducer<GenericRecord> producer = new FlinkKafkaProducer<>(
    topic,
    // block 3
    GlueSchemaRegistryAvroSerializationSchema.forGeneric(schema, topic, configs),
    properties);

DataStream<GenericRecord> stream = env.addSource(consumer);
stream.addSink(producer);
env.execute();
```

將 Kinesis Data Streams 與 Apache Flink 整合

您可以使用 Managed Service for Apache Flink，搭配 Kinesis Data Streams 作為來源或接收器。

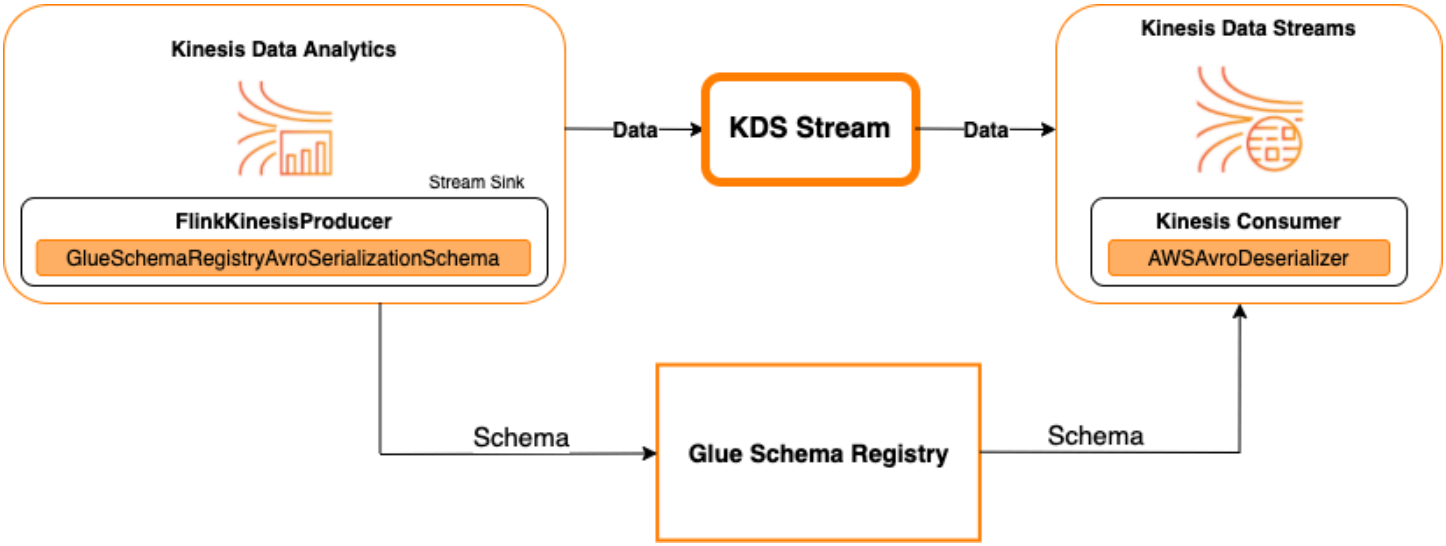
Kinesis Data Streams 做為來源

下圖顯示將 Kinesis Data Streams 與 Managed Service for Apache Flink 整合，並將 Kinesis Data Streams 作為來源。



Kinesis Data Streams 做為接收器

下圖顯示將 Kinesis Data Streams 與 Managed Service for Apache Flink 整合，並將 Kinesis Data Streams 作為接收器。



若要將 Kinesis Data Streams 與 Managed Service for Apache Flink 整合，並將 Kinesis Data Streams 作為來源或將 Kinesis Data Streams 作為接收器，請對程式碼進行下列變更。將粗體程式碼區塊新增到類似區段中的各自程式碼中。

如果 Kinesis Data Streams 是來源，請使用還原序列化程式程式碼 (區塊 2)。如果 Kinesis Data Streams 是接收器，請使用序列化程式程式碼 (區塊 3)。

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

String streamName = "stream";
Properties consumerConfig = new Properties();
consumerConfig.put(AWSConfigConstants.AWS_REGION, "aws-region");
consumerConfig.put(AWSConfigConstants.AWS_ACCESS_KEY_ID, "aws_access_key_id");
consumerConfig.put(AWSConfigConstants.AWS_SECRET_ACCESS_KEY, "aws_secret_access_key");
consumerConfig.put(ConsumerConfigConstants.STREAM_INITIAL_POSITION, "LATEST");

// block 1
Map<String, Object> configs = new HashMap<>();
configs.put(AWSSchemaRegistryConstants.AWS_REGION, "aws-region");
configs.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, true);
configs.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
 AvroRecordType.GENERIC_RECORD.getName());

FlinkKinesisConsumer<GenericRecord> consumer = new FlinkKinesisConsumer<>(
    streamName,
    // block 2
    GlueSchemaRegistryAvroDeserializationSchema.forGeneric(schema, configs),
    properties);

FlinkKinesisProducer<GenericRecord> producer = new FlinkKinesisProducer<>(
    // block 3
    GlueSchemaRegistryAvroSerializationSchema.forGeneric(schema, topic, configs),
    properties);
producer.setDefaultStream(streamName);
producer.setDefaultPartition("0");

DataStream<GenericRecord> stream = env.addSource(consumer);
stream.addSink(producer);
env.execute();
```

使用案例：與 AWS Lambda 整合

若要使用 AWS Lambda 函數作為 Apache Kafka/Amazon MSK 消費者，並使用 AWS Glue 結構描述登錄檔還原序列化 Avro 編碼訊息，請造訪 MSK Labs 頁面。

使用案例：AWS Glue Data Catalog

AWS Glue 資料表支援您可以手動指定的結構描述，也可以參考 AWS Glue 結構描述登錄檔。結構描述登錄檔與 Data Catalog 整合，可讓您在建立或更新 Data Catalog 中的 AWS Glue 資料表或分割區時

選用地使用儲存在結構描述登錄檔中的結構描述。若要識別結構描述登錄檔中的結構描述定義，您至少需要知道它所屬結構描述的 ARN。包含結構描述定義的結構描述版本，可以透過其 UUID 或版本號碼來參考。總有一個結構描述版本，即「最新」版本，可以在不知道其版本號碼或 UUID 的情況下查詢。

呼叫 `CreateTable` 或 `UpdateTable` 操作時，您將傳遞一個 `TableInput` 結構，其中包含 `StorageDescriptor`，它可能有一個新增到結構描述登錄檔中現有結構描述的 `SchemaReference`。同樣地，當您呼叫 `GetTable` 或 `GetPartition` API，回應可能包含結構描述和 `SchemaReference`。使用結構描述參考建立資料表或分割區時，Data Catalog 會嘗試擷取此結構描述參考的結構描述。如果它無法在結構描述登錄檔中找到結構描述，它會在 `GetTable` 回應中傳回空的結構描述；否則回應將同時具有結構描述和結構描述參考。

您可以在 AWS Glue 主控台中執行下列動作。

若要執行這些操作並建立、更新或檢視結構描述資訊，您必須向提供 `GetSchemaVersion` API 許可的呼叫使用者授予 IAM 角色。

新增資料表或更新資料表的結構描述

從現有的結構描述新增資料表，會將資料表繫結至特定的結構描述版本。註冊新的結構描述版本後，您可以從 AWS Glue 主控台 View table (檢視資料表) 頁面或使用 UpdateTable 行動（Python：更新表）API 更新此資料表定義。

從現有的結構描述新增資料表

您可以使用 AWS Glue 主控台或 `CreateTable` API，從登錄檔中的結構描述版本建立 AWS Glue 資料表。

AWS Glue API

呼叫 `CreateTable` API 時，您將傳遞包含 `StorageDescriptor` 的 `TableInput`，它對於結構描述登錄檔中現有的結構描述會有一個 `SchemaReference`。

AWS Glue 主控台

使用 AWS Glue 主控台建立資料表

1. 登入 AWS Management Console，並前往 https://console.aws.amazon.com/glue/ 開啟 AWS Glue 主控台。
2. 在導覽窗格中，於 Data catalog 下選擇 Tables (資料表)。
3. 在 Add Tables (新增資料表) 選單中，選擇 Add table from existing schema (從現有結構描述新增資料表)。

4. 根據《AWS Glue 開發人員指南》設定資料表屬性和資料存放區。
5. 在 Choose a Glue schema (選擇 Glue 結構描述) 頁面上，選取結構描述所在的 Registry (登錄檔)。
6. 選擇 Schema name (結構描述名稱)，然後選取要套用之結構描述的 Version (版本)。
7. 檢閱結構描述預覽，然後選擇 Next (下一步)。
8. 檢閱和建立資料表。

套用至資料表的結構描述和版本會顯示在資料表清單的 Glue schema (Glue 結構描述) 欄。您可以檢視資料表以查看更多詳細資訊。

更新資料表的結構描述

有新的結構描述版本可用時，您可能需要使用 UpdateTable 行動（Python：更新表）API 或 AWS Glue 主控台來更新資料表的結構描述。

> **Important**
>
> 更新具有手動指定之 AWS Glue 結構描述的現有資料表的結構描述時，結構描述登錄檔中參考的新結構描述可能不相容。這可能會導致您的任務失敗。

AWS Glue API

呼叫 `UpdateTable` API 時，您將傳遞包含 `StorageDescriptor` 的 `TableInput`，它對於結構描述登錄檔中現有的結構描述會有一個 `SchemaReference`。

AWS Glue 主控台

從 AWS Glue 主控台更新資料表的結構描述：

1. 登入 AWS Management Console，並前往 https://console.aws.amazon.com/glue/ 開啟 AWS Glue 主控台。
2. 在導覽窗格中，於 Data catalog 下選擇 Tables (資料表)。
3. 從資料表清單中檢視資料表。
4. 按一下 Update schema (更新結構描述)，通知您有關新版本的方塊。
5. 檢閱目前資料結構描述和新資料結構描述之間的差異。
6. 選擇 Show all schema differences (顯示所有結構描述差異) 以查看更多詳細資訊。

7. 選擇 Save table (儲存資料表) 以接受新的版本。

使用案例：AWS Glue 串流

AWS Glue 串流會消耗來自串流來源的資料，並在寫入輸出接收器之前執行 ETL 操作。輸入串流來源可以使用資料表來指定，也可以透過指定來源配置的方式直接指定。

在該結構描述存在於 AWS Glue 結構描述登錄檔的情況下，AWS Glue 串流支援針對串流來源所建立的 Data Catalog 資料表。您可以在 AWS Glue 結構描述登錄檔中建立結構描述，並使用此結構描述來建立具有串流來源的 AWS Glue 表格。此 AWS Glue 表格可以用來作為 AWS Glue 串流任務的輸入，以供還原序列化輸入串流中的資料之用。

請注意，當 AWS Glue 結構描述登錄檔中的結構描述變更時，您必須重新啟動 AWS Glue 串流任務需求以反映結構描述中的變更。

使用案例：Apache Kafka Streams

Apache Kafka Streams API 是用於處理和分析 Apache Kafka 中儲存資料的用戶端程式庫。本節介紹 Apache Kafka Streams 與 AWS Glue 結構描述登錄檔的整合，可讓您在資料串流應用程式上管理和強制執行結構描述。如需 Apache Kafka Streams 的詳細資訊，請參閱 Apache Kafka Streams。

與 SerDes 資料庫整合

有一個 `GlueSchemaRegistryKafkaStreamsSerde` 類別，您可用於設定串流應用程式。

Kafka Streams 應用程式範例程式碼

在 Apache Kafka Streams 應用程式中使用 AWS Glue 結構描述登錄檔：

1. 設定 Kafka Streams 應用程式。

```
final Properties props = new Properties();
    props.put(StreamsConfig.APPLICATION_ID_CONFIG, "avro-streams");
    props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
    props.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG, 0);
    props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
 Serdes.String().getClass().getName());
    props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
 AWSKafkaAvroSerDe.class.getName());
    props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
```

```
    props.put(AWSSchemaRegistryConstants.AWS_REGION, "aws-region");
    props.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, true);
    props.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
 AvroRecordType.GENERIC_RECORD.getName());
 props.put(AWSSchemaRegistryConstants.DATA_FORMAT, DataFormat.AVRO.name());
```

2. 從主題 avro-input 建立串流。

```
StreamsBuilder builder = new StreamsBuilder();
final KStream<String, GenericRecord> source = builder.stream("avro-input");
```

3. 處理資料記錄 (範例會篩選出 favorite_color 值為 pink 或 amount 值為 15 的記錄)。

```
final KStream<String, GenericRecord> result = source
    .filter((key, value) -
> !"pink".equals(String.valueOf(value.get("favorite_color"))));
    .filter((key, value) -> !"15.0".equals(String.valueOf(value.get("amount"))));
```

4. 將結果寫回主題 avro-output。

```
result.to("avro-output");
```

5. 啟動 Apache Kafka Streams 應用程式。

```
KafkaStreams streams = new KafkaStreams(builder.build(), props);
streams.start();
```

實作結果

這些結果顯示記錄篩選程序，其在步驟 3 中篩選出 favorite_color 為「pink」或值為「15.0」的記錄。

篩選前的記錄：

```
{"name": "Sansa", "favorite_number": 99, "favorite_color": "white"}
{"name": "Harry", "favorite_number": 10, "favorite_color": "black"}
```

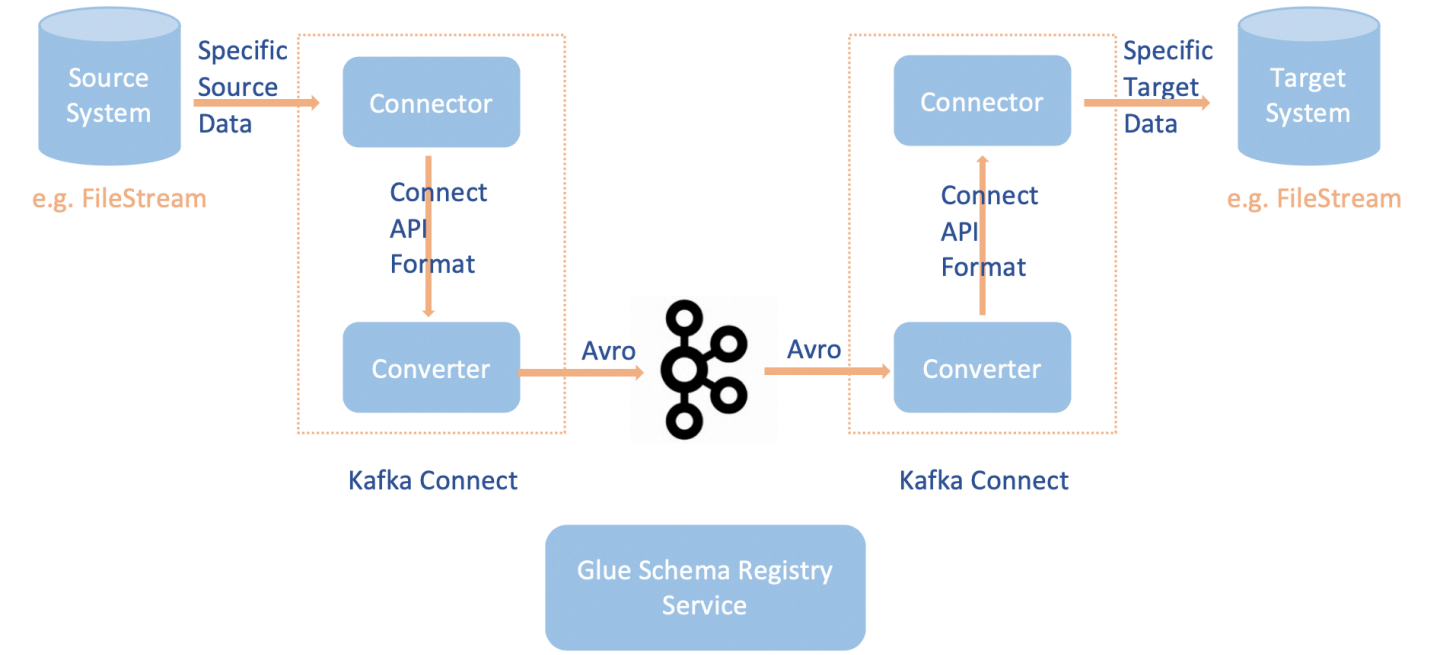
```
{"name": "Hermione", "favorite_number": 1, "favorite_color": "red"}
{"name": "Ron", "favorite_number": 0, "favorite_color": "pink"}
{"name": "Jay", "favorite_number": 0, "favorite_color": "pink"}

{"id": "commute_1","amount": 3.5}
{"id": "grocery_1","amount": 25.5}
{"id": "entertainment_1","amount": 19.2}
{"id": "entertainment_2","amount": 105}
 {"id": "commute_1","amount": 15}
```

篩選後的記錄：

```
{"name": "Sansa", "favorite_number": 99, "favorite_color": "white"}
{"name": "Harry", "favorite_number": 10, "favorite_color": "black"}
{"name": "Hermione", "favorite_number": 1, "favorite_color": "red"}
{"name": "Ron", "favorite_number": 0, "favorite_color": "pink"}

{"id": "commute_1","amount": 3.5}
{"id": "grocery_1","amount": 25.5}
{"id": "entertainment_1","amount": 19.2}
{"id": "entertainment_2","amount": 105}
```

使用案例：Apache Kafka Connect

將 Apache Kafka Connect 與 AWS Glue 結構描述登錄檔整合，可讓您從連接器取得結構描述資訊。Apache Kafka 轉換器指定 Apache Kafka 中的資料的格式，以及如何將其轉換成 Apache Kafka Connect 資料。每個 Apache Kafka Connect 使用者將需要根據他們希望他們的資料在載入或儲存到 Apache Kafka 時的格式，設定這些轉換器。透過這種方式，您可以定義自己的轉換器，將 Apache Kafka Connect 資料轉換為 AWS Glue 結構描述登錄檔中使用的類型 (例如：Avro) 並利用我們的序列化程式註冊其結構描述並進行序列化。然後轉換器也能夠使用我們的還原序列化程式將從 Apache Kafka 接收到的資料還原序列化，並將其轉換回 Apache Kafka Connect 資料。範例工作流程圖如下所示。



1. 安裝 aws-glue-schema-registry 專案，方法是複製 AWS Glue 結構描述登錄檔的 Github 儲存庫。

```
git clone git@github.com:awslabs/aws-glue-schema-registry.git
cd aws-glue-schema-registry
mvn clean install
mvn dependency:copy-dependencies
```

2. 如果您打算在獨立模式中使用 Apache Kafka Connect，請使用此步驟的下列指示來更新 connect-standalone.properties。如果您打算在分佈式模式下使用 Apache 卡夫卡 Connect，請使用相同的指令更新 connect-avro-distributed.properties。

   a. 將這些屬性也新增到 Apache Kafka 連接屬性檔案：

```
key.converter.region=aws-region
value.converter.region=aws-region
key.converter.schemaAutoRegistrationEnabled=true
value.converter.schemaAutoRegistrationEnabled=true
key.converter.avroRecordType=GENERIC_RECORD
value.converter.avroRecordType=GENERIC_RECORD
```

   b. 將下面的命令添加到 kafka-run-class.sh 下的啟動模式部分：

```
-cp $CLASSPATH:"<your AWS GlueSchema Registry base directory>/target/dependency/*"
```

3. 將下面的命令添加到 kafka-run-class. sh 下的啟動模式部分

```
-cp $CLASSPATH:"<your AWS GlueSchema Registry base directory>/target/dependency/*"
```

它應該如下所示：

```
# Launch mode
if [ "x$DAEMON_MODE" = "xtrue" ]; then
  nohup "$JAVA" $KAFKA_HEAP_OPTS $KAFKA_JVM_PERFORMANCE_OPTS $KAFKA_GC_LOG_OPTS
 $KAFKA_JMX_OPTS $KAFKA_LOG4J_OPTS -cp $CLASSPATH:"/Users/johndoe/aws-glue-schema-
registry/target/dependency/*" $KAFKA_OPTS "$@" > "$CONSOLE_OUTPUT_FILE" 2>&1 < /dev/
null &
else
  exec "$JAVA" $KAFKA_HEAP_OPTS $KAFKA_JVM_PERFORMANCE_OPTS $KAFKA_GC_LOG_OPTS
 $KAFKA_JMX_OPTS $KAFKA_LOG4J_OPTS -cp $CLASSPATH:"/Users/johndoe/aws-glue-schema-
registry/target/dependency/*" $KAFKA_OPTS "$@"
fi
```

4. 如果使用 bash，執行下面的命令來在您的 bash_profile 中設定您的 CLASSPATH。對於任何其他 Shell，請相應地更新環境。

```
echo 'export GSR_LIB_BASE_DIR=<>' >>~/.bash_profile
echo 'export GSR_LIB_VERSION=1.0.0' >>~/.bash_profile
echo 'export KAFKA_HOME=<your Apache Kafka installation directory>' >>~/.bash_profile
echo 'export CLASSPATH=$CLASSPATH:$GSR_LIB_BASE_DIR/avro-kafkaconnect-converter/
target/schema-registry-kafkaconnect-converter-$GSR_LIB_VERSION.jar:$GSR_LIB_BASE_DIR/
common/target/schema-registry-common-$GSR_LIB_VERSION.jar:$GSR_LIB_BASE_DIR/
avro-serializer-deserializer/target/schema-registry-serde-$GSR_LIB_VERSION.jar'
 >>~/.bash_profile
source ~/.bash_profile
```

5. (選用) 如果您想要使用簡單的檔案來源進行測試，請複製檔案來源連接器。

```
git clone https://github.com/mmolimar/kafka-connect-fs.git
cd kafka-connect-fs/
```

a. 在來源連接器組態下，將資料格式編輯為 Avro、檔案讀取器編輯為 AvroFileReader 並從您正在讀取的檔案路徑中更新範例 Avro 物件。例如：

```
vim config/kafka-connect-fs.properties
```

```
fs.uris=<path to a sample avro object>
policy.regexp=^.*\.avro$
file_reader.class=com.github.mmolimar.kafka.connect.fs.file.reader.AvroFileReader
```

b. 安裝來源連接器。

```
mvn clean package
echo "export CLASSPATH=\$CLASSPATH:\"\$(find target/ -type f -name '*.jar'| grep
 '\-package' | tr '\n' ':')\"" >>~/.bash_profile
source ~/.bash_profile
```

c. 更新 *<your Apache Kafka installation directory>*/config/connect-file-sink.properties 下的接收器屬性會更新主題名稱和輸出檔案名稱。

```
file=<output file full path>
topics=<my topic>
```

6. 啟動來源連接器 (在此範例中，它是檔案來源連接器)。

```
$KAFKA_HOME/bin/connect-standalone.sh $KAFKA_HOME/config/connect-
standalone.properties config/kafka-connect-fs.properties
```

7. 執行接收器連接器 (在此範例中，它是檔案接收器連接器)。

```
$KAFKA_HOME/bin/connect-standalone.sh $KAFKA_HOME/config/connect-
standalone.properties $KAFKA_HOME/config/connect-file-sink.properties
```

對於一個例子卡夫卡 Connect 用法，看看在 Github 存儲庫的架構註冊表集成測試文件夾下的 run-local-tests .sh 腳本。AWS Glue

從第三方結構描述登錄檔遷移至 AWS Glue 結構描述登錄檔

從第三方結構描述登錄檔遷移至 AWS Glue 結構描述登錄檔，對現有、目前的第三方結構描述登錄檔具有相依性。如果在使用第三方結構描述登錄檔傳送的 Apache Kafka 主題中有記錄，則消費者需要第

三方結構描述登錄檔來還原序列化這些記錄。AWSKafkaAvroDeserializer 提供指定次要還原序列化程式類別的能力，此類別指向第三方還原序列化程式並用於還原序列化這些記錄。

淘汰第三方結構描述有兩個條件。首先，只有在使用第三方結構描述登錄檔的 Apache Kafka 主題中的記錄不再被任何消費者需要後，才會發生淘汰。其次，取決於針對這些主題指定的保留期間，Apache Kafka 主題的老化可能會導致淘汰。請注意，如果您有無限保留的主題，您仍然可以遷移至 AWS Glue 結構描述登錄檔，但您將無法淘汰第三方結構描述登錄檔。因應措施是，您可以使用應用程式或 Mirror Maker 2 讀取目前主題，並使用 AWS Glue 結構描述登錄檔產生新主題。

從第三方結構描述登錄檔遷移至 AWS Glue 結構描述登錄檔：

1. 建立 AWS Glue 結構描述登錄檔中的登錄檔，或使用預設登錄檔。
2. 停止消費者。修改它以包含 AWS Glue 結構描述登錄檔做為主要的還原序列化程式，並包含第三方結構描述登錄檔做為次要。
   - 設定消費者屬性。在此範例中，secondary_deserializer 設定為不同的還原序列化程式。行為如下：消費者從 Amazon MSK 擷取記錄，並首先嘗試使用 AWSKafkaAvroDeserializer。如果無法為 AWS Glue 結構描述登錄檔讀取包含 Avro 結構描述 ID 的魔術位元組結構描述，則 AWSKafkaAvroDeserializer 會嘗試使用 secondary_deserializer 中提供的還原序列化程式類別。次要還原序列化程式專屬的屬性也需要在消費者屬性中提供，例如 schema_registry_url_config 和 specific_avro_reader_config，如下所示。

```
consumerProps.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
 StringDeserializer.class.getName());
consumerProps.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
 AWSKafkaAvroDeserializer.class.getName());
consumerProps.setProperty(AWSSchemaRegistryConstants.AWS_REGION,
 KafkaClickstreamConsumer.gsrRegion);
consumerProps.setProperty(AWSSchemaRegistryConstants.SECONDARY_DESERIALIZER,
 KafkaAvroDeserializer.class.getName());
consumerProps.setProperty(KafkaAvroDeserializerConfig.SCHEMA_REGISTRY_URL_CONFIG,
 "URL for third-party schema registry");
consumerProps.setProperty(KafkaAvroDeserializerConfig.SPECIFIC_AVRO_READER_CONFIG,
 "true");
```

3. 重新啟動消費者。
4. 停止生產者並將生產者指向 AWS Glue 結構描述登錄檔。
   a. 設定生產者屬性。在這個範例中，生產者將使用 default-registry 和自動註冊結構描述版本。

```
producerProps.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
 StringSerializer.class.getName());
producerProps.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
 AWSKafkaAvroSerializer.class.getName());
producerProps.setProperty(AWSSchemaRegistryConstants.AWS_REGION, "us-east-2");
producerProps.setProperty(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
 AvroRecordType.SPECIFIC_RECORD.getName());
producerProps.setProperty(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING,
 "true");
```

5. (選用) 手動將現有的結構描述和結構描述版本從目前的第三方結構描述登錄檔移至 AWS Glue 結構描述登錄檔，無論是 AWS Glue 結構描述登錄檔中的預設登錄檔或 AWS Glue 結構描述登錄檔中的特定非預設登錄檔。這可以透過以 JSON 格式從第三方結構描述登錄檔匯出結構描述並使用AWS Management Console或 AWS CLI 在 AWS Glue 結構描述登錄檔中建立新結構描述來完成。

   如果您需要使用 AWS CLI 和AWS Management Console針對新建立的結構描述版本啟用與先前結構描述版本的相容性檢查，或者當生產者使用開啟結構描述版本的自動註冊的新結構描述傳送訊息時，這個步驟可能很重要。
6. 啟動生產者。

連線至資料

AWS Glue連線是資料目錄物件，可儲存特定資料存放區的登入認證、URI 字串、虛擬私人雲端 (VPC) 資訊等。 AWS Glue 編目器、工作和開發端點會使用連線來存取特定類型的資料存放區。您可以針對來源和目標使用連線，並在多個爬蟲程式或擷取、轉換和載入 (ETL) 任務中重複使用相同的連線。

AWS Glue 支援下列連線類型：

- Amazon DocumentDB
- Amazon OpenSearch 服務，與星火一起 AWS Glue 使用。
- Amazon Redshift
- Azure 宇宙，適用於具有 AWS Glue ETL 工作的 NoSQL 適用的 Azure 宇宙資料庫
- 天青 SQL，與 AWS Glue 星火使用。
- 谷歌 BigQuery，與 AWS Glue 星火使用。
- JDBC
- Kafka
- MongoDB
- MongoDB Atlas
- Salesforce
- SAP HANA，與火花一起 AWS Glue 使用。
- 雪花，用 AWS Glue 於火花。
- 太數據有利，使用 AWS Glue 火花時。
- Vertica，與火花一起 AWS Glue 使用。
- 各種 Amazon Relational Database Service (Amazon RDS) 方案。
- 網路 (將連線指定至 Amazon Virtual Private Cloud (Amazon VPC) 內的資料來源)
- Aurora (如果正在使用原生 JDBC 驅動程式則支援。並非所有驅動程式功能都可用)

您也可以使用 AWS Glue Studio 為連接器建立連線。連接器是一個選用程式碼套件，可以協助存取AWS Glue Studio中的資料存放區。如需詳細資訊，請參閱搭配 AWS Glue Studio 使用連接器和連線

如需如何連線到內部部署資料庫的詳細資訊，請參閱如何使用 AWS GlueAWS Big Data Blog 網站存取和分析內部部署資料存放區。

本節包括可協助您使用 AWS Glue 連線的下列主題：

- AWS Glue 連線屬性
- 在 AWS Secrets Manager 中存放連線憑證
- 新增 AWS Glue 連線
- 測試 AWS Glue 連線
- 設定 AWS 呼叫通過您的 VPC
- 在 VPC 連線至 JDBC 資料存放區
- 使用 MongoDB 或 MongoDB Atlas 連線
- 使用 VPC 端點網路爬取 Amazon S3 資料存放區
- 針對 AWS Glue 中的連線問題進行故障診斷
- 教學課程：使用 AWS Glue Elasticsearch Connector

AWS Glue 連線屬性

本主題包括 AWS Glue 連線屬性的相關資訊。

主題

- 所需連線屬性
- AWS Glue JDBC 連線屬性
- AWS Glue MongoDB 和 MongoDB Atlas 連線屬性
- 銷售力量連線內容
- Snowflake 連線
- Vertica 連線
- SAP HANA 連線
- Azure SQL 連線
- Teradata Vantage 連線
- OpenSearch 服務連接
- Azure Cosmos 連線
- AWS Glue SSL 連線屬性
- 用於用戶端驗證的 Apache Kafka 連線屬性

- 谷歌 BigQuery 連接
- Vertica 連線

所需連線屬性

在 AWS Glue 主控台定義連線時，您必須為下列屬性提供值：

連線名稱

輸入您的連線的不重複名稱。

連線類型

選擇 JDBC 或其中一種特定的連線類型。

如需有關 JDBC 連線類型的詳細資訊，請參閱 the section called “JDBC 連線屬性”

選擇 Network (網路) 連接到 Amazon Virtual Private Cloud 環境 (Amazon VPC) 內的資料來源。

視您選擇的類型而定，AWS Glue 主控台會顯示其他必要的欄位。例如，如果您選擇 Amazon RDS，那麼您就必須選擇資料庫引擎。

需要 SSL 連線

選擇此選項時，AWS Glue 必須驗證連線已透過信任的 Secure Sockets Layer (SSL) 連線。

如需詳細資訊，包含在您選擇此選項時可用的額外選項，請參閱 the section called “SSL 連線屬性”。

選取 MSK 叢集 (僅限 Amazon Managed Streaming for Apache Kafka (MSK))

從另一個 AWS 帳戶指定 MSK 叢集。

Kafka 引導伺服器 URL (僅限 Kafka)

指定以逗號分隔的引導伺服 URL 清單。請加上連接埠號碼。例如：b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094, b-2.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094, b-3.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094

AWS Glue JDBC 連線屬性

AWS Glue 可以透過 JDBC 連線連接到以下資料存放區：

- Amazon Redshift
- Amazon Aurora
- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL
- 雪花，使用 AWS Glue 爬蟲時。
- Aurora (如果正在使用原生 JDBC 驅動程式則支援。並非所有驅動程式功能都可用)
- Amazon RDS for MariaDB

> **Important**
>
> 目前，ETL 任務只能使用一個子網路中的 JDBC 連線。如果您的任務中有多個資料存放區，它們必須位於同一個子網路，或者可從該子網路存取。
> 如果您選擇為 AWS Glue 爬蟲程式引入自己的 JDBC 驅動程式版本，爬蟲程式將使用 AWS Glue 任務和 Amazon S3 中的資源，以確保您提供的驅動程式在環境中執行。帳戶中將反映資源的額外使用量。此外，提供您的 JDBC 驅動程式，並不代表爬蟲程式能夠運用驅動程式的所有功能。驅動程式僅限於在資料型錄中定義連線中所述的屬性。

下列是 JDBC 連線類型的額外屬性。

JDBC URL

輸入您的 JDBC 資料存放區的 URL。對於大多數資料庫引擎而言，此欄位為以下格式。在此格式中，將 *protocol*、*host*、*port* 和 *db_name* 替換為您自己的資訊。

```
jdbc:protocol://host:port/db_name
```

依據資料庫引擎而定，可能需要不同的 JDBC URL 格式。此格式在使用冒號 (:) 和斜線 (/) 或不同關鍵字以指定資料庫方面，可以稍有不同。

如果是 JDBC 要連接到資料存放區，需要資料存放區中的 db_name。db_name 用於搭配 username 與 password 來建立網路連線。連線時，AWS Glue 可存取資料存放區內地其他資料庫，以執行爬蟲程式或執行 ETL 任務。

以下 JDBC URL 範例顯示多種資料庫引擎的語法。

- 若要連接到具有 dev 資料庫的 Amazon Redshift 叢集資料存放區：

  ```
  jdbc:redshift://xxx.us-east-1.redshift.amazonaws.com:8192/dev
  ```

- 若要連接到具有 employee 資料庫的 Amazon RDS for MySQL 資料存放區：

  ```
  jdbc:mysql://xxx-cluster.cluster-xxx.us-east-1.rds.amazonaws.com:3306/
  employee
  ```

- 若要連接到具有 employee 資料庫的 Amazon RDS for PostgreSQL 資料存放區：

  ```
  jdbc:postgresql://xxx-cluster.cluster-xxx.us-
  east-1.rds.amazonaws.com:5432/employee
  ```

- 若要連接到具有 employee 服務名稱的 Amazon RDS for Oracle 資料存放區：

  ```
  jdbc:oracle:thin://@xxx-cluster.cluster-xxx.us-
  east-1.rds.amazonaws.com:1521/employee
  ```

  Amazon RDS for Oracle 的語法可依照下列模式。在這些模式中，使用您自己的資訊取代 *host*、*port*、*service_name* 和 *SID*。

  - `jdbc:oracle:thin://@`*host*`:`*port*`/`*service_name*
  - `jdbc:oracle:thin://@`*host*`:`*port*`:`*SID*

- 若要連接到具有 employee 資料庫的 Amazon RDS for Microsoft SQL Server 資料存放區：

  ```
  jdbc:sqlserver://xxx-cluster.cluster-xxx.us-
  east-1.rds.amazonaws.com:1433;databaseName=employee
  ```

  Amazon RDS for SQL Server 的語法可依照下列模式。在這些模式中，使用您自己的資訊取代 *server_name*、*port* 和 *db_name*。

  - `jdbc:sqlserver://`*server_name*`:`*port*`;database=`*db_name*
  - `jdbc:sqlserver://`*server_name*`:`*port*`;databaseName=`*db_name*

- 若要連線到employee資料庫的 Amazon Aurora PostgreSQL 執行處理，請指定資料庫執行處理的端點、連接埠和資料庫名稱：

  ```
  jdbc:postgresql://employee_instance_1.xxxxxxxxxxxx.us-
  east-2.rds.amazonaws.com:5432/employee
  ```

- 若要使用 Amazon RDS for MariaDB 資料庫連線至資料倉employee庫，請指定資料庫執行個體的端點、連接埠和資料庫名稱：

```
jdbc:mysql://xxx-cluster.cluster-xxx.aws-
region.rds.amazonaws.com:3306/employee
```

- > **Warning**
  >
  > 雪花 JDBC 連線只有 AWS Glue 搜尋器才支援。在 AWS Glue 工作中使用雪花連接器時，請使用雪花連線類型。

  若要連線到 sample 資料庫中的 Snowflake 執行個體，請指定 Snowflake 執行個體的端點、使用者、資料庫名稱和角色名稱。您可以選擇是否新增 warehouse 參數。

  ```
  jdbc:snowflake://account_name.snowflakecomputing.com/?
  user=user_name&db=sample&role=role_name&warehouse=warehouse_name
  ```

  > **Important**
  >
  > 對於透過 JDBC 實現的 Snowflake 連線，會強制執行 URL 中的參數順序，且必須按 user、db、role_name 和 warehouse 排序參數。

- 若要使用 AWS 私人連結連線至sample資料庫的 Snowflake 執行個體，請指定雪花 JDBC URL，如下所示：

  ```
  jdbc:snowflake://account_name.region.privatelink.snowflakecomputing.com/?
  user=user_name&db=sample&role=role_name&warehouse=warehouse_name
  ```

使用者名稱

> **Note**
>
> 我們建議您使用 AWS 密碼來儲存連線認證，而不是直接提供您的使用者名稱和密碼。如需詳細資訊，請參閱 在 AWS Secrets Manager 中存放連線憑證。

提供有 JDBC 資料存放區存取許可的使用者名稱。

密碼

輸入提供有 JDBC 資料存放區存取許可的使用者名稱之密碼。

連線埠

輸入在 JDBC URL 中使用的連接埠，以連線至 Amazon RDS Oracle 執行個體。此欄位只有在針對 Amazon RDS Oracle 執行個體選取了 Require SSL connection (需要 SSL 連線) 時才會顯示。

VPC

選擇包含您的資料存放區的虛擬私有雲端 (VPC) 的名稱。AWS Glue 主控台會列出目前區域中所有的 VPC。

> **Important**
>
> 處理主控關閉的 JDBC 連線時 AWS，例如使用 Snowflake 的資料，您的 VPC 應該有一個 NAT 閘道，將流量分割為公有和私有子網路。公用子網路用於連線至外部來源，而內部子網路則用於處理 AWS Glue。如需有關為外部連線設定 Amazon VPC 的資訊，請參閱使用 NAT 裝置連線至網際網路或其他網路和 為 Amazon RDS 資料存放區的 JDBC 連線設定 Amazon VPC AWS Glue。

子網路

選擇 VPC 中包含您的資料存放區的子網路。AWS Glue 主控台列出您 VPC 中的資料存放區的所有子網路。

安全群組

選擇與您的資料存放區關聯的安全群組。AWS Glue 需要一或多個安全群組以及可讓 AWS Glue 連線的傳入來源規則。AWS Glue 主控台會列出所有已授權傳入存取 VPC 的安全群組。AWS Glue 將這些安全群組關聯至連接您 VPC 子網路的彈性網路界面。

JDBC 驅動程式類別名稱：選用

提供自訂 JDBC 驅動程式類別名稱：

- Postgres – org.postgresql.Driver
- MySQL – com.mysql.jdbc.Driver、com.mysql.cj.jdbc.Driver
- Redshift – com.amazon.redshift.jdbc.Driver、com.amazon.redshift.jdbc42.Driver
- 甲骨文-甲骨文 .jdbc. 驅動程序。 OracleDriver

- SQL 服務器-微軟 SQL 服務器。ServerDriver

JDBC 驅動程式 S3 路徑：選用

將 Amazon S3 位置提供給自訂 JDBC 驅動程式。這是 .jar 檔案的絕對路徑。如果您想要提供自己的 JDBC 驅動程式，以連線至爬蟲程式支援之資料庫的資料來源，您可以為參數 `customJdbcDriverS3Path` 和 `customJdbcDriverClassName` 指定值。使用客戶提供的 JDBC 驅動程式僅限於所需的 所需連線屬性。

AWS Glue MongoDB 和 MongoDB Atlas 連線屬性

下列是 MongoDB 或 MongoDB Atlas 連線類型的額外屬性。

MongoDB URL

輸入您的 MongoDB 或 MongoDB Atlas 資料存放區的網址：

- 若是 MongoDB：mongodb://host:port/database。主機可以是主機名稱、IP 地址或 UNIX 域通訊端。如果連接字串沒有指定連接埠，則會使用預設的 MongoDB 連接埠 27017。
- 若是 MongoDB Atlas：mongodb+srv://server.example.com/database。主機可以是遵循對應於 DNS SRV 記錄的主機名稱。SRV 格式不需要連接埠，而且會使用預設的 MongoDB 連接埠 27017。

使用者名稱

> **Note**
>
> 我們建議您使用 AWS 密碼來儲存連線認證，而不是直接提供您的使用者名稱和密碼。如需詳細資訊，請參閱 在 AWS Secrets Manager 中存放連線憑證。

提供有 JDBC 資料存放區存取許可的使用者名稱。

密碼

輸入提供有 MongoDB 或 MongoDB Atlas 資料存放區存取許可的使用者名稱之密碼。

銷售力量連線內容

以下是 Salesforce 連線類型的其他內容。

- ENTITY_NAME(字串)-(必要) 用於讀取/寫入。您的物件在 Salesforce 中的名稱。
- API_VERSION(字串)-(必要) 用於讀取/寫入。您想要使用的其餘 API 版本。
- SELECTED_FIELDS (列表<String>) -默認值：空 (選擇 *) 。用於讀取。您要為物件選取的欄。
- FILTER_PREDICATE(字串)-預設值：空白。用於讀取。它應該是在星火 SQL 格式。
- QUERY(字串)-預設值：空白。用於讀取。全火花 SQL 查詢。
- PARTITION_FIELD (字符串) -用於讀取。要用來分割查詢的欄位。
- LOWER_BOUND (字符串) -用於讀取。所選分割區欄位的包含下限值。
- UPPER_BOUND (字符串) -用於讀取。所選分割區欄位的獨佔上限值。
- NUM_PARTITIONS(整數)-預設值：1. 用於讀取。要讀取的分割區數目。
- IMPORT_DELETED_RECORDS (字符串) -默認值：假。用於讀取。在查詢時獲取刪除記錄。
- WRITE_OPERATION(字串)-預設值：插入。用於寫入。值應該是插入, 更新, 更新, 刪除.
- ID_FIELD_NAMES (字符串) -默認值：空。僅適用於 UPSERT。

Snowflake 連線

下列屬性可用來設定 AWS Glue ETL 工作中使用的雪花連線。對 Snowflake 進行網路爬取時，請使用 JDBC 連線。

Snowflake URL

Snowflake 端點的 URL。如需有關 Snowflake 端點 URL 的詳細資訊，請參閱 Snowflake 文件中的 Connecting to Your Accounts。

AWS 秘密

秘密中的秘密名稱 AWS Secrets Manager。 AWS Glue 將連接到雪花使用`sfUser`和你的秘密`sfPassword`鑰。

Snowflake 角色 (選用)

連線時 AWS Glue 會使用雪花安全性角色。

使用 AWS PrivateLink設定與 Amazon VPC 中託管的 Snowflake 端點的連線時，請使用下列屬性。

VPC

選擇包含您的資料存放區的虛擬私有雲端 (VPC) 的名稱。AWS Glue 主控台會列出目前區域中所有的 VPC。

子網路

選擇 VPC 中包含您的資料存放區的子網路。AWS Glue 主控台列出您 VPC 中的資料存放區的所有子網路。

安全群組

選擇與您的資料存放區關聯的安全群組。AWS Glue 需要一或多個安全群組以及可讓 AWS Glue 連線的傳入來源規則。AWS Glue 主控台會列出所有已授權傳入存取 VPC 的安全群組。AWS Glue 將這些安全群組關聯至連接您 VPC 子網路的彈性網路界面。

Vertica 連線

使用下列屬性來設定 AWS Glue ETL 工作的 Vertica 連線。

Vertica 主機

Vertica 安裝的主機名稱。

Vertica 連接埠

您可透過該連接埠安裝 Vertica。

AWS 秘密

秘密中的秘密名稱 AWS Secrets Manager。 AWS Glue 將使用您的秘密鑰連接到 Vertica。

設定連至 Amazon VPC 中託管之 Vertica 端點的連線時，請使用下列屬性。

VPC

選擇包含您的資料存放區的虛擬私有雲端 (VPC) 的名稱。AWS Glue 主控台會列出目前區域中所有的 VPC。

子網路

選擇 VPC 中包含您的資料存放區的子網路。AWS Glue 主控台列出您 VPC 中的資料存放區的所有子網路。

安全群組

選擇與您的資料存放區關聯的安全群組。AWS Glue 需要一或多個安全群組以及可讓 AWS Glue 連線的傳入來源規則。AWS Glue 主控台會列出所有已授權傳入存取 VPC 的安全群組。AWS Glue 將這些安全群組關聯至連接您 VPC 子網路的彈性網路界面。

SAP HANA 連線

請使用下列內容來設定 AWS Glue ETL 工作的 SAP HANA 連線。

SAP HANA URL

SAP JDBC URL。

SAP HANA JDBC URL 會採用的格式為 `jdbc:sap://saphanaHostname:saphanaPort/?databaseName=saphanaDBname,ParameterNa`

AWS Glue 需要下列 JDBC 網址參數：

- `databaseName`：要連線之 SAP HANA 的預設資料庫。

AWS 秘密

秘密中的秘密名稱 AWS Secrets Manager。 AWS Glue 將使用您的秘密鑰連接到 SAP HANA。

設定連至 Amazon VPC 中託管之 SAP HANA 端點的連線時，請使用下列屬性：

VPC

選擇包含您的資料存放區的虛擬私有雲端 (VPC) 的名稱。AWS Glue 主控台會列出目前區域中所有的 VPC。

子網路

選擇 VPC 中包含您的資料存放區的子網路。AWS Glue 主控台列出您 VPC 中的資料存放區的所有子網路。

安全群組

選擇與您的資料存放區關聯的安全群組。AWS Glue 需要一或多個安全群組以及可讓 AWS Glue 連線的傳入來源規則。AWS Glue 主控台會列出所有已授權傳入存取 VPC 的安全群組。AWS Glue 將這些安全群組關聯至連接您 VPC 子網路的彈性網路界面。

Azure SQL 連線

您可以使用下列屬性來設定 AWS Glue ETL 工作的 Azure SQL 連線。

Azure SQL URL

Azure SQL 端點的 JDBC URL。

此 URL 必須採用下列格式：`jdbc:sqlserver://databaseServerName:databasePort;databaseName=azuresqlDBna`

AWS Glue 需要下列 URL 屬性：

- `databaseName`：要連線之 Azure SQL 的預設資料庫。

如需有關 Azure SQL 受控執行個體之 JDBC URL 的詳細資訊，請參閱 Microsoft 文件。

AWS 秘密

秘密中的秘密名稱 AWS Secrets Manager。 AWS Glue 將使用您的密鑰連接到 Azure SQL。

Teradata Vantage 連線

您可以使用下列內容來設定 ETL 工作的 Teradata 遠端連線。 AWS Glue

Teradata URL

若要連線至 Teradata 執行個體，請指定資料庫執行個體的主機名稱和相關的 Teradata 參數：

`jdbc:teradata://teradataHostname/ParameterName=ParameterValue,ParameterName=Pa`

AWS Glue 支援下列 JDBC 網址參數：

- `DATABASE_NAME`：要連線之 Teradata 的預設資料庫。
- `DBS_PORT`：指定 Teradata 連接埠 (如果非標準)。

AWS 秘密

秘密中的秘密名稱 AWS Secrets Manager。 AWS Glue 將使用您的秘密鑰連接到 Teradata 華帝。

設定連至 Amazon VPC 中託管之 Teradata Vantage 端點的連線時，請使用下列屬性：

VPC

選擇包含您的資料存放區的虛擬私有雲端 (VPC) 的名稱。AWS Glue 主控台會列出目前區域中所有的 VPC。

子網路

選擇 VPC 中包含您的資料存放區的子網路。AWS Glue 主控台列出您 VPC 中的資料存放區的所有子網路。

安全群組

選擇與您的資料存放區關聯的安全群組。AWS Glue 需要一或多個安全群組以及可讓 AWS Glue 連線的傳入來源規則。AWS Glue 主控台會列出所有已授權傳入存取 VPC 的安全群組。AWS Glue 將這些安全群組關聯至連接您 VPC 子網路的彈性網路界面。

OpenSearch 服務連接

使用下列屬性來設定 AWS Glue ETL 工作的 OpenSearch 服務連線。

網域端點

Amazon OpenSearch 服務域端點將具有以下默認表單 https://search-*domainName-unstructuredIdContent*。 *##. #*馬遜. 如需識別網域端點的詳細資訊，請參閱 Amazon OpenSearch 服務文件中的建立和管理 Amazon OpenSearch 服務網域。

連線埠

端點上開啟的連接埠。

AWS 秘密

秘密中的秘密名稱 AWS Secrets Manager。 AWS Glue 將使用您的密鑰連接到 OpenSearch 服務。

設定與 Amazon VPC 中託管的 OpenSearch 服務端點的連線時，請使用下列屬性：

VPC

選擇包含您的資料存放區的虛擬私有雲端 (VPC) 的名稱。AWS Glue 主控台會列出目前區域中所有的 VPC。

子網路

選擇 VPC 中包含您的資料存放區的子網路。AWS Glue 主控台列出您 VPC 中的資料存放區的所有子網路。

安全群組

選擇與您的資料存放區關聯的安全群組。AWS Glue 需要一或多個安全群組以及可讓 AWS Glue 連線的傳入來源規則。AWS Glue 主控台會列出所有已授權傳入存取 VPC 的安全群組。AWS Glue 將這些安全群組關聯至連接您 VPC 子網路的彈性網路界面。

Azure Cosmos 連線

使用下列內容來設定 AWS Glue ETL 工作的 Azure 宇宙連線。

Azure Cosmos DB 帳戶端點 URI

用來連線至 Azure Cosmos 的端點。如需詳細資訊，請參閱 Azure 文件。

AWS 秘密

秘密中的秘密名稱 AWS Secrets Manager。 AWS Glue 將使用您的密鑰連接到 Azure 宇宙。

AWS Glue SSL 連線屬性

以下是 Require SSL connection (需要 SSL 連線) 屬性的相關詳細資訊。

如果您不需要 SSL 連線，AWS Glue 在使用 SSL 加密與資料存放區的連線時會忽略錯誤。請參閱適用於您資料存放區的文件以取得組態說明。當您選取此選項時，如果 AWS Glue 無法連線，開發端點中的任務執行、爬蟲程式或 ETL 陳述式即失敗。

> **Note**
>
> Snowflake 依預設支援 SSL 連線，因此此屬性不適用於 Snowflake。

此選項會在 AWS Glue 用戶端驗證。對於 JDBC 連線，AWS Glue 僅使用憑證和主機名稱驗證透過 SSL 連線。SSL 連線支援適用於：

- Oracle Database
- Microsoft SQL Server
- PostgreSQL
- Amazon Redshift
- MySQL (僅限 Amazon RDS 執行個體)
- Amazon Aurora MySQL (僅限 Amazon RDS 執行個體)
- Amazon Aurora PostgreSQL (僅限 Amazon RDS 執行個體)
- 卡夫卡, 其中包括 Amazon Managed Streaming for Apache Kafka
- MongoDB

> **Note**
>
> 若要啟用 Amazon RDS Oracle 資料存放區以使用 Require SSL connection (需要 SSL 連線)，您必須建立並連接選項群組至 Oracle 執行個體。
>
> 1. 登入 AWS Management Console 並開啟 Amazon RDS 主控台，網址為 https://console.aws.amazon.com/rds/。
> 2. 新增 Option group (選項群組) 至 Amazon RDS Oracle 執行個體。如需有關如何在 Amazon RDS 主控台新增選項群組的詳細資訊，請參閱建立選項群組。
> 3. 將 Option (選項) 新增至 SSL 選項群組。您為 SSL 指定的連接埠稍後會在您為 Amazon RDS Oracle 執行個體建立 AWS Glue JDBC 連線 URL 時用到。如需有關如何在 Amazon RDS 主控台新增選項的詳細資訊，請參閱 Amazon RDS 使用者指南中的將選項新增至選項群組。如需 Oracle SSL 選項的詳細資訊，請參閱 Amazon RDS 使用者指南中的 Oracle SSL。
> 4. 在 AWS Glue 主控台上建立一個至 Amazon RDS Oracle 執行個體的連線。在連線定義中，選擇 Require SSL connection (需要 SSL 連線)。根據要求，輸入您在 Amazon RDS Oracle SSL 選項中使用的 Port (連接埠)。

如果針對連線選取 Require SSL connection (需要 SSL 連線)，下列其他選用屬性即為可用。

在 S3 中自訂 JDBC 憑證

如果您有目前用於內部部署或雲端資料庫進行 SSL 通訊的憑證，您可以將該憑證用於 AWS Glue 資料來源或目標的 SSL 連線。輸入包含自訂根憑證的 Amazon Simple Storage Service (Amazon S3) 位置。AWS Glue 會使用此憑證來建立資料庫的 SSL 連線。AWS Glue 只會處理 X.509 憑證。憑證必須為 DER 編碼，並以 base64 編碼 PEM 格式提供。

如果此欄位保留空白，將會使用預設憑證。

自訂 JDBC 憑證字串

輸入 JDBC 資料庫特定的憑證資訊。這是網域比對或辨別名稱 (DN) 比對使用的字串。若為 Oracle 資料庫，此字串會映射到 `tnsnames.ora` 檔案安全區段中的 `SSL_SERVER_CERT_DN` 參數。若為 Microsoft SQL Server，此字串會做為 `hostNameInCertificate` 使用。

以下是 Oracle 資料庫 `SSL_SERVER_CERT_DN` 參數的範例。

```
cn=sales,cn=OracleContext,dc=us,dc=example,dc=com
```

Kafka 私有 CA 憑證位置

如果您有目前用於 Kafka 資料存放區進行 SSL 通訊的憑證，您可以將該憑證用於 AWS Glue 連線。此選項對於 Kafka 資料存放區而言是必需的，對於資料存放區而言是可選的 Amazon Managed Streaming for Apache Kafka 。輸入包含自訂根憑證的 Amazon Simple Storage Service (Amazon S3) 位置。AWS Glue 會使用此憑證來建立資料庫的 SSL 連線。AWS Glue 只會處理 X.509 憑證。憑證必須為 DER 編碼，並以 base64 編碼 PEM 格式提供。

略過憑證驗證

選取 Skip certificate validation (略過憑證驗證) 核取方塊可略過 AWS Glue 執行的自訂憑證驗證。如果您選擇驗證，AWS Glue 會驗證憑證的簽章演算法和主體公開金鑰演算法。如果憑證驗證失敗，使用該連線的任何 ETL 任務或爬蟲程式都會失敗。

允許的簽章演算法僅限 SHA256withRSA、SHA384withRSA 或 SHA512withRSA。針對主體公開金鑰演算法，金鑰長度必須至少為 2048。

Kafka 用戶端金鑰存放區位置

用於 Kafka 用戶端身分驗證的用戶端金鑰存放區檔案的 Amazon S3 位置。路徑的格式必須為 s3://bucket/prefix/filename.jks。其檔案名稱結尾必須是 .jks 副檔名。

Kafka 用戶端金鑰存放區密碼 (選用)

存取提供的金鑰存放區的密碼。

Kafka 用戶端金鑰密碼 (選用)

金鑰存放區可以由多個金鑰組成，所以這是用來存取用於 Kafka 伺服器端金鑰之用戶端金鑰的密碼。

用於用戶端驗證的 Apache Kafka 連線屬性

建立 Apache Kafka 連線時，AWS Glue 支援簡單驗證及安全性階層 (SASL) 架構進行驗證。SASL 框架支持各種身份驗證機制，並 AWS Glue 提供了 SCRAM（用戶名和密碼），GSSAPI（Kerberos 協議）和普通協議。

用 AWS Glue Studio 於設定下列其中一種用戶端驗證方法。如需詳細資訊，請參閱 AWS Glue Studio 使用指南中的建立連接器的連接。

- 無 - 無身分驗證。如果為進行測試而建立連線，此方法會很有用。
- SASL/SCRAM-SHA-512 - 選擇此身分驗證方法將允許您指定身分驗證憑證。有兩種可用選項：

  - 使用 AWS Secrets Manager (建議使用)-如果您選取此選項，您可以將使用者名稱和密碼儲存在 AWS Secrets Manager 中，並在需要時AWS Glue存取它們。指定存放 SSL 或 SASL 驗證憑證的秘密。如需詳細資訊，請參閱 在 AWS Secrets Manager 中存放連線憑證。
  - 請直接提供使用者名稱和密碼。
- SASL/GSSAPI (Kerberos) - 如果您選取此選項，則可以選取 keytab 檔案和 krb5.conf 檔案的位置，然後輸入 Kerberos 主體名稱和 Kerberos 服務名稱。keytab 檔案和 krb5.conf 檔案的位置必須位於 Amazon S3 位置。由於 MSK 尚不支援 SASL/GSSAPI，此選項僅適用於客戶受管的 Apache Kafka 叢集。如需詳細資訊，請參閱 MIT Kerberos 文件：Keytab。
- SASL/PLANY-選擇此驗證方法來指定認證身份證明。有兩種可用選項：
  - 使用 AWS Secrets Manager (建議使用)-如果您選取此選項，您可以將您的認證儲存在 AWS Secrets Manager 中，並在需要時 AWS Glue 存取這些資訊。指定存放 SSL 或 SASL 驗證憑證的秘密。
  - 直接提供用戶名和密碼。
- SSL 用戶端身分驗證 - 如果您選取此選項，則可以透過瀏覽 Amazon S3 來選取 Kafka 用戶端金鑰存放區的位置。或者，您可以輸入 Kafka 用戶端金鑰存放區密碼和 Kafka 用戶端金鑰密碼。

谷歌 BigQuery 連接

下列屬性用於設定 AWS Glue ETL 工作中使用的 Google BigQuery 連線。如需詳細資訊，請參閱 the section called “BigQuery 連線”。

AWS 秘密

秘密中的秘密名稱 AWS Secrets Manager。 AWS Glue ETL 工作將 BigQuery 使用您的秘密密`credentials`鑰連接到谷歌。

Vertica 連線

下列屬性可用來設定 AWS Glue ETL 工作中使用的 Vertica 連線。如需更多詳細資訊，請參閱 the section called “Vertica 連線”。

在 AWS Secrets Manager 中存放連線憑證

我們建議您使用 AWS Secrets Manager 以提供資料存放區的連線憑證。這樣做讓 AWS Glue 可以在執行時間存取 ETL 任務和爬蟲程式執行的秘密，並協助保護憑證的安全。

先決條件

若要將 AWS Glue 與 Secrets Manager 搭配使用，您必須授與 AWS Glue IAM 角色擷取秘密值的許可。AWS 受管政策 `AWSGlueServiceRole` 不包含 AWS Secrets Manager 許可。如需 IAM 政策的範例，請參閱《AWS Secrets Manager 使用者指南》中的 Example: Permission to retrieve secret values (範例：擷取秘密值的許可)。

根據您的網路設定而定，您可能還需要建立 VPC 端點，以在 VPC 與 Secrets Manager 之間建立私有連線。如需詳細資訊，請參閱使用 AWS Secrets Manager VPC 端點。

建立的秘密為 AWS Glue

1. 若要建立秘密，請遵循建立和管理秘密中的指示於《AWS Secrets Manager 使用者指南》。下列範例 JSON 示範如何在 Plaintext 標籤指定憑證，建立秘密為 AWS Glue。

   ```
   {
     "username": "EXAMPLE-USERNAME",
     "password": "EXAMPLE-PASSWORD"
   }
   ```

2. 使用 AWS Glue Studio 介面將機密與連線建立關聯。如需詳細說明，請參閱《AWS Glue Studio 使用者指南》中的建立連接器的連線。

新增 AWS Glue 連線

您可以以程式設計方式在 AWS Glue for Spark 中連線至資料來源。如需更多資訊，請參閱AWS Glue for Spark 中 ETL 的連線類型和選項

您也可以使用 AWS Glue 主控台新增、編輯、刪除和測試連線。如需 AWS Glue 連線的詳細資訊，請參閱 連線至資料。

新增 AWS Glue 連線

1. 請登入 AWS Management Console 並開啟 AWS Glue 主控台，網址為 https://console.aws.amazon.com/glue/。
2. 在導覽窗格中，於 Data catalog (Data Catalog) 下選擇 Connections (連線)。
3. 選擇 Add connection (新增連線)，然後完成精靈，依 the section called “AWS Glue 連線屬性” 中所述輸入連線屬性。

連接到 Amazon Redshift AWS Glue Studio

> **Note**
>
> 您可以使AWS Glue用 Spark 從以外的 Amazon Redshift 數據庫中讀取和寫入表AWS Glue Studio。要以編程方式 Amazon Redshift 配置AWS Glue工作，請參閱Redshift 連線。

AWS Glue提供的內建支援 Amazon Redshift。 AWS Glue Studio提供視覺化介面來連線 Amazon Redshift、編寫資料整合工作，以及在AWS Glue Studio無伺服器 Spark 執行階段上執行這些工作。

主題

建立 Amazon Redshift 連線

需要的許可

使用 Amazon Redshift 叢集和 Amazon Redshift 無伺服器環境需要其他權限。如需有關如何向 ETL 任務新增許可的詳細資訊，請參閱 Review IAM permissions needed for ETL jobs。

- 紅移：DescribeClusters
- 無紅移伺服器：ListWorkgroups
- 無紅移伺服器：ListNamespaces

概觀

添加 Amazon Redshift 連接時，您可以選擇現有的 Amazon Redshift 連接或在AWS Glue Studio中添加數據源-Redshift 節點時創建一個新的連接。

AWS Glue同時支援 Amazon Redshift 叢集和 Amazon Redshift 無伺服器環境。建立連線時， Amazon Redshift 無伺服器環境會在連線選項旁顯示無伺服器標籤。

如需有關如何建立 Amazon Redshift 連線的詳細資訊，請參閱將資料移入和移出 Amazon Redshift。

建立 Amazon Redshift 來源節點

需要的許可

使用 Amazon Redshift 資料來源的 AWS Glue Studio 任務需要額外的許可。如需有關如何向 ETL 任務新增許可的詳細資訊，請參閱 Review IAM permissions needed for ETL jobs。

需要下列許可才能使用 Amazon Redshift 連線。

- redshift-data:ListSchemas
- redshift-data:ListTables
- redshift-data:DescribeTable
- redshift-data:ExecuteStatement
- redshift-data:DescribeStatement
- redshift-data:GetStatementResult

新增 Amazon Redshift 資料來源

若要新增資料來源 – Amazon Redshift 節點：

1. 選擇 Amazon Redshift 存取類型：
   - 直接資料連線 (建議)：如果您要直接存取 Amazon Redshift 資料，請選擇此選項。這是建議的選項，也是預設選項。
   - Data Catalog tables：如果您有要使用的資料型錄資料表，請選擇此選項。
2. 如果選擇直接資料連線，請選擇 Amazon Redshift 資料來源的連線。假設連線已存在，您可以從現有的連線中進行選擇。如果您需要建立連線，請選擇建立 Redshift 連線。如需詳細資訊，請參閱 Overview of using connectors and connections 。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。隨即會顯示連線的相關資訊，包括 URL、安全群組、子網路、可用區域、說明，以及建立的 (UTC) 和上次更新的 (UTC) 時間戳記。
3. 選擇 Amazon Redshift 來源選項：
   - 選擇單一資料表：此資料表包含您要從單一 Amazon Redshift 資料表存取的資料。
   - 輸入自訂查詢：可讓您根據自訂查詢從多個 Amazon Redshift 資料表存取資料集。
4. 如果您選擇單一資料表，請選擇 Amazon Redshift 結構描述。可供選擇的結構描述清單由選取的資料表決定。

或者，選擇輸入自訂查詢。選擇此選項可從多個 Amazon Redshift 資料表存取自訂資料集。當您選擇此選項時，請輸入 Amazon Redshift 查詢。

連線至 Amazon Redshift 無伺服器環境時，請將下列許可新增至自訂查詢：

```
GRANT SELECT ON ALL TABLES IN <schema> TO PUBLIC
```

您可以選擇推論結構描述，根據您輸入的查詢讀取結構描述。您也可以選擇開啟 Redshift 查詢編輯器以輸入 Amazon Redshift 查詢。如需詳細資訊，請參閱 Querying a database using the query editor。

5. 在效能和安全性中，選擇 Amazon S3 暫存目錄和 IAM 角色。
   - Amazon S3 暫存目錄：選擇暫存資料的 Amazon S3 位置。
   - IAM 角色：選擇可寫入您選取之 Amazon S3 位置的 IAM 角色。
6. 在自訂 Redshift 參數 - 選用中，輸入參數和值。

建立目 Amazon Redshift 標節點

需要的許可

AWS Glue Studio 使用 Amazon Redshift 資料目標的工作需要其他權限。如需有關如何向 ETL 任務新增許可的詳細資訊，請參閱 Review IAM permissions needed for ETL jobs。

需要下列權限才能使用 Amazon Redshift 連線。

- 標記數據：ListSchemas
- 標記數據：ListTables

新增目 Amazon Redshift 標節點

若要建立 Amazon Redshift 目標節點：

1. 選擇現有 Amazon Redshift 表格作為目標，或輸入新表格名稱。
2. 如果您使用資料目標 - Redshift 目標節點，您可從以下選項中選擇：

- 附加：如果資料表已存在，請將所有新資料以插入方式傾印到資料表中。如果資料表不存在，請建立資料表，然後插入所有新資料。

  此外，如果您要更新 (UPSERT) 目標資料表中的現有記錄，請勾選此方塊。資料表必須先存在，否則操作將會失敗。

- 合併：AWS Glue 會根據您指定的條件，將資料更新或附加至目標資料表。

  > **Note**
  >
  > 若要在中使用合併動作AWS Glue，您必須啟用 Amazon Redshift 合併功能。如需如何為 Amazon Redshift 執行個體啟用合併的指示，請參閱 MERGE (預覽)。

  選擇選項：

  - 選擇索引鍵和簡單動作：選擇要用作來源資料與目標資料集之間相符索引鍵的資料欄。

    符合時指定下列選項：

    - 使用來源中的資料更新目標資料集中的記錄。
    - 刪除目標資料集中的記錄。

    不符合時指定下列選項：

    - 將來源資料作為新列插入目標資料集。
    - 什麼都不做。

  - 輸入自訂 MERGE 陳述式：然後您可以選擇驗證 Merge 陳述式，以驗證陳述式是有效還是無效。

- 截斷：如果資料表已存在，請先清除目標資料表的內容來截斷資料表資料。如果截斷成功，則插入所有資料。如果資料表不存在，請建立資料表並插入所有資料。如果截斷未成功，則操作將會失敗。
- 刪除：如果資料表已存在，請刪除資料表中繼資料和資料。如果刪除成功，則插入所有資料。如果資料表不存在，請建立資料表並插入所有資料。如果刪除未成功，則操作將會失敗。
- 建立：使用預設名稱建立新資料表。如果資料表名稱已存在，則建立一個新資料表，並在名稱後加上名稱後置詞 `job_datetime` 以確保唯一性。這會將所有資料插入新資料表中。如果資料表存在，則最終的資料表名稱將會附加後置詞。如果資料表不存在，則會建立資料表。無論哪種情況，都會建立新資料表。

進階選項

請參閱 Using the Amazon Redshift Spark connector on AWS Glue。

在 AWS Glue Studio 中連線至 Azure Cosmos DB

AWS Glue 會提供 Azure Cosmos DB 的內建支援。AWS Glue Studio 會提供視覺化介面以連線至 Azure Cosmos DB for NoSQL、撰寫資料整合工作，以及在 AWS Glue Studio 無伺服器 Spark 執行期上執行此類工作。

主題

建立 Azure Cosmos DB 連線

先決條件：

- 在 Azure 中，您將需要識別或產生 Azure Cosmos DB 索引鍵 `cosmosKey`，以供 AWS Glue 使用。如需詳細資訊，請參閱《Azure 文件》中的安全存取 Azure Cosmos DB 中的資料。

設定連至 Azure Cosmos DB 的連線：

1. 在 AWS Secrets Manager 中，使用 Azure Cosmos DB 索引鍵建立密碼。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *`secretName`*，以便進行下一個步驟。
   - 在選取鍵/值組時，請使用 *`cosmosKey`* 值來建立 `spark.cosmos.accountKey` 金鑰對。
2. 在 AWS Glue 主控台中，依照 the section called “新增 AWS Glue 連線” 中的步驟建立連線。建立連線之後，請保留連線名稱 *`connectionName`*，以便未來在 AWS Glue 中使用。
   - 選取連線類型時，請選取 Azure Cosmos DB。
   - 選取 AWS 機密時，請提供 *`secretName`*。

建立 Azure Cosmos DB 來源節點

必要先決條件

- 使用 AWS Secrets Manager 密碼設定的 AWS Glue Azure Cosmos DB 連線 (如上一節 the section called “建立 Azure Cosmos DB 連線” 中所述)。
- 針對您任務的適當許可，以讀取連線所使用的秘密。
- 您想要讀取的 Azure Cosmos DB for NoSQL 容器。您將需要容器的識別資訊。

  An Azure Cosmos NoSQL 容器由資料庫和容器識別。連線至 Azure Cosmos for NoSQL API 時，您必須提供資料庫名稱 *cosmosDBName* 和容器名稱 *cosmosContainerName*。

新增 Azure Cosmos DB 資料來源

新增資料來源 – Azure Cosmos DB 節點：

1. 選擇 Azure Cosmos DB 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 Azure Cosmos DB 連線。如需詳細資訊，請參閱前一 the section called “建立 Azure Cosmos DB 連線” 節。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。
2. 選擇 Cosmos DB 資料庫名稱：提供您想要讀取的資料庫名稱 *cosmosDBName*。
3. 選擇 Azure Cosmos 資料庫容器：提供您想要讀取的容器名稱 *cosmosContainerName*。
4. 或者，選擇 Azure Cosmos DB 自訂查詢：提供 SQL SELECT 查詢，以便從 Azure Cosmos DB 擷取特定資訊。
5. 在自訂 Azure Cosmos 屬性中，視需要輸入參數和值。

建立 Azure Cosmos DB 目標節點

必要先決條件

- 使用 AWS Secrets Manager 密碼設定的 AWS Glue Azure Cosmos DB 連線 (如上一節 the section called “建立 Azure Cosmos DB 連線” 中所述)。
- 針對您任務的適當許可，以讀取連線所使用的秘密。
- 您想要寫入的 Azure Cosmos DB 資料表。您將需要容器的識別資訊。您必須先建立容器，再呼叫連線方法。

An Azure Cosmos NoSQL 容器由資料庫和容器識別。連線至 Azure Cosmos for NoSQL API 時，您必須提供資料庫名稱 *cosmosDBName* 和容器名稱 *cosmosContainerName*。

新增 Azure Cosmos DB 資料目標

新增資料目標 – Azure Cosmos DB 節點：

1. 選擇 Azure Cosmos DB 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 Azure Cosmos DB 連線。如需詳細資訊，請參閱前一 the section called “建立 Azure Cosmos DB 連線” 節。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。
2. 選擇 Cosmos DB 資料庫名稱：提供您想要讀取的資料庫名稱 *cosmosDBName*。
3. 選擇 Azure Cosmos 資料庫容器：提供您想要讀取的容器名稱 *cosmosContainerName*。
4. 在自訂 Azure Cosmos 屬性中，視需要輸入參數和值。

進階選項

您可以在建立 Azure Cosmos DB 節點時提供進階選項。這些選項與為 AWS Glue for Spark 指令碼進行程式設計時的選項相同。

請參閱 the section called “Azure Cosmos DB 連線”。

在 AWS Glue Studio 中連線至 Azure SQL

AWS Glue 會提供 Azure SQL 的內建支援。AWS Glue Studio 會提供視覺化介面以連線至 Azure SQL、撰寫資料整合工作，以及在 AWS Glue Studio 無伺服器 Spark 執行期上執行此類工作。

主題

- 建立 Azure SQL 連線
- 建立 Azure SQL 來源節點
- 建立 Azure SQL 目標節點
- 進階選項

建立 Azure SQL 連線

若要從 AWS Glue 連線至 Azure SQL，您將需要在 AWS Secrets Manager 密碼中建立並儲存 Azure SQL 憑證，然後將該密碼與 Azure SQL AWS Glue 連線建立關聯。

設定連至 Azure SQL 的連線：

1. 在 AWS Secrets Manager 中，使用您的 Azure SQL 憑證建立密碼。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *secretName*，以便進行下一個步驟。

   - 在選取鍵/值組時，請使用 *azuresqlUsername* 值來建立 `user` 金鑰對。
   - 在選取鍵/值組時，請使用 *azuresqlPassword* 值來建立 `password` 金鑰對。

2. 在 AWS Glue 主控台中，依照 the section called “新增 AWS Glue 連線” 中的步驟建立連線。建立連線之後，請保留連線名稱 *connectionName*，以便未來在 AWS Glue 中使用。

   - 選取連線類型時，請選取 Azure SQL。
   - 提供 Azure SQL URL時，請提供 JDBC 端點 URL。

     此 URL 必須採用下列格式：`jdbc:sqlserver://`*databaseServerName*`:`*databasePort*`;databaseName=`*azuresqlDB*

     AWS Glue 需要具有下列 URL 屬性：

     - `databaseName`：要連線之 Azure SQL 的預設資料庫。

     如需有關 Azure SQL 受控執行個體之 JDBC URL 的詳細資訊，請參閱 Microsoft 文件。
   - 選取 AWS 機密時，請提供 *secretName*。

建立 Azure SQL 來源節點

必要先決條件

- 使用 AWS Secrets Manager 密碼設定的 AWS Glue Azure SQL 連線 (如上一節 the section called “建立 Azure SQL 連線” 中所述)。
- 針對您任務的適當許可，以讀取連線所使用的秘密。
- 您想要讀取的 Azure SQL 資料表 *tableName*。

Azure SQL 資料表由資料庫、結構描述及資料表名稱識別。連線至 Azure SQL 時，您必須提供資料庫名稱和資料表名稱。如果結構描述不是預設的 "public"，您也必須提供結構描述。資料庫會透過 *connectionName* 中的 URL 屬性提供，而結構描述和資料表名稱會透過 dbtable 提供。

新增 Azure SQL 資料來源

新增資料來源 – Azure SQL 節點：

1. 選擇 Azure SQL 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 Azure SQL 連線。如需詳細資訊，請參閱前一 the section called “建立 Azure SQL 連線” 節。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。
2. 選擇 Azure SQL 來源選項：
   - 選擇單一資料表：從單一資料表存取所有資料。
   - 輸入自訂查詢：根據自訂查詢從多個資料表存取資料集。
3. 如果您選擇單一資料表，請輸入 *tableName*。

   如果您選擇輸入自訂查詢，請輸入 TransactSQL SELECT 查詢。
4. 在自訂 Azure SQL 屬性中，視需要輸入參數和值。

建立 Azure SQL 目標節點

必要先決條件

- 使用 AWS Secrets Manager 密碼設定的 AWS Glue Azure SQL 連線 (如上一節 the section called “建立 Azure SQL 連線” 中所述)。
- 針對您任務的適當許可，以讀取連線所使用的秘密。
- 您想要寫入的 Azure SQL 資料表 *tableName*。

Azure SQL 資料表由資料庫、結構描述及資料表名稱識別。連線至 Azure SQL 時，您必須提供資料庫名稱和資料表名稱。如果結構描述不是預設的 "public"，您也必須提供結構描述。資料庫會透過 *connectionName* 中的 URL 屬性提供，而結構描述和資料表名稱會透過 dbtable 提供。

新增 Azure SQL 資料目標

新增資料目標 – Azure SQL 節點：

1. 選擇 Azure SQL 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 Azure SQL 連線。如需詳細資訊，請參閱前一 the section called “建立 Azure SQL 連線” 節。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。
2. 透過提供 *tableName* 來設定資料表名稱。
3. 在自訂 Azure SQL 屬性中，視需要輸入參數和值。

進階選項

您可以在建立 Azure SQL 節點時提供進階選項。這些選項與為 AWS Glue for Spark 指令碼進行程式設計時的選項相同。

請參閱 the section called “Azure SQL 連線”。

連接到谷歌 BigQuery AWS Glue Studio

> **Note**
>
> 您可以使AWS Glue用 Spark 在 AWS Glue 4.0 及更高版本中從谷歌 BigQuery 讀取和寫入表。若要透過程式設定 Google BigQuery 的AWS Glue工作，請參閱 BigQuery 連線。

AWS Glue Studio提供視覺化介面來連線 BigQuery、編寫資料整合工作，以及在AWS Glue Studio無伺服器 Spark 執行階段上執行這些工作。

主題

- 建立 BigQuery 連線
- 建立 BigQuery 來源節點
- 建立 BigQuery 目標節點
- 進階選項

建立 BigQuery 連線

若要從 AWS Glue 連線至 Google BigQuery，必須在 AWS Secrets Manager 秘密中建立並儲存您的 Google Cloud Platform 憑證，然後將該秘密與 Google BigQuery AWS Glue 連線建立關聯。

設定連至 BigQuery 的連線：

1. 在 Google Cloud Platform 中建立並識別相關資源：

   - 建立或識別 GCP 專案，其中應包含您要連線之 BigQuery 資料表。
   - 啟用 BigQuery API。如需詳細資訊，請參閱 Use the BigQuery Storage Read API to read table data。
2. 在 Google Cloud Platform 中建立和匯出服務帳戶憑證：

   您可以使用 BigQuery 憑證精靈來加速此步驟：Create credentials。

   若要在 GCP 中建立服務帳戶，請依照 Create service accounts 中提供的教學課程進行操作。

   - 在選取專案時，請選取包含您 BigQuery 資料表的專案。
   - 在為您的服務帳戶選取 GCP IAM 角色時，請新增或建立角色，其會授予供執行 BigQuery 任務的適當許可，以讀取、寫入或建立 BigQuery 資料表。

   若要為您的服務帳戶建立憑證，請依照 Create a service account key 中提供的教學課程進行操作。

   - 在選取金鑰類型時，請選取 JSON。

   您現在應已下載 JSON 檔案，該檔案中包含您服務帳戶的憑證。其看起來與下列類似：

```
{
  "type": "service_account",
  "project_id": "*****",
  "private_key_id": "*****",
  "private_key": "*****",
  "client_email": "*****",
  "client_id": "*****",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "*****",
```

```
  "universe_domain": "googleapis.com"
}
```

3. 對您下載的憑證檔案進行 base64 編碼。在 AWS CloudShell 工作階段或類似工作階段中，您可以透過執行 `cat credentialsFile.json | base64 -w 0` 來從命令列進行此動作。保留此命令的輸出，*credentialString*。
4. 在 AWS Secrets Manager 中使用您的 Google Cloud Platform 憑證建立秘密。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *secretName*，以便進行下一個步驟。
   - 在選取鍵值對時，請使用 *credentialString* 值來建立 `credentials` 鍵對。
5. 在 AWS Glue Data Catalog 中，依照 https://docs.aws.amazon.com/glue/latest/dg/console-connections.html 中的步驟建立連線。建立連線之後，請保留連線名稱 *connectionName*，以便進行下一個步驟。
   - 選取連線類型時，請選取 Google BigQuery。
   - 選取 AWS 機密時，請提供 *secretName*。
6. 授予與您 AWS Glue 任務許可相關聯的 IAM 角色，以讀取 *secretName*。
7. 在您的 AWS Glue 任務組態中，提供 *connectionName* 作為其他網路連線。

建立 BigQuery 來源節點

必要先決條件

- BigQuery 類型 AWS Glue Data Catalog 連線
- 您的 Google BigQuery 憑證的 AWS Secrets Manager 秘密，由連線使用。
- 針對您任務的適當許可，以讀取連線所使用的秘密。
- 您要讀取的資料表名稱和資料集，以及對應的 Google Cloud 專案。

新增 BigQuery 資料來源

新增資料來源 – BigQuery 節點：

1. 選擇您 BigQuery 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 BigQuery 連線。如需詳細資訊，請參閱 Overview of using connectors and connections 。

選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。

2. 識別您要讀取的 BigQuery 資料，然後選擇 BigQuery 來源選項

   - 選擇單一資料表 – 讓您可從資料表中提取所有資料。
   - 輸入自訂查詢 – 讓您可藉由提供查詢來自訂要擷取的資料。

3. 描述您要讀取的資料

   (必要) 將父系專案設定為包含資料表的專案，或設定為計費父系專案 (若相關)。

   若您選擇單一資料表，請以下列格式將資料表設定為 Google BigQuery 資料表的名稱：`[dataset].[table]`

   若您選擇查詢，請將其提供給查詢。在您的查詢中，請參考具有完整資料表名稱的資料表，格式為：`[project].[dataset].[tableName]`。

4. 提供 BigQuery 屬性

   若您選擇單一資料表，您便無需提供其他屬性。

   若您選擇查詢，則必須提供下列自訂 Google BigQuery 屬性：

   - 將 `viewsEnabled` 設定為 true。
   - 將 `materializationDataset` 設定為資料集。透過 AWS Glue 連線提供之認證驗證的 GCP 主體，必須能在此資料集中建立資料表。

建立 BigQuery 目標節點

必要先決條件

- BigQuery 類型 AWS Glue Data Catalog 連線
- 您的 Google BigQuery 憑證的 AWS Secrets Manager 秘密，由連線使用。
- 針對您任務的適當許可，以讀取連線所使用的秘密。
- 您要寫入的資料表名稱和資料集，以及對應的 Google Cloud 專案。

新增 BigQuery 資料目標

新增資料目標 – BigQuery 節點：

1. 選擇您 BigQuery 資料目標的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 BigQuery 連線。如需詳細資訊，請參閱 Overview of using connectors and connections 。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。
2. 識別您要寫入的 BigQuery 資料表，然後選擇一個寫入方法。

   - 直接 – 使用 BigQuery Storage Write API 直接寫入 BigQuery。
   - 間接 – 寫入 Google Cloud Storage，然後複製至 BigQuery。

   若您要間接寫入，請提供具有暫時 GCS 儲存貯體的目的地 GCS 位置。您將需在 AWS Glue 連線中提供其他配置。如需詳細資訊，請參閱 Using indirect write with Google BigQuery。
3. 描述您要讀取的資料

   (必要) 將父系專案設定為包含資料表的專案，或設定為計費父系專案 (若相關)。

   若您選擇單一資料表，請以下列格式將資料表設定為 Google BigQuery 資料表的名稱：`[dataset].[table]`

進階選項

您可以在建立 BigQuery 節點時提供進階選項。這些選項與 Spark 腳本編程時可用 AWS Glue 的選項相同。

請參閱 AWS Glue 開發人員指南中的 BigQuery 連線選項參考。

在 AWS Glue Studio 中連線至 MongoDB

AWS Glue 會提供 MongoDB 的內建支援。AWS Glue Studio 會提供視覺化介面以連線至 MongoDB、撰寫資料整合工作，以及在 AWS Glue Studio 無伺服器 Spark 執行期上執行此類工作。

主題

- 建立 MongoDB 連線
- 建立 MongoDB 來源節點

- 建立 MongoDB目標節點
- 進階選項

建立 MongoDB 連線

先決條件：

- 如果 Mongo DB 執行個體位於 Amazon VPC 中，請設定 Amazon VPC 以允許 AWS Glue 工作與 MongoDB 執行個體通訊，使流量不會周遊公有網際網路。

  在 Amazon VPC 中，識別或建立 AWS Glue 將在執行工作時使用的 VPC、子網路及安全群組。此外，您也需要確保 Amazon VPC 已完成設定，以允許 MongoDB 執行個體與此位置之間的網路流量。根據您的網路配置，這可能需要變更安全群組規則、網路 ACL、NAT 閘道及對等連線。

設定連至 MongoDB 的連線：

1. 或者，在 AWS Secrets Manager 中，使用 MongoDB 憑證建立密碼。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *secretName*，以便進行下一個步驟。

   - 在選取鍵/值組時，請使用 *mongodbUser* 值來建立 `username` 金鑰對。

     在選取鍵/值組時，請使用 *mongodbPass* 值來建立 `password` 金鑰對。
2. 在 AWS Glue 主控台中，依照 the section called “新增 AWS Glue 連線” 中的步驟建立連線。建立連線之後，請保留連線名稱 *connectionName*，以便未來在 AWS Glue 中使用。

   - 選取連線類型時，請選取 MongoDB 或 MongoDB Atlas。
   - 選取 MongoDB URL 或 MongoDB Atlas URL 時，請提供 MongoDB 執行個體的主機名稱。

     MongoDB URL 會以 `mongodb://`*mongoHost*`:`*mongoPort*`/`*mongoDBname* 格式提供。

     MongoDB Atlas URL 會以 `mongodb+srv://`*mongoHost*`:`*mongoPort*`/`*mongoDBname* 格式提供。

     您可選用 *mongoDBname* 針對連線提供預設資料庫。
   - 如果您選擇建立 Secrets Manager 密碼，請選擇 AWS Secrets Manager 憑證類型。

     然後，在 AWS 密碼中提供 *secretName*。

- 如果您選擇提供使用者名稱和密碼，請提供 *mongodbUser* 和 *mongodbPass*。

3. 在下列情況中，您可能需要其他組態：

   - Amazon VPC 中託管於 AWS 的 MongoDB 執行個體
     - 您將需要向定義 MongoDB 安全憑證的 AWS Glue 連線提供 Amazon VPC 連線資訊。建立或更新連線時，請在網路選項中設定 VPC、子網路及安全群組。

建立 AWS Glue MongoDB 連線後，您將需要執行下列步驟，才能執行 AWS Glue 工作：

- 在視覺化編輯器中處理 AWS Glue 工作時，您必須提供工作的 Amazon VPC 連線資訊，才能連線至 MongoDB。在 Amazon VPC 中識別合適的位置，並將其提供給 AWS Glue MongoDB 連線。
- 如果您選擇建立 Secrets Manager 密碼，請授予與 AWS Glue 工作權限相關聯的 IAM 角色，以讀取 *secretName*。

建立 MongoDB 來源節點

必要先決條件

- AWS Glue MongoDB 連線 (如上一節 the section called “建立 MongoDB 連線” 中所述)。
- 如果您選擇建立 Secrets Manager 密碼，請針對工作授予適當的權限，以讀取連線所使用的密碼。
- 您想要讀取的 MongoDB 集合。您將需要集合的識別資訊。

  MongoDB 集合由資料庫名稱與集合名稱 *mongodbName*、*mongodbCollection* 識別。

新增 MongoDB 資料來源

新增資料來源 – MongoDB 節點：

1. 選擇 MongoDB 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 MongoDB 連線。如需詳細資訊，請參閱前一 the section called “建立 MongoDB 連線” 節。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。

2. 選擇資料庫。輸入 *mongodbName*。
3. 選擇集合。輸入 *mongodbCollection*。

4. 選擇分割程式、分割區大小 (MB) 及分割區金鑰。如需有關分割區參數的詳細資訊，請參閱 the section called “"connectionType": "mongodb" as Source”。
5. 在自訂 MongoDB 屬性中，視需要輸入參數和值。

建立 MongoDB目標節點

必要先決條件

- 使用 AWS Secrets Manager 密碼設定的 AWS Glue MongoDB 連線 (如上一節 the section called “建立 MongoDB 連線” 中所述)。
- 針對您任務的適當許可，以讀取連線所使用的秘密。
- 您想要寫入的 MongoDB 資料表 *tableName*。

新增 MongoDB 資料目標

新增資料目標 – MongoDB 節點：

1. 選擇 MongoDB 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 MongoDB 連線。如需詳細資訊，請參閱前一 the section called “建立 MongoDB 連線” 節。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。
2. 選擇資料庫。輸入 *mongodbName*。
3. 選擇集合。輸入 *mongodbCollection*。
4. 選擇分割程式、分割區大小 (MB) 及分割區金鑰。如需有關分割區參數的詳細資訊，請參閱 the section called “"connectionType": "mongodb" as Source”。
5. 選擇重試寫入 (如有需要)。
6. 在自訂 MongoDB 屬性中，視需要輸入參數和值。

進階選項

您可以在建立 MongoDB 節點時提供進階選項。這些選項與為 AWS Glue for Spark 指令碼進行程式設計時的選項相同。

請參閱 the section called “MongoDB 連線”。

在 AWS Glue Studio 中連線至 OpenSearch Service

AWS Glue 會提供 Amazon OpenSearch Service 的內建支援。AWS Glue Studio 會提供視覺化介面以連線至 Amazon OpenSearch Service、撰寫資料整合任務，以及在 AWS Glue Studio 無伺服器 Spark 執行期上執行此類任務。此功能與 OpenSearch Service Serverless 不相容。

主題

建立 OpenSearch 服務連線

先決條件：

- 依照 Amazon 服務文件中的指示，識別網域端*##AoS port* 和連接埠、AoSport 或建立資源。OpenSearch 如需有關建立網域的詳細資訊，請參閱 Amazon OpenSearch 服務文件中的建立和管理 Amazon OpenSearch 服務網域。

  Amazon OpenSearch 服務域端點將具有以下默認表單 https://search-*domainName*-*unstructuredIdContent*。 *##. #*馬遜. 如需識別網域端點的詳細資訊，請參閱 Amazon OpenSearch 服務文件中的建立和管理 Amazon OpenSearch 服務網域。

  識別或產生您網域的 HTTP 基本身分驗證憑證 *aosUser* 和 *aosPassword*。

若要設定 OpenSearch 服務的連線：

1. 在中AWS Secrets Manager，使用您的 OpenSearch 服務認證建立密碼。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *secretName*，以便進行下一個步驟。

   - 在選取鍵/值組時，請使用 *aosUser* 值來建立 opensearch.net.http.auth.user 金鑰對。
   - 在選取鍵/值組時，請使用 *aosPassword* 值來建立 opensearch.net.http.auth.pass 金鑰對。
2. 在 AWS Glue 主控台中，依照 the section called “新增 AWS Glue 連線” 中的步驟建立連線。建立連線之後，請保留連線名稱 *connectionName*，以便未來在 AWS Glue 中使用。

- 選取連線類型時，請選取 OpenSearch 服務。
- 選取網域端點時，請提供 *aosEndpoint*。
- 選取連接埠時，請提供 *aosPort*。
- 選取 AWS 機密時，請提供 *secretName*。

建立 OpenSearch Service 來源節點

必要先決條件

- 使用 AWS Secrets Manager 密碼設定的 AWS Glue OpenSearch Service 連線 (如上一節 the section called “建立 OpenSearch 服務連線” 中所述)。
- 針對您任務的適當許可，以讀取連線所使用的秘密。
- 您想要讀取的 OpenSearch Service 索引 *aosIndex*。

新增 OpenSearch Service 資料來源

新增 資料來源 – OpenSearch Service 節點：

1. 選擇 OpenSearch Service 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 OpenSearch Service 連線。如需詳細資訊，請參閱前一 the section called “建立 OpenSearch 服務連線” 節。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。
2. 提供您想要讀取的索引。
3. 您可以選擇提供查詢 (OpenSearch 查詢)，以產生更特定的結果。如需有關寫入 OpenSearch 查詢的詳細資訊，請參閱 the section called “從 OpenSearch 服務中讀取”。
4. 在自訂 OpenSearch Service 屬性中，視需要輸入參數和值。

建立 OpenSearch Service 目標節點

必要先決條件

- 使用 AWS Secrets Manager 密碼設定的 AWS Glue OpenSearch Service 連線 (如上一節 the section called “建立 OpenSearch 服務連線” 中所述)。
- 針對您任務的適當許可，以讀取連線所使用的秘密。

- 您想要寫入的 OpenSearch Service 索引 *aosIndex*。

新增 OpenSearch Service 資料目標

新增 資料目標 – OpenSearch Service 節點：

1. 選擇 OpenSearch Service 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 OpenSearch Service 連線。如需詳細資訊，請參閱前一 the section called “建立 OpenSearch 服務連線” 節。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。
2. 提供您想要讀取的索引。
3. 在自訂 OpenSearch Service 屬性中，視需要輸入參數和值。

進階選項

您可以在建立 OpenSearch Service 節點時提供進階選項。這些選項與為 AWS Glue for Spark 指令碼進行程式設計時的選項相同。

請參閱 the section called “OpenSearch 服務連接”。

在中連接到銷售力量 AWS Glue Studio

Salesforce 提供客戶關係管理 (CRM) 軟體，可協助您進行銷售、客戶服務、電子商務等。如果您是 Salesforce 使用者，您可以連線 AWS Glue 到您的 Salesforce 帳戶。然後，您可以使用 Salesforce 做為 ETL 工作中的資料來源或目的地。執行這些工作以在 Salesforce 與 AWS 服務或其他支援的應用程式之間傳輸資料。

主題

- AWS Glue 對銷售力量的支持
- 包含用於建立和使用連線的 API 作業的政策
- 設定銷售力量
- 設定銷售連線
- 從銷售實體讀取
- 寫信給銷售人員
- 銷售力連線選項

- 銷售連接器的限制
- 為銷售力量設置 JWT 承載 OAuth 流程

AWS Glue 對銷售力量的支持

AWS Glue 支持銷售力量，如下所示：

作為來源支援？

是。您可以使用 AWS Glue ETL 工作來查詢來自 Salesforce 的資料。

作為目標支持？

是。您可以使用 AWS Glue ETL 將記錄寫入銷售力量。

支援的 API 版本

支援下列 API 版本

- V58.0
- V59.0
- V60.0

包含用於建立和使用連線的 API 作業的政策

下列範例政策說明建立和使用連線所需的 AWS IAM 許可。如果您要建立新角色，請建立包含下列項目的政策：

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

您也可以使用下列 IAM 政策來允許存取權：

- AWSGlueServiceRole— 授予對代表您執行各種 AWS Glue 程序所需之資源的存取權。這些資源包括 AWS Glue Amazon S3、IAM、 CloudWatch 日誌和 Amazon EC2。如果您遵循此原則中指定之資源的命名慣例，則 AWS Glue 處理程序將具有必要的權限。此政策通常會連接至定義爬蟲程式、工作和開發端點時所指定的角色。
- AWSGlueConsoleFullAccess— 當附加原則的身分識別使用 AWS 管理主控台時，授與對 AWS Glue 資源的完整存取權。如果您依照此政策中指定的資源命名慣例，使用者就能擁有完整的主控台功能。此原則通常會附加至 AWS Glue 主控台的使用者。

設定銷售力量

您必須符合下列需求，才能用 AWS Glue 來傳輸資料至 Salesforce 或從 Salesforce 傳輸資料：

最低需求

以下是最低要求：

- 您有一個銷售力量帳戶。
- 您的帳戶已啟用 API 存取權。企業版、無限制版、開發人員和效能版本預設會啟用 API 存取權。
- 您的 Salesforce 帳戶可讓您安裝連線的應用程式。如果您無法存取此功能，請聯絡您的 Salesforce 管理員。如需詳細資訊，請參閱 Salesforce 說明中的已連線應用程式。

如果您符合這些需求，就可以連線 AWS Glue 到您的 Salesforce 帳戶了。 AWS Glue 使用受管理的連線應用程式處 AWS 理剩餘需求。

受 AWS 管理的連線應用程式

AWS 受管理的連線應用程式可協助您以更少的步驟建立 Salesforce 連線。在 Salesforce 中，連線的應用程式是一種架構，可授權外部應用程式 (例如 AWS Glue) 存取您的 Salesforce 資料。

- 使用 AWS Glue 主控台建立 Salesforce 連線。
- 設定連線時，請將 OAuth 授予類型設定為授權碼。

設定銷售連線

若要設定銷售力量連線：

1. 在 AWS 密碼管理員中，使用下列詳細資料建立密碼：
   a. 對於 JWT_TOKEN 授予類型-密鑰應包含 JWT_TOKEN 密鑰及其值。
   b. 對於 AuthorizationCode 授予類型：對於客戶管理的連接應用程序，該密鑰應包含已連接的應用程序消費者密鑰以USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET作為密鑰。對於受 AWS 管理的連接應用程序，則為空密或具有某些臨時值的密碼。
   c. 附註：您必須在中為每個連線建立密碼 AWS Glue。
2. 在資 AWS Glue 料目錄中，依照下列步驟建立連線：
   a. 選取連線類型時，請選取 Salesforce。
   b. 提供您要連線到的銷售力量的執行個體 URL。
   c. 提供銷售環境。
   d. 選取 AWS Glue 可承擔並具有下列動作許可的 AWS IAM 角色：
   e. 選取您要用於連線的 OAuth2 授與類型。授權類型決定如何 AWS Glue 與 Salesforce 通訊以要求存取您的資料。您的選擇會影響您在建立連線之前必須符合的需求。您可以選擇下列其中一種類型：
      - JWT_BEARER 授予類型：此授權類型適用於自動化案例，因為它允許以 Salesforce 執行個體中特定使用者的權限預先建立 JSON 網頁權杖 (JWT)。創建者可以控制 JWT 的有效期限。AWS Glue 能夠使用 JWT 獲取用於調用 Salesforce API 的訪問令牌。

        此流程要求使用者已在其 Salesforce 執行個體中建立連線的應用程式，以便為使用者發出以 JWT 為基礎的存取權杖。

        有關為 JWT 承載 OAuth 流程創建連接應用程序的信息，請參閱用於集成的 OAuth 2.0 JWT 承載流程。 server-to-server 若要使用 Salesforce 連線的應用程式設定 JWT 承載流程，請參閱。為銷售力量設置 JWT 承載 OAuth 流程
      - Au@@ thorIZATION_CODE 授予類型：此授權類型被認為是「三條腿」的 OAuth，因為它依賴於將用戶重定向到第三方授權服務器來驗證用戶。它通過 AWS Glue 控制台創建連接時使用。根據預設，建立連線的使用者可能會依賴 AWS Glue 連線的應用程式 (AWS Glue 受管理的用戶端應用程式)，除了 Salesforce 執行個體 URL 以外，他們不需要提供任何 OAuth 相關資訊。 AWS Glue 主控台會將使用者重新導向至 Salesforce，使用者必須在其中登入，並允許要求 AWS Glue 的權限存取其 Salesforce 執行個體。

使用者仍然可以選擇在 Salesforce 中建立自己的連線應用程式，並在透過 AWS Glue 主控台建立連線時提供自己的用戶端 ID 和用戶端密碼。在這個案例中，他們仍會被重新導向至 Salesforce 以登入並授權存 AWS Glue 取其資源。

此授權類型會產生刷新令牌和訪問令牌。訪問令牌壽命短，並且可以在沒有用戶交互使用刷新令牌的情況下自動刷新。

有關為授權碼 OAuth 流程創建連接應用程序的信息，請參閱為授權碼和憑據流程配置連接的應用程序。

f. 選擇`secretName`要用於此連接的內 AWS Glue 容以放置令牌。

g. 如果您要使用網路，請選取網路選項。授予與您的 AWS Glue 工作相關聯的 IAM 角色以進行讀取`secretName`。

3. 授予與您的 AWS Glue 工作相關聯的 IAM 角色以進行讀取`secretName`。

4. 在您的 AWS Glue 工作組態中，提供`connectionName`為其他網路連線。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

從銷售實體讀取

必要條件

您想要讀取的銷售支援對象。您將需要物件名稱，例如`Account`或`Case`或`Opportunity`。

範例：

```
salesforce_read = glueContext.create_dynamic_frame.from_options(
    connection_type="salesforce",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "Account",
        "API_VERSION": "v60.0"
    }
```

分割查詢

您可以提供額外的 Spark 選項PARTITION_FIELD、、LOWER_BOUNDUPPER_BOUND，以及NUM_PARTITIONS如果您想要在 Spark 中使用並發功能。使用這些參數，原始查詢將被分成可以由 Spark 任務同時執行的子查詢的數NUM_PARTITIONS量。

- PARTITION_FIELD: 要用來分割查詢的欄位名稱。
- LOWER_BOUND: 所選分割區欄位的包含下限值。

  對於時間戳字段，我們接受 Spark SQL 查詢中使用的星火時間戳格式。

  有效值的例子：

  ```
  "TIMESTAMP \"1707256978123\""
  "TIMESTAMP '2024-02-06 22:02:58.123 UTC'"
  "TIMESTAMP \"2018-08-08 00:00:00 Pacific/Tahiti\"
  "TIMESTAMP \"2018-08-08 00:00:00\""
  "TIMESTAMP \"-123456789\" Pacific/Tahiti"
  "TIMESTAMP \"1702600882\""
  ```
- UPPER_BOUND：所選分割區欄位的獨佔上限值。
- NUM_PARTITIONS: 分割區的數量。

範例：

```
salesforce_read = glueContext.create_dynamic_frame.from_options(
    connection_type="salesforce",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "Account",
        "API_VERSION": "v60.0",
```

```
        "PARTITION_FIELD": "SystemModstamp"
        "LOWER_BOUND": "TIMESTAMP '2021-01-01 00:00:00 Pacific/Tahiti'"
        "UPPER_BOUND": "TIMESTAMP '2023-01-10 00:00:00 Pacific/Tahiti'"
        "NUM_PARTITIONS": "10"
    }
```

寫信給銷售人員

先決條件

您想要寫入的銷售支援對象。您將需要物件名稱，例如Account或Case或Opportunity。

Salesforce 連接器支援四種寫入作業：

- INSERT
- UPSERT (更新插入)
- UPDATE
- DELETE

使用UPSERT寫入作業時，ID_FIELD_NAMES必須提供以指定記錄的外部 ID 欄位。

範例

```
salesforce_write = glueContext.write_dynamic_frame.from_options(
    frame=frameToWrite,
    connection_type="salesforce",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "Account",
        "API_VERSION": "v60.0",
        "WRITE_OPERATION": "INSERT"
    }
```

銷售力連線選項

下列是適用於 Salesforce 的連線選項：

- ENTITY_NAME(字串)-(必要) 用於讀取/寫入。您的物件在 Salesforce 中的名稱。
- API_VERSION(字串)-(必要) 用於讀取/寫入。您想要使用的其餘 API 版本。
- SELECTED_FIELDS (列表<String>) -默認值：空 (選擇 *) 。用於讀取。您要為物件選取的欄。

- FILTER_PREDICATE(字串)-預設值：空白。用於讀取。它應該是在星火 SQL 格式。
- QUERY(字串)-預設值：空白。用於讀取。全火花 SQL 查詢。
- PARTITION_FIELD（字符串）-用於讀取。用於對查詢進行分區的欄位。
- LOWER_BOUND（字符串）-用於讀取。所選分割區欄位的包含下限值。
- UPPER_BOUND（字符串）-用於讀取。所選分割區欄位的獨佔上限值。
- NUM_PARTITIONS(整數)-預設值：1. 用於讀取。要讀取的分割區數目。
- IMPORT_DELETED_RECORDS（字符串）-默認值：假。用於讀取。要在查詢時獲取刪除記錄。
- WRITE_OPERATION(字串)-預設值：插入。用於寫入。值應該是插入, 更新, 更新, 刪除.
- ID_FIELD_NAMES（字符串）-默認值：空。僅適用於 UPSERT。

銷售連接器的限制

下列是 Salesforce 連接器的限制：

- 我們只支持星火 SQL 和不支持銷售 SOQL。
- 不支援任務書籤。

為銷售力量設置 JWT 承載 OAuth 流程

如需啟用與 OAuth 2.0 JSON 網頁權杖的 server-to-server 整合，請參閱 Salesforce 公開文件。

建立PEM檔案的證書/金key pair

建立PEM檔案的證書/金key pair

```
openssl req -newkey rsa:4096 -new -nodes -x509 -days 3650 -keyout key.pem -out cert.pem
```

使用 JWT 建立與 Salesforce 連線的應用程式

1. 登入 Salesforce 並按一下右上角的設定齒輪，然後選取 [設定]。
2. 在左側，導覽至 [應用程式管理員]。(平台工具 > 應用程式 > 應用程式管理員)
3. 選擇新增連線應用程式。
4. 提供一個應用程序名稱，讓其餘的 auto 填充。
5. 勾選「啟用 OAuth 設定」方塊。
6. 設定回呼網址。它不會用於 JWT，因此您可以使用 https://localhost。

7. 勾選「使用數位簽名」方塊。
8. 上傳先前建立的憑證 .pem 檔案。
9. 新增必要的權限：
   a. 透過 API 管理使用者資料。
   b. 存取自訂權限 (自訂權限)。
   c. 存取身分識別 URL 服務 (ID、設定檔、電子郵件、地址、電話)。
   d. 訪問唯一的用戶標識符（OpenID）。
   e. 隨時執行請求（刷新令牌，離線訪問）。
10核取 [為指名使用者發行 JSON Web Token (JWT) 存取權杖] 核取方塊。
11選擇儲存。
12選擇繼續。
13選擇管理消費者詳細資訊。
14複製消費者金鑰 (用戶端 ID)。
15複製消費者密碼 (用戶端密碼)。
16按一下 Cancel (取消)。

生成一個 JSON 網絡令牌（JWT）

1. 將 key pair 轉換為 pkcs12 (出現提示時設定匯出密碼)。

```
openssl pkcs12 -export -in cert.pem -inkey key.pem -name jwtcert > jwtcert.p12
```

2. 從 pkcs12 建立 Java 金鑰儲存庫 (在出現提示時設定目標金鑰庫密碼，並提供原始碼儲存庫密碼之前的匯出密碼)。

```
keytool -importkeystore -srckeystore jwtcert.p12 -destkeystore keystore.jks -
srcstoretype pkcs12 -alias jwtcert
```

3. 確認密鑰庫 .jks 包含 jwtcert 別名（出現提示時輸入先前的目標密鑰庫密碼）。

```
keytool -keystore keystore.jks -list
```

4. 使用 Salesforce 文件中提供的 Java 類別 JWT 範例來產生已簽署的權杖。
   a. 視需要編輯磁面馬拉線中的值：
      - 克萊馬雷 [0] = 用戶端識別碼

- 克萊馬雷 [1] = 銷售人員使用者識別碼
- 克萊馬雷 [2] = 銷售人員登錄網址
- 克萊馬雷 [4] = 自紀元以來的到期日，以毫米為單位。3660624000000 是 2085-12-31。

b. 將路徑/到密鑰庫替換為密鑰庫 .jks 的正確路徑。

c. 將密鑰庫密碼替換為您輸入的目標密鑰庫密碼

d. 用您輸入的源密鑰庫密碼替換私密密鑰密碼

e. 編譯程式碼。該代碼取決於用於 base64 編碼的 Apache 共享編解碼器。

```
javac -classpath ".:./commons-codec-1.16.1.jar" JWTExample.java
```

f. 執行程式碼。

```
java -classpath ".:commons-codec-1.16.1.jar" JWTExample
```

5. 創建連接的應用程序和 JWT 後，仍然需要對該應用程序授權用戶。有關兩種方法，請參閱 https://mannharleen.github.io/2020-03-03-salesforce-jwt/ 中的步驟 3。

通過上述步驟完成後，這應該輸出一個 JSON 網絡令牌（JWT），可用於從 Salesforce 獲取訪問令牌。

示例輸入：

```
export password for pkcs12: awsglue
destination keystore password for jks: awsglue
source keystore password for jks: awsglue

claimArray[0] = “client-id”;
claimArray[1] = “my@email.com”;
claimArray[2] = "https://login.salesforce.com“;
claimArray[3] = "3660624000000";

path to keystore: ./keystore.jks
keystore password: awsglue
privatekey password: awsglue
```

示例輸出：

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0Ij
```

有用的鏈接：

- https://www.base64encode.org/
- https://jwt.io/
- https://help.salesforce.com/s/articleView?id=sf.remoteaccess_oauth_jwt_flow.htm

JWTExample.java:

```
import org.apache.commons.codec.binary.Base64;
import java.io.*;
import java.security.*;
import java.text.MessageFormat;

public class JWTExample {

  public static void main(String[] args) {

    String header = "{\"alg\":\"RS256\"}";
    String claimTemplate = "'{'\"iss\": \"{0}\", \"sub\": \"{1}\", \"aud\": \"{2}\",
 \"exp\": \"{3}\"'}'";

    try {
      StringBuffer token = new StringBuffer();

      //Encode the JWT Header and add it to our string to sign
      token.append(Base64.encodeBase64URLSafeString(header.getBytes("UTF-8")));

      //Separate with a period
      token.append(".");

      //Create the JWT Claims Object
      String[] claimArray = new String[5];
      claimArray[0] = "value";
      claimArray[1] = "my@email.com";
      claimArray[2] = "https://login.salesforce.com";
      claimArray[3] = Long.toString( ( System.currentTimeMillis()/1000 ) + 300);
      MessageFormat claims;
      claims = new MessageFormat(claimTemplate);
      String payload = claims.format(claimArray);

      //Add the encoded claims object
      token.append(Base64.encodeBase64URLSafeString(payload.getBytes("UTF-8")));
```

```
      //Load the private key from a keystore
      KeyStore keystore = KeyStore.getInstance("JKS");
      keystore.load(new FileInputStream("./keystore.jks"), "awsglue".toCharArray());
      PrivateKey privateKey = (PrivateKey) keystore.getKey("jwtcert",
 "awsglue".toCharArray());

      //Sign the JWT Header + "." + JWT Claims Object
      Signature signature = Signature.getInstance("SHA256withRSA");
      signature.initSign(privateKey);
      signature.update(token.toString().getBytes("UTF-8"));
      String signedPayload = Base64.encodeBase64URLSafeString(signature.sign());

      //Separate with a period
      token.append(".");

      //Add the encoded signature
      token.append(signedPayload);

      System.out.println(token.toString());

    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

在 AWS Glue Studio 中連線至 SAP HANA

AWS Glue 會提供 SAP HANA 的內建支援。AWS Glue Studio 會提供視覺化介面以連線至 SAP HANA、撰寫資料整合工作，以及在 AWS Glue Studio 無伺服器 Spark 執行期上執行此類工作。

主題

- 建立 SAP HANA 連線
- 建立 SAP HANA 來源節點
- 建立 SAP HANA 目標節點
- 進階選項

建立 SAP HANA 連線

若要從 AWS Glue 連線至 SAP HANA，您將需要在 AWS Secrets Manager 密碼中建立並儲存 SAP HANA 憑證，然後將該密碼與 SAP HANA AWS Glue 連線建立關聯。您將需要設定 SAP HANA 服務與 AWS Glue 之間的網路連線。

先決條件：

- 如果 SAP HANA 服務位於 Amazon VPC 中，請設定 Amazon VPC 以允許 AWS Glue 工作與 SAP HANA 服務進行通訊，使流量不會周遊公有網際網路。

  在 Amazon VPC 中，識別或建立 AWS Glue 將在執行工作時使用的 VPC、子網路及安全群組。此外，您也需要確保 Amazon VPC 已完成設定，以允許 SAP HANA 端點與此位置之間的網路流量。您的工作將需要與 SAP HANA JDBC 連接埠建立 TCP 連線。如需有關 SAP HANA 連接埠的詳細資訊，請參閱 SAP HANA 文件。根據您的網路配置，這可能需要變更安全群組規則、網路 ACL、NAT 閘道及對等連線。

設定連至 SAP HANA 的連線：

1. 在 AWS Secrets Manager 中，使用 SAP HANA 憑證建立密碼。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *secretName*，以便進行下一個步驟。

   - 在選取鍵/值組時，請使用 *saphanaUsername* 值來建立 `user` 金鑰對。
   - 在選取鍵/值組時，請使用 *saphanaPassword* 值來建立 `password` 金鑰對。

2. 在 AWS Glue 主控台中，依照 the section called “新增 AWS Glue 連線” 中的步驟建立連線。建立連線之後，請保留連線名稱 *connectionName*，以便未來在 AWS Glue 中使用。

   - 選取連線類型時，請選取 SAP HANA。
   - 提供 SAP HANA URL 時，請提供執行個體的 URL。

     SAP HANA JDBC URL 會採用的格式為 `jdbc:sap://`*saphanaHostname*`:`*saphanaPort*`/?`*databaseName*`=`*saphanaDBname*`,`*Paramete*

     AWS Glue 需要下列 JDBC URL 參數：

     - `databaseName`：要連線之 SAP HANA 的預設資料庫。
   - 選取 AWS 機密時，請提供 *secretName*。

建立 AWS Glue SAP HANA 連線之後，您將需要執行下列步驟，才能執行 AWS Glue 工作：

- 授予與您 AWS Glue 任務許可相關聯的 IAM 角色，以讀取 *secretName*。

建立 SAP HANA 來源節點

必要先決條件

- 使用 AWS Secrets Manager 密碼設定的 AWS Glue SAP HANA 連線 (如上一節 the section called "建立 SAP HANA 連線" 中所述)。
- 針對您任務的適當許可，以讀取連線所使用的秘密。
- 您想要讀取的 SAP HANA 資料表 *tableName* 或查詢 *targetQuery*。

  您可以在表單 *schemaName*.*tableName* 中使用 SAP HANA 資料表名稱和結構描述名稱來指定資料表。如果資料表位於預設結構描述 "public" 中，則不需要結構描述名稱和 "." 分隔符號。呼叫此 *tableIdentifier*。請注意，在 connectionName 中，資料庫會以 JDBC URL 參數形式提供。

新增 SAP HANA 資料來源

新增資料來源 – SAP HANA 節點：

1. 選擇 SAP HANA 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 SAP HANA 連線。如需詳細資訊，請參閱前一 the section called "建立 SAP HANA 連線" 節。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。
2. 選擇 SAP HANA 來源 選項：
   - 選擇單一資料表：從單一資料表存取所有資料。
   - 輸入自訂查詢：根據自訂查詢從多個資料表存取資料集。
3. 如果您選擇單一資料表，請輸入 *tableName*。

   如果您選擇輸入自訂查詢，請輸入 SQL SELECT 查詢。
4. 在自訂 SAP HANA 屬性中，視需要輸入參數和值。

建立 SAP HANA 目標節點

必要先決條件

- 使用 AWS Secrets Manager 密碼設定的 AWS Glue SAP HANA 連線 (如上一節 the section called “建立 SAP HANA 連線” 中所述)。
- 針對您任務的適當許可，以讀取連線所使用的秘密。
- 您想要寫入的 SAP HANA 資料表 *tableName*。

  您可以在表單 *schemaName*.*tableName* 中使用 SAP HANA 資料表名稱和結構描述名稱來指定資料表。如果資料表位於預設結構描述 "public" 中，則不需要結構描述名稱和 "." 分隔符號。呼叫此 *tableIdentifier*。請注意，在 connectionName 中，資料庫會以 JDBC URL 參數形式提供。

新增 SAP HANA 資料目標

新增資料目標 – SAP HANA 節點：

1. 選擇 SAP HANA 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 SAP HANA 連線。如需詳細資訊，請參閱前一 the section called “建立 SAP HANA 連線” 節。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。
2. 透過提供 *tableName* 來設定資料表名稱。
3. 在自訂 Teradata 屬性中，視需要輸入參數和值。

進階選項

您可以在建立 SAP HANA 節點時提供進階選項。這些選項與為 AWS Glue for Spark 指令碼進行程式設計時的選項相同。

請參閱 the section called “SAP HANA 連線”。

在 AWS Glue Studio 中連線至 Snowflake

> **Note**
>
> 您可以使用 AWS Glue for Spark 從 AWS Glue 4.0 及更新版本中的 Snowflake 讀取和寫入資料表。若要使用 AWS Glue 任務以程式設計方式設定 Snowflake 連線，請參閱Redshift 連線。

AWS Glue 提供 Snowflake 的內建支援。AWS Glue Studio 提供視覺化介面來連線至 Snowflake、撰寫資料整合任務，以及在 AWS Glue Studio 無伺服器 Spark 執行期上執行這些任務。

主題

建立 Snowflake 連線

在 AWS Glue Studio 中新增資料來源 - Snowflake 節點時，您可以選擇現有的 AWS Glue Snowflake 連線或建立新連線。您必須選擇 SNOWFLAKE 類型連線，而不是設定為連線至 Snowflake 的 JDBC 類型連線。遵循下列程序以建立 AWS Glue Snowflake 連線：

建立 Snowflake 連線

1. 在 Snowflake 中產生一個使用者、*snowflakeUser*和密碼 *snowflakePassword*。
2. 確定此使用者將與哪個 Snowflake 倉儲 (*snowflakeWarehouse*) 互動。可將其設定為 Snowflake 中 *snowflakeUser* 的 DEFAULT_WAREHOUSE，或是記住此資訊以供下一步使用。
3. 在 AWS Secrets Manager 中，使用您的 Snowflake 憑證建立機密。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *secretName*，以便進行下一個步驟。
   - 選取鍵值對時，使用索引鍵 sfUser 為 *snowflakeUser* 建立鍵值對。
   - 選取鍵值對時，使用索引鍵 sfPassword 為 *snowflakePassword* 建立鍵值對。
   - 選取鍵值對時，使用索引鍵 sfWarehouse 為 *snowflakeWarehouse* 建立鍵值對。如果 Snowflake 中設定了預設值，則不需要此操作。
4. 在 AWS Glue Data Catalog 中，依照新增 AWS Glue 連線中的步驟建立連線。建立連線之後，請保留連線名稱 *connectionName*，以便進行下一個步驟。
   - 選取連線類型時，請選取 Snowflake。
   - 選取 Snowflake URL 時，請提供 Snowflake 執行個體的主機名稱。URL 將在表單 *account_identifier*.snowflakecomputing.com 中使用 hostname。
   - 選取 AWS 機密時，請提供 *secretName*。

建立 Snowflake 來源節點

需要的許可

使用 Snowflake 資料來源的 AWS Glue Studio 任務需要額外的許可。如需有關如何向 ETL 任務新增許可的詳細資訊，請參閱 Review IAM permissions needed for ETL jobs。

SNOWFLAKE AWS Glue 連線會使用 AWS Secrets Manager 機密來提供憑證資訊。您在 AWS Glue Studio 中的任務和資料預覽版角色必須具有讀取此機密的許可。

新增 Snowflake 資料來源

先決條件：

- Snowflake 憑證的 AWS Secrets Manager 機密
- Snowflake 類型 AWS Glue Data Catalog 連線

若要新增資料來源 – Snowflake 節點：

1. 選擇 Snowflake 資料來源的連線。假設連線已存在，您可以從現有的連線中進行選擇。如果您需要建立連線，請選擇建立 Snowflake 連線。如需詳細資訊，請參閱 Overview of using connectors and connections 。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。隨即會顯示連線的相關資訊，包括 URL、安全群組、子網路、可用區域、說明，以及建立的 (UTC) 和上次更新的 (UTC) 時間戳記。
2. 選擇 Snowflake 來源選項：
   - 選擇單一資料表：此資料表包含您要從單一 Snowflake 資料表存取的資料。
   - 輸入自訂查詢：可讓您根據自訂查詢從多個 Snowflake 資料表存取資料集。
3. 如果您選擇單一資料表，請輸入 Snowflake 結構描述的名稱。

   或者，選擇輸入自訂查詢。選擇此選項可從多個 Snowflake 資料表存取自訂資料集。當您選擇此選項時，請輸入 Snowflake 查詢。
4. 在效能和安全性選項 (選用) 中，
   - 啟用查詢下推：選擇是否要將工作卸載至 Snowflake 執行個體。
5. 在自訂 Snowflake 屬性 (選用) 中，根據需要輸入參數和值。

建立 Snowflake 目標節點

需要的許可

使用 Snowflake 資料來源的 AWS Glue Studio 任務需要額外的許可。如需有關如何向 ETL 任務新增許可的詳細資訊，請參閱 Review IAM permissions needed for ETL jobs。

SNOWFLAKE AWS Glue 連線會使用 AWS Secrets Manager 機密來提供憑證資訊。您在 AWS Glue Studio 中的任務和資料預覽版角色必須具有讀取此機密的許可。

新增 Snowflake 資料目標

若要建立 Snowflake 目標節點：

1. 選擇現有的 Snowflake 資料表作為目標，或輸入新的資料表名稱。
2. 當您使用資料目標 - Snowflake 目標節點時，您可以從下列選項中進行選擇：

   - 附加：如果資料表已存在，請將所有新資料以插入方式傾印到資料表中。如果資料表不存在，請建立資料表，然後插入所有新資料。
   - 合併：AWS Glue 會根據您指定的條件，將資料更新或附加至目標資料表。

     選擇選項：

     - 選擇索引鍵和簡單動作：選擇要用作來源資料與目標資料集之間相符索引鍵的資料欄。

       符合時指定下列選項：

       - 使用來源中的資料更新目標資料集中的記錄。
       - 刪除目標資料集中的記錄。

       不符合時指定下列選項：

       - 將來源資料作為新列插入目標資料集。
       - 什麼都不做。
     - 輸入自訂 MERGE 陳述式：然後您可以選擇驗證 Merge 陳述式，以驗證陳述式是有效還是無效。
   - 截斷：如果資料表已存在，請先清除目標資料表的內容來截斷資料表資料。如果截斷成功，則插入所有資料。如果資料表不存在，請建立資料表並插入所有資料。如果截斷未成功，則操作將會失敗。
   - 刪除：如果資料表已存在，請刪除資料表中繼資料和資料。如果刪除成功，則插入所有資料。如果資料表不存在，請建立資料表並插入所有資料。如果刪除未成功，則操作將會失敗。

進階選項

請參閱 AWS Glue 開發人員指南中的 Snowflake 連線。

在 AWS Glue Studio 中連線至 Teradata Vantage

AWS Glue 會提供 Teradata Vantage 的內建支援。AWS Glue Studio 會提供視覺化介面以連線至 Teradata、撰寫資料整合工作，以及在 AWS Glue Studio 無伺服器 Spark 執行期上執行此類工作。

主題

- 建立 Teradata Vantage 連線
- 建立 Teradata 來源節點
- 建立 Teradata 目標節點
- 進階選項

建立 Teradata Vantage 連線

要從中連接到 Teradata 華帝 AWS Glue，您需要在密碼中創建和存儲您的 Teradata 憑據，然後將該機 AWS Secrets Manager 密與 Teradata 連接相關聯。 AWS Glue

先決條件：

- 如果您要透過 Amazon VPC 存取 Teradata 環境，請設定 Amazon VPC 以允許您的 AWS Glue 任務與 Teradata 環境進行通訊。我們不建議透過公有網際網路存取 Teradata 環境。

  在 Amazon VPC 中，識別或建立 AWS Glue 將在執行任務時使用的 VPC、子網路和安全群組。此外，您也需要確保 Amazon VPC 已完成設定，以允許 Teradata 執行個體與此位置之間的網路流量。您的任務將需要與 Teradata 用戶端連接埠建立 TCP 連線。如需有關 Teradata 連接埠的詳細資訊，請參閱 Teradata 文件。

  根據您的網路配置，安全 VPC 連線可能需要變更 Amazon VPC 和其他網路服務。如需有關 AWS 連線的詳細資訊，請參閱 Teradata 文件中的AWS 連線選項。

若要設定 T AWS Glue eradata 連線：

1. *### Teradata ############### AWS Glue ## Teradata ########*如需詳細資訊，請參閱《Teradata 文件》中的 Vantage 安全概觀。

2. 在中 AWS Secrets Manager，使用您的 Teradata 認證建立密碼。若要在 Secrets Manager 中建立密碼，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 密碼中提供的教學課程。建立機密之後，請保留機密名稱 *secretName*，以便進行下一個步驟。

   - 在選取鍵/值組時，請使用 *teradataUsername* 值來建立 `user` 金鑰對。
   - 在選取鍵/值組時，請使用 *teradataPassword* 值來建立 `password` 金鑰對。

3. 在主 AWS Glue 控台中，依照中的步驟建立連線the section called “新增 AWS Glue 連線”。建立連線之後，請保留連線名稱 *connectionName*，以便進行下一個步驟。

   - 選取連線類型時，請選取 Teradata。
   - 提供 JDBC URL 時，請提供執行個體的 URL。您也可在 JDBC URL 中，針對特定逗號分隔的連線參數進行硬編碼。URL 必須符合下列格式：`jdbc:teradata://`*teradataHostname*`/`*ParameterName*`=`*ParameterValue*`,`*ParameterN*

     支援的 URL 參數包括：

     - `DATABASE`：依預設要存取之主機的資料庫名稱。
     - `DBS_PORT`：在非標準連接埠上執行時所使用的資料庫連接埠。
   - 選取憑證類型時，請選取 AWS Secrets Manager，然後將 AWS 密碼 設定為 *secretName*。

4. 在下列情況中，您可能需要其他組態：

   - 對於 AWS 在 Amazon VPC 上託管的 Teradata 執行個體
     - 您需要向定義 Teradata 安全登入資料的 AWS Glue 連線提供 Amazon VPC 連線資訊。建立或更新連線時，請在網路選項中設定 VPC、子網路及安全群組。

建立 Teradata 來源節點

必要先決條件

- 使用 AWS Secrets Manager 密碼設定的 AWS Glue Teradata Vantage 連線 (如上一節 the section called “建立 Teradata Vantage 連線” 中所述)。
- 針對您任務的適當許可，以讀取連線所使用的秘密。
- 您想要讀取的 Teradata 資料表 *tableName* 或查詢 *targetQuery*。

新增 Teradata 資料來源

新增資料來源 – Teradata 節點：

1. 選擇 Teradata 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立新連線。如需詳細資訊，請參閱前一 the section called “建立 Teradata Vantage 連線” 節。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。
2. 選擇 Teradata 來源選項：
   - 選擇單一資料表：從單一資料表存取所有資料。
   - 輸入自訂查詢：根據自訂查詢從多個資料表存取資料集。
3. 如果您選擇單一資料表，請輸入 *tableName*。

   如果您選擇輸入自訂查詢，請輸入 SQL SELECT 查詢。
4. 在自訂 Teradata 屬性中，視需要輸入參數和值。

建立 Teradata 目標節點

必要先決條件

- 使用 AWS Secrets Manager 密碼設定的 AWS Glue Teradata Vantage 連線 (如上一節 the section called “建立 Teradata Vantage 連線” 中所述)。
- 針對您任務的適當許可，以讀取連線所使用的秘密。
- 您想要寫入的 Teradata 資料表 *tableName*。

新增 Teradata 資料目標

新增資料目標 – Teradata 節點：

1. 選擇 Teradata 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 Teradata 連線。如需詳細資訊，請參閱 Overview of using connectors and connections 。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。
2. 透過提供 *tableName* 來設定資料表名稱。

3. 在自訂 Teradata 屬性中，視需要輸入參數和值。

進階選項

您可以在建立 Teradata 節點時提供進階選項。這些選項與為 AWS Glue for Spark 指令碼進行程式設計時的選項相同。

請參閱 the section called “Teradata Vantage 連線”。

在 AWS Glue Studio 中連線至 Vertica

AWS Glue 會提供 Vertica 的內建支援。AWS Glue Studio 會提供視覺化介面以連線至 Vertica、撰寫資料整合工作，以及在 AWS Glue Studio 無伺服器 Spark 執行期上執行此類工作。

主題

建立 Vertica 連線

先決條件：

- 從資料庫讀取和寫入資料庫時用於暫存空間的 Amazon S3 儲存貯體或資料夾 (稱為 *tempS3Path*)。

> **Note**
>
> 在 AWS Glue 工作資料預覽中使用 Vertica 時，可能無法從 *tempS3Path* 中自動移除暫存檔案。若要確保移除暫存檔案，請選擇資料預覽窗格中的結束工作階段，直接結束資料預覽工作階段。
> 如果無法保證資料預覽工作階段會直接結束，請考慮設定 Amazon S3 生命週期組態以移除舊資料。根據最長任務執行期和餘裕，我們建議移除超過 49 小時的資料。如需有關設定 Amazon S3 生命週期的詳細資訊，請參閱《Amazon S3 文件》中的管理您的儲存空間生命週期。

- 具有 Amazon S3 路徑適當許可的 IAM 政策可與您的 AWS Glue 任務角色建立關聯。

- 如果 Vertica 執行個體位於 Amazon VPC 中，請設定 Amazon VPC 以允許 AWS Glue 任務與 Vertica 執行個體進行通訊，使流量不會周遊公有網際網路。

  在 Amazon VPC 中，識別或建立 AWS Glue 將在執行任務時使用的 VPC、子網路及安全群組。此外，您也需要確保 Amazon VPC 已完成設定，以允許 Vertica 執行個體與此位置之間的網路流量。您的任務將需要與 Vertica 用戶端連接埠建立 TCP 連線 (預設 5433)。根據您的網路配置，這可能需要變更安全群組規則、網路 ACL、NAT 閘道及對等連線。

設定連至 Vertica 的連線：

1. 在 AWS Secrets Manager 中，使用 Vertica 憑證 *verticaUsername* 和 *verticaPassword* 建立密碼。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *secretName*，以便進行下一個步驟。

   - 在選取鍵/值組時，請使用 *verticaUsername* 值來建立 `user` 金鑰對。
   - 在選取鍵/值組時，請使用 *verticaPassword* 值來建立 `password` 金鑰對。

2. 在 AWS Glue 主控台中，依照 the section called “新增 AWS Glue 連線” 中的步驟建立連線。建立連線之後，請保留連線名稱 *connectionName*，以便進行下一個步驟。

   - 選取連線類型時，請選取 Vertica。
   - 選取 Vertica 主機後，請提供 Vertica 安裝的主機名稱。
   - 選取 Vertica 連接埠時，您可透過該連接埠安裝 Vertica。
   - 選取 AWS 機密時，請提供 *secretName*。

3. 在下列情況中，您可能需要其他組態：

   - Amazon VPC 中託管於 AWS 的 Vertica 執行個體
     - 向定義 Vertica 安全憑證的 AWS Glue 連線提供 Amazon VPC 連線資訊。建立或更新連線時，請在網路選項中設定 VPC、子網路及安全群組。

您將需要執行下列步驟，才能執行 AWS Glue 任務：

- 將與 AWS Glue 任務權限相關聯的 IAM 角色授予 *tempS3Path*。
- 授予與您 AWS Glue 任務許可相關聯的 IAM 角色，以讀取 *secretName*。

建立 Vertica 來源節點

必要先決條件

- Vertica 類型 AWS Glue Data Catalog 連線 *connectionName* 和 Amazon S3 暫存位置 *Temps3Path* (如上一節 the section called “建立 Vertica 連線” 中所述)。
- 您想要讀取的 Vertica 資料表 *tableName* 或查詢 *targetQuery*。

新增 Vertica 資料來源

新增資料來源 – Vertica 節點：

1. 選擇 Vertica 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 Vertica 連線。如需詳細資訊，請參閱前一 the section called “建立 Vertica 連線” 節。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。
2. 選擇包含資料表的資料庫。
3. 選擇 Amazon S3 中的暫存區域，將 S3A URI 輸入 *tempS3Path*。
4. 選擇 Verica 來源。
   - 選擇單一資料表：從單一資料表存取所有資料。
   - 輸入自訂查詢：根據自訂查詢從多個資料表存取資料集。
5. 如果您選擇單一資料表，請輸入 *tableName*，然後選擇性選取結構描述。

   如果您選擇輸入自訂查詢，請輸入 SQL SELECT 查詢，然後選擇性選取結構描述。
6. 在自訂 Vertica 屬性中，視需要輸入參數和值。

建立 Vertica 目標節點

必要先決條件

- Vertica 類型 AWS Glue Data Catalog 連線 *connectionName* 和 Amazon S3 暫存位置 *Temps3Path* (如上一節 the section called “建立 Vertica 連線” 中所述)。

新增 Vertica 資料目標

新增資料目標 – Vertica 節點：

1. 選擇 Vertica 資料來源的連線。由於您已建立連線，其應可用於下拉式清單中。如果您需要建立連線，請選擇建立 Vertica 連線。如需詳細資訊，請參閱前一 the section called “建立 Vertica 連線” 節。

   選擇連線之後，您可以按一下檢視屬性來檢視連線屬性。
2. 選擇包含資料表的資料庫。
3. 選擇 Amazon S3 中的暫存區域，將 S3A URI 輸入 *tempS3Path*。
4. 輸入 *tableName*，然後選擇性選取結構描述。
5. 在自訂 Vertica 屬性中，視需要輸入參數和值。

進階選項

您可以在建立 Vertica 節點時提供進階選項。這些選項與為 AWS Glue for Spark 指令碼進行程式設計時的選項相同。

請參閱 the section called “Vertica 連線”。

將連接器和連線與 AWS Glue Studio 搭配使用

AWS Glue 使用 JDBC 連線為最常用資料存放區 (例如 Amazon Redshift、Amazon Aurora、Microsoft SQL Server、MySQL、MongoDB 和 PostgreSQL) 提供內建支援。AWS Glue 也允許您在擷取、轉換和載入 (ETL) 任務中使用自訂 JDBC 驅動程式。對於原生不支援的資料存放區 (例如 SaaS 應用程式)，您可以使用連接器。

連接器是一個選用程式碼套件，可以協助存取 AWS Glue Studio 中的資料存放區。您可以訂閱 AWS Marketplace中提供的多個連接器。

建立 ETL 工作時，您可以使用原生支援的資料存放區、來 AWS Marketplace源的連接器或您自己的自訂連接器。如果您使用連接器，則必須先為連接器建立連線。連線包含連到特定資料存放區所需的屬性。您在 ETL 任務中將連線用於資料來源和資料目標。連接器和連線搭配運作，以方便存取資料存放區。

主題

- 使用連接器和連線的概觀

- 新增連接器至 AWS Glue Studio
- 可用的連線
- 建立連接器的連線
- 使用自訂連接器編寫任務
- 管理連接器和連線
- 開發自訂連接器
- 在 AWS Glue Studio 中使用連接器和連線的限制

使用連接器和連線的概觀

連線包含連到特定資料存放區所需的屬性。當您建立連線時，連線會儲存在 AWS Glue Data Catalog 中。您先選擇連接器，然後根據該連接器建立連線。

您可以在中訂閱非原生支援的資料存放區的連接器 AWS Marketplace，然後在建立連線時使用這些連接器。開發人員也可以建立自己的連接器，而且您可以在建立連線時使用它們。

> **Note**
>
> 使用自訂或中的連 AWS Marketplace 接器建立的連線AWS Glue Studio會顯示在類型設定為的AWS Glue控制台中UNKNOWN。

下列步驟說明在 AWS Glue Studio 中使用連接器的整體程序：

1. 在中訂閱連接器 AWS Marketplace，或開發您自己的連接器並將其上傳到AWS Glue Studio。如需詳細資訊，請參閱 新增連接器至 AWS Glue Studio。
2. 檢閱連接器使用資訊。您可以在連接器產品頁面的 Usage (用途) 索引標籤上找到此資訊。例如，如果您按一下此產品頁面上的 [使用量] 索引標籤 [Google AWS Glue 連接器] BigQuery，您可以在 [其他資源] 區段中看到有關使用此連接器的部落格連結。其他連接器可能包含指向 Overview (概觀) 區段的指示連結，如 Cloudwatch Logs connector for AWS Glue 連接器產品頁面上所示。
3. 建立連線。您可以選擇要使用哪個連接器，並提供連線的額外資訊，例如登入憑證、URI 字串和虛擬私有雲端 (VPC) 資訊。如需詳細資訊，請參閱 建立連接器的連線。
4. 為您的任務建立 IAM 角色。任務會承擔您在建立 IAM 角色時所指定的角色許可。這個 IAM 角色必須具有必要許可，才能對資料存放區進行驗證、從中擷取資料，以及寫入資料。
5. 建立 ETL 任務並設定 ETL 任務的資料來源屬性。依照自訂連接器提供者的指示，提供連線選項和驗證資訊。如需詳細資訊，請參閱 使用自訂連接器編寫任務。

6. 透過新增轉換或其他資料存放區來自訂您的 ETL 任務，如使用 AWS Glue Studio 視覺化 ETL中所述。
7. 如果將連接器用於資料目標，請為 ETL 任務設定資料目標屬性。依照自訂連接器提供者的指示，提供連線選項和驗證資訊。如需詳細資訊，請參閱 the section called “使用自訂連接器編寫任務”。
8. 藉由設定任務屬性來自訂任務執行環境，如修改任務屬性中所述。
9. 執行任務。

新增連接器至 AWS Glue Studio

連接器是一段程式碼，可促進資料存放區和 AWS Glue 之間的通訊。您可以訂閱中提供的連接器 AWS Marketplace，也可以建立自己的自訂連接器。

主題

- 訂閱連接器 AWS Marketplace
- 建立自訂連接器

訂閱連接器 AWS Marketplace

AWS Glue Studio可以輕鬆地從中添加連接器 AWS Marketplace。

將連接器從加入 AWS Marketplace 至 AWS Glue Studio

1. 在 AWS Glue Studio 主控台中，在主控台導覽窗格中選擇 Connectors (連接器)。
2. 在連接器頁面上，選擇移至 AWS Marketplace。
3. 在 AWS Marketplace「精選產品」中，選擇您要使用的連接器。您可以選擇其中一個精選連接器，或使用搜尋。您可以搜尋連接器的名稱或類型，也可以使用選項來縮小搜尋結果。

   如果您想要使用其中一個精選連接器，請選擇 View product (檢視產品)。如果您使用搜尋來尋找連接器，請選擇連接器的名稱。
4. 在連接器的產品頁面上，使用索引標籤來檢視連接器的相關資訊。如果您決定購買此連接器，請選擇 Continue to Subscribe (繼續訂閱)。
5. 提供付款資訊，然後選擇 Continue to Configure (繼續進行設定)。
6. 在 Configure this software (設定此軟體) 頁面上，選擇部署方法和要使用的連接器版本。然後選擇 Continue to Launch (繼續啟動)。
7. 在 Launch this software (啟動此軟體) 頁面上，您可以檢閱由連接器提供者所提供的 Usage Instructions (使用指示)。當您準備好繼續時，請選擇 在 AWS Glue Studio 中啟用連線。

一段時間後，主控台會在 AWS Glue Studio 中顯示 Create marketplace connection (建立市集連線) 頁面。

8. 建立使用此連接器的連線，如建立連接器的連線中所述。

   或者，您也可以選擇 Activate connector only (僅啟動連接器)，在此時略過建立連線。您稍後必須建立連線，才能使用連接器。

建立自訂連接器

您也可以建置自己的連接器，然後將連接器程式碼上傳到 AWS Glue Studio。

自訂連接器會透過 AWS Glue Spark 執行時間 API 整合至 AWS Glue Studio。AWS Glue Spark 執行時間可讓您插入任何符合 Spark、Athena 或 JDBC 介面的連接器。它可讓您傳入任何可用於自訂連接器的連線選項。

您可以使用 AWS Glue 連線封裝所有連線屬性並提供連線名稱給您的 ETL 任務。與 Data Catalog 連線整合可讓您在單一 Spark 應用程式或不同應用程式之間，跨多個呼叫使用相同的連線屬性。

您可以為連線指定其他選項。AWS Glue Studio 產生的任務指令碼包含 `Datasource` 項目，該項目使用連線以指定的連線選項插入連接器。例如：

```
Datasource = glueContext.create_dynamic_frame.from_options(connection_type =
"custom.jdbc", connection_options = {"dbTable":"Account","connectionName":"my-custom-
jdbc-
connection"}, transformation_ctx = "DataSource0")
```

將自訂連接器新增至 AWS Glue Studio

1. 為您的自訂連接器建立程式碼。如需詳細資訊，請參閱 開發自訂連接器。
2. 新增對 AWS Glue 功能的支援至您的連接器。以下是這些功能的一些範例，以及如何在 AWS Glue Studio 產生的任務指令碼中使用它們：

   - 資料類型映射 – 您的連接器可以在從基礎資料存放區讀取欄時對欄進行類型轉換。例如，當剖析記錄並建構 `DynamicFrame` 時，`{"INTEGER":"STRING"}` 的 `dataTypeMapping` 會將類型 `Integer` 的所有欄轉換為類型 `String` 的欄。這可以協助使用者將欄轉換為他們選擇的類型。

     ```
     DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type
     = "custom.jdbc", connection_options = {"dataTypeMapping":{"INTEGER":"STRING"}",
     ```

```
connectionName":"test-connection-jdbc"}, transformation_ctx = "DataSource0")
```

- 並行讀取的分割 – AWS Glue 允許透過分割欄上的資料，從資料存放區並行讀取資料。您必須指定分割區欄、分割區下界、分割區上限，以及分割區的數目。此功能讓您能夠利用資料並行性和分配給 Spark 應用程式的多個 Spark 執行器。

```
DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type
= "custom.jdbc", connection_options = {"upperBound":"200","numPartitions":"4",
"partitionColumn":"id","lowerBound":"0","connectionName":"test-connection-jdbc"},
transformation_ctx = "DataSource0")
```

- 用 AWS Secrets Manager 於儲存認證 — 資料目錄連線也可以包`secretId`含儲存在中的密碼 AWS Secrets Manager。 AWS 密碼可以安全地存儲身份驗證和認證信息，並 AWS Glue 在運行時將其提供給。或者，您也可以從 Spark 指令碼指定 `secretId`，如下所示：

```
DataSource = glueContext.create_dynamic_frame.from_options(connection_type
= "custom.jdbc", connection_options = {"connectionName":"test-connection-jdbc",
 "secretId"-> "my-secret-id"}, transformation_ctx = "DataSource0")
```

- 使用列述詞和欄投影篩選來源資料 – AWS Glue Spark 執行時間也允許使用者下推 SQL 查詢，使用使用列述詞和欄投影在來源篩選資料。這可讓您的 ETL 任務更快從支援下推的資料存放區載入篩選過的資料。下推到 JDBC 資料來源的範例 SQL 查詢是：`SELECT id, name, department FROM department WHERE id < 200.`

```
DataSource = glueContext.create_dynamic_frame.from_options(connection_type =
"custom.jdbc", connection_options = {"query":"SELECT id, name, department FROM
 department
WHERE id < 200","connectionName":"test-connection-jdbc"}, transformation_ctx =
"DataSource0")
```

- 任務書籤 – AWS Glue 支援從 JDBC 來源增量載入資料。AWS Glue 會追蹤資料存放區的最後處理記錄，並在後續的 ETL 任務執行中處理新的資料記錄。任務書籤使用主索引鍵作為書籤索引鍵的預設欄，前提是此欄會按順序新增或減少。如需任務書籤的詳細資訊，請參閱 AWS Glue 開發人員指南中的任務書籤。

```
DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
"custom.jdbc", connection_options = {"jobBookmarkKeys":["empno"],
 "jobBookmarkKeysSortOrder"
:"asc", "connectionName":"test-connection-jdbc"}, transformation_ctx =
 "DataSource0")
```

3. 將自訂連接器封裝為 JAR 檔案，然後將檔案上傳到 Amazon S3。
4. 測試您的自訂連接器。如需詳細資訊，請參閱 Glue GitHub 自訂連接器：本機驗證測試指南中的說明。
5. 在 AWS Glue Studio 主控台中，在主控台導覽窗格中選擇 Connectors (連接器)。
6. 在 Connectors (連接器) 頁面上，選擇 Create custom connector (建立自訂連接器)。
7. 在 Create custom connector (建立自訂連接器) 頁面上，輸入以下資訊：
   - 指向 Amazon S3 中自訂程式碼 JAR 檔案位置的路徑。
   - AWS Glue Studio 將使用的連接器的名稱。
   - 您的連接器類型，可以是 JDBC、Spark, 或 Athena。
   - 您自訂程式碼中的入口點名稱，AWS Glue Studio 將呼叫以使用連接器。
     - 對於 JDBC 連接器，此欄位應該是 JDBC 驅動程式的類別名稱。
     - 對於 Spark 連接器，此欄位應該是當您使用 `format` 運算子載入 Spark 資料來源時使用的完整資料來源類別名稱或其別名。
   - (僅限 JDBC) 資料存放區的 JDBC 連線使用的基本 URL。
   - (選用) 自訂連接器的描述。
8. 選擇 Create connector (建立連接器)。
9. 從 Connectors (連接器) 頁面，建立使用此連接器的連線，如建立連接器的連線中所述。

可用的連線

建立連接器的連線時，可使用下列連線：

- Amazon Aurora：可擴展的高效能關聯式資料庫引擎，具備內建安全、備份與還原，以及記憶體加速功能。
- Amazon DocumentDB：可擴展、高可用性及全受管的文件資料庫服務，支援 MongoDB 和 SQL API。
- Amazon Redshift：可擴展、高可用性及全受管的文件資料庫服務，支援 MongoDB 和 SQL API。
- Azure SQL：Microsoft Azure 的雲端型關聯式資料庫服務，能提供可擴展、可靠及安全的資料儲存和管理功能。
- Cosmos DB：Microsoft Azure 的全域分佈雲端資料庫服務，能提供可擴展、高效能的資料儲存和查詢功能。
- Google BigQuery — 無伺服器雲端資料倉儲，可在大型資料集上執行快速 SQL 查詢。

- JDBC：關聯式資料庫管理系統 (RDBMS)，可使用 Java API 進行連線和與資料連線互動。
- Kafka：用於即時資料串流和傳訊的開放原始碼串流處理平台。
- MariaDB：社群開發的 MySQL 分支，可提供增強的效能、可擴展性及功能。
- MongoDB：跨平台的文件導向資料庫，可提供高擴展性、彈性及效能。
- MongoDB Atlas：MongoDB 提供的雲端型資料庫即服務 (DBAs) 產品，可簡化 MongoDB 部署的管理和擴展。
- Microsoft SQL Server：Microsoft 的關聯式資料庫管理系統 (RDBMS)，可提供強大的資料儲存，分析及報告功能。
- MySQL：開放原始碼關聯式資料庫管理系統 (RDBMS)，可廣泛用於 Web 應用程式，並以其可靠性和可擴展性聞名。
- 網路：網路資料來源表示可由資料整合平台存取的網路存取資源或服務。
- OpenSearch— OpenSearch 資料來源是 OpenSearch 可以連線到資料並從中擷取資料的應用程式。
- Oracle：Oracle Corporation 的關聯式資料庫管理系統 (RDBMS)，可提供強大的資料儲存，分析及報告功能。
- PostgreSQL：開放原始碼關聯式資料庫管理系統 (RDBMS)，可提供強大的資料儲存，分析及報告功能。
- Salesforce — Salesforce 提供客戶關係管理 (CRM) 軟體，可協助您進行銷售、客戶服務、電子商務等。如果您是 Salesforce 使用者，您可以連線 AWS Glue 到您的 Salesforce 帳戶。然後，您可以使用 Salesforce 做為 ETL 工作中的資料來源或目的地。執行這些工作以在 Salesforce 與 AWS 服務或其他支援的應用程式之間傳輸資料。
- SAP HANA：記憶體資料庫和分析平台，可提供快速的資料處理、進階分析和即時資料整合。
- Snowflake：雲端型資料倉儲，可提供可擴展、高效能的資料儲存和分析服務。
- Teradata：關聯式資料庫管理系統 (RDBMS)，可提供高效能的資料儲存，分析及報告功能。
- Vertica：專為大數據分析而設計的欄式導向分析資料倉儲，可提供快速的查詢效能、進階分析和可擴展性。

建立連接器的連線

AWS Glue 連線是針對特定資料存放區儲存連線資訊的 Data Catalog 物件。連線會儲存登入憑證、URI 字串、虛擬私有雲端 (VPC) 資訊等。在 Data Catalog 中建立連線，就可輕鬆指定每次您建立工作時的所有連線詳細資料。

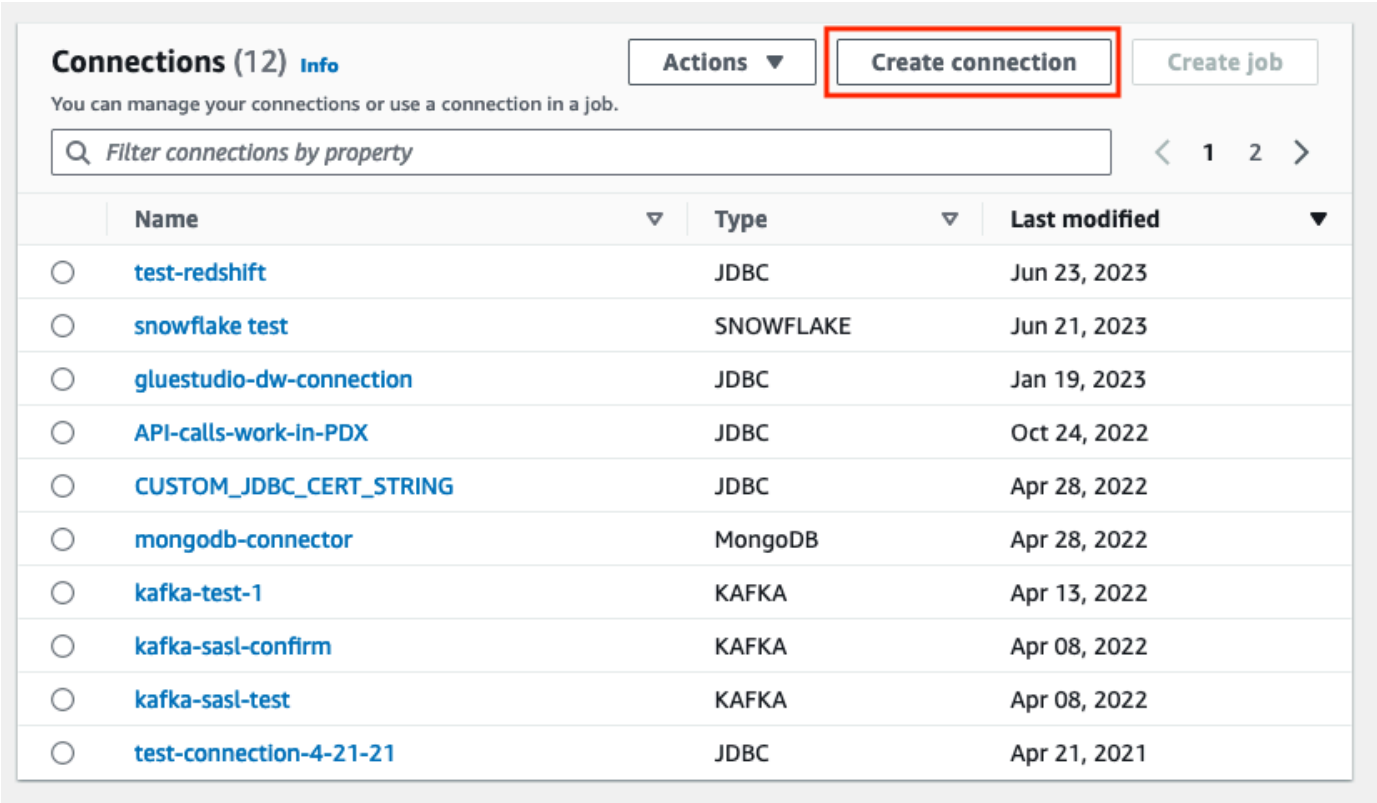
建立連接器的連線

1. 在 AWS Glue Studio 主控台中，在主控台導覽窗格中選擇 Connectors (連接器)。在連線區段中，選擇建立連線。
2. 在建立資料連線精靈的步驟 1 中，選擇您想要建立連線的資料來源。以下提供多種檢視可用資料來源的方法，包含：

   - 透過選擇索引標籤篩選可用的資料來源。預設會選取所有連接器。
   - 切換清單即可以清單形式檢視資料來源，或切換回網格即可在網格版面中檢視可用的連接器。
   - 使用搜尋列來縮小資料來源清單範圍。當您輸入時，系統會顯示搜尋相符項目，並從檢視中移除不相符的來源。

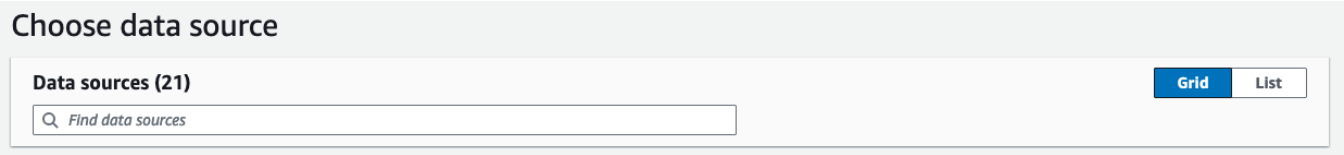
   選擇資料來源後，請選擇下一步。
3. 在精靈的步驟 2 中設定連線。

   輸入連線詳細資訊。視您選取的連接器類型而定，系統會提示您輸入其他資訊：



4. 在建立資料連線精靈的步驟 1 中，選擇您想要建立連線的資料來源。以下提供多種檢視可用資料來源的方法。依預設，您將會在網格版面中看到所有可用的資料來源。您也可以：

- 切換清單即可以清單形式檢視資料來源，或切換回網格即可在網格版面中檢視可用的連接器。
- 使用搜尋列來縮小資料來源清單範圍。當您輸入時，系統會顯示搜尋相符項目，並從檢視中移除不相符的來源。

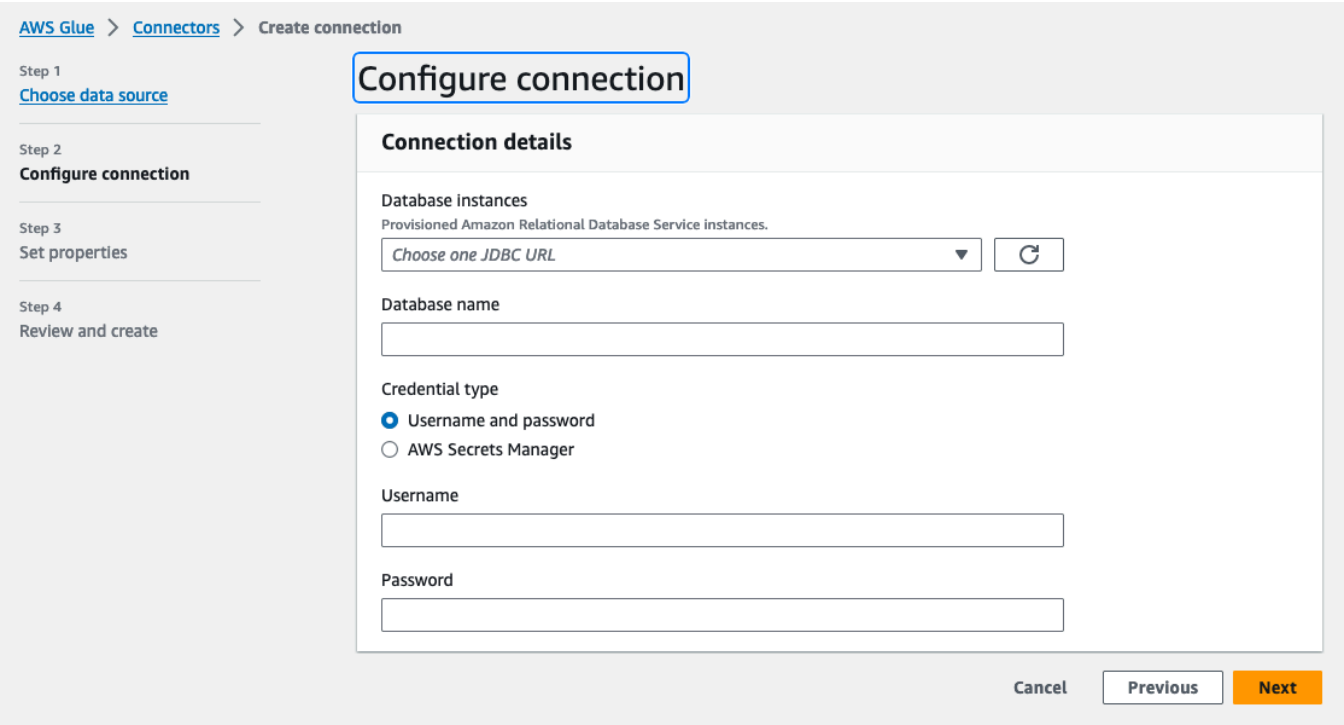


選擇資料來源後，請選擇下一步。

5. 在精靈的步驟 2 中設定連線。

輸入連線詳細資訊。視您選取的連接器類型而定，系統可能會要求您輸入其他連線資訊。其中包括：

- 連線詳細資料：這些欄位將會根據您要連線的資料來源而變更。例如，如果您要連線至 Amazon DocumentDB 資料庫，則要輸入 Amazon DocumentDB URL。如果您要連線至 Amazon Aurora，則要選擇資料庫執行個體並輸入資料庫名稱。下列為 Amazon Aurora 所需的連線詳細資料：



- 憑證類型：選擇使用者名稱和密碼或 AWS Secrets Manager。輸入要求的驗證資訊。
- 對於使用 JDBC 的連接器，請輸入為資料存放區建立 JDBC URL 所需的資訊。

- 如果您使用虛擬私有雲端 (VPC)，請輸入 VPC 的網路資訊。

6. 在精靈的步驟 3 中設定連線屬性。您可以新增說明和標籤作為此步驟的選用部分。名稱為必填項目，會預先填入預設值。選擇 Next (下一步)。
7. 檢閱連線來源、詳細資料及屬性。如果您需要進行任何變更，請針對精靈中的步驟選擇編輯。準備就緒後，請選擇建立連線。

   選擇 Create connection (建立連線)。

   您將返回 Connectors (連接器) 頁面，而且資訊橫幅會指出已建立的連線。您現在可以在 AWS Glue Studio 任務中使用連線。

建立 Kafka 連線

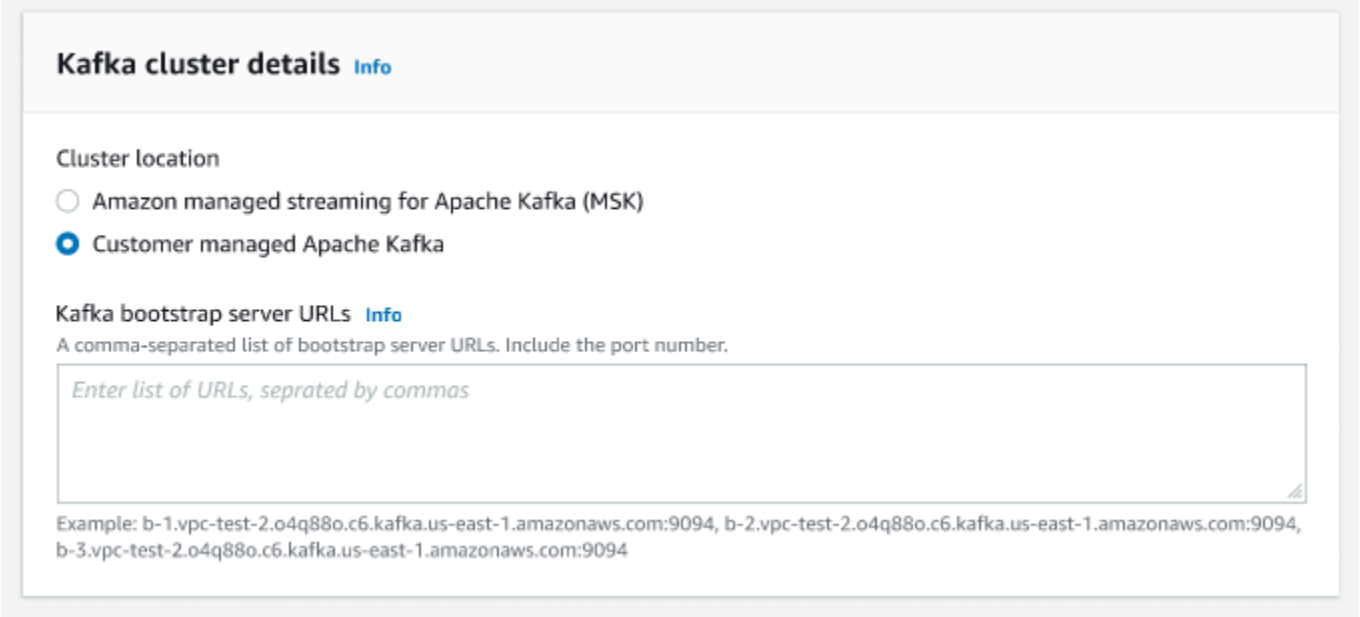
建立 Kafka 連線時，從下拉式選單中選取 Kafka 將會顯示要設定的其他設定：

- Kafka 叢集詳細資訊
- 身分驗證
- 加密
- 網路選項

設定 Kafka 叢集詳細資訊

1. 選擇叢集位置。您可以選擇 Amazon managed streaming for Apache Kafka (MSK) 叢集或客戶受管的 Apache Kafka 叢集。如需 Amazon Managed streaming for Apache Kafka 的詳細資訊，請參閱 Amazon managed streaming for Apache Kafka (MSK)。

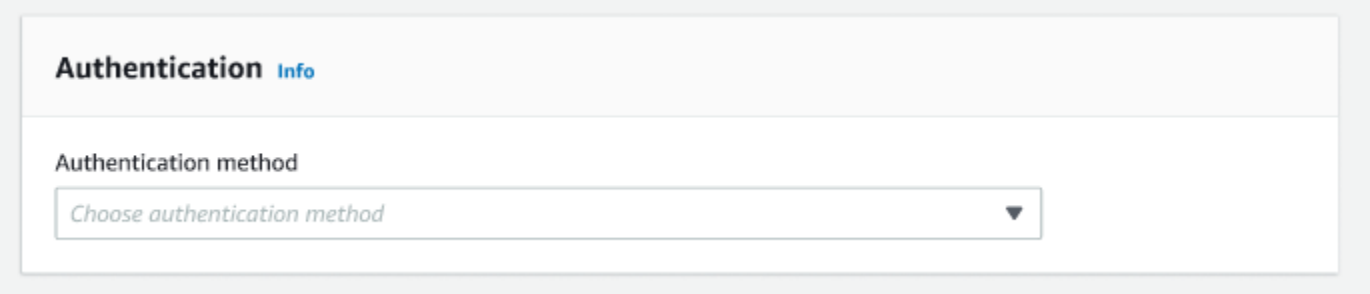
   > **Note**
   >
   > Amazon Managed Streaming for Apache Kafka 僅支援 TLS 及 SASL/SCRAM-SHA-512 驗證方法。



2. 輸入您的 Kafka Bootstrap 伺服器的 URL。您可以用逗號分隔每個伺服器，進而輸入多個伺服器。附加 `:<port number>` 即可在 URL 的結尾處加上連接埠號碼，

   例如：`b-1.vpc-test-2.034a88o.kafka-us-east-1.amazonaws.com:9094`

選取身分驗證方法

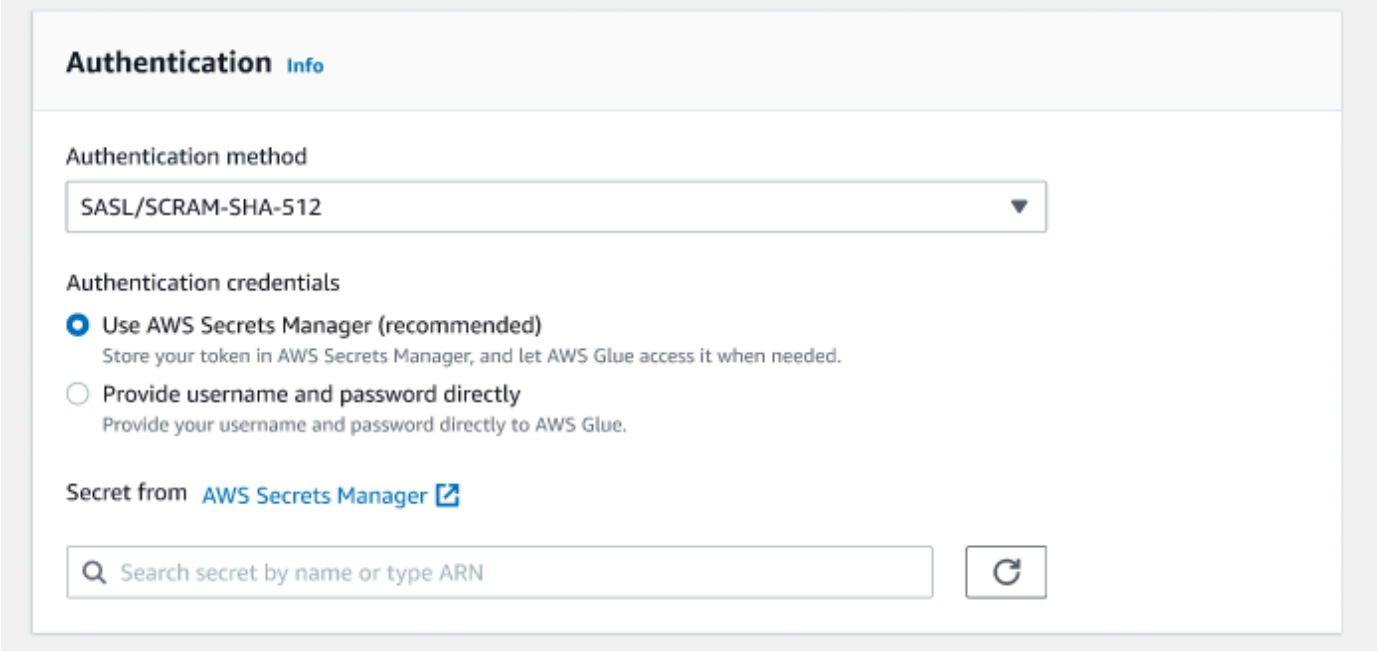


AWS Glue 支援 Simple Authentication and Security Layer (SASL) 架構來進行身分驗證。SASL 框架支持各種身份驗證機制，並 AWS Glue 提供了 SCRAM（用戶名和密碼），GSSAPI（Kerberos 協議）和普通（用戶名和密碼）協議。

從下拉式選單中選擇身分驗證方法時，可以選取以下用戶端身分驗證方法：

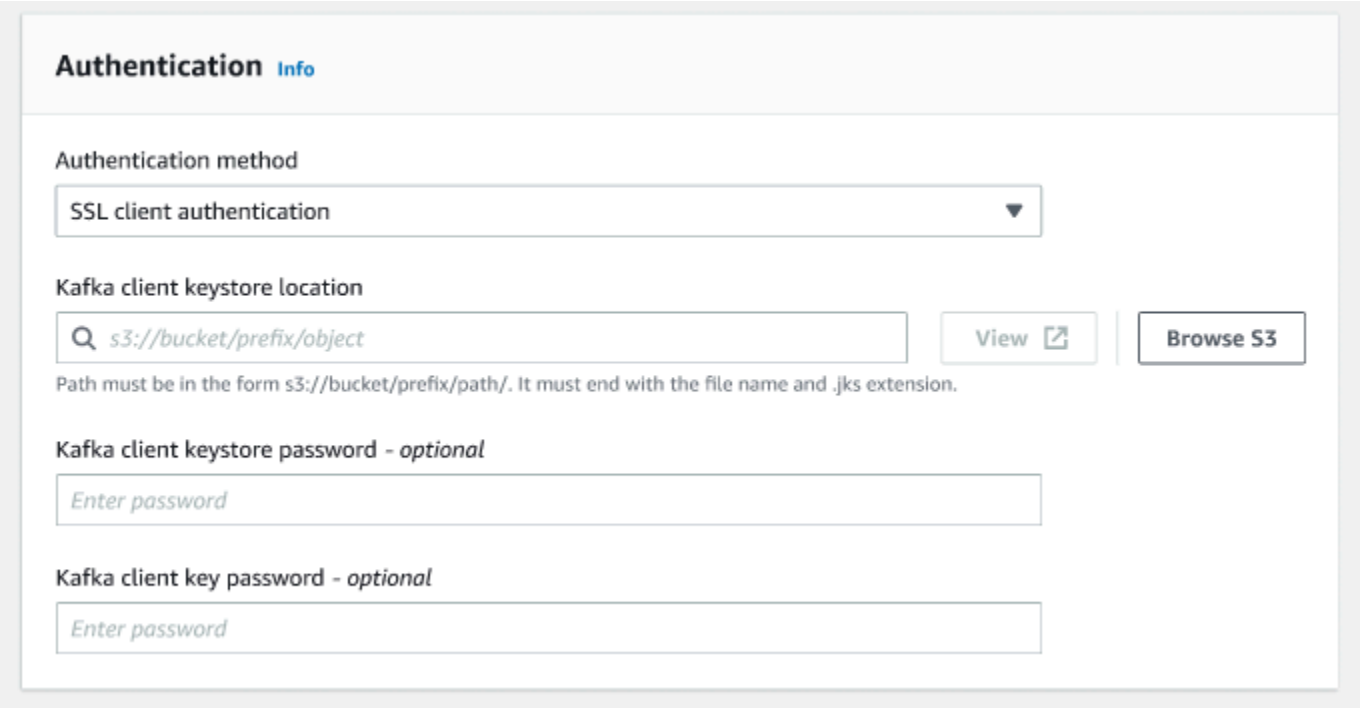
- 無 - 無身分驗證。如果為進行測試而建立連線，此方法會很有用。
- SASL/SCRAM-SHA-512 – 選擇此驗證方法來指定驗證憑證。有兩種可用選項：

- 使用 AWS Secrets Manager (建議使用)-如果您選取此選項，您可以將您的認證儲存在 AWS Secrets Manager 中，並在需要時AWS Glue存取這些資訊。指定存放 SSL 或 SASL 驗證憑證的秘密。



  - 直接提供用戶名和密碼。

- SASL/GSSAPI (Kerberos) - 如果您選取此選項，則可以選取 keytab 檔案和 krb5.conf 檔案的位置，然後輸入 Kerberos 主體名稱和 Kerberos 服務名稱。keytab 檔案和 krb5.conf 檔案的位置必須位於 Amazon S3 位置。由於 MSK 尚不支援 SASL/GSSAPI，此選項僅適用於客戶受管的 Apache Kafka 叢集。如需詳細資訊，請參閱 MIT Kerberos 文件：Keytab。
- SASL/PLAIN-選擇此驗證方法來指定認證證明資料。有兩種可用選項：
  - 使用 AWS Secrets Manager (建議使用)-如果您選取此選項，您可以將您的認證儲存在 AWS Secrets Manager 中，並在需要時 AWS Glue 存取這些資訊。指定存放 SSL 或 SASL 驗證憑證的秘密。
  - 直接提供用戶名和密碼。
- SSL 用戶端身分驗證 - 如果您選取此選項，則可以透過瀏覽 Amazon S3 來選取 Kafka 用戶端金鑰存放區的位置。或者，您可以輸入 Kafka 用戶端金鑰存放區密碼和 Kafka 用戶端金鑰密碼。



設定加密設定

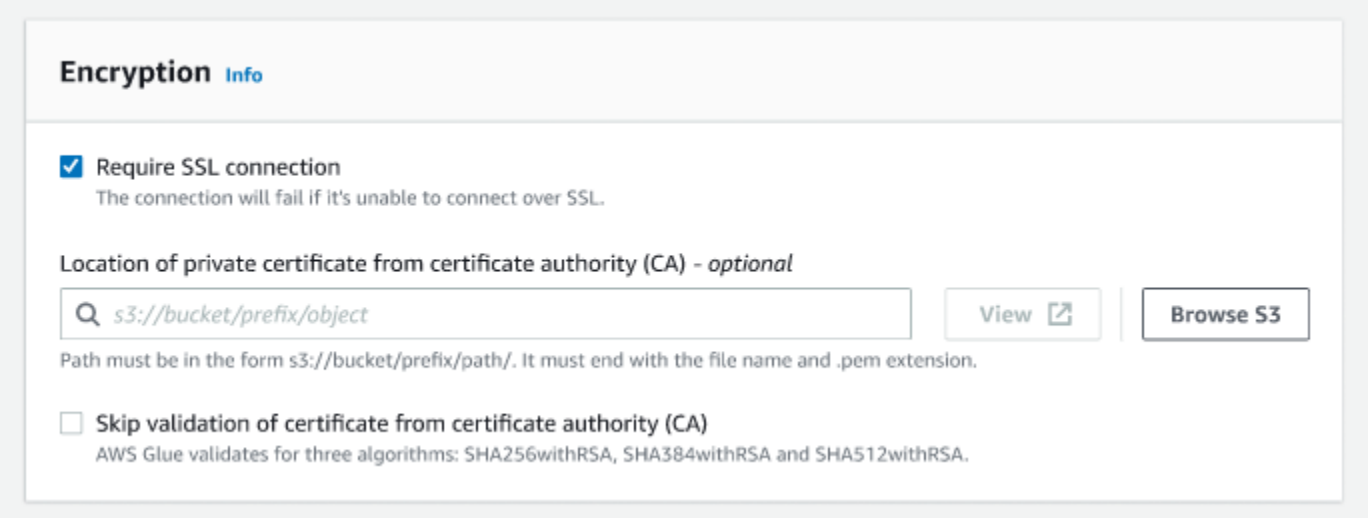
1. 如果 Kafka 連線需要 SSL 連線，請選取 Require SSL connection (需要 SSL 連線) 核取方塊。請注意，如果連線無法透過 SSL 連接，則連線將會失敗。用於加密的 SSL 可以與任何一種驗證方法 (SASL/SCRAM-SHA-512、SASL/GSSAPI、SASL/ 普通或 SSL 用戶端驗證) 搭配使用，而且是選用的。

   如果身分驗證方法設定為 SSL client authentication (SSL 用戶端身分驗證)，則將會自動選取此選項，並將其停用以防止任何變更。

2. (選用)。從憑證授權機構 (CA) 中，選擇私有憑證的位置。請注意，憑證必須位於 S3 位置。選擇 Browse (瀏覽) 從連接的 S3 儲存貯體中選擇檔案。該路徑的格式必須是 `s3://bucket/prefix/filename.pem`。其檔案名稱結尾必須是 .pem 副檔名。

3. 您可以選擇略過憑證授權機構 (CA) 的憑證驗證。選擇 Skip validation of certificate from certificate authority (CA) (略過憑證授權機構 (CA) 的憑證驗證) 核取方塊。如果未勾選此方塊，則 AWS Glue 會驗證三種演算法的憑證：

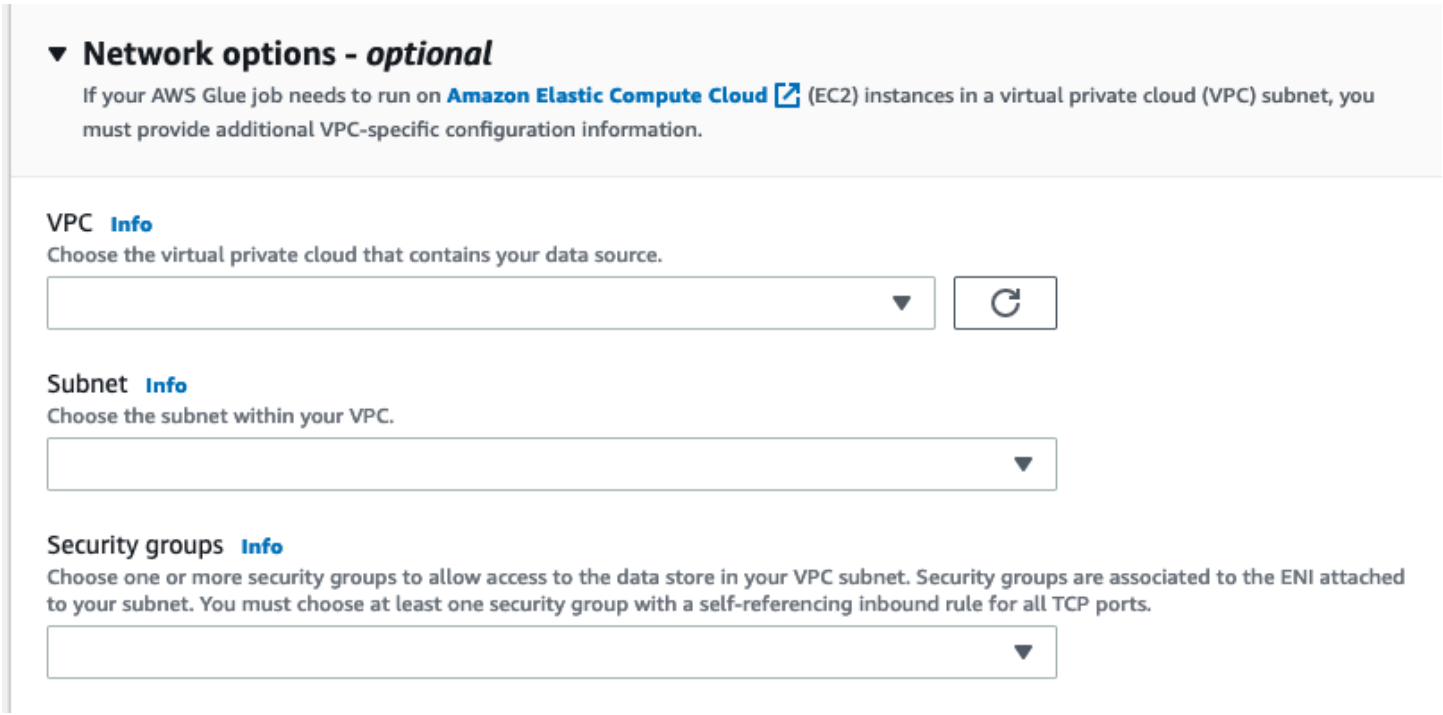
   - SHA256withRSA
   - SHA384withRSA
   - SHA512withRSA



(選用) 網路選項

以下是設定 VPC、子網路和安全群組的可選步驟。如果您的 AWS Glue 任務需要在虛擬私有雲端 (VPC) 子網路中的 Amazon EC2 執行個體上執行，您必須提供其他 VPC 特定的組態資訊。

1. 選擇包含您的資料來源的 VPC (虛擬私有雲端)。
2. 選擇 VPC 中的子網路。
3. 選擇一個或多個允許存取 VPC 子網路中資料存放區的安全群組。安全群組與連接到子網路的 ENI 相關聯。您必須至少選擇一個安全群組並為所有 TCP 連接埠建立自我引用的傳入規則。



使用自訂連接器編寫任務

您可以針對 AWS Glue Studio 中的資料來源節點和資料目標節點，使用連接器和連線。

主題

- 建立將連接器用於資料來源的任務
- 為使用連接器的節點設定來源屬性
- 為使用連接器的節點設定目標屬性

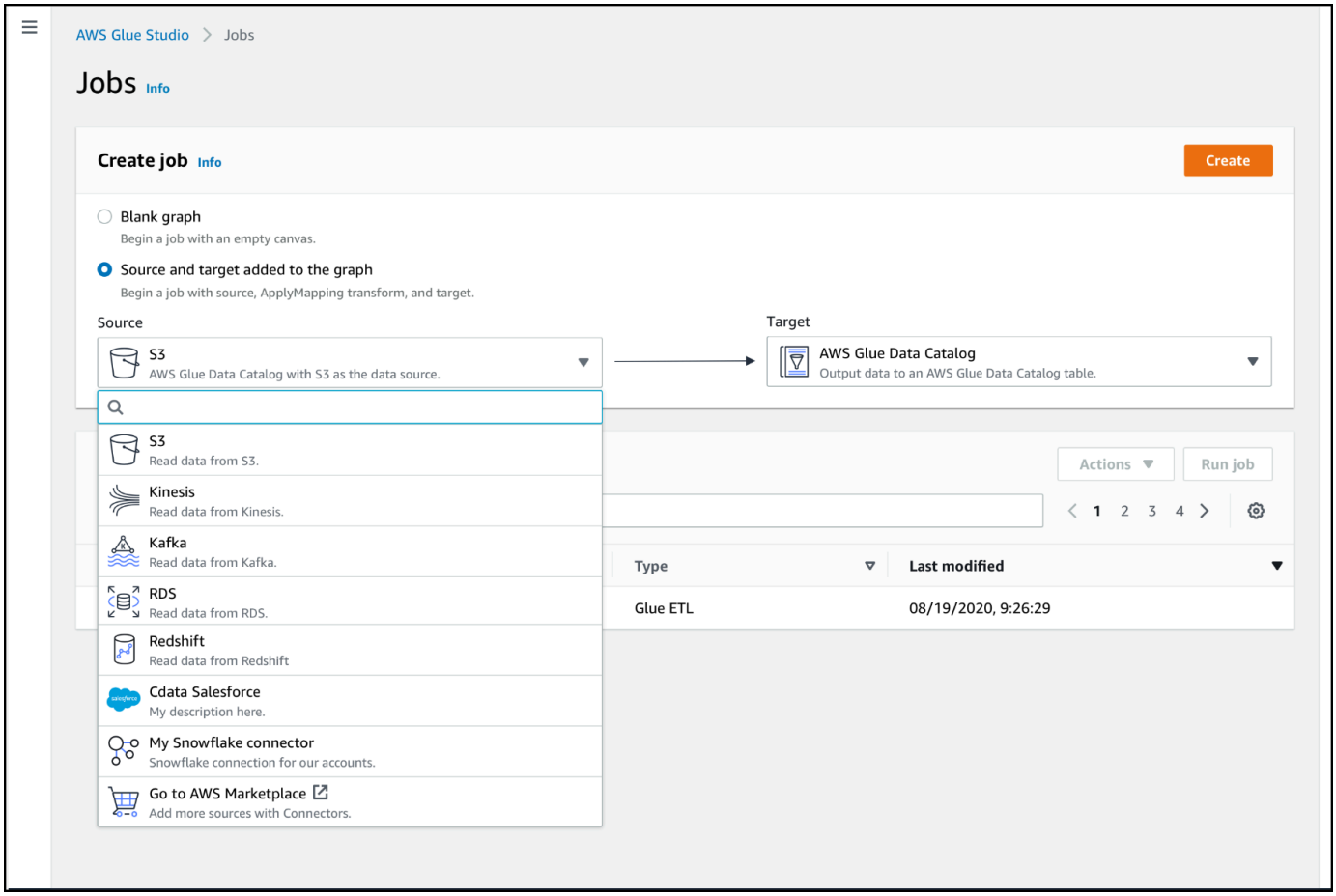
建立將連接器用於資料來源的任務

建立新任務時，您可以為資料來源和資料目標選擇連接器。

建立將連接器用於資料來源或資料目標的任務

1. 請登入 AWS Management Console 並開啟AWS Glue Studio主控台，網址為 https://console.aws.amazon.com/gluestudio/。
2. 在 Connectors (連接器) 頁面 Your connections (您的連線) 資源清單中，選擇您想要在任務中使用的連線，然後選擇 Create job (建立任務)。

或者，在 AWS Glue Studio Jobs (任務) 頁面的 Create job (建立任務) 下，選擇 Source and target added to the graph (新增至圖表的來源和目標)。在 Source (來源) 下拉式清單中，選擇您想要在任務中使用的自訂連接器。您也可以為 Target (目標) 選擇連接器。



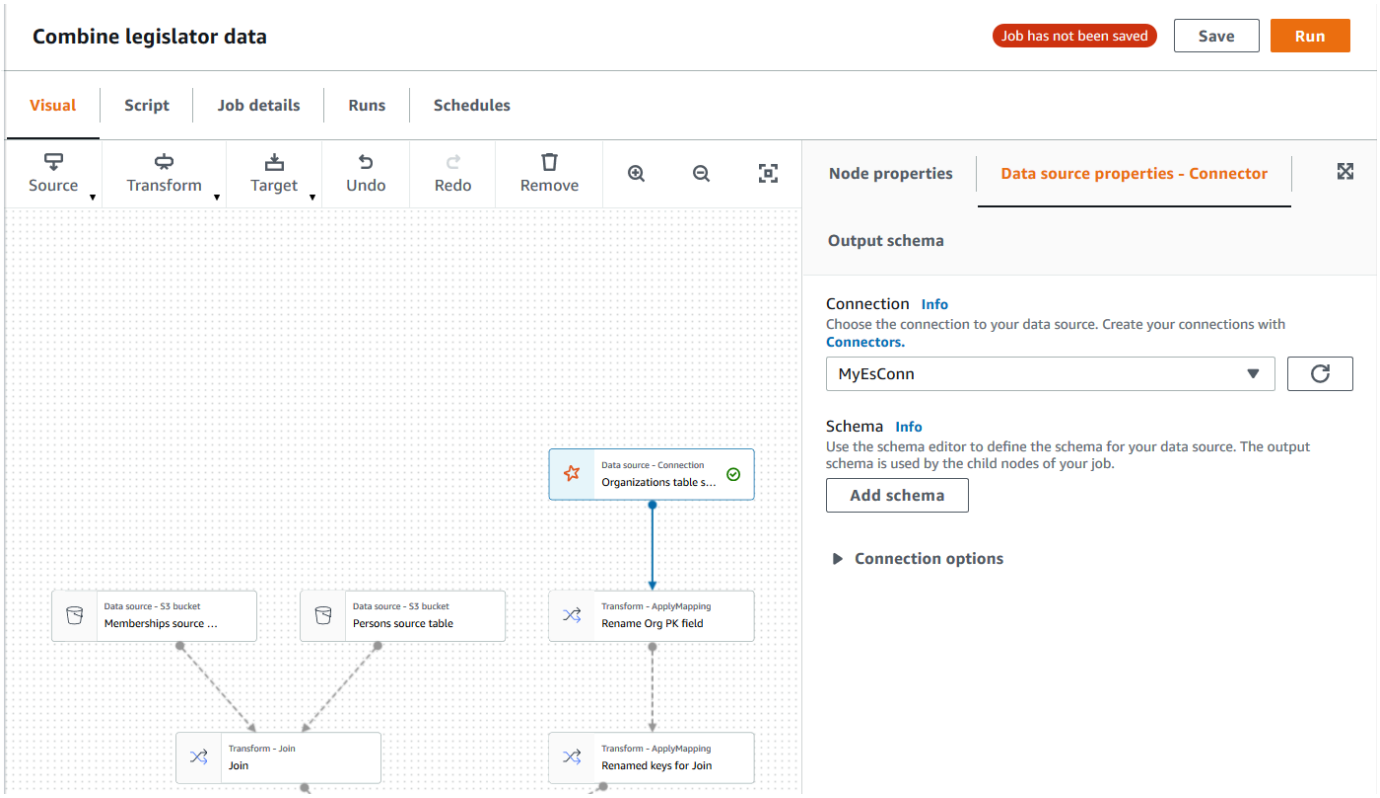
3. 選擇 Create (建立) 以開啟視覺化任務編輯器。
4. 設定資料來源節點，如為使用連接器的節點設定來源屬性中所述。
5. 透過新增轉換、其他資料存放區和資料目標來繼續建立 ETL 任務，如使用 AWS Glue Studio 視覺化 ETL中所述。
6. 藉由設定任務屬性來自訂任務執行環境，如修改任務屬性中所述。
7. 儲存並執行任務。

為使用連接器的節點設定來源屬性

建立將連接器用於資料來源的任務後，視覺化任務編輯器會顯示任務圖，其中包含為連接器設定的資料來源節點。您必須設定該節點的資料來源屬性。

為使用連接器的資料來源節點設定屬性

1. 在任務圖中選擇連接器資料來源節點，或新增節點，然後選擇 Node type (節點類型) 的連接器。然後，在右側的節點詳細資訊面板中，選擇 Data source properties (資料來源屬性) 索引標籤 (如果尚未選取)。



2. 在 Data source properties (資料來源屬性) 索引標籤上，選擇要用於此任務的連線。

   輸入每個連線類型所需的其他資訊：

   JDBC

   - Data source input type (資料來源輸入類型)：選擇提供資料表名稱或 SQL 查詢做為資料來源。根據您的選擇，您接著必須提供下列額外資訊：
     - Table name (資料表名稱)：資料來源中的資料表名稱。如果資料來源不使用術語表格，請提供適當資料結構的名稱，如自訂連接器使用資訊 (可在中找到 AWS Marketplace) 所示。
     - Filter predicate (篩選述詞)：讀取資料來源時要使用的條件子句，類似於 WHERE 子句，用來擷取資料的子集。
     - Query code (查詢程式碼)：輸入 SQL 查詢，以用來從資料來源擷取特定資料集。基本的 SQL 查詢範例是：

```
SELECT column_list FROM
                            table_name WHERE where_clause
```

- Schema (結構描述)：由於 AWS Glue Studio 使用儲存在連線中的資訊來存取資料來源，而不是從 Data Catalog 資料表中擷取中繼資料資訊，因此您必須提供資料來源的結構描述中繼資料。選擇 Add schema (新增結構描述) 開啟結構描述編輯器。

  如需如何使用結構描述編輯器的指示，請參閱編輯自訂轉換節點的結構描述。

- Partition column (分割區欄)：(選用) 您可以選擇分割資料讀取，方法是提供 Partition column (分割區欄)、Lower bound (下限)、Upper bound (上限)，以及 Number of partitions (分割區數量)。

  `lowerBound` 和 `upperBound` 值用於決定分割區步幅，而不是用於篩選資料表中的列。資料表中的所有列都會進行分割並傳回。

  > **Note**
  >
  > 欄分割為用於讀取資料的查詢新增一個額外的分割條件。使用查詢而不是資料表名稱時，您應該驗證查詢是否適用於指定的分割條件。例如：
  >
  > - 如果您的查詢格式為 `"SELECT col1 FROM table1"`，則透過在使用分割區欄的查詢結尾附加 `WHERE` 子句來測試查詢。
  > - 如果您的查詢格式為 `"SELECT col1 FROM table1 WHERE col2=val"`，則透過使用 `AND` 和使用分割區欄的表達式擴展 `WHERE` 子句來測試查詢。

- Data type casting (資料類型轉換)：如果資料來源使用的是 JDBC 中不可用的資料類型，請使用本區段指定來自資料來源的資料類型應該如何轉換為 JDBC 資料類型。您可以指定最多 50 個不同的資料類型轉換。資料來源中使用相同資料類型的所有欄都會以相同的方式轉換。

  例如，如果您在資料來源中有三個使用 `Float` 資料類型的欄，而且您指出 `Float` 資料類型應轉換為 JDBC `String` 資料類型，則使用 `Float` 資料類型的全部三個欄都會轉換為 `String` 資料類型。

- Job bookmark keys (任務書籤索引鍵)：任務書籤可協助 AWS Glue 維護狀態資訊，以及防止重新處理舊資料。指定一個或多個資料行作為書籤索引鍵。AWS Glue Studio 使用書籤索引鍵追蹤在該任務執行期間所處理的資料。您用於自訂書籤索引鍵的任何欄必須是嚴格單調地增加或減少，但允許有間隙。

如果您輸入多個書籤索引鍵，則它們會結合在一起，形成單一複合索引鍵。複合任務書籤索引鍵不應包含重複的欄。如果您沒有指定書籤索引鍵，依預設，AWS Glue Studio 會使用主索引鍵做為書籤索引鍵，前提是主索引鍵依序遞增或遞減 (沒有間隙)。如果資料表沒有主索引鍵，但已啟用任務書籤屬性，您必須提供自訂任務書籤索引鍵。否則，搜尋要用作預設值的主索引鍵將會失敗，而且任務執行將會失敗。

- Job bookmark keys sorting order (任務書籤索引鍵排序順序)：選擇索引鍵值是依序增加還是減少。

Spark

- Schema (結構描述)：由於 AWS Glue Studio 使用儲存在連線中的資訊來存取資料來源，而不是從 Data Catalog 資料表中擷取中繼資料資訊，因此您必須提供資料來源的結構描述中繼資料。選擇 Add schema (新增結構描述) 開啟結構描述編輯器。

  如需如何使用結構描述編輯器的指示，請參閱編輯自訂轉換節點的結構描述。

- Connection options (連線選項)：視需要輸入其他索引鍵值組，以提供額外的連線資訊或選項。例如，您可以輸入資料庫名稱、資料表名稱、使用者名稱和密碼。

  例如 OpenSearch，您輸入下列鍵值配對，如中the section called “ 教學課程：使用 AWS Glue Elasticsearch Connector ”所述：

  - `es.net.http.auth.user` : *`username`*
  - `es.net.http.auth.pass` : *`password`*
  - `es.nodes` : `https://`*`<Elasticsearch endpoint>`*
  - `es.port` : `443`
  - `path`: *`<Elasticsearch resource>`*
  - `es.nodes.wan.only` : `true`

有關要使用的最小連接選項的示例，請參閱 MinimalSparkConnectorTest.scala 上的示例測試腳本 GitHub，其中顯示了通常在連接中提供的連接選項。

Athena

- Table name (資料表名稱)：資料來源中的資料表名稱。如果您使用連接器從 Athena-CloudWatch 日誌中讀取，則需要輸入表格名稱`all_log_streams`。

- Athena schema name (Athena 結構描述名稱)：選擇您 Athena 資料來源中對應至包含資料表之資料庫中的結構描述。如果您使用連接器從 Athena-CloudWatch 記錄檔讀取，您可以輸入類似的結構描述名稱`/aws/glue/`*`name`*。
- Schema (結構描述)：由於 AWS Glue Studio 使用儲存在連線中的資訊來存取資料來源，而不是從 Data Catalog 資料表中擷取中繼資料資訊，因此您必須提供資料來源的結構描述中繼資料。選擇 Add schema (新增結構描述) 開啟結構描述編輯器。

  如需如何使用結構描述編輯器的指示，請參閱編輯自訂轉換節點的結構描述。
- Additional connection options (其他連線選項)：視需要輸入其他索引鍵值組，以提供額外的連線資訊或選項。

  舉例來說，請參閱此`README.md`檔案，網址為 https://github.com/aws-samples/ aws-glue-samples /樹/主/開發/ GlueCustomConnectors Athena。在本文件的步驟中，範例程式碼顯示了所需的最少連線選項，也就是 `tableName`、`schemaName` 以及 `className`。程式碼範例會將這些選項指定為 `optionsMap` 變數的一部分，但您可以為您的連線指定它們，然後使用該連線。

3. (選用) 提供必要資訊之後，您可以選擇節點詳細資訊面板中的 Output schema (輸出結構描述) 索引標籤來檢視為資料來源產生的資料結構描述。此索引標籤上顯示的結構描述將由您新增至任務圖的任何子節點使用。
4. (選用) 設定節點屬性和資料來源屬性之後，您可以選擇節點詳細資訊面板中的 Data preview (資料預覽) 索引標籤來預覽資料來源中的資料集。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。使用此功能需支付相關費用，並且在您提供 IAM 角色後立即開始計費。

為使用連接器的節點設定目標屬性

如果您將連接器用於資料目標類型，則必須設定資料目標節點的屬性。

為使用連接器的資料目標節點設定屬性

1. 在任務圖中選擇連接器資料目標節點。然後，在右側的節點詳細資訊面板中，選擇 Data target properties (資料目標屬性) 索引標籤 (如果尚未選取)。
2. 在 Data target properties (資料目標屬性) 索引標籤上，選擇用來寫入目標的連線。

   輸入每個連線類型所需的其他資訊：

JDBC

- Connection (連線)：選擇要用於連接器的連線。如需如何建立連線的詳細資訊，請參閱建立連接器的連線。
- Table name (資料表名稱)：資料目標中的資料表名稱。如果資料目標不使用「表格」一詞，請提供適當資料結構的名稱，如自訂連接器使用資訊 (可在中找到 AWS Marketplace) 所示。
- Batch size (批次大小) (選用)：輸入單一作業中要在目標資料表中插入的列數或記錄數。預設值為 1000 列。

Spark

- Connection (連線)：選擇要用於連接器的連線。如果您先前未建立連線，請選擇 Create connection (建立連線) 建立一個。如需如何建立連線的詳細資訊，請參閱建立連接器的連線。
- Connection options (連線選項)：視需要輸入其他索引鍵值組，以提供額外的連線資訊或選項。您可以輸入資料庫名稱、資料表名稱、使用者名稱和密碼。

  例如 OpenSearch，您輸入下列鍵值配對，如中the section called “ 教學課程：使用 AWS Glue Elasticsearch Connector ”所述：

  - `es.net.http.auth.user` : *`username`*
  - `es.net.http.auth.pass` : *`password`*
  - `es.nodes` : `https://`*`<Elasticsearch endpoint>`*
  - `es.port` : `443`
  - path: *`<Elasticsearch resource>`*
  - `es.nodes.wan.only` : `true`

  有關要使用的最小連接選項的示例，請參閱 MinimalSparkConnectorTest.scala 上的示例測試腳本 GitHub，其中顯示了通常在連接中提供的連接選項。

3. 提供必要資訊之後，您可以選擇節點詳細資訊面板中的 Output schema (輸出結構描述) 索引標籤來檢視為資料來源產生的資料結構描述。

管理連接器和連線

您使用 AWS Glue 的連線頁面管理您的連接器和連線。

主題

檢視連接器和連線詳細資訊

您可以在 Connectors (連接器) 頁面上的 Your connectors (您的連接器) 和 Your connections (您的連線) 資源資料表中檢視連接器和連線的摘要資訊。若要檢視詳細資訊，請執行以下步驟。

檢視連接器或連線詳細資訊

1. 在 AWS Glue Studio 主控台中，在主控台導覽窗格中選擇 Connectors (連接器)。
2. 選擇您想要檢視詳細資訊的連接器或連線。
3. 選擇 Actions (動作)，然後選擇 View details (檢視詳細資訊) 以開啟該連接器或連線的詳細資訊頁面。
4. 在詳細資訊頁面上，您可以選擇 Edit (編輯) 或 Delete (刪除) 連接器或連線。
   - 對於連接器，您可以選擇 Create connection (建立連線)，以建立使用連接器的新連線。
   - 對於連線，您可以選擇建立任務，以建立使用此連線的任務。

編輯連接器和連線

您使用 Connectors (連接器) 頁面來變更儲存在連接器和連線中的資訊。

修改連接器或連線

1. 在 AWS Glue Studio 主控台中，在主控台導覽窗格中選擇 Connectors (連接器)。
2. 選擇您要變更的連接器或連線。
3. 選擇動作，然後選擇編輯。

   您也可以選擇 View details (檢視詳細資訊)，並在連接器或連線詳細資訊頁面上，您可以選擇 Edit (編輯)。
4. 在 Edit connector (編輯連接器) 或 Edit connection (編輯連線) 頁面上，更新資訊，然後選擇 Save (儲存)。

刪除連接器和連線

您使用 Connectors (連接器) 頁面來刪除連接器和連線。如果您刪除連接器，則也應該刪除為該連接器建立的所有連線。

從 AWS Glue Studio 中移除連接器

1. 在 AWS Glue Studio 主控台中，在主控台導覽窗格中選擇 Connectors (連接器)。
2. 選擇您要刪除的連接器或連線。
3. 選擇動作，然後選擇刪除。

   您也可以選擇 View details (檢視詳細資訊)，並在連接器或連線詳細資訊頁面上，您可以選擇 Delete (刪除)。
4. 確認您想要移除連接器或連線，方法是輸入 **`Delete`**，然後選擇 Delete (刪除)。

   刪除連接器時，也會刪除為該連接器建立的所有連線。

使用已刪除連線的任何任務將無法再運作。您可以編輯任務以使用其他資料存放區，也可以移除任務。如需有關如何刪除任務的詳細資訊，請參閱刪除任務。

如果您刪除連接器，這不會取消 AWS Marketplace中對連接器的訂閱。若要移除已刪除連接器的訂閱，請依照取消連接器的訂閱中的指示進行。

取消連接器的訂閱

從中刪除連線和連接器之後AWS Glue Studio， AWS Marketplace 如果您不再需要連接器，則可以在中取消訂閱。

> **Note**
>
> 如果您取消訂閱連接器，這不會從您的帳戶移除連接器或連線。使用連接器和相關連線的任何任務將無法再使用連接器，而且將會失敗。
> 在取消訂閱或重新訂閱連接器之前 AWS Marketplace，您應該先刪除與該 AWS Marketplace 產品相關聯的現有連線和連接器。

若要在中取消訂閱連接器 AWS Marketplace

1. 請在以下位置登入 AWS Marketplace 主控台：https://console.aws.amazon.com/marketplace。

2. 選擇 Manage subscriptions (管理訂閱)。
3. 在 Manage subscriptions (管理訂閱) 頁面上，選擇您想要取消的連接器訂閱旁的 Manage (管理)。
4. 選擇 Actions (動作)，再選擇 Cancel Subscriptions (取消訂閱)。
5. 選取核取方塊，確認執行中的執行個體會向您的帳戶收費，然後選擇 Yes, cancel subscription (是的，取消訂閱)。

開發自訂連接器

您可以編寫程式碼以從資料存放區讀取資料或將資料寫入資料存放區，以及將用於 AWS Glue Studio 任務的資料格式化。您可以為 Spark、Athena 和 JDBC 資料存放區建立連接器。張貼在上的範例程式碼 GitHub 提供您需要實作之基本介面的概觀。

您將需要一個本機開發環境來建立您的連接器程式碼。您可以使用任何 IDE，甚至只是命令行編輯器來編寫連接器。開發環境的範例包括：

- 本機 Scala 環境搭配本機 AWS Glue ETL Maven 程式庫，如 AWS Glue 開發人員指南的在本機開發 Scala 中所述。
- IntelliJ IDE，透過從 https://www.jetbrains.com/idea/ 下載 IDE 取得。

主題

- 開發 Spark 連接器
- 開發 Athena 連接器
- 開發 JDBC 連接器
- 搭配 AWS Glue Studio 使用自訂連接器的範例
- 開發AWS Glue連接器 AWS Marketplace

開發 Spark 連接器

您可以創建一個星火 DataSource API V2（星火 2.4）讀取數據的星火連接器。

建立自訂 Spark 連接器

遵循AWS Glue GitHub 範例程式庫中的步驟，以開發星火連接器，位於 https://github.com/aws-samples/ aws-glue-samples /樹/主//開發/火花/ GlueCustomConnectors readme.md。

開發 Athena 連接器

您可以建立 Athena 連接器，以供 AWS Glue 和 AWS Glue Studio 用於查詢自訂資料來源。

建立自訂 Athena 連接器

請依照AWS Glue GitHub 範例程式庫中的步驟來開發 Athena 連接器，該連接器位於 https://github.com/aws-samples/ aws-glue-samples /樹/主//開發/ GlueCustomConnectors 雅典娜。

開發 JDBC 連接器

您可以建立使用 JDBC 存取資料存放區的連接器。

建立自訂 JDBC 連接器

1. 在您的本機開發環境中安裝 AWS Glue Spark 執行時間程式庫。請參閱AWS Glue GitHub 範例程式庫中的說明，網址為 https://github.com/aws-samples/ aws-glue-samples /樹/主/開發/ GlueCustomConnectors readme.md。GlueSparkRuntime
2. 實作負責從資料來源擷取資料的 JDBC 驅動程式。請參閱適用於 Java SE 8 的 Java 文件。

   在您的程式碼中建立一個入口點，供 AWS Glue Studio 用來找到您的連接器。Class name (類別名稱) 欄位應該是 JDBC 驅動程式的完整路徑。
3. 使用 `GlueContext` API 來透過連接器讀取資料。如有必要，使用者可以在 AWS Glue Studio 主控台中新增更多輸入選項以設定連到資料來源的連線。如需說明如何使用自訂 JDBC 連接器讀取和寫入 JDBC 資料庫的程式碼範例，請參閱自訂和 AWS Marketplace connec tionType 值。

搭配 AWS Glue Studio 使用自訂連接器的範例

如需使用自訂連接器的範例，請參閱下列部落格：

- 使用 AWS Glue 為您的資料存放區開發、測試及部署自訂連接器
- Apache Hudi：使用 AWS Glue 自訂連接器寫入 Apache Hudi 資料表
- 谷歌 BigQuery：使用AWS Glue自定義連接器將數據從谷歌遷移 BigQuery 到 Amazon S3
- Snowflake (JDBC)：使用 Snowflake 和 AWS Glue 執行資料轉換
- SingleStore：使 SingleStore 用和構建快速 ETL AWS Glue
- Salesforce： 搭配使用 CData JDBC 自訂連接器與 AWS Glue，將 Salesforce 資料擷取至 Amazon S3 -

- MongoDB：使用 Amazon DocumentDB (with MongoDB compatibility) 和 MongoDB 建置 AWS Glue Spark ETL 任務
- Amazon Relational Database Service （Amazon RDS）：通過為 Amazon RDS 帶來自己的 JDBC 驅動程序來構建AWS Glue星火 ETL 任務
- MySQL (JDBC): https://github.com/aws-samples/ aws-glue-samples /blob /主/開發/火花/SQL/ 斯卡拉 GlueCustomConnectors SparkConnectorMy

開發AWS Glue連接器 AWS Marketplace

身為 AWS 合作夥伴，您可以建立自訂連接器，並將其上傳 AWS Marketplace 至銷售給AWS Glue客戶。

開發連接器程式碼的程序與自訂連接器的程序相同，但是上傳和驗證連接器程式碼的程序比較詳細。請參閱 GitHub 網站上的建立連接器中 AWS Marketplace的指示。

在 AWS Glue Studio 中使用連接器和連線的限制

當您使用的自訂連接器或連接器來源時 AWS Marketplace，請注意下列限制：

- 針對自訂連接器建立的連線不支援 testConnection API。
- 自訂連接器不支援 Data Catalog 連線密碼加密。
- 如果您為使用 JDBC 連接器的資料來源節點指定篩選述詞，則無法使用任務書籤。
- 在 AWS Glue Studio 使用者介面之外，不支援建立 Marketplace 連線。

使用視覺化 ETL 任務來連線至資料來源

建立新任務時，您可以在 AWS Glue 中編輯視覺化 ETL 任務時，使用連線功能來連線至資料。您可以透過新增使用連接器讀入資料的來源節點，以及用於指定資料寫出位置的目標節點來執行此操作。

主題

- 修改資料來源節點的屬性
- 將 Data Catalog 資料表用於資料來源
- 將連接器用於資料來源
- 將 Amazon S3 中的檔案用於資料來源
- 使用串流資料來源

- 參考

修改資料來源節點的屬性

若要指定資料來源屬性，您需先在任務圖表中選擇資料來源節點。然後，在節點詳細資訊面板的右側，設定節點屬性。

修改資料來源節點的屬性

1. 前往新任務或已儲存任務的視覺化編輯器。
2. 在任務圖表中選擇資料來源節點。
3. 在節點詳細資訊面板中選擇 Node properties (節點屬性) 索引標籤，然後輸入下列資訊：
   - Name (名稱)：(選用) 輸入要與任務圖表中節點產生關聯的名稱。此名稱在此任務的所有節點中應該是唯一的。
   - Node type (節點類型)：節點類型決定節點所執行的動作。在 Node type (節點類型) 的選項清單中，選擇其中一個列於 Data source (資料來源) 標題下的值。
4. 設定 Data source properties (資料來源屬性) 資訊。如需詳細資訊，請參閱下列章節：
   - 將 Data Catalog 資料表用於資料來源
   - 將連接器用於資料來源
   - 將 Amazon S3 中的檔案用於資料來源
   - 使用串流資料來源
5. (選用) 設定節點屬性和資料來源屬性之後，您可以選擇節點詳細資訊面板中的 Output schema (輸出結構描述) 索引標籤來檢視資料來源的結構描述。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。如果您尚未在 Job details (任務詳細資訊) 索引標籤上指定 IAM 角色，系統會提示您在此輸入 IAM 角色。
6. (選用) 設定節點屬性和資料來源屬性之後，您可以選擇節點詳細資訊面板中的 Data preview (資料預覽) 索引標籤來預覽資料來源中的資料集。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。使用此功能需支付相關費用，並且在您提供 IAM 角色後立即開始計費。

將 Data Catalog 資料表用於資料來源

對於除 Amazon S3 和連接器以外的所有資料來源，資料表必須存在於 AWS Glue Data Catalog，以取得您選擇的來源類型。AWS Glue 不會建立 Data Catalog 資料表。

根據 Data Catalog 資料表設定資料來源節點

1. 前往新任務或已儲存任務的視覺化編輯器。
2. 在任務圖表中選擇資料來源節點。
3. 選擇 Data source properties (資料來源屬性) 索引標籤，然後輸入下列資訊：

   - S3 source type (S3 來源類型)：(僅適用於 Amazon S3 資料來源) 選擇 Select a Catalog table (選取目錄資料表) 以使用現有 AWS Glue Data Catalog 資料表。
   - Database (資料庫)：在 Data Catalog 中選擇包含您要用於此任務之來源資料表的資料庫。您可以使用搜尋欄位來依名稱搜尋資料庫。
   - Table (資料表)：從清單中選擇與來源資料相關聯的資料表。此資料表必須已存在於 AWS Glue Data Catalog。您可以使用搜尋欄位來依名稱搜尋資料表。
   - Partition predicate (分割區述詞)：(僅適用於 Amazon S3 資料來源) 輸入以 Spark SQL 為基礎的布林表達式，該表達式僅包含分割欄，例如：`"(year=='2020' and month=='04')"`
   - Temporary directory (暫時目錄): (僅適用於 Amazon Redshift 資料來源) 輸入 Amazon S3 中工作目錄位置的路徑，您的 ETL 任務可以在此處寫入暫時的中繼結果。
   - Role associated with the cluster (與叢集相關聯的角色): (僅適用於 Amazon Redshift 資料來源) 輸入您的 ETL 任務要使用的角色，該角色包含 Amazon Redshift 叢集的許可。如需更多詳細資訊，請參閱 the section called “資料來源和資料目標許可”。

將連接器用於資料來源

如果您選取 Node type (節點類型) 的連接器，請遵循使用自訂連接器編寫任務中的說明以完成設定資料來源屬性。

將 Amazon S3 中的檔案用於資料來源

如果您選擇 Amazon S3 做為資料來源，則可以選擇下列其中一項：

- Data Catalog 資料庫和資料表。
- Amazon S3 中的儲存貯體、資料夾或檔案。

如果您使用 Amazon S3 儲存貯體作為資料來源，則 AWS Glue 會從其中一個檔案，或使用您指定為範例檔案的檔案中，偵測位於指定位置之資料的結構描述。結構描述偵測會發生在您使用 Infer schema (推斷結構描述) 按鈕時。如果變更 Amazon S3 位置或範例檔案，則必須選擇 Infer schema (推斷結構描述)，以使用新資訊執行結構描述偵測。

設定直接從 Amazon S3 中的檔案讀取的資料來源節點

1. 前往新任務或已儲存任務的視覺化編輯器。
2. 在任務圖表中為 Amazon S3 來源選擇資料來源節點。
3. 選擇 Data source properties (資料來源屬性) 索引標籤，然後輸入下列資訊：

   - S3 source type (S3 來源類型)：(僅適用於 Amazon S3 資料來源) 選擇 S3 location (S3 位置) 選項。
   - S3 URL：輸入包含任務資料的 Amazon S3 儲存貯體、資料夾或檔案的路徑。您可以選擇 Browse S3 (瀏覽 S3)，從您的帳戶可用位置選取路徑。
   - Recursive (遞迴)：如果您想要 AWS Glue 從 S3 位置的子資料夾中的檔案讀取資料，請選擇此選項。

     如果子資料夾包含分割的資料，AWS Glue 不會將資料夾名稱中指定的任何分割區資訊新增至 Data Catalog 。例如，請試想在 Amazon S3 中有以下資料夾：

     ```
     S3://sales/year=2019/month=Jan/day=1
     S3://sales/year=2019/month=Jan/day=2
     ```

     如果選擇 Recursive (遞迴)，然後選取 `sales` 資料夾作為您的 S3 位置，則 AWS Glue 會讀取所有子資料夾中的資料，但不會為年份、月份或日建立分割區。
   - Data format (資料格式)：選擇儲存資料的格式。您可以選擇 JSON、CSV 或 Parquet。您選取的值會告訴 AWS Glue 任務如何從來源檔案讀取資料。

     > **Note**
     >
     > 如果您沒有為資料選取正確的格式，則 AWS Glue 可能會正確推斷結構描述，但任務將無法正確解析來源檔案中的資料。

     根據您選擇的格式，您可以輸入其他組態選項。

     - JSON (JavaScript 物件標記法)
       - JsonPath：輸入指向用來定義資料表結構描述之物件的 JSON 路徑。JSON 路徑表達式一律以將 XPath 表達式與 XML 文件搭配使用的相同方式，來參照 JSON 結構。JSON 路徑中的「根成員物件」一律稱為 $，即使它是物件或陣列。您可以用點標記法或括號標記法來撰寫 JSON 路徑。

如需 JSON 路徑的詳細資訊，請參閱 GitHub 網站上的 JsonPath。

- Records in source files can span multiple lines (來源檔案中的記錄可以跨越多行)：如果單項記錄可以跨越 CSV 檔案中的多行，請選擇此選項。

- CSV (逗號分隔值)
  - Delimiter (分隔符號)：輸入字元來表示用什麼分隔列中的每個欄項目，例如 `;` 或 `,`。
  - Escape character (逸出字元)：輸入用作逸出字元的字元。此字元表示緊接在逸出字元後面的字元應該依照字面解讀，不應該被解譯為分隔符號。
  - Quote character (引號字元)：輸入用來將個別字串群組成單一值的字元。例如，如果在您的 CSV 檔案中有像 `"This is a single value"` 的值，您可以選擇 Double quote (") (雙引號 (("))。
  - Records in source files can span multiple lines (來源檔案中的記錄可以跨越多行)：如果單項記錄可以跨越 CSV 檔案中的多行，請選擇此選項。
  - First line of source file contains column headers (來源檔案的第一行包含欄標題)：如果 CSV 檔案中的第一列包含欄標頭而非資料，請選擇此選項。
- Parquet (Apache Parquet 直欄式儲存)

  沒有額外的設定可以為儲存在 Parquet 格式的資料進行設定。

- Partition predicate (分割區述詞)：若要分割從資料來源讀取的資料，請輸入以 Spark SQL 為基礎的布林表達式，其中只包含分割欄，例如：`"(year=='2020' and month=='04')"`
- Advanced options (進階選項)：如果您需要 AWS Glue 根據特定檔案偵測資料的結構描述，請展開此區段。
  - Schema inference (結構描述推斷)：如果您想使用特定的檔案而不是讓 AWS Glue 選擇檔案，請選擇選項 Choose a sample file from S3 (從 S3 選擇範例檔案)。
  - Auto-sampled file (自動取樣檔案)：輸入 Amazon S3 中要用於推斷結構描述的檔案路徑。

  如果您要編輯資料來源節點並變更選取的範例檔案，請選擇 Reload schema (重新載入結構描述) 以使用新的範例檔案來偵測結構描述。

4. 選擇 Infer schema (推斷結構描述) 按鈕，以從 Amazon S3 中的來源檔案偵測結構描述。如果變更 Amazon S3 位置或範例檔案，則必須再次選擇 Infer schema (推斷結構描述) 以使用新資訊推斷結構描述。

使用串流資料來源

您可以建立串流擷取、轉換和載入 (ETL) 任務，讓它連續執行並從 Amazon Kinesis Data Streams、Apache Kafka 和 Amazon Managed Streaming for Apache Kafka (Amazon MSK) 的串流來源使用資料。

設定串流資料來源的屬性

1. 前往新任務或已儲存任務的視覺化圖表編輯器。
2. 在 Kafka 或 Kinesis Data Streams 的圖形中選擇資料來源節點。
3. 選擇 Data source properties (資料來源屬性) 索引標籤，然後輸入下列資訊：

   Kinesis

   - Kinesis source type (Kinesis 來源類型)：選擇選項 Stream details (串流詳細資訊) 以使用直接存取串流來源，或選擇 Data Catalog table (Data Catalog 資料表) 以改用儲存在其中的資訊。

     如果選擇 Stream details (串流詳細資訊)，則指定下列其他資訊。

     - 資料串流位置：選擇串流是否與目前的使用者相關聯，或是與不同的使用者相關聯。
     - 區域：選擇串流所 AWS 區域 在的位置。此資訊用於建構存取資料串流的 ARN。
     - Stream ARN (串流 ARN)：輸入 Kinesis 資料串流的 Amazon Resource Name (ARN)。如果串流位於目前帳戶內，可以從下拉式清單中選擇串流名稱。您可以使用搜尋欄位來依名稱或 ARN 搜尋資料串流。
     - Data format (資料格式)：從清單中選擇資料串流使用的格式。

       AWS Glue 會自動從串流資料偵測結構描述。

     如果選擇 Data Catalog table (Data Catalog 資料表)，指定下列其他資訊。

     - Database (資料庫)：(選用) 在 AWS Glue Data Catalog 中選擇資料庫，其中包含與串流資料來源相關聯的資料表。您可以使用搜尋欄位來依名稱搜尋資料庫。
     - Table (資料表)：(選用) 從清單中選擇與來源資料相關聯的資料表。此資料表必須已存在於 AWS Glue Data Catalog 中。您可以使用搜尋欄位來依名稱搜尋資料表。
     - Detect schema (偵測結構描述)：選擇此選項可讓 AWS Glue 偵測來自串流資料的結構描述，而不是使用 Data Catalog 資料表中的結構描述資訊。如果您選擇 Stream details (串流詳細資訊) 選項，則自動啟用此選項。
   - Starting position (開始位置)：依預設，ETL 任務會使用 Earliest (最早) 選項，這表示它會從串流中最早期的可用記錄開始讀取資料。您可以改為選擇 Latest (最新)，這表示 ETL 任務應該從串流中的最新記錄之後開始讀取。

- Window size (時段大小)：依預設 ETL 任務以 100 秒的時段處理和寫出資料。這樣可以有效處理資料，並且可在資料到達時間比預期晚時執行彙總。您可以修改此間隔大小，以提高適時性或彙總正確性。

  AWS Glue 串流工作會使用檢查點而非工作書籤來追蹤已讀取的資料。

- Connection options (連線選項)：展開此區段以新增索引鍵-值配對，以指定其他連線選項。如需您可以在此指定哪些選項的相關資訊，請參閱《AWS Glue 開發人員指南》中的 "connectionType": "kinesis"。

Kafka

- Apache Kafka source (Apache Kafka 來源)：選擇選項 Stream details (串流詳細資訊) 以使用直接存取串流來源，或選擇 Data Catalog table (Data Catalog 資料表) 來改用儲存在其中的資訊。

  如果選擇 Data Catalog table (Data Catalog 資料表)，指定下列其他資訊。

  - Database (資料庫)：(選用) 在 AWS Glue Data Catalog 中選擇資料庫，其中包含與串流資料來源相關聯的資料表。您可以使用搜尋欄位來依名稱搜尋資料庫。
  - Table (資料表)：(選用) 從清單中選擇與來源資料相關聯的資料表。此資料表必須已存在於 AWS Glue Data Catalog 中。您可以使用搜尋欄位來依名稱搜尋資料表。
  - Detect schema (偵測結構描述)：選擇此選項可讓 AWS Glue 偵測來自串流資料的結構描述，而不是使用 Data Catalog 資料表中的結構描述資訊。如果您選擇 Stream details (串流詳細資訊) 選項，則自動啟用此選項。

  如果選擇 Stream details (串流詳細資訊)，則指定下列其他資訊。

  - 連線名稱 (Connection name)：選擇包含 Kafka 資料串流的存取和身分驗證資訊的 AWS Glue 連線。您必須將此連線與 Kafka 串流資料來源搭配使用。如果連線不存在，您可以使用 AWS Glue 主控台為您的 Kafka 資料串流建立連線。
  - Topic name (主題名稱)：輸入要讀取的主題名稱。
  - Data format (資料格式)：選擇從 Kafka 事件資料流讀取資料時使用的格式。

- Starting position (開始位置)：預設情況下，ETL 任務會使用 Earliest (最早) 選項，這表示它會從串流中最早期的可用記錄開始讀取資料。您可以改為選擇 Latest (最新)，這表示 ETL 任務應該從串流中的最新記錄之後開始讀取。

- Window size (時段大小)：依預設 ETL 任務以 100 秒的時段處理和寫出資料。這樣可以有效處理資料，並且可在資料到達時間比預期晚時執行彙總。您可以修改此間隔大小，以提高適時性或彙總正確性。

  AWS Glue 串流任務使用檢查點而不是任務書籤來追蹤已讀取的資料。

- Connection options (連線選項)：展開此區段以新增索引鍵-值配對，以指定其他連線選項。如需您可以在此指定哪些選項的相關資訊，請參閱《AWS Glue 開發人員指南》中的 "connectionType": "kinesis"。

> **Note**
>
> 資料預覽目前不支援串流資料來源。

參考

最佳實務

- 建置 ETL 服務管道以使用 AWS Glue 將資料從 Amazon S3 增量載入至 Amazon Redshift

ETL 程式設計

- AWS Glue 中 ETL 的連線類型和選項
- JDBC connectionType 值
- 用於將資料移入和移出 Amazon Redshift 的進階選項

使用自己的 JDBC 驅動程式新增 JDBC 連線

使用 JDBC 連線時，您可以使用自己的 JDBC 驅動程式。當 AWS Glue 爬蟲程式使用的預設驅動程式無法連線到資料庫時，您就可以使用自己的 JDBC 驅動程式。例如，如果您想將 SHA-256 與 Postgres 資料庫搭配使用，而較舊的 Postgres 驅動程式不支援此功能，則可以使用自己的 JDBC 驅動程式。

支援的資料來源

支援的資料來源	不支援的資料來源
MySQL	Snowflake
Postgres	
Oracle	
Redshift	
SQL Server	
Aurora*	

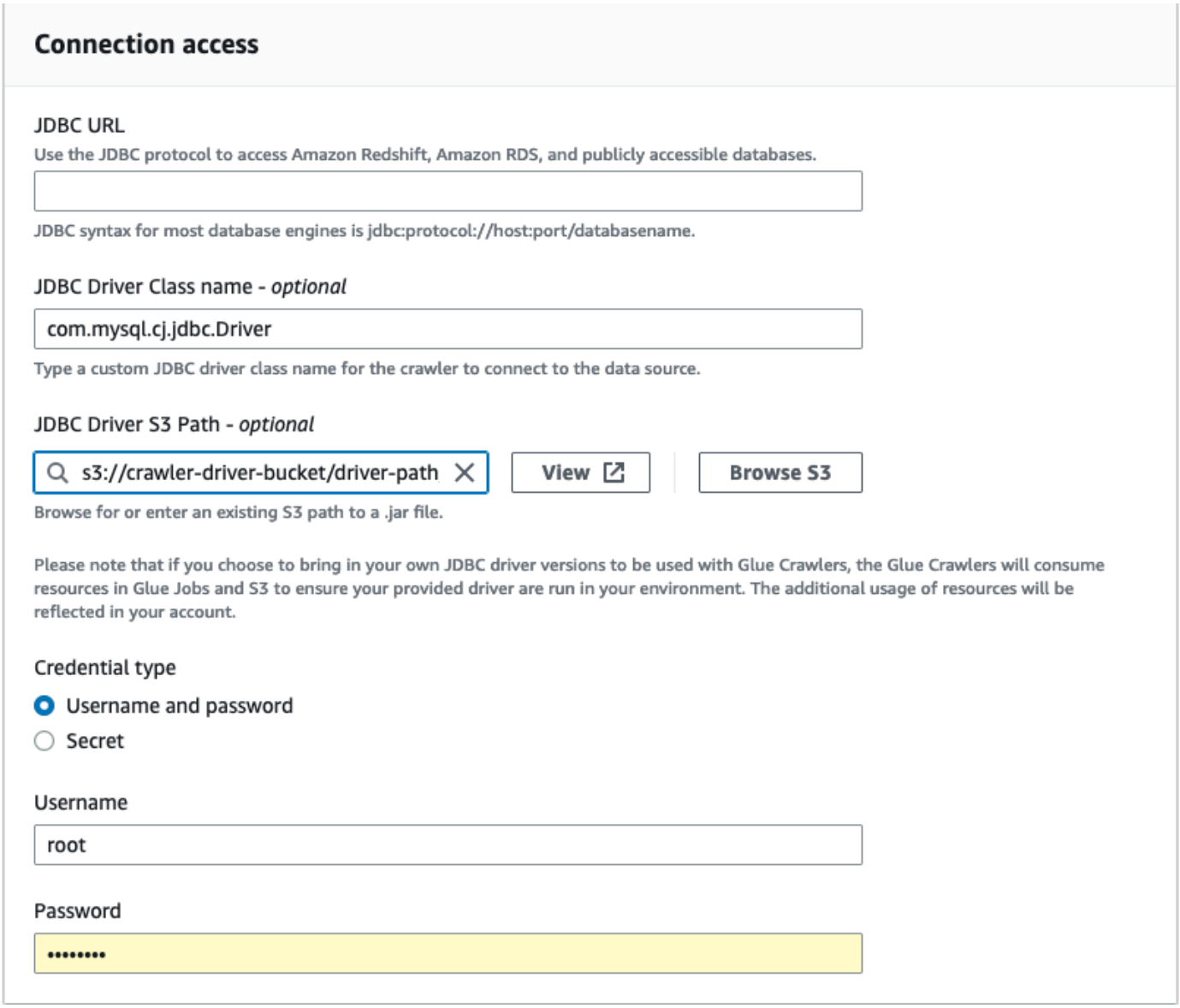
* 如果正在使用原生的 JDBC 驅動程式，則支援此功能。並非所有驅動程式功能都可供運用。

將 JDBC 驅動程式新增至 JDBC 連線

> **Note**
>
> 如果您選擇引入自己的 JDBC 驅動程式版本，AWS Glue 爬蟲程式將使用 AWS Glue 任務和 Amazon S3 儲存貯體中的資源，以確保您提供的驅動程式在環境中執行。帳戶中將反映資源的額外使用量。AWS Glue 爬蟲程式和任務的費用屬於 AWS Glue 計費類別。此外，提供您的 JDBC 驅動程式，並不代表爬蟲程式能夠運用驅動程式的所有功能。

若要將自己的 JDBC 驅動程式新增至 JDBC 連線：

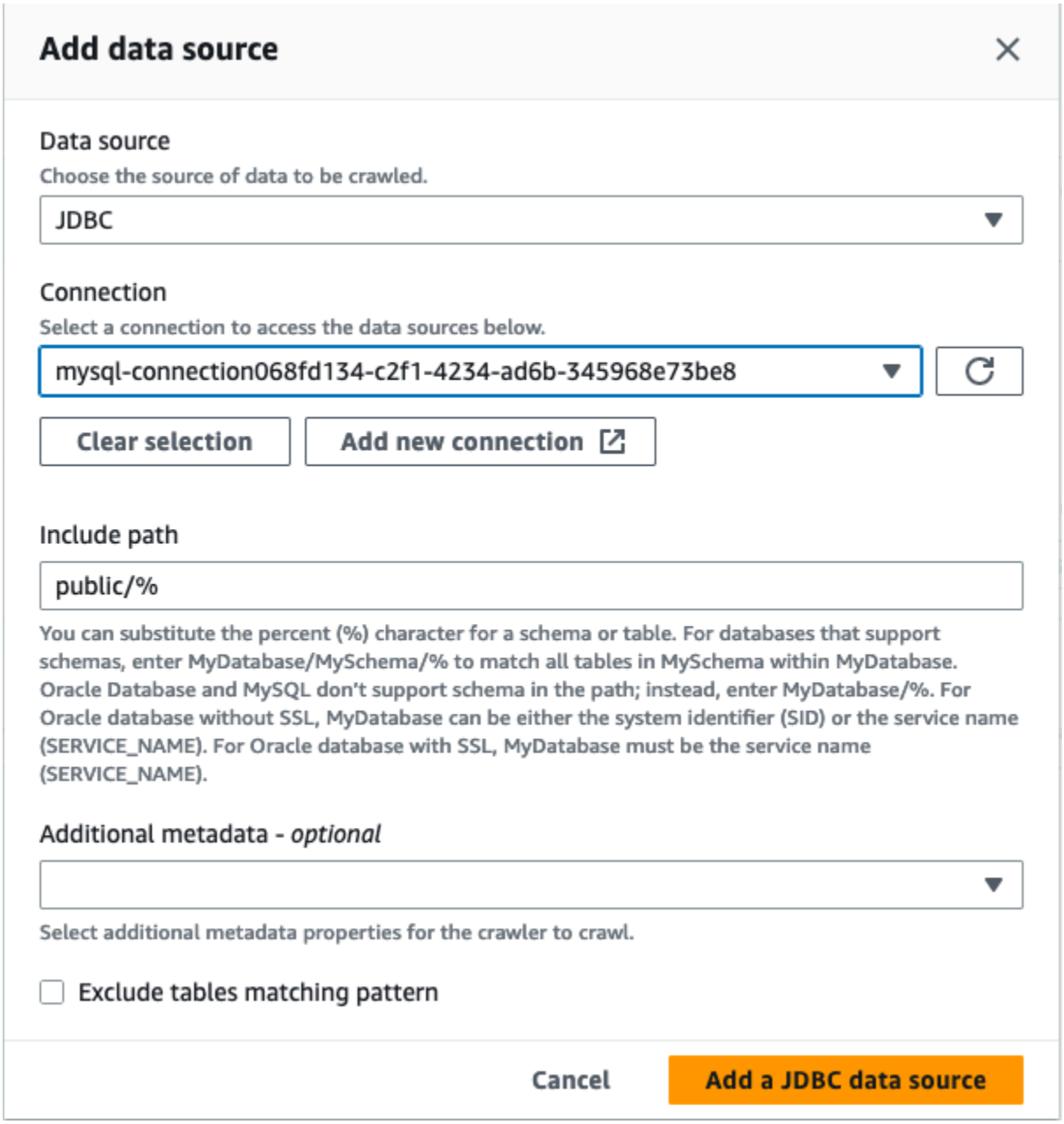
1. 將 JDBC 驅動程式檔案新增至 Amazon S3 位置。您可以建立儲存貯體和/或資料夾，或使用現有的儲存貯體和/或資料夾。
2. 在 AWS Glue 主控台中，選擇左側選單中資料型錄下的連線，然後建立新連線。
3. 填寫連線屬性欄位，然後選擇 JDBC 作為連線類型。
4. 在連線存取中，輸入 JDBC URL 和 JDBC 驅動程式類別名稱：選用。驅動程式類別名稱必須是 AWS Glue 爬蟲程式支援的資料來源。



5. 選擇 JDBC 驅動程式 Amazon S3 路徑中 JDBC 驅動程式所在的 Amazon S3 路徑：選用欄位。
6. 如果輸入使用者名稱和密碼或機密，請填寫「憑證」類型的欄位。完成時，選擇建立連線。

> **Note**
>
> 目前不支援測試連線。使用您提供的 JDBC 驅動程式針對資料來源進行網路爬取時，爬蟲程式會略過此步驟。

7. 將新建立的連線新增至爬蟲程式。在 AWS Glue 主控台中，從左側選單選擇資料型錄下的爬蟲程式，然後建立新的爬蟲程式。
8. 在新增爬蟲程式精靈的步驟 2 中，選擇新增資料來源。



9. 選擇 JDBC 作為資料來源，然後選擇在先前步驟中建立的連線。完成
10. 若要將您自己的 JDBC 驅動程式與 AWS Glue 爬行者程式搭配使用，請將下列權限新增至爬行者程式所使用的角色：

    - 授予下列任務動作的許可：`CreateJob`、`DeleteJob`、`GetJob`、`GetJobRun`、`StartJobRun`。
    - 授予 IAM 動作的許可：`iam:PassRole`
    - 授予 Amazon S3 動作的許可：`s3:DeleteObjects`、`s3:GetObject`、`s3:ListBucket`、`s3:PutObject`。

- 在 IAM 政策中授予服務主體對儲存貯體/資料夾的存取權。

IAM 政策範例：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name/driver-parent-folder/driver.jar",
                "arn:aws:s3:::bucket-name"
            ]
        }
    ]
}
```

11. 如果您使用的是 VPC，則必須透過建立介面端點並將其新增至路由表來允許對 AWS Glue 端點的存取。如需詳細資訊，請參閱 Creating an interface VPC endpoint for AWS Glue
12. 如果您在資料目錄中使用加密，請建立 AWS KMS 介面端點並將其新增至路由表。如需詳細資訊，請參閱為 AWS KMS建立 VPC 端點。

測試 AWS Glue 連線

作為最佳實務，在您於 ETL 任務中使用 AWS Glue 連線時，請使用 AWS Glue 主控台測試連線。AWS Glue 會在連線中使用參數確認連線可存取資料存放區，並回報任何錯誤。如需 AWS Glue 連線的詳細資訊，請參閱 連線至資料。

測試 AWS Glue 連線

1. 請登入 AWS Management Console 並開啟 AWS Glue 主控台，網址為 https://console.aws.amazon.com/glue/。
2. 在導覽窗格中，於資料型錄下選擇連線。您也可以在導覽窗格中選擇資料型錄上方的資料連線。
3. 在連線中選取所需連線旁邊的核取方塊，然後選擇動作。在下拉式選單中，選擇測試連線。
4. 在「測試連線」對話方塊中，選取角色或選擇「建立 IAM 角色」以移至 AWS Identity and Access Management (IAM) 主控台以建立新角色。角色必須有資料存放區的許可。
5. 選擇確認。

   然後測試就會開始，但可能需要幾分鐘的時間才能完成。如果測試失敗，請選擇疑難排解以檢視問題的解決步驟。
6. 選擇「記錄檔」以檢視記錄 CloudWatch。您必須具有必要的 IAM 許可才能檢視日誌。如需詳細資訊，請參閱 AWSAmazon CloudWatch 日誌使用者指南中的 CloudWatch 日誌受管 (預先定義) 政策。

設定 AWS 呼叫通過您的 VPC

特殊任務參數 `disable-proxy-v2` 允許您透過 VPC，將呼叫路由至服務，例如 Amazon S3、CloudWatch 和 AWS Glue。依預設，AWS Glue 會使用本機代理透過 AWS Glue VPC 來傳送流量以從 Amazon S3 下載指令碼和程式庫，將要求傳送至 CloudWatch 以發佈日誌和指標，以及將要求傳送至 AWS Glue 以存取資料目錄。即使您的 VPC 未設定通往其他 AWS 服務 (如 Amazon S3、CloudWatch 和 AWS Glue) 的適當路由，此代理也可允許任務正常運作。AWS Glue 現會提供參數，讓您關閉此行為。如需詳細資訊，請參閱 AWS Glue 使用的任務參數。AWS Glue 將繼續使用本機代理來發佈 AWS Glue 任務的 CloudWatch 日誌。

> **Note**
>
> - 在 AWS Glue 2.0 及更新版本中，此功能支援 AWS Glue 任務。在使用此功能時，您需要確保自己的 VPC 已透過 NAT 或服務 VPC 端點設定通往 Amazon S3 的路由。
> - 已棄用的任務參數 `disable-proxy` 僅會透過您的 VPC 將您的呼叫路由至 Amazon S3，以下載指令碼和程式庫。建議改用新參數 `disable-proxy-v2`。

範例使用方式

建立具有 disable-proxy-v2 的 AWS Glue 任務：

```
aws glue create-job \
    --name no-proxy-job \
    --role GlueDefaultRole \
    --command "Name=glueetl,ScriptLocation=s3://my-bucket/glue-script.py" \
    --connections Connections="traffic-monitored-connection" \
    --default-arguments '{"--disable-proxy-v2" : "true"}'
```

在 VPC 連線至 JDBC 資料存放區

通常，您是在 Amazon Virtual Private Cloud (Amazon VPC) 內部建立資源，因此無法經由公有網際網路進行存取。依預設，AWS Glue 無法存取 VPC 之內的資源。要讓 AWS Glue 存取 VPC 中的資源，您必須提供其他 VPC 特定的組態資訊，包括 VPC 子網路 ID 和安全群組 ID。AWS Glue 會使用此資訊設定彈性網路界面，讓您的函式可安全連接至私有 VPC 中的其他資源。

使用 VPC 端點時，請將其新增至路由表。如需詳細資訊，請參閱 Creating an interface VPC endpoint for AWS Glue 和 必要條件。

在資料型錄中使用加密時，建立 KMS 介面端點並將其新增至路由表。如需詳細資訊，請參閱 Creating a VPC endpoint for AWS KMS。

使用彈性網路介面存取 VPC 資料

當 AWS Glue 連線至 VPC 中的 JDBC 資料存放區時，AWS Glue 會在您的帳戶中建立彈性網路界面 (加上字首 Glue_) 以存取您的 VPC 資料。只要此網路界面連接至 AWS Glue，您就無法將它刪除。在建立彈性網路界面的過程中，AWS Glue 會讓一個或多個安全群組與其建立關聯。要讓 AWS Glue 建立網路界面，與資源關聯的安全群組必須以來源規則允許傳入存取。此規則包含與資源關聯的安全群組。因此彈性網路界面就能以相同的安全群組存取您的資料存放區。

要讓 AWS Glue 與其元件通訊，請指定一個安全群組並為所有 TCP 連接埠建立自我參考的傳入規則。建立自我參考的規則後，您就可以將資源限制給 VPC 中相同的安全群組，不對所有網路開放。VPC 的預設安全群組可能已經有了 ALL Traffic 的自我參考傳入規則。

您可以在 Amazon VPC 主控台建立規則。要透過 AWS Management Console 更新規則設定，請瀏覽至 VPC 主控台 (https://console.aws.amazon.com/vpc/) 並選取適合的安全群組。指定 ALL TCP 的傳入規則，其來源則為相同的安全群組名稱。如需安全群組規則的詳細資訊，請參閱 VPC 的安全群組。

每個彈性網路界面都會指派一個私有 IP 地址，而此地址來自您在子網路中指定的 IP 地址範圍。網路界面不會指派任何公有 IP 地址。AWS Glue 需要網際網路存取 (例如存取無 VPC 端點的 AWS 服務)。您

可以在 VPC 中設定網路位址轉譯 (NAT) 執行個體，或使用 Amazon VPC NAT 閘道。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的 NAT 閘道。您無法直接將連接至 VPC 的網際網路閘道做為子網路路由表中的路由使用，原因是網路介面需要公有 IP 地址。

VPC 網路屬性 `enableDnsHostnames` 和 `enableDnsSupport` 必須設定為 true。如需詳細資訊，請參閱以 VPC 使用 DNS。

> **Important**
>
> 請勿將資料存放區置於公有子網路或是無網際網路存取的私有子網路。反之，請僅將其連接至透過 NAT 執行個體或 Amazon VPC NAT 閘道存取網際網路的私有子網路。

彈性網路介面屬性

要建立彈性網路界面，您必須提供以下屬性：

VPC

包含資料存放區之 VPC 的名稱。

子網

包含資料存放區之 VPC 的子網路。

安全群組

與資料存放區關聯的安全群組。AWS Glue 將這些安全群組關聯至連接您 VPC 子網路的彈性網路界面。為了讓 AWS Glue 元件通訊並防止從其他網路存取，至少一個所選安全群組必須指定適用於所有 TCP 連接埠的自我參考傳入規則。

如需以 Amazon Redshift 管理 VPC 的詳細資訊，請參閱在 Amazon Virtual Private Cloud (VPC) 管理叢集。

如需使用 Amazon Relational Database Service (Amazon RDS) 管理 VPC 的詳細資訊，請參閱在 VPC 中使用 Amazon RDS 資料庫執行個體。

使用 MongoDB 或 MongoDB Atlas 連線

建立 MongoDB 或 MongoDB Atlas 連線後，您就可以在 ETL 任務中使用此連線。您可以在 AWS Glue Data Catalog 建立資料表並為資料表的 `connection` 屬性指定 MongoDB 或 MongoDB Atlas 連線。

AWS Glue 將您的連線 `url` 和憑證存放在 MongoDB 連線中。連線 URI 格式如下：

- 若是 MongoDB：mongodb://host:port/database。主機可以是主機名稱、IP 地址或 UNIX 域通訊端。如果連接字串沒有指定連接埠，則會使用預設的 MongoDB 連接埠 27017。
- 若是 MongoDB Atlas：mongodb+srv://server.example.com/database。主機可以是遵循對應於 DNS SRV 記錄的主機名稱。SRV 格式不需要連接埠，而且會使用預設的 MongoDB 連接埠 27017。

此外，您也可以在工作指令碼中指定選項。如需更多詳細資訊，請參閱 the section called “MongoDB 連線”。

使用 VPC 端點網路爬取 Amazon S3 資料存放區

出於安全性、稽核或控制目的，您可能希望 Amazon S3 資料存放區或 Amazon S3 支援的 Data Catalog 資料表只能透過 Amazon Virtual Private Cloud 環境 (Amazon VPC) 存取。本主題說明如何使用 `Network` 連線類型，在 VPC 端點中建立和測試 Amazon S3 資料存放區或 Amazon S3 支援的資料目錄資料表的連線。

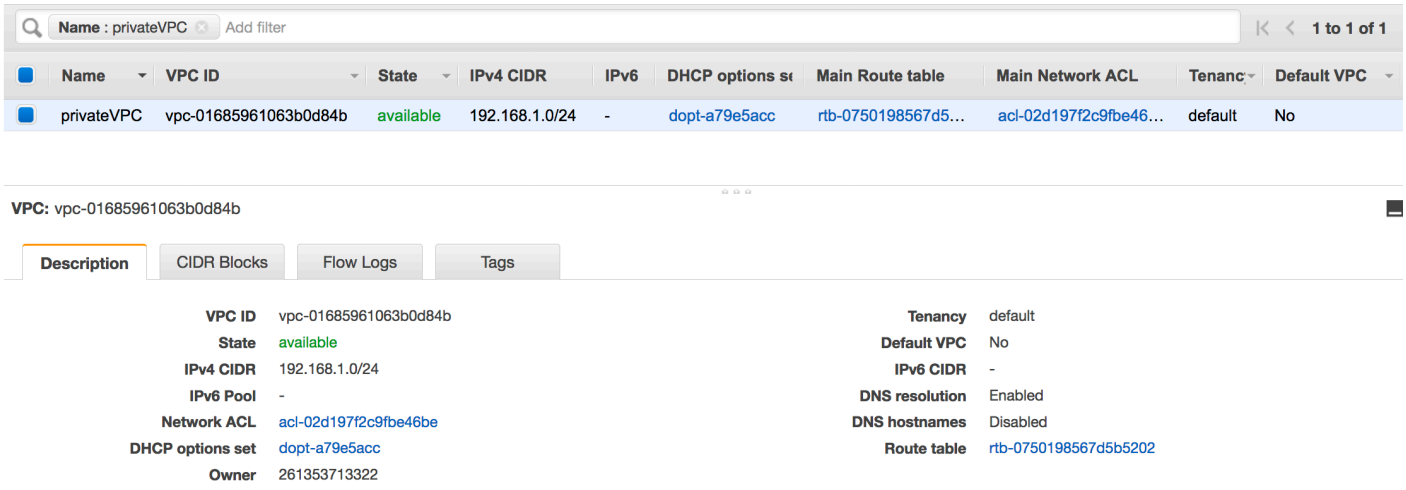
執行下列工作，以在資料存放區上執行爬蟲程式：

- the section called “必要條件”
- the section called “建立與 Amazon S3 的連線”
- the section called “測試與 Amazon S3 的連線”
- the section called “為 Amazon S3 資料存放區建立爬蟲程式”
- the section called “執行爬蟲程式”

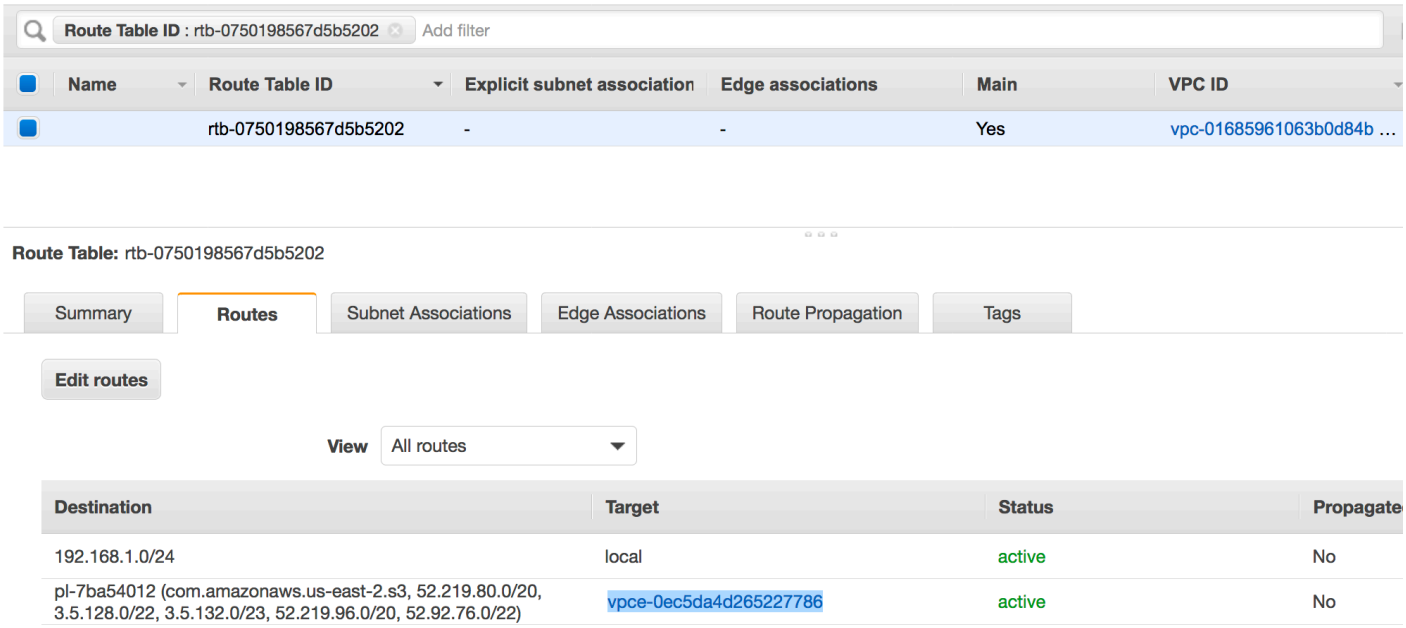
必要條件

檢查您是否符合這些先決條件，以便將 Amazon S3 資料存放區或 Amazon S3 支援的資料型錄資料表設定為透過 Amazon Virtual Private Cloud 環境 (Amazon VPC) 存取。

- 已設定的 VPC。例如：vpc-01685961063b0d84b。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的 Amazon VPC 入門。
- 連接到 VPC 的 Amazon S3 端點。例如：vpc-01685961063b0d84b。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的適用於 Amazon S3 的端點。



- 指向 VPC 端點的路由項目。例如，VPC 端點 (vpce-0ec5da4d265227786) 所使用的路由表中的 vpce-0ec5da4d265227786。



- 連接到 VPC 的網路 ACL 允許流量。
- 連接至 VPC 的安全群組允許流量。

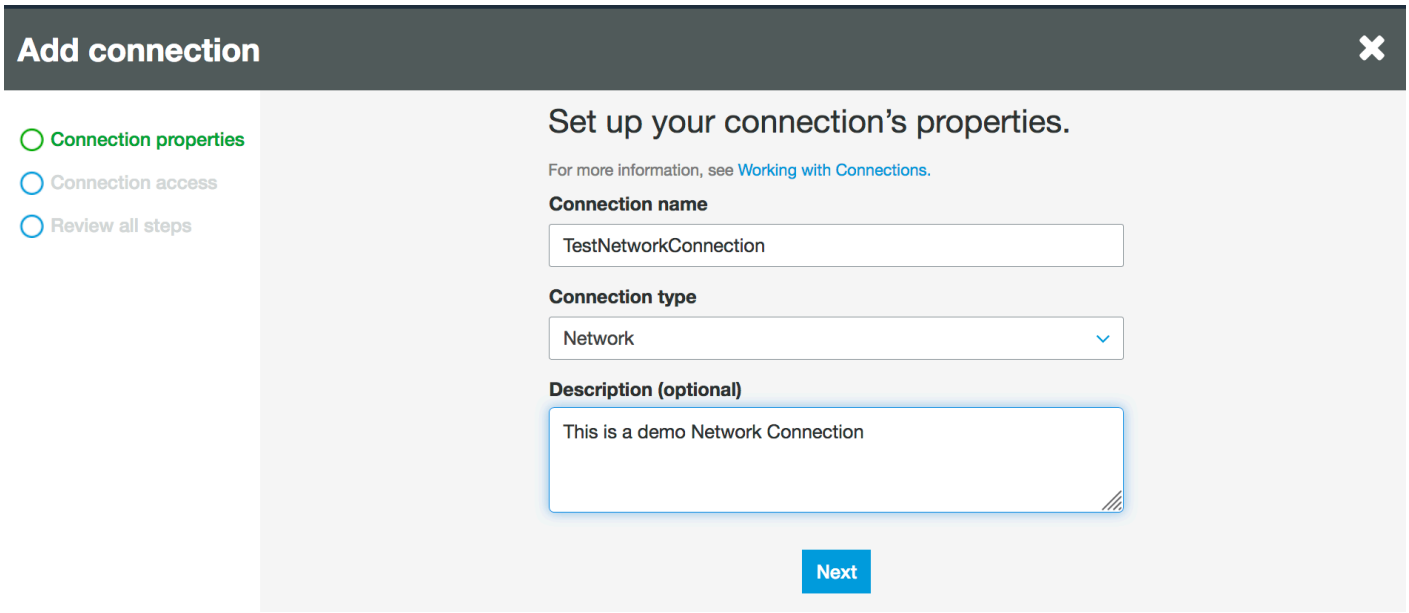
建立與 Amazon S3 的連線

通常，您是在 Amazon Virtual Private Cloud (Amazon VPC) 內部建立資源，因此無法經由公有網際網路進行存取。依預設，AWS Glue 無法存取 VPC 之內的資源。要讓 AWS Glue 存取 VPC 中的資源，您必須提供其他 VPC 特定的組態資訊，包括 VPC 子網路 ID 和安全群組 ID。建立 `Network` 連線時需要指定以下資訊：

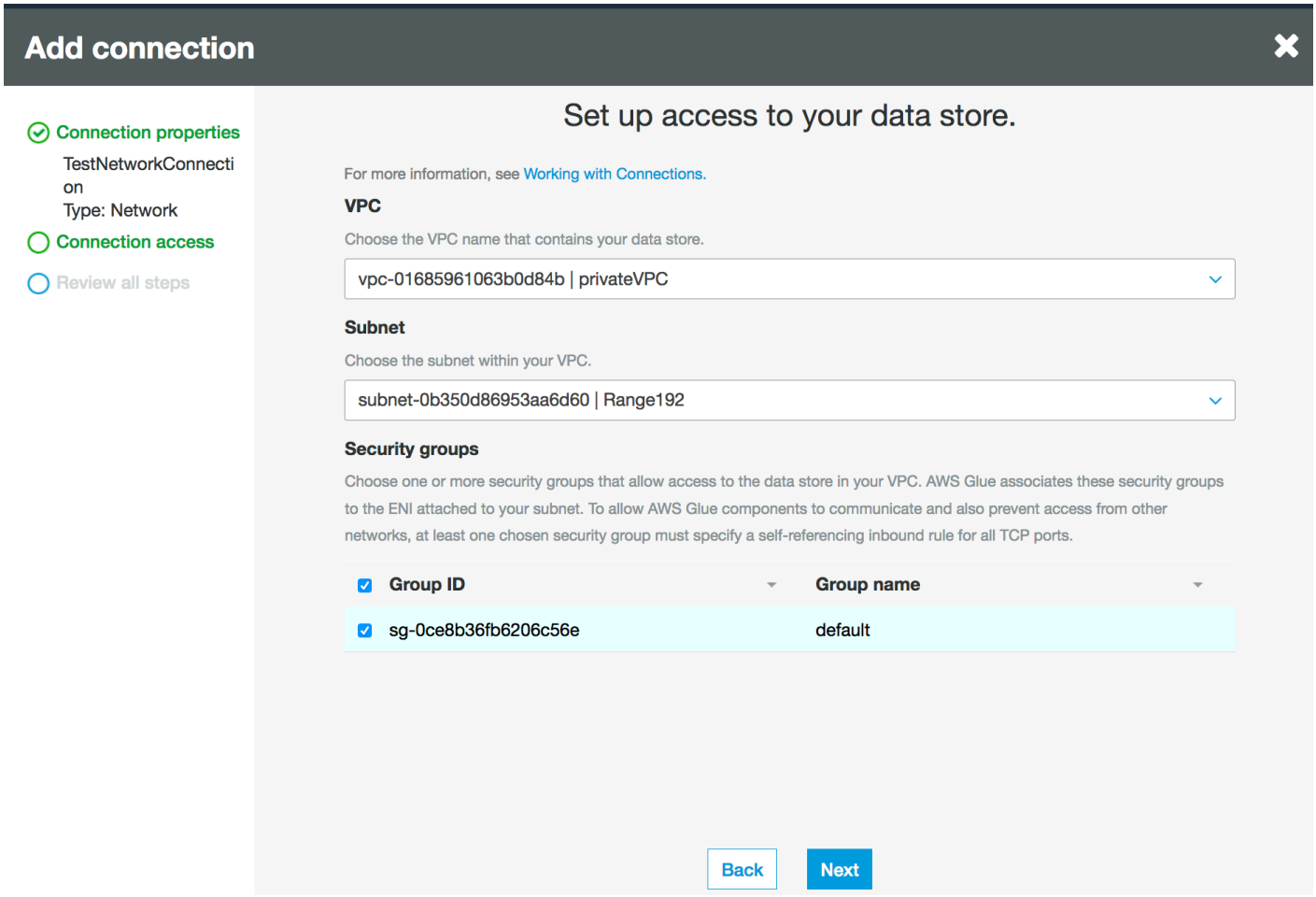
- VPC ID
- VPC 內的子網路
- 安全群組

設定 Network 連線：

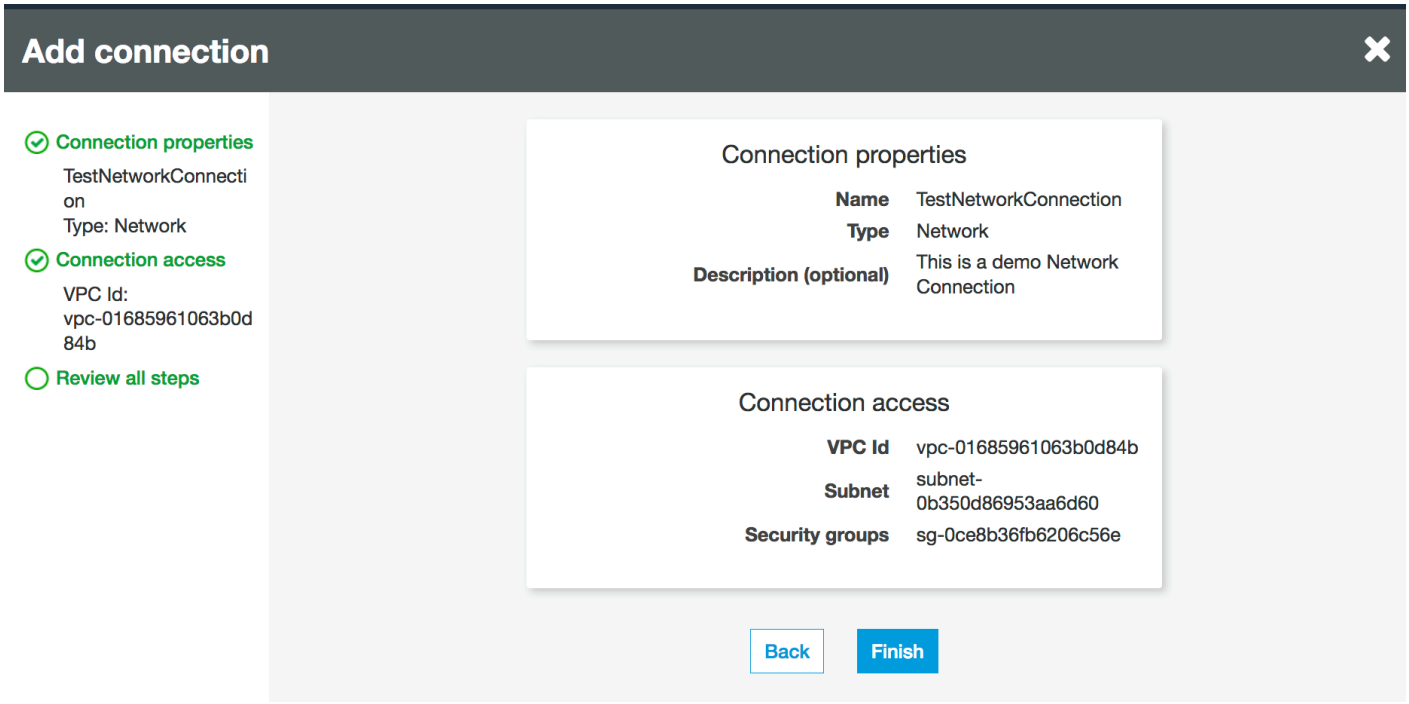
1. 選擇 AWS Glue 主控台導覽窗格中的 Add connection (新增連線)。
2. 輸入連線名稱，選擇 Network (網路) 做為連線類型。選擇 Next (下一步)。



3. 設定 VPC、子網路和安全群組資訊。
   - VPC：選擇包含資料存放區的 VPC 名稱。
   - 子網路：選擇 VPC 中的子網路。
   - 安全群組：選擇一個或多個允許存取 VPC 中資料存放區的安全群組。



4. 選擇 Next (下一步)。
5. 驗證連線資訊，然後選擇 Finish (完成)。



測試與 Amazon S3 的連線

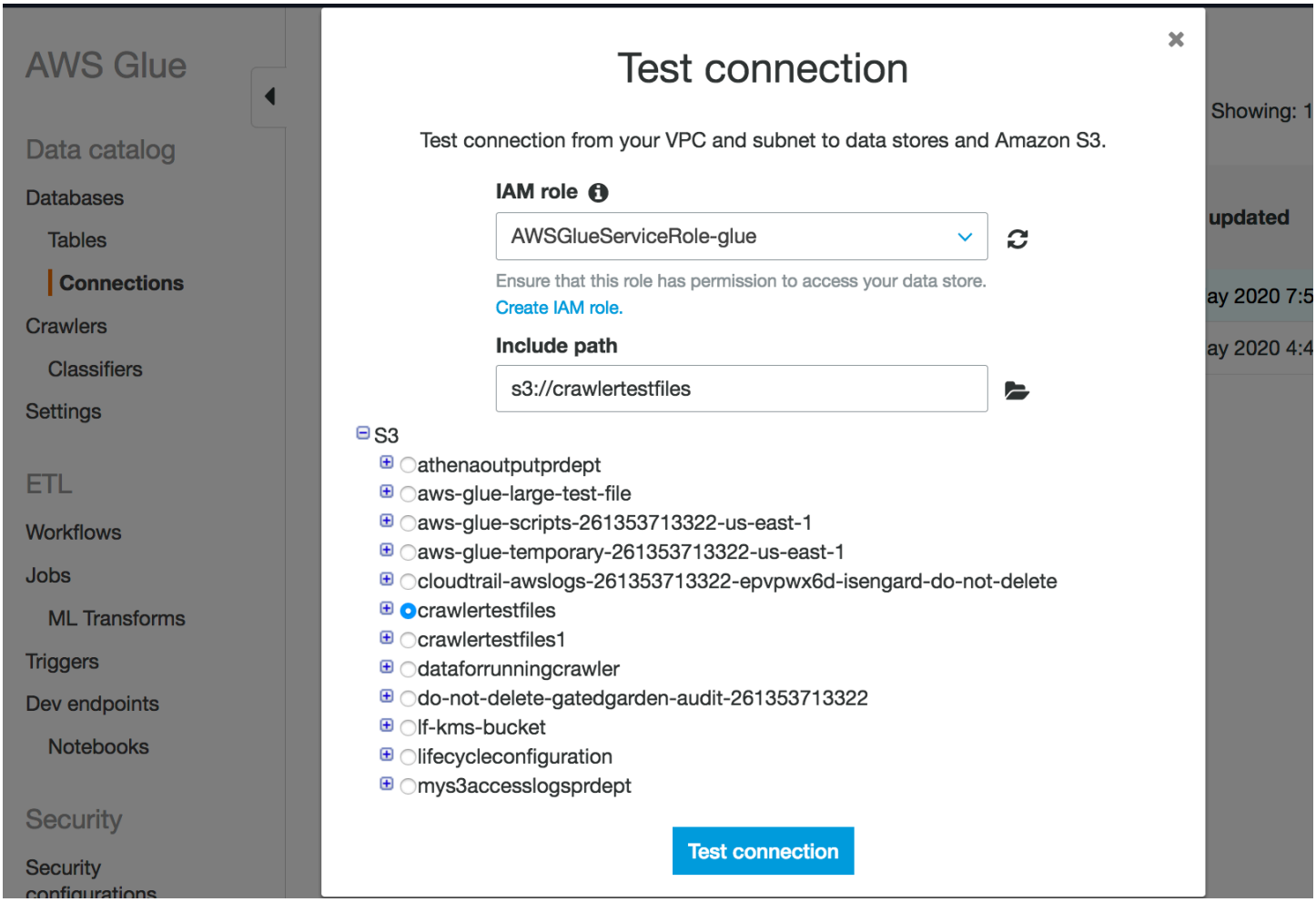
建立您的 Network 連線後，您可以在 VPC 端點中測試與 Amazon S3 資料存放區的連線。

測試連線時，可能會發生下列錯誤：

- INTERNET CONNECTION ERROR：表示網際網路連線問題
- INVALID BUCKET ERROR：表示 Amazon S3 儲存貯體發生問題
- S3 CONNECTION ERROR：表示無法連線至 Amazon S3
- INVALID CONNECTION TYPE：表示連線類型沒有預期的值，NETWORK
- INVALID CONNECTION TEST TYPE：表示網路連線測試類型有問題
- INVALID TARGET：表示尚未正確指定 Amazon S3 儲存貯體

測試 Network 連線：

1. 在 AWS Glue 主控台中選取 Network (網路) 連線。
2. 選擇 Test connection (測試連線)。
3. 選擇您在上一步驟中建立的 IAM 角色，並指定 Amazon S3 儲存貯體。
4. 選擇 Test connection (測試連線) 開始測試。可能需要幾分鐘才能顯示結果。



如果您收到錯誤，請檢查下列項目：

- 系統會為所選角色提供正確的權限。
- 已提供正確的 Amazon S3 儲存貯體。
- 安全群組和網路 ACL 允許所需的輸入和輸出流量。
- 您指定的 VPC 已連線到 Amazon S3 VPC 端點。

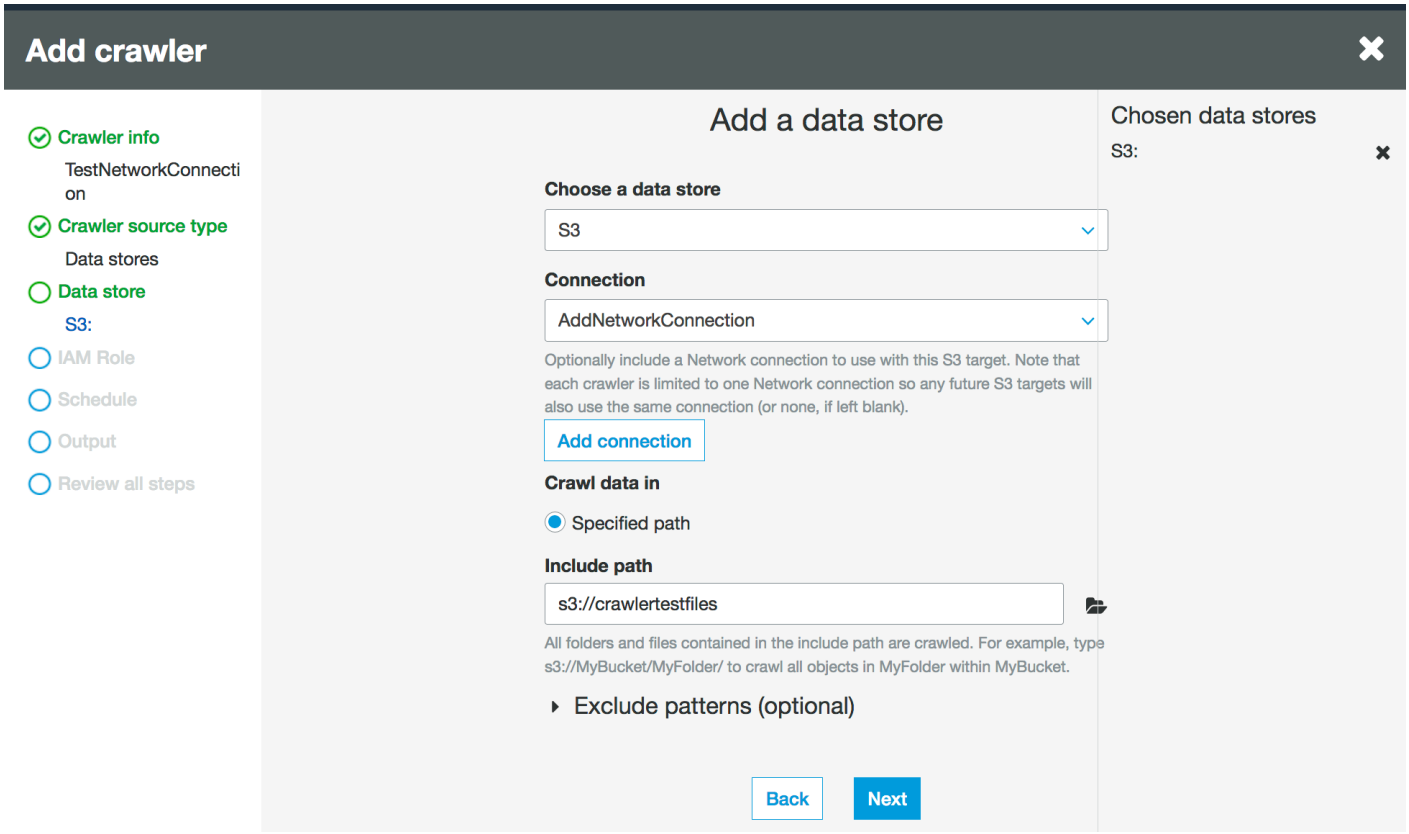
成功測試連線後，您就可以建立爬蟲程式。

為 Amazon S3 資料存放區建立爬蟲程式

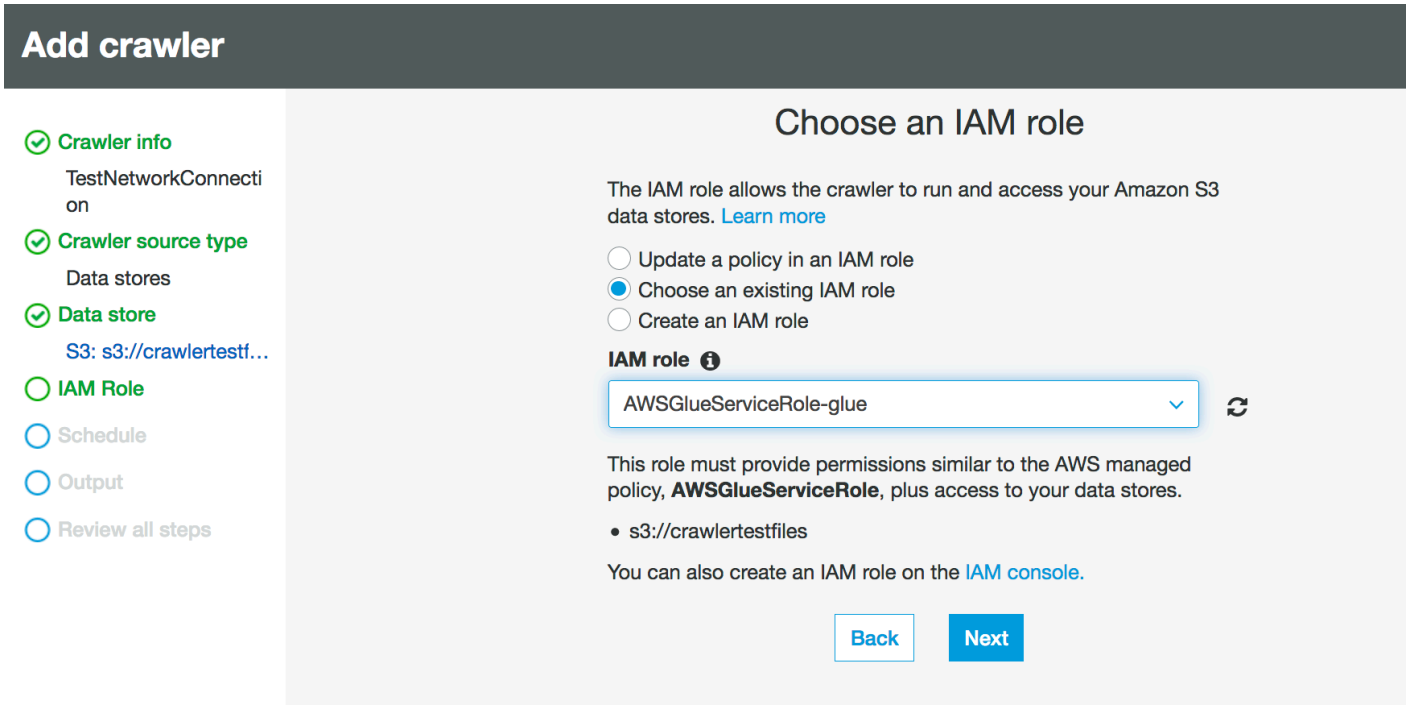
您現在可以建立爬蟲程式來指定您已建立的 `Network` 連線。如需建立爬蟲程式的詳細資訊，請參閱設定爬行者程式。

1. 首先選擇 AWS Glue 主控台導覽窗格中的 Crawlers (爬蟲程式)。

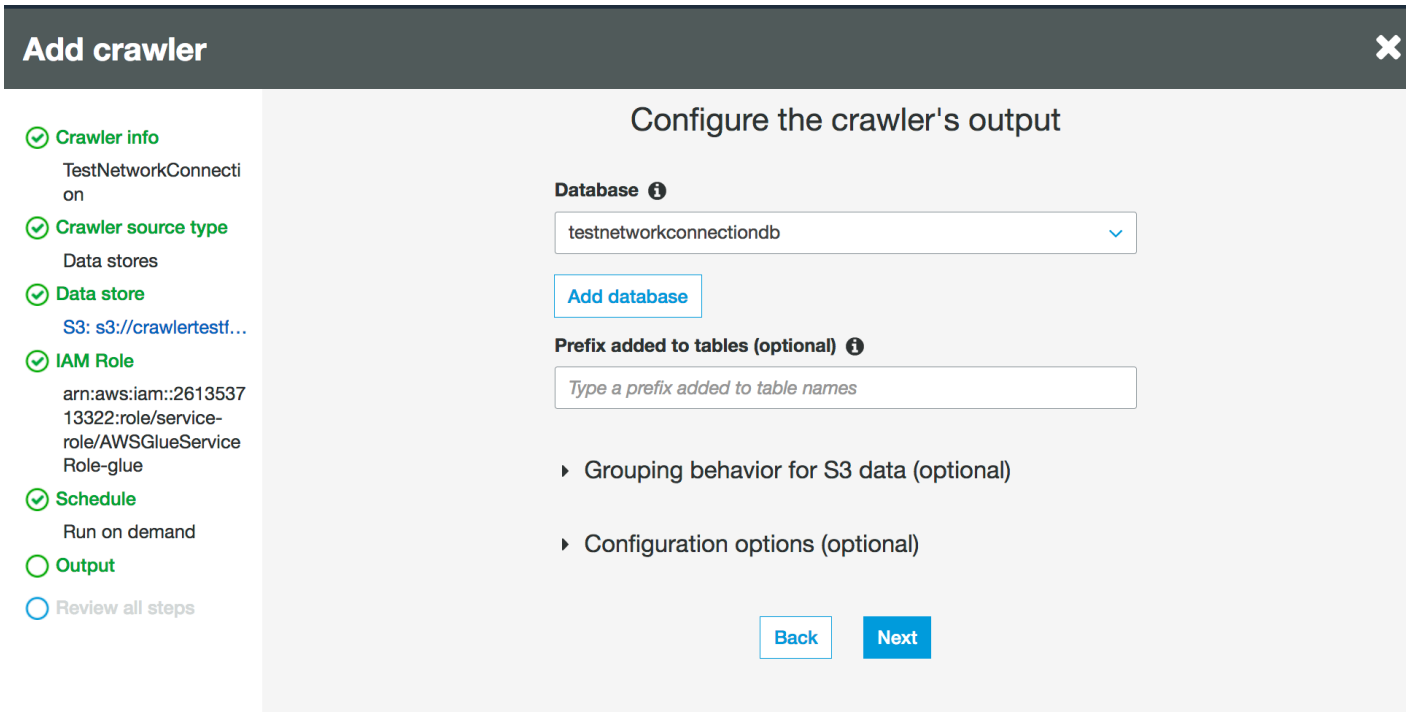
2. 選擇 Add crawler (新增爬蟲程式)。
3. 指定爬蟲程式名稱，然後選擇 Next (下一步)。
4. 系統詢問資料來源時，請選擇 S3，然後指定 Amazon S3 儲存貯體字首和您先前建立的連線。



5. 如果需要，請在相同的網路連線上新增其他資料存放區。
6. 選擇 IAM 角色。IAM 角色必須允許存取 AWS Glue 服務和 Amazon S3 儲存貯體。如需詳細資訊，請參閱 the section called “設定爬行者程式”。



7. 定義爬蟲程式的排程。
8. 在 Data Catalog 中選擇既有的資料庫，或建立新的資料庫項目。

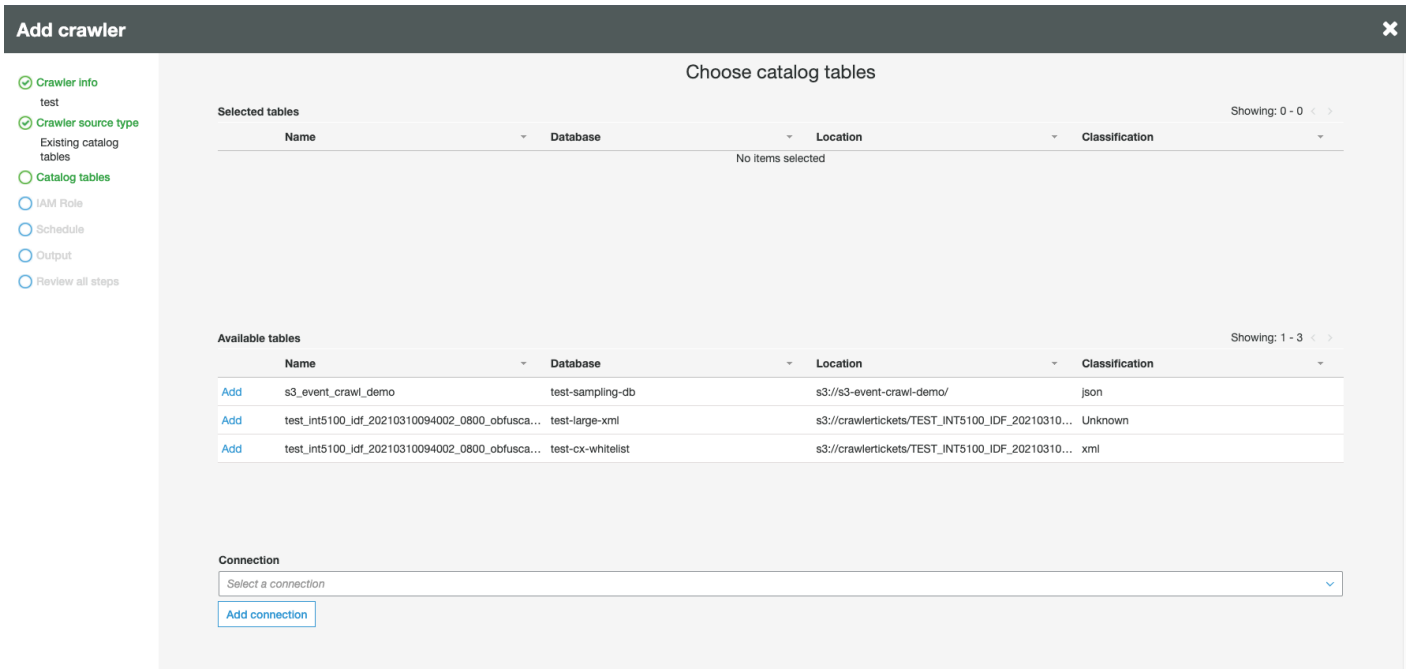


9. 完成剩餘的設定。

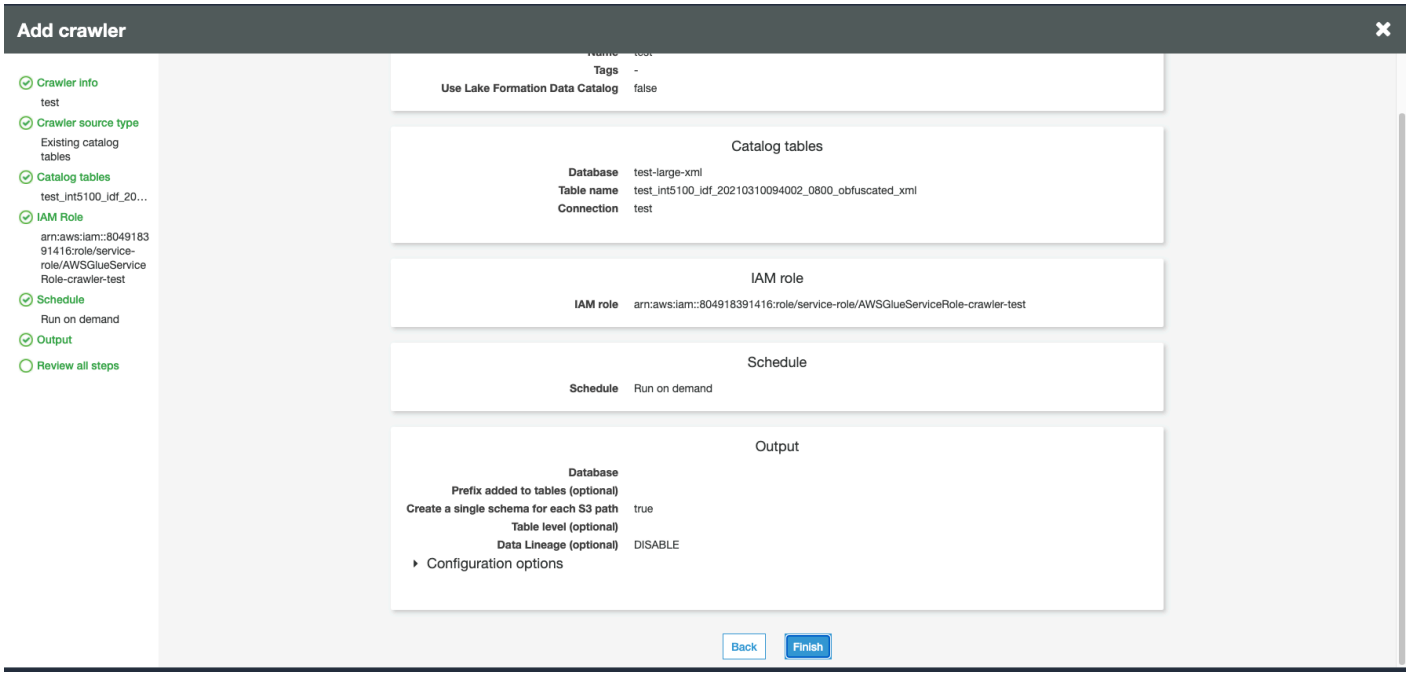
為 Amazon S3 支援的 Data Catalog 資料表建立爬蟲程式

您現在可以建立爬蟲程式來指定您已建立的 `Network` 連線和目錄來源類型。如需建立爬蟲程式的詳細資訊，請參閱 設定爬行者程式。

1. 首先選擇 AWS Glue 主控台導覽窗格中的 Crawlers (爬蟲程式)。
2. 選擇 Add crawler (新增爬蟲程式)。
3. 指定爬蟲程式名稱，然後選擇 Next (下一步)。
4. 當系統要求提供爬蟲程式來源類型時，選擇 Existing catalog tables (現有目錄資料表)，並指定要從可用資料表清單中搜尋的現有目錄資料表。

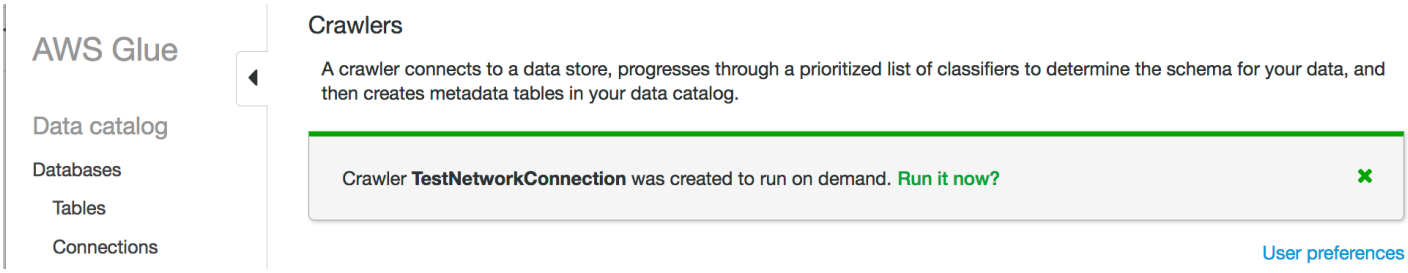


5. 選擇 IAM 角色。IAM 角色必須允許存取 AWS Glue 服務和 Amazon S3 儲存貯體。如需詳細資訊，請參閱 the section called “設定爬行者程式”。
6. 定義爬蟲程式的排程。
7. 在 Data Catalog 中選擇既有的資料庫，或建立新的資料庫項目。
8. 完成剩餘的設定並審查您的步驟。



執行爬蟲程式

執行您的爬蟲程式。



故障診斷

如需使用 VPC 閘道與 Amazon S3 儲存貯體相關的疑難排解，請參閱為什麼我無法使用閘道 VPC 端點連接到 S3 儲存貯體？

針對 AWS Glue 中的連線問題進行故障診斷

當 AWS Glue 爬蟲程式或任務使用連線屬性來存取資料存放區時，可能會在您嘗試連線時發生錯誤。當 AWS Glue 在您指定的虛擬私有雲端 (VPC) 和子網路中建立彈性網路界面時，會使用子網路中的私有 IP 地址。 連線中指定的安全群組將套用至每個彈性網路界面。檢查安全群組是否允許對外存取，以及是否允許連線到資料庫叢集。

此外，Apache Spark 需要驅動程式和執行器節點之間的雙向連線。安全群組的需求之一是允許所有 TCP 連接埠的輸入規則。您可以利用自我參考的安全群組，將安全群組的來源限制為安全群組本身，以防止向全世界開放。

以下是您可採取的一些典型動作以排除連線問題：

- 檢查您的連線的連接埠位址。
- 檢查您連線或秘密中的使用者名稱和密碼字串。
- 對於 JDBC 資料存放區，驗證它允許傳入的連線。
- 確認您的資料存放區可在 VPC 內存取。
- 如果您使用 AWS Secrets Manager 存放連線憑證，請確定 AWS Glue 的 IAM 角色具有存取秘密的許可。如需詳細資訊，請參閱《AWS Secrets Manager 使用者指南》中的 Example: Permission to retrieve secret values (範例：擷取秘密值的許可)。根據您的網路設定而定，您可能還需要建立 VPC 端點，以在 VPC 與 Secrets Manager 之間建立私有連線。如需詳細資訊，請參閱使用 AWS Secrets Manager VPC 端點。

教學課程：使用 AWS Glue Elasticsearch Connector

Elasticsearch 是受歡迎的開源搜尋和分析引擎，適用於例如日誌分析、即時應用程式監控及點擊流分析等使用案例。您可以在 AWS Glue Studio 中設定 AWS Glue Elasticsearch Connector，利用 OpenSearch 作為擷取、轉換和載入 (ETL) 任務的資料存放區。此連接器可免費從 AWS Marketplace 取得。

> **Note**
>
> AWS Marketplace Elasticsearch Spark Connector 已棄用。請改用 AWS Glue Elasticsearch Connector。

在本教學課程中，我們將展示如何以最少的步驟連接到您的 Amazon OpenSearch Service 節點。

主題

- 先決條件
- 步驟 1：(選用) 為您的 OpenSearch 叢集資訊建立 AWS 秘密
- 步驟 2：訂閱連接器
- 步驟 3：在 AWS Glue Studio 中啟用連接器和建立連線

- 步驟 4：設定 ETL 任務的 IAM 角色
- 步驟 5：建立使用 OpenSearch 連線的任務
- 步驟 6：執行任務

先決條件

若要使用本教學課程，您必須具備下列項目：

- AWS Glue Studio 的存取權限
- 能夠存取 AWS Cloud 中的 OpenSearch 叢集
- (選用) 存取 AWS Secrets Manager。

步驟 1：(選用) 為您的 OpenSearch 叢集資訊建立 AWS 秘密

若要安全地存放和使用您的連線憑證，請將您的憑證儲存在 AWS Secrets Manager。您建立的秘密將在教學課程稍後由連線使用。憑證鍵值對將作為正常連線選項輸入 AWS Glue Elasticsearch Connector。

如需建立秘密的詳細資訊，請參閱 AWS Secrets Manager 使用者指南中的使用 AWS Secrets Manager 建立和管理秘密。

建立 AWS 秘密

1. 登入 AWS Secrets Manager 主控台。
2. 在服務簡介頁面或 Secrets (秘密) 清單頁面上，選擇 Store a new secret (存放新的秘密)。
3. 在 Store a new secret (存放新的秘密) 頁面上，選擇 Other type of secret (其他秘密類型)。這個選項表示您必須提供秘密架構和詳細資訊。
4. 新增金鑰和數值配對，以取得 OpenSearch 叢集使用者名稱。例如：

   ```
   es.net.http.auth.user: #####
   ```
5. 選擇 + Add row (+ 新增列)，然後輸入密碼的另一個鍵值對。例如：

   ```
   es.net.http.auth.pass: ##
   ```
6. 選擇 Next (下一步)。
7. 輸入秘密名稱。例如：my-es-secret。您也可以選擇輸入說明。

   記錄秘密名稱 (此名稱稍後會在本教學課程中使用)，然後選擇 Next (下一步)。

8. 再次選擇 Next (下一步)，然後選擇 Store (存放) 建立秘密。

下一個步驟

步驟 2：訂閱連接器

步驟 2：訂閱連接器

AWS Glue Elasticsearch Connector 可從 AWS Marketplace 免費取得。

在 AWS Marketplace 上訂閱 AWS Glue Elasticsearch Connector

1. 如果您尚未設定 AWS 帳戶以使用 License Manager，請執行下列作業：
   a. 開啟位於 https://console.aws.amazon.com/license-manager 的 AWS License Manager 主控台。
   b. 選擇 Create customer managed license (建立客戶受管授權)。
   c. 在 IAM permissions (one-time setup)(IAM 許可 (一次性設定)) 畫面中，選取 I grant AWS License Manager the required permissions (我要授予 AWS License Manager 這些必要許可)，然後選擇 Grant permissions (授予許可)。

      如果您沒有看到此視窗，表示您已經設定必要的許可。
2. 開啟位於 https://console.aws.amazon.com/gluestudio/ 的 AWS Glue Studio 主控台。
3. 在 AWS Glue Studio 主控台中，展開選單圖示 (☰)。然後在導覽窗格中選擇 Connectors (連接器)。
4. 在連接器頁面上，選擇移至 AWS Marketplace。
5. 在 AWS Marketplace 的搜尋 AWS Glue Studio 產品區段內，輸入 AWS Glue Elasticsearch 連接器，然後按 Enter 鍵。
6. 選擇連接器的名稱 AWS Glue Elasticsearch Connector。
7. 在連接器的產品頁面上，使用索引標籤來檢視連接器的相關資訊。準備好繼續時，請選擇 Continue to Subscribe (繼續訂閱)。
8. 檢閱軟體的使用條款。按一下 Accept Terms (接受條款)。
9. 訂閱程序完成後，您將看到一條通知："Thank you for subscribing to this product! You can now configure your software." (感謝您訂閱此產品！您現在可進行軟體設定。) 橫幅上方是按鈕 Continue to Configuration (繼續進行設定)。選擇 Continue to Configuration (繼續進行設定)。

10. 請在 Configure this software (設定此軟體) 頁面上選擇 Fulfillment (履行) 選項。您可選擇 AWS Glue 1.0/2.0 或 AWS Glue 3.0。然後選擇 Continue to Launch (繼續啟動)。

下一步驟

步驟 3：在 AWS Glue Studio 中啟用連接器和建立連線

步驟 3：在 AWS Glue Studio 中啟用連接器和建立連線

在您選擇 Continue to Launch (繼續啟動) 後，您會在 AWS Marketplace 中看到 Launch this software (啟動此軟體) 頁面。在 AWS Glue Studio 中使用連結啟動連接器後，您會建立連線。

在 AWS Glue Studio 中部署連接器和建立連線

1. 在 AWS Marketplace 主控台的 Launch this software (啟動此軟體) 頁面，選擇Usage Instructions (使用說明)，然後選擇出現的視窗中的連結。

   您的瀏覽器將重新導向到 AWS Glue Studio 主控台 Create marketplace connection (建立市集連線) 頁面。
2. 輸入連線的名稱。例如：my-es-connection。
3. 在 Connection access (連線存取) 區段，對於 Connection credential type (連線憑證類型) 選擇 User name and password (使用者名稱和密碼)。
4. 對於 AWS secret (AWS 秘密)，輸入您的秘密名稱。例如：my-es-secret。
5. 在 Network options (網路選項) 區段中，輸入要連線到 OpenSearch 叢集的 VPC 資訊。
6. 選擇 Create connection and activate connector (建立連線並啟動連接器)。

下一步驟

步驟 4：設定 ETL 任務的 IAM 角色

步驟 4：設定 ETL 任務的 IAM 角色

建立 AWS Glue ETL 任務時，您可以指定 AWS Identity and Access Management (IAM) 角色，以供任務使用。角色必須授與任務使用的所有資源的存取權，包括 Amazon S3 (適用於任何來源、目標、指令碼、驅動程式檔案和暫存目錄)，以及 AWS Glue Data Catalog 物件。

AWS Glue ETL 任務的擔任 IAM 角色也必須能夠存取上一節中建立的秘密。根據預設，AWS 受管角色 `AWSGlueServiceRole` 無法存取秘密。若要設定秘密的存取控制，請參閱 AWS Secrets Manager 的身分驗證與存取控制和限制對特定秘密的存取。

設定 ETL 任務的 IAM 角色

1. 設定the section called “檢閱 ETL 任務所需的 IAM 許可”中所述的許可。
2. 設定搭配 AWS Glue Studio 使用連接器時所需的額外許可，如 the section called “使用連接器所需的許可” 中所述。

下一步驟

步驟 5：建立使用 OpenSearch 連線的任務

步驟 5：建立使用 OpenSearch 連線的任務

為您的 ETL 任務建立角色後，您可以在 AWS Glue Studio 中建立任務，來使用 Open Spark ElasticSearch 的連線和連接器。

如果您的任務在 Amazon Virtual Private Cloud (Amazon VPC) 內執行，請確保 VPC 設定正確。如需更多詳細資訊，請參閱the section called “為您的 ETL 任務設定 VPC”。

建立使用 Elasticsearch Spark Connector 的任務

1. 在 AWS Glue Studio 中，選擇 Connectors (連接器)。
2. 在 Your connections (您的連線) 清單中，選取您剛才建立的連線，然後選擇 Create job (建立任務)。
3. 在視覺化任務編輯器中，選擇 [資料來源] 節點。在右側，於 Data source properties - Connector (資料來源屬性 - 連接器) 索引標籤上，設定連接器的其他資訊。

    a. 選擇 Add schema (新增結構描述)，然後在資料來源中輸入資料集的結構描述。連線不會使用存放在 Data Catalog 中的資料表，這表示 AWS Glue Studio 不知道資料的結構描述。您必須手動提供此結構描述資訊。如需如何使用結構描述編輯器的指示，請參閱the section called “編輯自訂轉換節點的結構描述”。

    b. 展開 Connection options (連線選項)。

    c. 選擇 Add new option (新增選項)，然後輸入未在 AWS 秘密中輸入的連接器所需資訊：

        - es.nodes：https://<OpenSearch 網域端點>

- es.port：443
- path：test
- es.nodes.wan.only: true

如需這些連線選項的說明，請參閱：https://www.elastic.co/guide/en/elasticsearch/hadoop/current/configuration.html。

4. 將目標節點新增至圖形。

   您的資料目標可以是 Amazon S3，也可以使用來自 AWS Glue Data Catalog 或連接器的資訊將資料寫入不同位置。例如，您可以使用 Data Catalog 資料表來寫入 Amazon RDS 中的資料庫，或者您可以使用連接器做為資料目標來寫入 AWS Glue 原生不支援的資料存放區。

   如果您為資料目標選擇連接器，則必須選擇為該連接器建立的連線。此外，如果連接器提供者要求，您必須新增選項，以提供其他資訊給連接器。如果您使用的連線包含 AWS 秘密的資訊，則您不需要在連線選項中提供使用者名稱和密碼驗證。

5. 可以選擇新增其他資料來源和一或多個轉換節點，如the section called “編輯 AWS Glue 受管資料轉換節點”中所述。
6. 如the section called “修改任務屬性”所述設定任務屬性，從步驟 3 開始，然後儲存任務。

下一步驟

步驟 6：執行任務

步驟 6：執行任務

儲存任務後，您可以執行任務來執行 ETL 任務。

執行為 AWS Glue Elasticsearch Connector 建立的任務

1. 使用 AWS Glue Studio 主控台，在視覺化編輯器頁面上，選擇 Run (執行)。
2. 在成功橫幅中，選擇 Run Details (執行詳細資訊)，或者您可以選擇視覺化編輯器的 Runs (執行)索引標籤，以檢視任務執行的相關資訊。

使用互動式 AWS Glue 工作階段建立

使用 AWS Glue 中的互動式工作階段，資料工程師可以比之前更快、更輕鬆地編寫 AWS Glue 任務。

主題

AWS Glue 互動式工作階段概觀

使用 AWS Glue 互動式工作階段，您可以快速建置、測試和執行資料準備和分析應用程式。互動式工作階段提供用於建置和測試資料準備之擷取、轉換和載入 (ETL) 指令碼的程式化和視覺化介面。互動式工作階段執行 Apache Spark 分析應用程式，並提供對遠端 Spark 執行階段環境的隨需存取權。AWS Glue 透明地管理這些互動式工作階段的無伺服器 Spark。

互動式工作階段具有彈性，因此您可以從選擇的環境建置和測試應用程式。您可以透過和 API 建立和使用互動式工作階段。 AWS Command Line Interface 您可以使用與 Jupyter 相容的筆記本，以視覺化方式編寫和測試您的筆記本指令碼。互動式工作階段提供開放原始碼 Jupyter 核心，該核心幾乎可以整合 Jupyter 所做的任何位置，包括整合 IntelliJ 和 VS 程式碼等 PyCharm IDE。這可讓您在本機環境中編寫程式碼，並在互動式工作階段後端順暢地執行程式碼。

使用互動式工作階段 API，客戶可以透過程式設計方式執行使用 Apache Spark 分析的應用程式，而無需管理 Spark 基礎設施。您可以在單一互動式工作階段內執行一或多個 Spark 陳述式。

因此，互動式工作階段提供更快速、便宜、靈活的方式來建置和執行資料準備和分析應用程式。若要了解如何使用互動式工作階段，請參閱本節中的文件。AWS Glue 支援的魔術命令

限制

- 互動式工作階段中不支援任務書籤。
- 不支援使用建立筆記本工作。 AWS Command Line Interface

AWS Glue 互動式工作階段入門

這些章節說明如何執行 AWS Glue 本機互動式工作階段。

在本機設定互動式工作階段的先決條件

下列是安裝互動式工作階段的先決條件：

- 支援的 Python 版本為 3.6 至 3.10 以上版本。
- 請參閱以下章節來取得 MacOS/Linux 和 Windows 的相關指示。

安裝 Jupyter 和 AWS Glue 互動式工作階段 Jupyter 內核

使用以下命令在本機安裝核心。

`install-glue-kernels` 命令會同時為 Pyspark 和 Spark 核心安裝 jupyter kernelspec，並在正確的目錄中安裝標誌。

```
pip3 install --upgrade jupyter boto3 aws-glue-sessions
```

```
install-glue-kernels
```

執行 Jupyter

若要執行 Jupyter 筆記本，請完成以下步驟。

1. 執行下列命令以啟動 Jupyter 筆記本。

   ```
   jupyter notebook
   ```

2. 選擇 New (新增)，然後選擇其中一個 AWS Glue 核心以開始針對 AWS Glue 編碼。

設定工作階段憑證和區域

MacOS/Linux 指示

AWS Glue 互動式工作階段需要與 AWS Glue 任務和開發端點相同的 IAM 許可。使用下列兩種方式之一指定與互動式工作階段搭配使用的角色：

1. 使用 `%iam_role` 和 `%region` 魔術命令
2. 使用 `~/.aws/config` 中額外的行

使用魔術命令設定工作階段角色

在第一個儲存格中，在執行的第一個儲存格中鍵入 `%iam_role <YourGlueServiceRole>`。

Configuring a session role with **`~/.aws/config`** (使用 設定工作階段角色)

AWS Glue互動式工作階段的服務角色可以在筆記本本身中指定，或與 AWS CLI 組態一起儲存。如果您有一個通常與 AWS Glue 任務一起使用的角色，那麼這就是該角色。如果您沒有用於 AWS Glue 任務的角色，請遵循本指南：《設定用於 AWS Glue 的 IAM 許可》來進行設定。

若要將此角色設定為互動式工作階段的預設角色：

1. 在文字編輯器中開啟 `~/.aws/config`。
2. 尋找您用於 AWS Glue 的設定檔。如果您不使用設定檔，請使用 `[Default]` 設定檔。
3. 在設定檔中為您打算使用的角色新增一行，如 `glue_role_arn=<AWSGlueServiceRole>`。
4. [選用]：如果您的設定檔沒有設定預設區域，建議使用 `region=us-east-1` 新增一個區域，將 `us-east-1` 替換為您所需的區域。
5. 儲存組態。

如需詳細資訊，請參閱 具備 IAM 的互動式工作階段。

Windows 指示

AWS Glue 互動式工作階段需要與 AWS Glue 任務和開發端點相同的 IAM 許可。使用下列兩種方式之一指定與互動式工作階段搭配使用的角色：

1. 使用 `%iam_role` 和 `%region` 魔術命令

2. 使用 ~/.aws/config 中額外的行

使用魔術命令設定工作階段角色

在第一個儲存格中，在執行的第一個儲存格中鍵入 %iam_role <YourGlueServiceRole>。

使用 **~/.aws/config** 設定工作階段角色

AWS Glue互動式工作階段的服務角色可以在筆記本本身中指定，或與 AWS CLI 組態一起儲存。如果您有一個通常與 AWS Glue 任務一起使用的角色，那麼這就是該角色。如果您沒有用於 AWS Glue 任務的角色，請遵循本指南：設定用於 AWS Glue 的 IAM 許可，來進行設定。

若要將此角色設定為互動式工作階段的預設角色：

1. 在文字編輯器中開啟 ~/.aws/config。
2. 尋找您用於 AWS Glue 的設定檔。如果您不使用設定檔，請使用 [Default] 設定檔。
3. 在設定檔中為您打算使用的角色新增一行，如 glue_role_arn=<AWSGlueServiceRole>。
4. [選用]：如果您的設定檔沒有設定預設區域，建議使用 region=us-east-1 新增一個區域，將 us-east-1 替換為您所需的區域。
5. 儲存組態。

如需詳細資訊，請參閱 具備 IAM 的互動式工作階段。

從互動式工作階段預覽版升級

在 0.27 版本發佈時，核心已升級並擁有新名稱。要清理內核的預覽版本，請從終端或 PowerShell運行以下內容。

> **Note**
>
> 如果您的核心是需要自訂服務模型的任何其他 AWS Glue 預覽版的一部分，移除核心會移除自訂服務模型。

```
# Remove Old Glue Kernels
jupyter kernelspec remove glue_python_kernel
jupyter kernelspec remove glue_scala_kernel
```

```
# Remove Custom Model
cd ~/.aws/models
rm -rf glue/
```

使用 SageMaker Studio 的互動式工作階段

AWS Glue 互動式工作階段是隨選、無伺服器的 Apache Spark 執行階段環境，資料科學家和工程師可以使用它來快速建置、測試和執行資料準備和分析應用程式。您可以啟動 Amazon SageMaker Studio 典型筆記本，以啟 AWS Glue 動互動式工作階段。

有關詳情，請參閱使用AWS Glue互動式工作階段準備資料。

將互動式工作階段與 Microsoft Visual Studio Code 搭配使用

先決條件

- 安裝 AWS Glue 互動式工作階段，並確認它能搭配 Jupyter 筆記本運作。
- 使用 Jupyter 下載並安裝與 Visual Studio Code。如需詳細資訊，請參閱 VS Code 中的 Jupyter 筆記本。

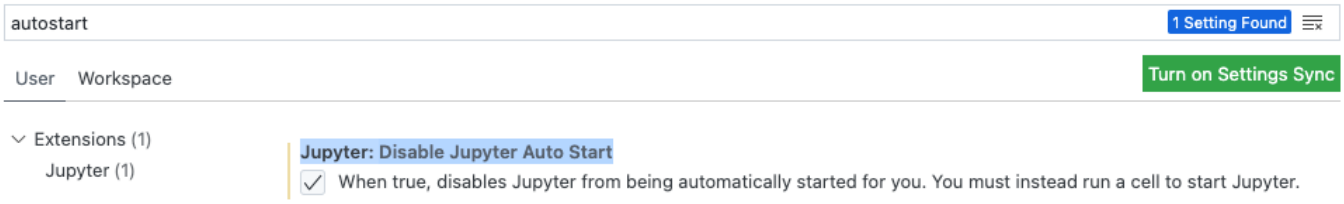
開始使用 VSCode 的互動式工作階段

1. AutoStart 在 VS 代碼中禁用 Jupyter。

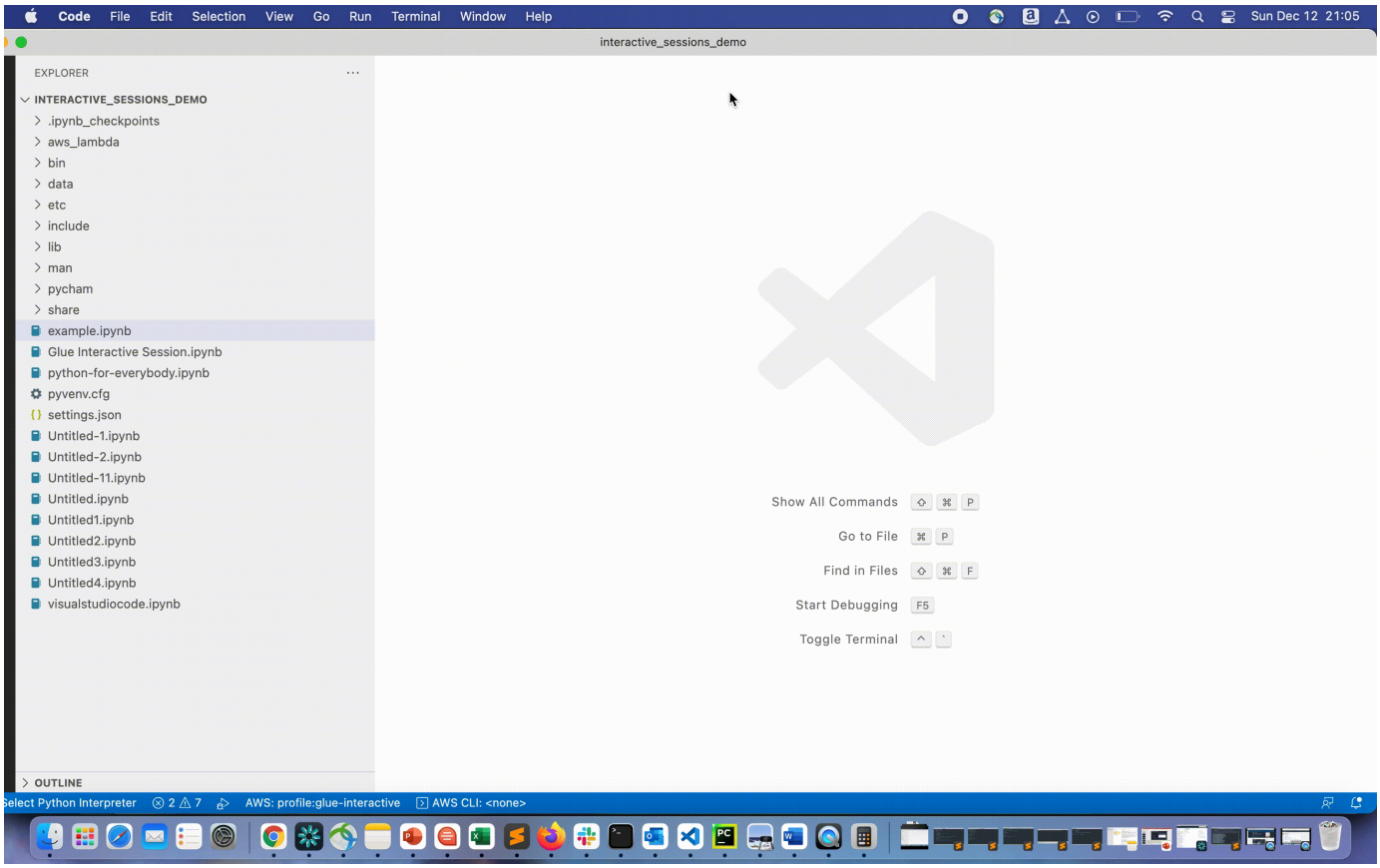
   在 Visual Studio Code 中，Jupyter 核心會自動啟動，這會阻止魔術命令生效，因為工作階段已經開始。若要在 Windows 上停用自動啟動，請前往檔案 > 偏好設定 > 延伸模組 > Jupyter > 用滑鼠右鍵按一下 Jupyter，然後選擇延伸模組設定。

   在 MacOS 上，前往程式碼 > 設定 > 延伸模組 > Jupyter > 用滑鼠右鍵按一下 Jupyter，然後選擇延伸模組設定。

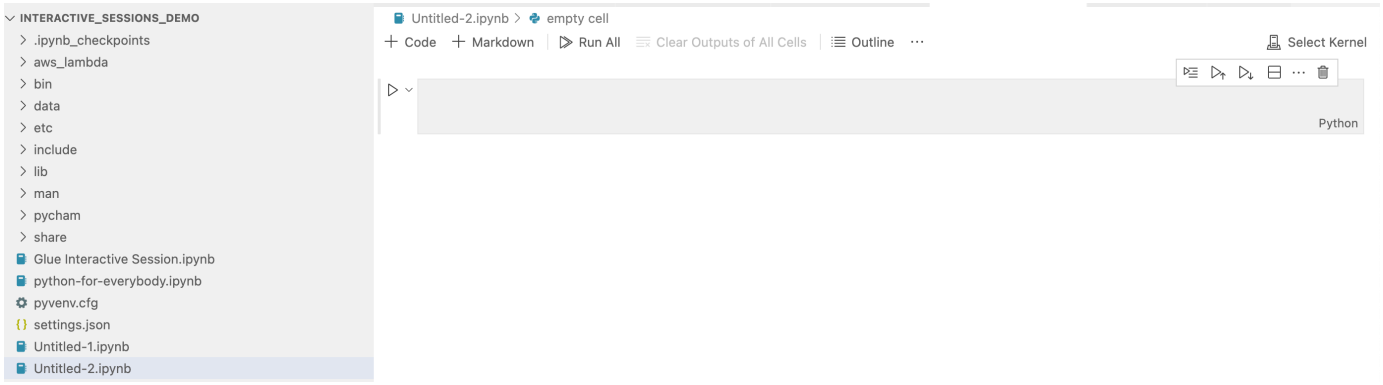
   向下捲動，直到顯示 Jupyter：停用 Jupyter 自動啟動。勾選標示為「如果為 true，將停用 Jupyter 自動啟動。您必須改為執行儲存格來啟動 Jupyter。」的核取方塊。



2. 前往 File (檔案) > New File (新建檔案) > Save (儲存)，以您選擇的名稱將此檔案儲存為 `.ipynb` 副檔名或在 Select a language (選取一種語言) 下選取 Jupyter 並儲存檔案。



3. 按兩下該檔案。會顯示出 Jupyter shell，並打開一個筆記本。



4. 當您在 Windows 上首次建立檔案時，依預設不會選取核心。按一下 Select Kernel (選取核心)，即可顯示出可用的核心清單。選擇 Glue PySpark。

   在 MacOS 上，如果沒有看到 Glue PySpark 核心，請嘗試下列步驟：

   1. 執行本機 Jupyter 工作階段以取得 URL。

例如，執行下列命令以啟動 Jupyter 筆記本。

```
jupyter notebook
```

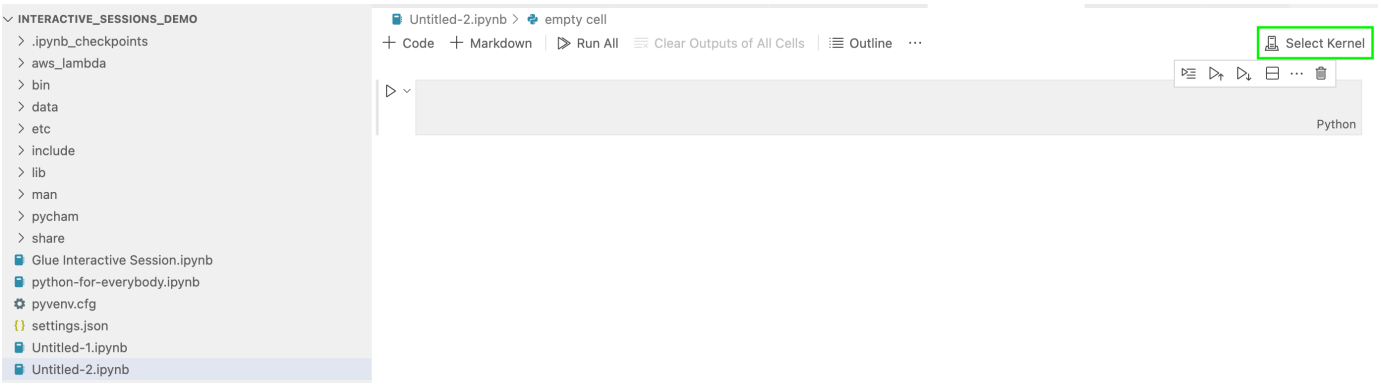
當筆記本第一次執行時，您會看到類似 `http://localhost:8888/?token=3398XXXXXXXXXXXXXXX` 的 URL。

複製 URL。

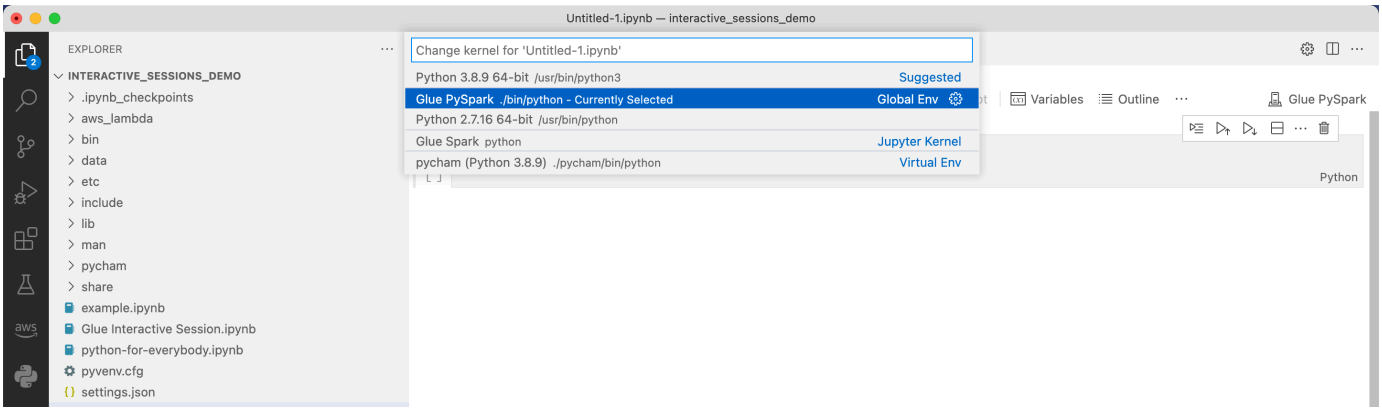
2. 在 VS Code 中，按一下目前的核心，然後選取另一個核心...，接著再選取現有的 Jupyter 伺服器...。貼上您從上述步驟複製的 URL。

   若顯示錯誤訊息，請參閱 VS Code Jupyter Wiki。

3. 如果成功，這會將核心設定為 Glue PySpark。



選擇 Glue PySpark 或 Glue 火花內核（分別為 Python 和斯卡拉）。



如果在下拉列表中沒有看到AWS Glue PySpark和 AWS GlueSpark 內核，請確保您已在上面的步驟中安裝了內AWS Glue核，或者您在 Visual Studio 代碼中

的python.defaultInterpreterPath設置是否正確。有關更多信息，請參閱 python。defaultInterpreterPath 設定說明。

5. 建立 AWS Glue 互動式工作階段。依照您在 Jupyter 筆記本中所做的相同方式繼續建立工作階段。在第一個儲存格頂部指定任何魔術命令，然後執行程式碼語句。

為 Jupyter 和 AWS Glue Studio 筆記本設定 AWS Glue 互動式工作階段

Jupyter 魔術命令簡介

Jupyter 魔術命令是可在儲存格的開頭或作為整個儲存格內文執行的命令。行魔術命令的開頭為 %，而儲存格魔術命令開頭為 %%。行魔術命令，如 %region 和 %connections，可在儲存格搭配多個魔術命令執行，或者像以下範列那樣搭配包含在儲存格內文中的程式碼。

```
%region us-east-2
%connections my_rds_connection
dy_f = glue_context.create_dynamic_frame.from_catalog(database='rds_tables',
 table_name='sales_table')
```

儲存格魔術命令必須使用整個儲存格，並且可以使命令跨越多行。%%sql 的範例位於下方。

```
%%sql
select * from rds_tables.sales_table
```

適用於 Jupyter 的 AWS Glue 互動式工作階段支援的魔術命令

以下是可以與適用於 Jupyter 筆記本的 AWS Glue 互動式工作階段搭配使用的魔術命令。

Sessions magics (工作階段魔術命令)

名稱	Type	描述
%help	N/A	傳回所有魔術命令的描述和輸入類型清單。
%profile	字串	在您的 AWS 組態中指定要用作認證提供者的設定檔。

名稱	Type	描述
%region	字串	指定要在其中初始化工作階段的 AWS 區域;。預設來自 ~/.aws/configure. 範例：%region us-west-1
%idle_timeout	Int	在執行儲存格後工作階段逾時之前的閒置分鐘數。預設的 Spark ETL 工作階段閒置逾時值為預設的逾時值，即 2880 分鐘 (48 小時)。針對其他工作階段類型，請參閱該工作階段類型的相關文件。 範例：%idle_timeout 3000
%session_id	N/A	傳回執行中工作階段的工作階段 ID。
%session_id_prefix	字串	定義一個字串，該字串將放在所有工作階段 ID 之前，格式為 [session_id_prefix]-[session_id]。如果未提供工作階段 ID，則會產生隨機 UUID。當您在 AWS Glue Studio中執行 Jupyter 筆記本時，不支援此魔術命令。 範例：%session_id_prefix 001
%status		傳回當前 AWS Glue 工作階段的狀態，包括其持續時間、組態和執行使用者/角色。
%stop_session		停止目前的工作階段。
%list_sessions		依名稱和 ID 列出所有目前執行中的工作階段。
%session_type	字串	將工作階段類型設為「串流」、「ETL」或「Ray」的其中之一。 範例：%session_type Streaming

名稱	Type	描述
%glue_version	字串	此工作階段要使用的 AWS Glue 版本。 範例：%glue_version 3.0

用於選取任務類型的魔術命令

名稱	Type	描述
%streaming	字串	將工作階段類型變更為 AWS Glue 串流。
%etl	字串	將工作階段類型變更為 AWS Glue ETL。
%glue_ray	字串	將 Ray 的階段作業類型變更 AWS Glue 為。請參閱 AWS Glue Ray 互動課程支持的魔術。

AWS Glue for Spark 組態魔術命令

%%configure 魔術命令是由工作階段的所有組態參數組成的 JSON 格式字典。可以在此處或透過個別魔術命令來指定每個參數。

名稱	Type	描述
%%configure	字典	指定由工作階段的所有組態參數組成的 JSON 格式字典。可以在此處或透過個別魔術命令來指定每個參數。 如需有關如何使用 %%configure 的參數與範例清單，請參閱下表：使用 %%configure。
%iam_role	字串	指定用來執行工作階段的 IAM 角色 ARN。預設來自 ~/.aws/configure。

名稱	Type	描述
		範例：`%iam_role AWSGlueServiceRole`
`%number_of_workers`	Int	當任務執行時所分配的已定義 worker_type 的工作者數目。還必須設定 `worker_type` 。預設的 `number_of_workers` 為 5。 範例：`%number_of_workers 2`
`%additional_python_modules`	清單	要包含在叢集中的以逗號分隔的其他 Python 模組清單 (可以來自 PyPI 或 S3)。 範例：`%additional_python_modules pandas, numpy` 。

名稱	Type	描述

%%tags	字串	將標籤新增至工作階段。指定大括號 {} 內的標籤。每個標籤名稱配對均以括號 (" ") 括住，並以逗號 (,) 區隔。
		
		`%%tags`
		`{"billing":"Data-Platform",`
		`"team":"analytics"}`
		
		使用 %status 魔術命令來檢視與工作階段相關聯的標籤。
		
		`%status`
		
		`Session ID: <sessionId>`
		`Status: READY`
		`Role: <example-role>`
		`CreatedOn: 2023-05-26 11:12:17.056000-07:00`
		`GlueVersion: 3.0`
		`Job Type: glueetl`
		`Tags: {'owner':'example-owner',`
		`'team':'analytics', 'billing':'Data-Platform'}`
		`Worker Type: G.4X`
		`Number of Workers: 5`
		`Region: us-west-2`
		`Applying the following default arguments:`
		`--glue_kernel_version 0.38.0`
		`--enable-glue-datacatalog true`
		`Arguments Passed: ['--glue_kernel_version: 0.38.0', '--enable-glue-datacatalog: true']`

名稱	Type	描述

%%assume_role	字典	指定 JSON 格式的字典或 IAM 角色 ARN 字串，以建立可取得跨帳戶存取權的工作階段。
		
		使用 ARN 的範例：
		
		`%%assume_role`
		`{`
		`  'arn:aws:iam::XXXXXXXXXXXX:`
		`role/AWSGlueServiceRole'`
		`}`
		
		使用憑證的範例：
		
		`%%assume_role`
		`{{`
		`  "aws_access_key_id" =`
		`"XXXXXXXXXXXX",`
		`  "aws_secret_access_key" =`
		`"XXXXXXXXXXXX",`
		`  "aws_session_token" =`
		`"XXXXXXXXXXXX"`
		`}}`

%%configure 儲存格魔術命令引數

%%configure 魔術命令是由工作階段的所有組態參數組成的 JSON 格式字典。可以在此處或透過個別魔術命令來指定每個參數。如需 %%configure 儲存格魔術命令支援的引數範例，請參閱以下內容。為工作指定的執行引數使用--前置詞。範例：

```
%%configure
{
   "--user-jars-first": "true",
   "--enable-glue-datacatalog": "false"
}
```

如需工作參數的詳細資訊，請參閱任務參數。

階段作業組

參數	類型	描述

`max_retries`	Int	如果此任務失敗，可重試的次數上限。
		`%%configure`
		`{`
		`  "max_retries": "0"`
		`}`

`max_concurrent_runs`	Int	一項任務可同時執行的數量上限。
		範例：
		`%%configure`
		`{`
		`  "max_concurrent_runs": "3"`
		`}`

會話參數

參數	類型	描述

`--enable-spark-ui`	布林值	啟用 Spark UI 以監控和偵錯 AWS Glue ETL 任務。
		`%%configure`
		`{`
		`  "--enable-spark-ui": "true"`
		`}`

`--spark-event-logs-path`	字串	指定 Amazon S3 路徑。使用 Spark UI 監控功能時。 範例：

參數	類型	描述

		%%configure
		{
		"--spark-event-logs-path":
		"s3://path/to/event/logs/"
		}

--script_location	字串	指定執行任務的指令碼 S3 路徑。
		
		範例：
		
		%%configure
		{
		"script_location": "s3://new-folder-here"
		}

--SECURITY_CONFIGURATION	字串	AWS Glue 安全性組態的名稱
		
		範例：
		
		%%configure
		{
		"--SECURITY_CONFIGURATION": *security-configuration-name* ,
		}

參數	類型	描述

--job-language	字串	指令碼程式設計語言。接受 'scala' 或 'python' 的值。預設為 'python'。
		
		範例：
		
		`%%configure`
		`{`
		`  "--job-language": "scala"`
		`}`

--class	字串	可作為您的 Scala 指令碼進入點的 Scala 類別。預設為 Null。
		
		範例：
		
		`%%configure`
		`{`
		`  "--class": "className"`
		`}`

--user-jars-first	布林值	在 classpath 中優先採用客戶的額外 JAR 檔案。預設為 Null。
		
		範例：
		
		`%%configure`
		`{`
		`  "--user-jars-first": "true"`
		`}`

參數	類型	描述

--use-postgres-driver	布林值	在類別路徑中設定 Postgres JDBC 驅動程式的優先順序，以避免與 JDBC 驅動程式發生衝突。 Amazon Redshift 預設為 Null。
		
		範例：
		
		`%%configure`
		`{`
		`  "--use-postgres-driver": "true"`
		`}`

--extra-files	List(string)	組態檔案等其他檔案的 Amazon S3 路徑；在 AWS Glue 執行您的指令碼前，即會將其複製至指令碼的工作目錄。
		
		範例：
		
		`%%configure`
		`{`
		`  "--extra-files": "s3://path/to/additional/files/"`
		`}`

參數	類型	描述

--job-bookmark-option	字串	控制任務書籤的行為。接受 "、job-bookmark-enable '或job-bookmark-disable' '的值。job-bookmark-pause預設值為 'job-bookmark-disable'。
		
		範例：
		
		`%%configure`
		`{`
		`  "--job-bookmark-option": "job-bookmark-enable"`
		`}`

--TempDir	字串	指定儲存貯體的 Amazon S3 路徑做為任務的暫時目錄。預設為 Null。
		
		範例：
		
		`%%configure`
		`{`
		`  "--TempDir": "s3://path/to/temp/dir"`
		`}`

參數	類型	描述

`--enable-s3-parquet-optimized-committer`	布林值	啟用 EMRFS Amazon S3 最佳化遞交者，以將 Parquet 資料寫入 Amazon S3。預設為 'true'。
		
		範例：
		
		`%%configure`
		`{`
		`  "--enable-s3-parquet-optimized-committer": "false"`
		`}`

`--enable-rename-algorithm-v2`	布林值	將 EMRFS 重新命名演算法版本設定為第 2 版。預設為 'true'。
		
		範例：
		
		`%%configure`
		`{`
		`  "--enable-rename-algorithm-v2": "true"`
		`}`

`--enable-glue-datacatalog`	布林值	可讓您使用 AWS Glue Data Catalog 作為 Apache Spark Hive 中繼存放區。
		
		範例：
		
		`%%configure`
		`{`
		`  --"enable-glue-datacatalog": "true"`
		`}`

參數	類型	描述

--enable-metrics	布林值	針對任務執行啟用任務分析的指標集合。預設為 'false'。
		
		範例：
		
		`%%configure`
		`{`
		`  "--enable-metrics": "true"`
		`}`

--enable-continuous-cloudwatch-log	布林值	啟用即時、持續的 AWS Glue 任務記錄。預設為 'false'。
		
		範例：
		
		`%%configure`
		`{`
		`  "--enable-continuous-cloudwatch-log": "true"`
		`}`

--enable-continuous-log-filter	布林值	當您建立或編輯已啟用持續記錄的任務時，指定標準篩選條件或無篩選條件。預設為 'true'。
		
		範例：
		
		`%%configure`
		`{`
		`  "--enable-continuous-log-filter": "true"`
		`}`

參數	類型	描述

--continuous-log-stream-prefix	字串	為啟用連續記錄的工作指定自訂 Amazon CloudWatch 記錄資料流前置詞。預設為 Null。
		範例：
		`%%configure`
		`{`
		`  "--continuous-log-stream-prefix": "prefix"`
		`}`

--continuous-log-conversionPattern	字串	為啟用連續記錄的任務指定自訂轉換日誌模式。預設為 Null。
		範例：
		`%%configure`
		`{`
		`  "--continuous-log-conversionPattern": "pattern"`
		`}`

參數	類型	描述

--conf	字串	控制 Spark 組態參數。適用於進階使用案例。--conf在每個參數之前使用。範例：
		
		%%configure
		{
		"--conf": "spark.hadoop.hive
		.metastore.glue.catalogid=1
		23456789012 --conf hive.meta
		store.client.factory.class=
		com.amazonaws.glue.catalog.
		metastore.AWSGlueDataCatalo
		gHiveClientFactory --conf
		hive.metastore.schema.verif
		ication=false"
		}

Spark 任務 (ETL 與串流) 魔術命令

名稱	Type	描述
%worker_type	字串	標準、G.1X 或 G.2X。還必須設定 number_of_workers 。預設 worker_type 是 G.1X。
%connections	List	指定以逗號分隔的清單，其中列出要在工作階段中使用的連線。 範例： %connections my_rds_connection dy_f = glue_context.create_dynamic _frame.from_catalog(databas e='rds_tables', table_nam e='sales_table')

名稱	Type	描述
%extra_py_files	清單	列出來自 Amazon S3 的以逗號分隔的其他 Python 檔案清單。
%extra_jars	清單	列出要包含在叢集中的以逗號分隔的其他 jar 清單。
%spark_conf	字串	為您的工作階段指定自訂 Spark 組態。例如 %spark_conf spark.serializer=org.apache.spark.serializer.KryoSerializer 。

適用於 Ray 任務的魔術命令

名稱	Type	描述
%min_workers	Int	分配給 Ray 任務的工作者數量下限。預設：1。 範例：%min_workers 2
%object_memory_head	Int	暖啟動後執行個體前端節點上可用記憶體的百分比。下限：0。上限：100。 範例：%object_memory_head 100
%object_memory_worker	Int	暖啟動後執行個體工作節點上可用記憶體的百分比。下限：0。上限：100。 範例：%object_memory_worker 100

Action magics (動作魔術命令)

名稱	Type	描述
%%sql	字串	執行 SQL 程式碼。初始 %%sql 魔術命令之後的所有行都將作為 SQL 程式碼的一部分傳遞。 範例：%%sql select * from rds_tables.sales_table

%matplot	Matplotlib 圖	使用 matplotlib 程式庫視覺化資料。
		
		範例：
		
		`import matplotlib.pyplot as plt`
		
		`# Set X-axis and Y-axis values`
		`x = [5, 2, 8, 4, 9]`
		`y = [10, 4, 8, 5, 2]`
		
		`# Create a bar chart`
		`plt.bar(x, y)`
		
		`# Show the plot`
		`%matplot plt`

%plotly	Plotly 圖	使用 plotly 程式庫視覺化資料。
		
		範例：
		
		`import plotly.express as px`
		
		`#Create a graphical figure`
		`fig = px.line(x=["a","b","c"],`
		` y=[1,3,2], title="sample`
		` figure")`
		
		`#Show the figure`
		`%plotly fig`

命名工作階段

AWS Glue互動式工作階段是 AWS 資源，需要名稱。每個工作階段的名稱應是唯一的，且可能受 IAM 管理員的限制。如需詳細資訊，請參閱 具備 IAM 的互動式工作階段。Jupyter 核心會為您自動產生唯一的工作階段名稱。但是，可以透過兩種方式手動命名工作階段：

1. 使用位於的 AWS Command Line Interface 配置文件`~.aws/config`。請參 AWS Config 使用 AWS Command Line Interface.
2. 使用 `%session_id_prefix` 魔術命令。請參閱 適用於 Jupyter 的 AWS Glue 互動式工作階段支援的魔術命令。

產生的工作階段名稱如下：

- 當提供字首和 session_id 時：工作階段名稱將是 {prefix}-{UUID}。
- 如果未提供任何資料：工作階段名稱將是 {UUID}。

前置工作階段名稱可讓您在 AWS CLI 或主控台中列出工作階段時辨識工作階段。

指定互動式工作階段的 IAM 角色

您必須指定 I AWS dentity and Access Management (IAM) 角色，才能與您搭配互動式工作階段執行的 AWS Glue ETL 程式碼搭配使用。

角色需要擁有執行 AWS Glue 任務所需的相同 IAM 許可。如需為 AWS Glue 任務和互動式工作階段建立角色的詳細資訊，請參閱《為 AWS Glue 建立 IAM 角色》。

IAM 角色可以使用兩種方式指定：

- 使用位於`~.aws/config`（推薦）的 AWS Command Line Interface 配置文件。如需詳細資訊，請參閱使用 ~/.aws/config 設定工作階段。

  > **Note**
  >
  > 使用 `%profile` 魔術命令時，該設定檔的 `glue_iam_role` 組態會被接受。

- 使用 %iam_role 魔術命令。如需詳細資訊，請參閱 適用於 Jupyter 的 AWS Glue 互動式工作階段支援的魔術命令。

使用命名設定檔設定工作階段

AWS Glue互動式工作階段使用與 AWS Command Line Interface 或 boto3 相同的認證，互動式工作階段會遵循並使用具名的設定檔，例如在 ~/.aws/config (Linux 和 MacOS) 或 %USERPROFILE%\.aws\config (Windows) 中 AWS CLI 找到的設定檔。如需詳細資訊，請參閱使用命名設定檔。

互動式工作階段透過允許在設定檔中指定 AWS Glue 服務角色和工作階段 ID 字首，來利用命名設定檔。若要設定設定檔角色，請為 iam_role 金鑰和/或 session_id_prefix 在命名設定檔中新增一行，如下所示。session_id_prefix 不需要引號。例如，若您想要新增 session_id_prefix，請輸入 session_id_prefix=myprefix 的值。

```
[default]
region=us-east-1
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
glue_iam_role=arn:aws:iam::<AccountID>:role/<GlueServiceRole>
session_id_prefix=<prefix_for_session_names>

[user1]
region=eu-west-1
aws_access_key_id=AKIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
glue_iam_role=arn:aws:iam::<AccountID>:role/<GlueServiceRoleUser1>
session_id_prefix=<prefix_for_session_names_for_user1>
```

如果您有產生憑證的自訂方法，您也可以在 ~/.aws/config 檔案中將設定檔設定為使用 credential_process 參數。例如：

```
[profile developer]
region=us-east-1
credential_process = "/Users/Dave/generate_my_credentials.sh" --username helen
```

您可以透過此處的 credential_process 參數找到有關取得憑證的詳細資訊：透過外部程序取得憑證。

如果區域或 iam_role 未在您使用的設定檔中設定，則您必須在執行的第一個儲存格中使用 %region 和 %iam_role 魔術命令來指定它們。

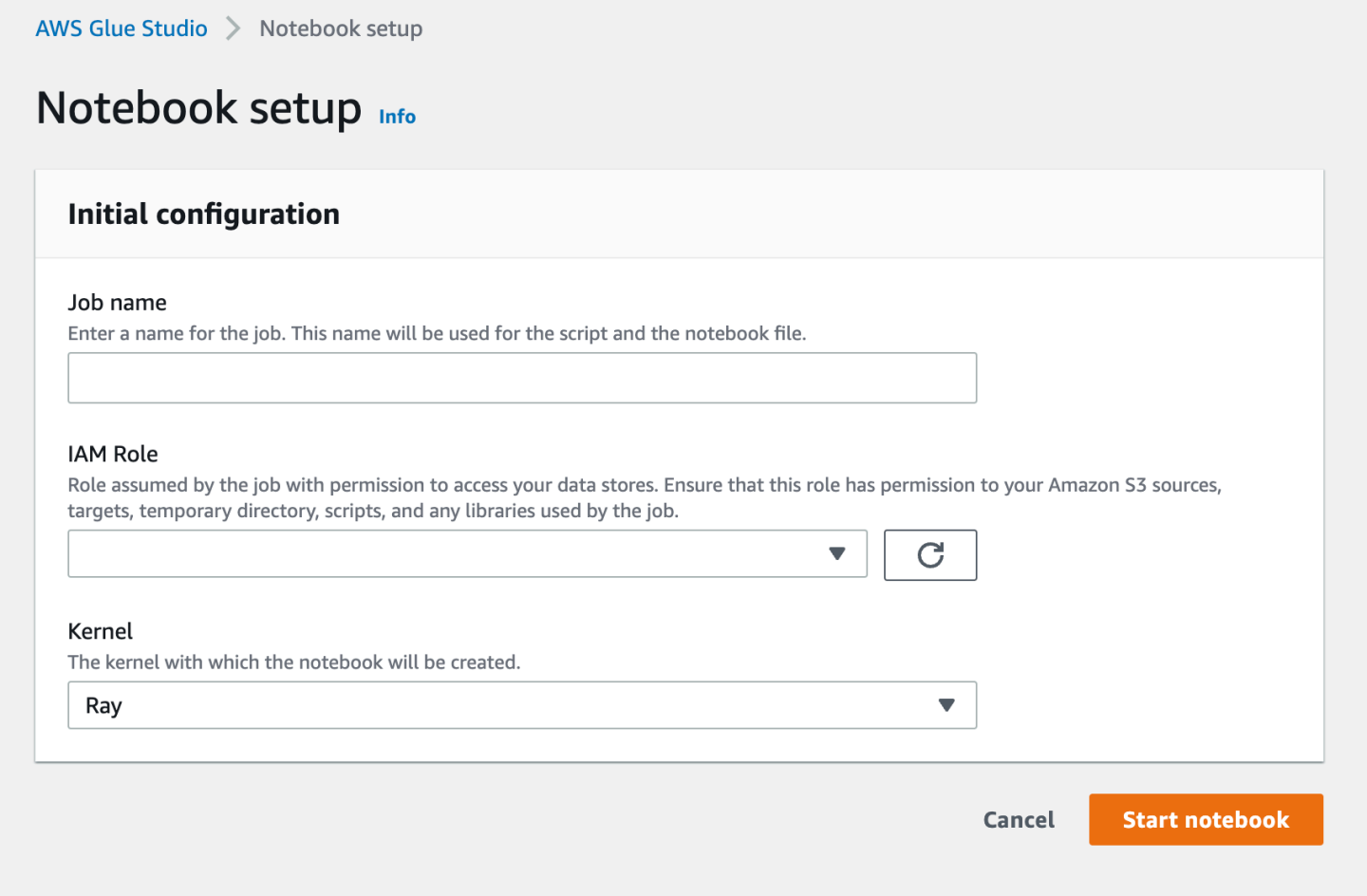
Ray 互動式工作階段入門 (預覽) AWS Glue

> **Warning**
>
> Ray 互動式工作階段的 AWS Glue 預覽結束了 2024 年 4 月 30 日。您將無法再為 Ray 建立新的互動式工 AWS Glue 作階段。

> **Note**
>
> AWS Glue in Ray 在美國東部 (維吉尼亞北部)、美國東部 (俄亥俄)、美國西部 (奧勒岡)、亞太區域 (東京) 和歐洲 (愛爾蘭) 提供。

AWS Glue Studio 主控台中的 Ray 互動工作階段

在「 AWS Glue Studio 主控台」的「工作」頁面中，選取現有的 Jupyter 記事本選項。這會開啟一個 Notebook setup (筆記本設定) 頁面，您可在其中選取自己的 Kernel (核心)。選取 Ray 核心以啟動 Ray 互動式工作階段。如需有關互動式工作階段及其使用方式的詳細資訊，請參閱 the section called “AWS Glue 互動式工作階段入門”。



使用 Jupyter 核心的 Ray 互動式工作階段

要在 AWS Glue Studio 控制台之外使用 Ray 內核，您需要安裝我們在 PyPI 上發布的`aws-glue-sessions`軟件包。如需有關使用核心套件的詳細資訊，請參閱 the section called “AWS Glue 互動式工作階段入門” 文件。

如要更新或安裝核心，請執行 `pip install --upgrade aws-glue-sessions`。您會需要 .37+ 版本才能使用 Ray 核心。

Ray 互動式工作階段可以存取與 Ray 任務相同的 Ray 程式庫和版本。在預覽版中，所有 Ray 互動式工作階段都將使用 Ray 2.4.0。

Ray 互動式工作階段逾時預設值

- (工作階段的) 預設逾時：8 小時。
- 空閒逾時預設值：1 小時。

AWS Glue Ray 互動式工作階段支援的魔術命令

適用於 AWS Glue Jupyter 核心的魔術命令，當其為 Ray 互動式工作階段提供支援時，與 Spark 工作階段的魔術命令類似。如需參考資訊，請參閱the section called “ 為 Jupyter 和 AWS Glue Studio 筆記本設定 AWS Glue 互動式工作階段 ”。

Sessions magics (工作階段魔術命令)

工作階段魔術命令大致上與 AWS Glue for Ray 預覽版之前的相同。如需有關此預覽版之外的工作階段魔法的詳細資訊，請參閱 the section called “適用於 Jupyter 的 AWS Glue 互動式工作階段支援的魔術命令”。我們引入了一種新的魔術命令，可將工作階段類型設定為 AWS Glue for Ray。

Name	類型	描述
`%glue_ray`	字串	將工作階段類型變更為 AWS Glue for Ray。

AWS GlueConfig 魔法

在互動式工作階段中設定 AWS Glue 的魔術命令，可能會在各個工作階段之間有所差異。目前，我們僅針對使用 AWS Glue for Ray 支援此現有魔術命令子集。

> **Warning**
>
> 已知問題：**`additional_python_modules`**
> 在 Ray 互動式工作階段預覽版中，使用 `additional_python_modules` 魔術命令會在儲存筆記本時造成問題。若要為 Ray 工作階段設定 Python 模組，請使用 `%%configure` 魔術命令來設定 the section called “Ray 任務參數” 中定義的 `pip-install` 參數。

Name	類型	描述
`%%configure`	字典	指定由工作階段的所有組態參數組成的 JSON 格式字典。可以在此處或透過個別魔術命令來指定每個參數。
`%iam_role`	字串	指定用來執行工作階段的 IAM 角色 ARN。預設來自 ~/.aws/configure

Name	類型	描述
%number_of_workers	int	當任務執行時所分配的已定義 worker_type 的工作者數目。還必須設定 worker_type 。
%worker_type	字串	在 AWS Glue for Ray 預覽版中，唯一支援的工作者類型為 Z.2X。
%additional_python_modules	列出	要包含在叢集中的以逗號分隔的其他 Python 模組清單 (可以來自 Pypi 或 S3)。

Action magics (動作魔術命令)

AWS Glue Ray 工作階段不支援任何動作魔術命令。

具備 IAM 的互動式工作階段

下列章節說明適用於 AWS Glue 互動式工作階段的安全考量。

主題

- 與互動式工作階段搭配使用的 IAM 主體
- 設定用戶端主體
- 設定執行時間角色
- 使您的會話私有 TagOnCreate
- IAM 政策考量

與互動式工作階段搭配使用的 IAM 主體

您會使用兩個與 AWS Glue 互動式工作階段搭配使用的 IAM 主體。

- Client principal (用戶端主體)：用戶端主體 (使用者或角色) 會授權 AWS Glue 用戶端互動式工作階段的 API 操作，用戶端使用主體的身分識別型憑證進行設定。例如，這可能是您通常用來存取 AWS Glue 主控台的 IAM 角色。這也可以是授與 IAM 中的使用者的角色，其認證用於 AWS Command Line Interface，或互動式工作階段 Jupyter 核心使用的用AWS Glue戶端。

- 執行時間角色：執行時間角色是用戶端主體傳遞給互動式工作階段 API 操作的 IAM 角色。AWS Glue 會使用此角色在工作階段中執行陳述式。例如，此角色可能是用來執行 AWS Glue ETL 任務的角色。

  如需詳細資訊，請參閱 設定執行時間角色。

設定用戶端主體

您必須將身分政策連接至用戶端主體，以允許其呼叫互動式工作階段 API。此角色必須擁有對您傳遞至如 `CreateSession` 等互動式工作階段 API 之執行角色的 `iam:PassRole` 存取權。例如，您可以將AWSGlueConsoleFullAccess受管政策附加到 IAM 角色，讓帳戶中附加政策的使用者存取您帳戶中建立的所有工作階段 (例如執行階段陳述式或取消陳述式)。

如果您想保護您的會話，並使其僅對某些 IAM 角色（例如與創建會話的用戶相關聯的角色）私有，則可以使用調用 AWS Glue Interactive Session 的基於標籤的授權控制 TagOnCreate。如需詳細資訊，請參閱 使您的會話私有 TagOnCreate 以擁有者標籤為基礎的受管理原則如何讓您的工作階段私有。TagOnCreate如需有關以身分識別為基礎的原則的詳細資訊，請參閱. AWS Glue

設定執行時間角色

您必須將 IAM 角色傳遞給 CreateSession API 作業，才能AWS Glue在互動式工作階段中假設和執行陳述式。角色應該擁有執行一般 AWS Glue 任務所需的相同 IAM 許可。例如，您可以使用允許代表您呼叫服務的AWSGlueServiceRole原則AWS Glue來建立 AWS 服務角色。如果您使用 AWS Glue 主控台，它會代為自動建立服務角色，或使用現有的服務角色。您也可以建立自己的 IAM 角色，並連接自己的 IAM 政策，以允許類似的許可。

如果你想保護你的會話，並使其私有只給誰創建會話的用戶，那麼你可以使用AWS Glue交互式會話的基於標籤的授權控制調用 TagOnCreate。如需詳細資訊，請參閱 使您的會話私有 TagOnCreate 以擁有者標籤為基礎的受管理原則如何讓您的工作階段私有。 TagOnCreate如需有關以身分為基礎的政策詳細資訊，請參閱 Glue 的身分識別原則 AWS。如果您要從 IAM 控制台自己創建執行角色，並且希望使用 TagOnCreate 功能將服務設為私有，請按照以下步驟操作。

1. 建立 IAM 角色並將角色類型設定為 `Glue`。
2. 附加此AWS Glue受管理策略：AwsGlueSessionUserRestrictedServiceRole
3. 在角色名稱前面加上策略名稱AwsGlueSessionUserRestrictedServiceRole。例如，您可以建立名稱為 AwsGlueSessionUserRestrictedServiceRole-myrole 的角色，並附加AWS Glue受管理的策略。AwsGlueSessionUserRestrictedServiceRole
4. 連接類似下列項目的信任政策，以允許 AWS Glue 擔任此角色：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "glue.amazonaws.com"
        ]
      },
      "Action": [
        "sts:AssumeRole"
      ]
    }
  ]
}
```

對於交互式會話 Jupyter 內核，您可以在個人資料中指定`iam_role`密鑰。 AWS Command Line Interface 如需詳細資訊，請參閱使用 ~/.aws/config 設定工作階段。如果您是使用 AWS Glue 筆記本與互動式工作階段進行互動，則您可以在您執行之第一個儲存格中的 `%iam_role` 魔術命令中傳遞執行角色。

使您的會話私有 TagOnCreate

AWS Glue 互動式工作階段支援對互動式工作階段進行標記和以標籤為基礎的授權 (TBAC) 作為命名資源。除了使用 TagResource 和 UntagResource API 的 TBAC 之外，AWS Glue互動式工作階段還支援僅在使用作業建立工作階段期間使用指定標記「標記」工作階段的 TagOnCreate CreateSession 功能。這也意味著這些標籤將被刪除 DeleteSession，又名 UntagOnDelete。

TagOnCreate 提供了強大的安全機制，使您的會話對會話的創建者私有。例如，您可以將具有「owner」 RequestTag 且值為 $ {AWs: userId} 的 IAM 政策附加到用戶端主體 (例如使用者)，以便只有在要求中提供具有相符來電者 userID 值的「擁有者」標記作為 userId 標籤時，才允許建立工作階段。 CreateSession 此政策允許 AWS Glue 互動式工作階段建立工作階段資源，並僅在工作階段建立期間使用 userId 標籤標記工作階段。除此之外，您還可以將 IAM 政策與「owner」 ResourceTag 附加到您在期間傳入的執行角色附加到會話的創建者（也就是值為 $ {AWS：userId} 的所有者標籤）範圍僅限於會話的訪問權限（例如運行語句）。 CreateSession

為了讓您更輕鬆地使用 TagOnCreate 功能，讓工作階段建立者的私有工作階段，請AWS Glue提供專門的受管理原則和服務角色。

如果您想要使用 IAM AssumeRole 主體建立AWS Glue互動式工作階段 (亦即，使用假設 IAM 角色提供的登入資料)，並且想要將工作階段設定為建立者為私有，請使用AWSGlueSessionUserRestrictedNotebookPolicy與和AWSGlueSessionUserRestrictedNotebookServiceRole分別類似的政策。這些政策AWS Glue允許使用 $ {aws:PrincipalTag} 來擷取擁有者標籤值。這需要您傳遞值為 $ {AWS: userId} 的使用者 ID 標籤，如 SessionTag 假設角色認證所示。請參閱 ID 工作階段標籤。如果您使用 Amazon EC2 執行個體與執行個體設定檔自動售貨登入資料，並且想要從 Amazon EC2 執行個體內建立工作階段或與工作階段互動，則您需要像假設角色憑證一樣傳遞值為 SessionTag $ {AWS: UserID} 的 UserID 標籤。

例如，如果您要使用 IAM AssumeRole 主要登入資料建立工作階段，並且想要使用 TagOnCreate 功能將服務設為私有，請依照下列步驟操作。

1. 從 IAM 主控台自行建立執行時間角色。請附加此AWS Glue受管理的策略，AwsGlueSessionUserRestrictedNotebookServiceRole並在角色名稱前面加上策略名稱AwsGlueSessionUserRestrictedNotebookServiceRole。例如，您可以建立名稱為 AwsGlueSessionUserRestrictedNotebookServiceRole-myrole 的角色，並附加AWS Glue受管理的策略。AwsGlueSessionUserRestrictedNotebookServiceRole
2. 連接類似下方的信任政策，以允許 AWS Glue 來擔任上述角色。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "glue.amazonaws.com"
        ]
      },
      "Action": [
        "sts:AssumeRole"
       ]
    }
   ]
}
```

3. 建立另一個以前綴命名的角色，AwsGlueSessionUserRestrictedNotebookPolicy並附加受AWS Glue管理的策略AwsGlueSessionUserRestrictedNotebookPolicy以使工作階段為私有。除了受管政策之外，請附加以下內嵌政策以允許 iam: PassRole 到您在步驟 1 中創建的角色。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/
AwsGlueSessionUserRestrictedNotebookServiceRole*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "glue.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

4. 將類似下列的信任政策連接到上述 IAM AWS Glue，以擔任此角色。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": [
                "glue.amazonaws.com"
            ]
        },
        "Action": [
            "sts:AssumeRole",
            "sts:TagSession"
        ]
    }]
}
```

> **Note**
>
> 或者，您可以使用單一角色 (例如，筆記本角色)，並附加上述受管理的原則AwsGlueSessionUserRestrictedNotebookServiceRole和AwsGlueSessionUserRestrictedNotebo
> 也請連接額外的內嵌政策，以允許您角色的 `iam:passrole` 連接至 AWS Glue。最後連接上述信任政策，以允許 `sts:AssumeRole` 和 `sts:TagSession`。

AWSGlueSessionUserRestrictedNotebookPolicy

只 AWSGlueSessionUserRestrictedNotebookPolicy 有當標籤鍵「owner」和值與主參與者 (使用者或角色) 的 AWS 使用者識別碼相符時，才能從記事本建立AWS Glue互動式工作階段。如需詳細資訊，請參閱您可以在何處使用政策變數。此政策會連接至 AWS Glue Studio 中建立 AWS Glue 互動式工作階段筆記本的主體 (使用者或角色)。此原則也允許對AWS Glue Studio筆記本有足夠的存取權，以便與 AWS 使用者識別碼相符的「owner」標籤值所建立的AWS Glue Studio互動式工作階段資源互動。工作階段建立之後，此政策會拒絕對變更或移除 AWS Glue 工作階段資源中 "owner" 標籤的許可。

AWSGlueSessionUserRestrictedNotebookServiceRole

AWSGlueSessionUserRestrictedNotebookServiceRole提供對AWS Glue Studio筆記本的足夠存取權，以便與「擁有者」標籤值與筆記本建立者 (AWS 使用者或角色) 的使用者識別碼相符的「擁有者」標籤值所建立的AWS Glue互動式工作階段資源互動。如需詳細資訊，請參閱您可以在何處使用政策變數。此服務角色政策附加到作為魔術傳遞給筆記本或作為執行角色傳遞給 CreateSession API 的角色。此原則也允許從記事本建立AWS Glue互動式工作階段，只有在標籤鍵「擁有者」和值符合主體的 AWS 使用者識別碼時才能建立互動式工作階段。工作階段建立之後，此政策會拒絕對變更或移除 AWS Glue 工作階段資源中 "owner" 標籤的許可。此政策還包括從 Amazon S3 儲存貯體寫入和讀取、撰寫 CloudWatch 日誌、建立和刪除所使用之 Amazon EC2 資源標籤的許可AWS Glue。

透過使用者政策將工作階段設定為私有

您可以附加AWSGlueSessionUserRestrictedPolicy至帳戶中每個使用者附加的 IAM 角色，以限制他們只建立擁有者標籤的工作階段，其值與自己的 $ {AWS: userID} 相符。而不是使用AWSGlueSessionUserRestrictedNotebookPolicy和，AWSGlueSessionUserRestrictedNotebookServiceRo需要使用AWSGlueSessionUserRestrictedPolicy與AWSGlueSessionUserRestrictedServiceRole分別類似的原則。如需詳細資訊，請參閱使用以身分為基礎的政策。此政策將對工作階段的存取權縮小到僅限建立者，亦即使用具有專屬 ${aws:userId} 之擁有者標籤建立工作階段的使用者 ${aws:userId}。如果您已按照中的步驟使用 IAM 主控台自行建立執行角色設定執行時間角色，則除了

附加AwsGlueSessionUserRestrictedPolicy受管政策之外，還會將下列內嵌政策附加到帳戶中的每個使用者，以允許iam:PassRole您先前建立的執行角色。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "iam:PassRole"
        ],
        "Resource": [
            "arn:aws:iam::*:role/AwsGlueSessionUserRestrictedServiceRole*"
        ],
        "Condition": {
            "StringLike": {
                "iam:PassedToService": [
                    "glue.amazonaws.com"
                ]
            }
        }
    }]
}
```

AWSGlueSessionUserRestrictedPolicy

只有在AWSGlueSessionUserRestrictedPolicy提供與其使用 AWS 者 ID 相符的標籤金鑰「owner」和值時，才提供使用 CreateSession API 建立AWS Glue互動式工作階段的存取權。此身分識別原則會附加至叫用 CreateSession API 的使用者。此原則也允許與使 AWS 用者識別碼相符的「擁有者」標籤和值所建立的AWS Glue互動式工作階段資源互動。工作階段建立之後，此政策會拒絕對變更或移除 AWS Glue 工作階段資源中 "owner" 標籤的許可。

AWSGlueSessionUserRestrictedServiceRole

AWSGlueSessionUserRestrictedServiceRole提供對工作階段以外的所有AWS Glue資源的完整存取權，並允許使用者僅建立和使用與使用者關聯的互動式工作階段。此原則也包含管理其他 AWS 服務中 Glue 資源所需的其他權限。AWS Glue該策略還允許向其他 AWS 服務中的AWS Glue資源添加標籤。

IAM 政策考量

互動式工作階段是 AWS Glue 中的 IAM 資源。由於其是 IAM 資源，因此對工作階段的存取和互動受 IAM 政策管控。根據連接至用戶端主體或管理員設定之執行角色的 IAM 政策，用戶端主體 (使用者或角色) 可以建立新的工作階段，並與其本身的工作階段和其他工作階段進行互動。

如果管理員已附加 IAM 政策 (例如 AWSGlueConsoleFullAccess 或允許存取 AWSGlueServiceRole 該帳戶中的所有AWS Glue資源)，則客戶主體將能夠彼此共同作業。例如，如果政策允許，一位使用者可以與其他使用者建立的工作階段進行互動。

如果您想根據您的特定需求設定政策，請參閱有關設定政策資源的 IAM 說明文件。例如，為了隔離屬於使用者的工作階段，您可以使用AWS Glue互動式工作階段支援的 TagOnCreate 功能。請參閱 使您的會話私有 TagOnCreate 。

互動式工作階段支援根據特定 VPC 條件限制建立工作階段。請參閱控制使用條件索引鍵來控制設定的政策。

將指令碼或筆記本轉換為 AWS Glue 任務

有兩種方式可將指令碼或筆記本轉換為 AWS Glue 任務：

- 使用 nbconvert 將您的 Jupyter `.ipynb` 筆記本文件檔案轉換為 `.py` 檔案。如需詳細資訊，請參閱 nbconvert：將筆記本轉換為其他格式。
- 將檔案上傳到 AWS Glue Studio 筆記本。
  - 在 AWS Glue Studio 主控台中，從導覽選單中選擇 Jobs (任務)。
  - 在 Create job (建立任務) 區段中，選擇 Jupyter Notebook (Jupyter 筆記本)。
  - 在 Options (選項) 區段中，選擇 Upload and edit an existing notebook (上傳並編輯現有的筆記本)。
  - 選取 Choose file (選擇檔案) 以上傳 `.ipynb` 檔案。

適用於串流的 AWS Glue 互動式工作階段

切換串流工作階段類型

使用 AWS Glue 互動式工作階段組態魔術命令 `%streaming`，定義您正在執行的任務並初始化串流互動式工作階段。

為互動式開發抽樣輸入串流

我們所衍生的其中一個工具可協助增強互動AWS Glue式工作階段中的互動式體驗，就是在下`GlueContext`方加入新的方法，以取得靜態資料流的快照 DynamicFrame。 `GlueContext`允許您檢查，互動和實施您的工作流程。

使用 GlueContext 類執行個體，您將能夠找到方法 getSampleStreamingDynamicFrame。此方法所需的引數為：

- dataFrame: 火花流 DataFrame
- options：請參閱下列可用的選項

可用選項包括：

- windowSize：這也稱為微量批持續時間。此參數將判定觸發前一個批次後串流查詢將等待的時長。此參數值必須小於 pollingTimeInMs。
- pollingTimeIn女士：該方法運行的總時間長度。它將至少觸發一個微批次，以從輸入串流中獲取樣本記錄。
- recordPollingLimit：此參數可協助您限制要從串流輪詢的記錄總數。
- (選用) 您也可以使用 writeStreamFunction 將此自訂函數應用至每個記錄抽樣函數。請參閱下列 Scala 和 Python 中的範例。

Scala

```
val sampleBatchFunction = (batchDF: DataFrame, batchId: Long) => {//Optional but
 you can replace your own forEachBatch function here}
val jsonString: String = s"""{"pollingTimeInMs": "10000", "windowSize": "5
 seconds"}"""
val dynFrame = glueContext.getSampleStreamingDynamicFrame(YOUR_STREAMING_DF,
 JsonOptions(jsonString), sampleBatchFunction)
dynFrame.show()
```

Python

```
def sample_batch_function(batch_df, batch_id):
       //Optional but you can replace your own forEachBatch function here
options = {
            "pollingTimeInMs": "10000",
            "windowSize": "5 seconds",
        }
glue_context.getSampleStreamingDynamicFrame(YOUR_STREAMING_DF, options,
 sample_batch_function)
```

> **Note**
>
> 當抽樣 DynFrame 為空白時，它可能是由以下幾個原因造成的：
>
> - 串流來源設定為「最新」，而且在抽樣週期期間沒有擷取任何新資料。
> - 輪詢時間不足，無法處理它擷取的記錄。除非整個批次處理完畢，否則資料不會顯示。

在互動式工作階段中執行串流應用程式

在 AWS Glue 互動式工作階段中，您可以執行 AWS Glue 串流應用程式，就像您在 AWS Glue 主控台中建立串流應用程式一樣。由於互動式工作階段是以工作階段為基礎，因此在執行時間遇到異常情形不會導致工作階段停止。反覆開發批次函數現在帶來了額外好處。例如：

```
def batch_function(data_frame, batch_id):
    log.info(data_frame.count())
    invalid_method_call()
glueContext.forEachBatch(frame=streaming_df, batch_function = batch_function, options =
 {**})
```

上面的範例中包含了對方法的無效使用，與將退出整個應用程式的常規 AWS Glue 任務不同的是，使用者的編碼內容和定義得到完全保留，且工作階段仍然可運作。無需引導新叢集並重新執行之前的所有轉換。這樣，您就可以專注於快速逐一查看批次函數實作，以獲得理想的結果。

需要注意的是，互動式工作階段會以封鎖的方式評估每個陳述句，以便讓工作階段一次只執行一個陳述句。由於串流查詢是連續的並且永不結束，因此具有作用中串流查詢的工作階段將無法處理任何後續陳述句，除非它們被中斷。您可以直接從 Jupyter 筆記本發出中斷命令，我們的核心會為您處理取消操作。

以下列正在等待執行的陳述句序列為範例：

```
Statement 1:
      val number = df.count()
      #Spark Action with deterministic result
      Result: 5

Statement 2:
      streamingQuery.start().awaitTermination()
```

```
      #Spark Streaming Query that will be executing continously
      Result: Constantly updated with each microbatch

Statement 3:
      val number2 = df.count()
      #This will not be executed as previous statement will be running indefinitely
```

在本機開發和測試 AWS Glue 任務指令碼

開發和測試 AWS Glue for Spark 工作指令碼時，可使用多個選項：

- AWS Glue Studio 主控台
  - Visual editor (視覺化編輯器)
  - 指令碼編輯器
  - AWS Glue Studio 筆記本
- 互動式工作階段
  - Jupyter 筆記本
- Docker 映像檔
  - 本機開發
  - 遠端開發
- AWS Glue Studio ETL 程式庫
  - 本機開發

您可根據自己的需求選擇上述任何選項。

如果不想要使用程式碼或不想使用大量程式碼，那麼 AWS Glue Studio 視覺化編輯器是個不錯的選擇。

如果您偏好互動式筆記本體驗，那麼 AWS Glue Studio 筆記本是個不錯的選擇。如需詳細資訊，請參閱搭配 AWS Glue Studio 和 AWS Glue 使用筆記本。如果想要使用自己的本機環境，那麼互動式工作階段是個不錯的選擇。如需詳細資訊，請參閱搭配 AWS Glue 使用互動式工作階段。

如果您偏好採用本機/遠端開發，那麼 Docker 映像檔是個不錯的選擇。這有助於您在任何地方開發和測試 AWS Glue for Spark 工作指令碼，而不會產生 AWS Glue 成本。

如果您偏好不使用 Docker 的本機開發模式，那麼在本機安裝 AWS Glue ETL 程式庫目錄是個不錯的選擇。

使用 AWS Glue Studio 開發

AWS Glue Studio 視覺化編輯器是一個新的圖形介面，有助於在 AWS Glue 更加輕鬆地內建立、執行、監控擷取、轉換和負載 (ETL) 任務。您可以用視覺化方式撰寫資料轉換工作流程，並在 AWS Glue 的 Apache Spark 型無伺服器 ETL 引擎上順暢地執行它們。您可以在任務的每個步驟中檢查結構描述和資料結果。如需詳細資訊，請參閱《AWS Glue Studio 使用者指南》。

使用互動式工作階段進行開發

使用互動式工作階段允許您自行選擇要建置和測試應用程式的環境。如需詳細資訊，請參閱搭配 AWS Glue 使用互動式工作階段。

使用 Docker 映像檔進行開發

> **Note**
>
> 本節中的指示尚未在 Microsoft Windows 作業系統上進行測試。
> 如需 Windows 平台上的本機開發和測試，請參閱部落格在未使用 AWS 帳戶的情況下，在本機建置 AWS Glue ETL 管道

對於已準備用於生產的資料平台而言，AWS Glue 任務的開發流程和 CI/CD 管道是關鍵的主題。您可以靈活地開發和測試 Docker 容器中的 AWS Glue 任務。AWS Glue 在 Docker Hub 上託管 Docker 映像檔，以運用其他公用程式設定開發環境。您可透過 AWS Glue ETL 程式庫，使用您偏好的 IDE、筆記本或 REPL。本主題說明如何使用 Docker 映像檔，開發和測試 Docker 容器中的 AWS Glue 4.0 版任務。

以下 Docker 映像檔可用於 Docker Hub 上的 AWS Glue。

- 對於 AWS Glue 4.0 版：`amazon/aws-glue-libs:glue_libs_4.0.0_image_01`
- 若為 AWS Glue 3.0 版：`amazon/aws-glue-libs:glue_libs_3.0.0_image_01`
- 若為 AWS Glue 2.0 版：`amazon/aws-glue-libs:glue_libs_2.0.0_image_01`

這些映像檔適用於 x86_64。建議您在此架構上進行測試。然而，在不支援的基礎映像上重製本機開發解決方案可能可行。

此範例說明在本機上使用 amazon/aws-glue-libs:glue_libs_4.0.0_image_01 並執行容器。此容器映像檔已通過 AWS Glue 3.3 版 Spark 任務的測試。該映像檔包含下列項目：

- Amazon Linux
- AWS Glue ETL 程式庫 (aws-glue-libs)
- Apache Spark 3.3.0
- Spark 歷史記錄伺服器
- Jupyter Lab
- Livy
- 與其他程式庫的相依性 (與 AWS Glue 任務系統中的同一組程式庫之間)

根據您的需求填妥下列其中一個區段：

- 將容器設定為使用 spark-submit
- 將容器設定為使用 REPL Shell (PySpark)
- 將容器設定為使用 Pytest
- 將容器設定為使用 Jupyter Lab
- 將容器設定為使用 Visual Studio Code

先決條件

在開始之前，請務必安裝 Docker 並確保 Docker 常駐程式正在執行。如需安裝說明，請參閱 Mac 或 Linux 專用的 Docker 文件。執行 Docker 的電腦負責管理 AWS Glue 容器。亦請確保執行 Docker 的主機上至少有 7 GB 的磁碟空間可供映像檔使用。

如需有關在本機開發 AWS Glue 程式碼的詳細資訊，請參閱本機開發限制。

設定 AWS

若要啟用來自容器的 AWS API 呼叫，請按照以下步驟設定 AWS 憑證。我們會在下列各節中使用此 AWS 具名設定檔。

1. 設定 AWS CLI，配置已命名的設定檔。如需 AWS CLI 組態的詳細資訊，請參閱 AWS CLI 文件中的組態和憑證檔案設定。
2. 在終端機執行下列命令：

```
PROFILE_NAME="<your_profile_name>"
```

您可能還需要設定 AWS_LEGION 環境變數，以指定要傳送請求的目標 AWS 區域。

設定和執行容器

透過 spark-submit 命令設定容器或執行 PySpark 程式碼需進一步完成下列步驟：

1. 從 Docker Hub 擷取映像檔
2. 執行容器。

從 Docker Hub 擷取映像檔

執行以下命令可從 Docker Hub 中擷取映像檔：

```
docker pull amazon/aws-glue-libs:glue_libs_4.0.0_image_01
```

執行容器

然後再使用此映像檔執行容器。您可根據自己的需求選擇下列任何選項。

spark-submit

執行 AWS Glue 任務指令碼的方法是在容器上執行 `spark-submit` 命令。

1. 編寫指令碼並將其儲存為 `/local_path_to_workspace` 目錄下的 `sample1.py`。本主題的附錄列有範本程式碼。

```
$ WORKSPACE_LOCATION=/local_path_to_workspace
$ SCRIPT_FILE_NAME=sample.py
$ mkdir -p ${WORKSPACE_LOCATION}/src
$ vim ${WORKSPACE_LOCATION}/src/${SCRIPT_FILE_NAME}
```

2. 執行下列命令可在容器上執行 `spark-submit` 命令以提交新的 Spark 應用程式：

```
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v $WORKSPACE_LOCATION:/
home/glue_user/workspace/ -e AWS_PROFILE=$PROFILE_NAME -e DISABLE_SSL=true
 --rm -p 4040:4040 -p 18080:18080 --name glue_spark_submit amazon/aws-glue-
libs:glue_libs_4.0.0_image_01 spark-submit /home/glue_user/workspace/src/
$SCRIPT_FILE_NAME
```

```
...22/01/26 09:08:55 INFO DAGScheduler: Job 0 finished: fromRDD at
 DynamicFrame.scala:305, took 3.639886 s
root
|-- family_name: string
|-- name: string
|-- links: array
| |-- element: struct
| | |-- note: string
| | |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
| |-- element: struct
| | |-- scheme: string
| | |-- identifier: string
|-- other_names: array
| |-- element: struct
| | |-- lang: string
| | |-- note: string
| | |-- name: string
|-- sort_name: string
|-- images: array
| |-- element: struct
| | |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
| |-- element: struct
| | |-- type: string
| | |-- value: string
|-- death_date: string

...
```

3. (選用) 設定 `spark-submit` 以符合您的環境。例如，可以將相依項與 `--jars` 組態一起傳遞。如需詳細資訊，請參閱 Spark 文件中的使用 spark-submit 啟動應用程式。

REPL Shell (Pyspark)

採互動式開發模式時，可執行 REPL (read-eval-print loop) Shell。

執行以下命令可在容器上執行 PySpark 命令以啟動 REPL Shell：

```
$ docker run -it -v ~/.aws:/home/glue_user/.aws -e AWS_PROFILE=$PROFILE_NAME -e
 DISABLE_SSL=true --rm -p 4040:4040 -p 18080:18080 --name glue_pyspark amazon/aws-glue-
libs:glue_libs_4.0.0_image_01 pyspark
...
 ____ __
 / __/__ ___ _____/ /__
 _\ \/ _ \/ _ `/ __/ '_/
 /__ / .__/\_,_/_/ /_/\_\  version 3.1.1-amzn-0
 /_/

Using Python version 3.7.10 (default, Jun 3 2021 00:02:01)
Spark context Web UI available at http://56e99d000c99:4040
Spark context available as 'sc' (master = local[*], app id = local-1643011860812).
SparkSession available as 'spark'.
>>>
```

Pytest

單位測試時，AWS Glue Spark 任務指令碼可使用 pytest。

執行下列命令以完成前置作業。

```
$ WORKSPACE_LOCATION=/local_path_to_workspace
$ SCRIPT_FILE_NAME=sample.py
$ UNIT_TEST_FILE_NAME=test_sample.py
$ mkdir -p ${WORKSPACE_LOCATION}/tests
$ vim ${WORKSPACE_LOCATION}/tests/${UNIT_TEST_FILE_NAME}
```

執行下列命令可在測試套件上執行 `pytest`：

```
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v $WORKSPACE_LOCATION:/home/glue_user/
workspace/ -e AWS_PROFILE=$PROFILE_NAME -e DISABLE_SSL=true --rm -p 4040:4040 -p
 18080:18080 --name glue_pytest amazon/aws-glue-libs:glue_libs_4.0.0_image_01 -c
 "python3 -m pytest"
starting org.apache.spark.deploy.history.HistoryServer,
 logging to /home/glue_user/spark/logs/spark-glue_user-
org.apache.spark.deploy.history.HistoryServer-1-5168f209bd78.out
*============================================================= test session starts
 ==============================================================
*platform linux -- Python 3.7.10, pytest-6.2.3, py-1.11.0, pluggy-0.13.1
rootdir: /home/glue_user/workspace
plugins: anyio-3.4.0
```

```
*collected 1 item  *

tests/test_sample.py . [100%]

============================================================= warnings summary
 =============================================================
tests/test_sample.py::test_counts
 /home/glue_user/spark/python/pyspark/sql/context.py:79: DeprecationWarning: Deprecated
 in 3.0.0. Use SparkSession.builder.getOrCreate() instead.
 DeprecationWarning)

-- Docs: https://docs.pytest.org/en/stable/warnings.html
========================================================== 1 passed, *1 warning* in
 21.07s ===========================================================
```

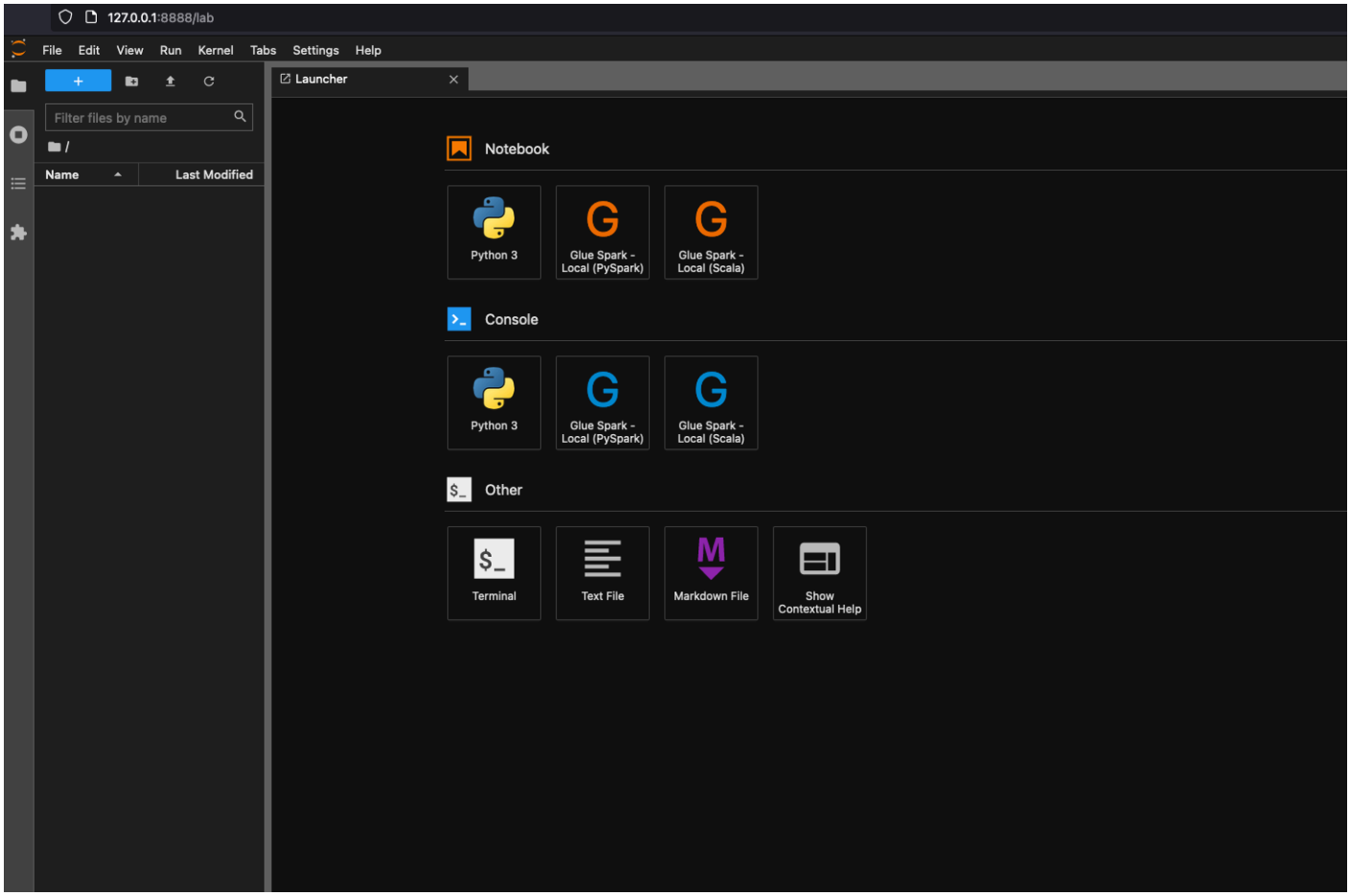
Jupyter Lab

您可在筆記本上啟動 Jupyter 以進行互動式開發和臨機操作查詢。

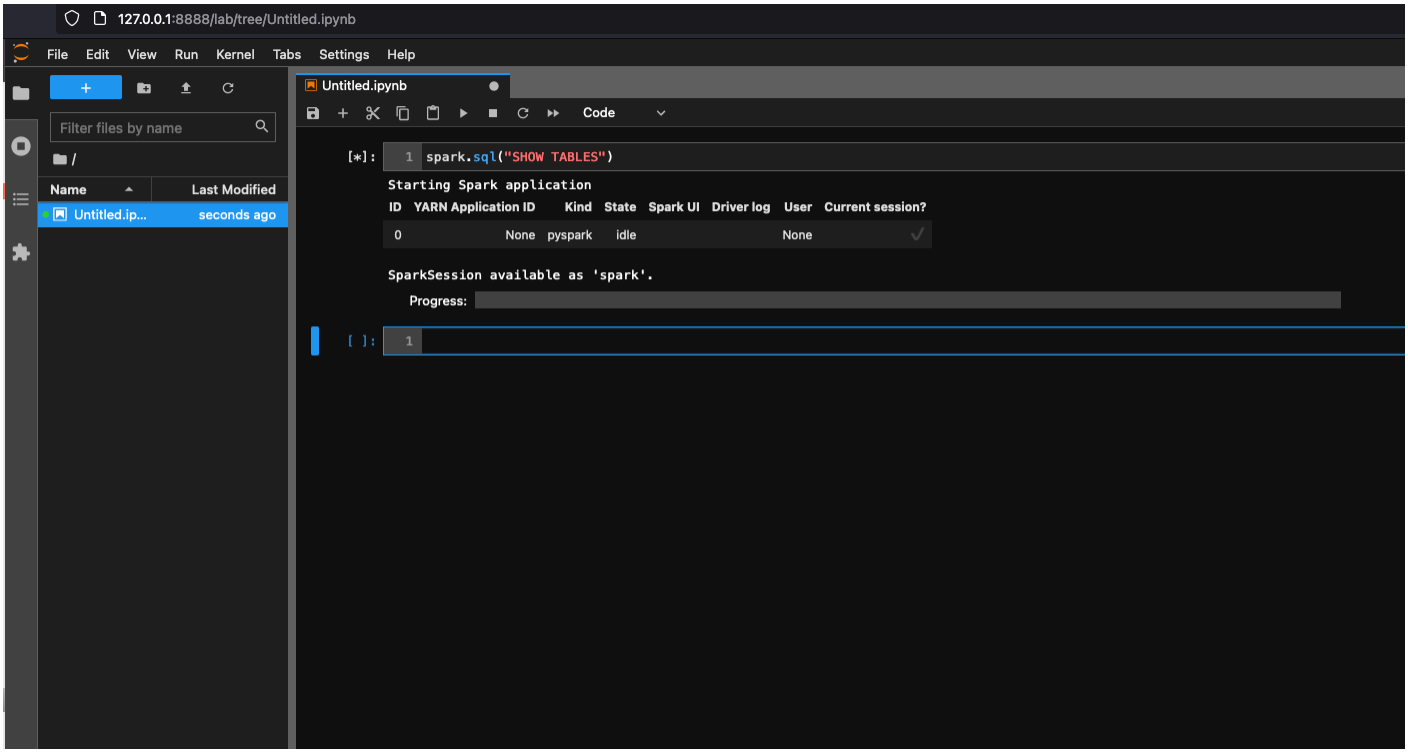
1. 執行下列命令可啟動 Jupyter Lab：

```
$ JUPYTER_WORKSPACE_LOCATION=/local_path_to_workspace/jupyter_workspace/
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v $JUPYTER_WORKSPACE_LOCATION:/
home/glue_user/workspace/jupyter_workspace/ -e AWS_PROFILE=$PROFILE_NAME -e
 DISABLE_SSL=true --rm -p 4040:4040 -p 18080:18080 -p 8998:8998 -p 8888:8888 --name
 glue_jupyter_lab amazon/aws-glue-libs:glue_libs_4.0.0_image_01 /home/glue_user/
jupyter/jupyter_start.sh
...
[I 2022-01-24 08:19:21.368 ServerApp] Serving notebooks from local directory: /home/
glue_user/workspace/jupyter_workspace
[I 2022-01-24 08:19:21.368 ServerApp] Jupyter Server 1.13.1 is running at:
[I 2022-01-24 08:19:21.368 ServerApp] http://faa541f8f99f:8888/lab
[I 2022-01-24 08:19:21.368 ServerApp] or http://127.0.0.1:8888/lab
[I 2022-01-24 08:19:21.368 ServerApp] Use Control-C to stop this server and shut down
 all kernels (twice to skip confirmation).
```

2. 使用本機電腦的 Web 瀏覽器開啟 http://127.0.0.1:8888/lab，即可查看 Jupyter Lab 使用者介面。



3. 選擇 Notebook (筆記本) 之下的 Glue Spark Local (PySpark) (本機 Glue Spark (PySpark))。即可開始使用互動式 Jupyter 筆記本使用者介面來開發程式碼。



將容器設為使用 Visual Studio Code

事前準備：

1. 安裝 Visual Studio Code。
2. 安裝 Python。
3. 安裝 Visual Studio Code Remote - Containers
4. 在 Visual Studio Code 中開啟 workspace (工作區) 資料夾。
5. 選擇設定。
6. 選擇 Workspace (工作區)。
7. 選擇 Open Settings (JSON) (開啟設定 (JSON))。
8. 將下列 JSON 貼上並儲存。

```
{
    "python.defaultInterpreterPath": "/usr/bin/python3",
    "python.analysis.extraPaths": [
        "/home/glue_user/aws-glue-libs/PyGlue.zip:/home/glue_user/spark/python/lib/
py4j-0.10.9-src.zip:/home/glue_user/spark/python/",
    ]
```

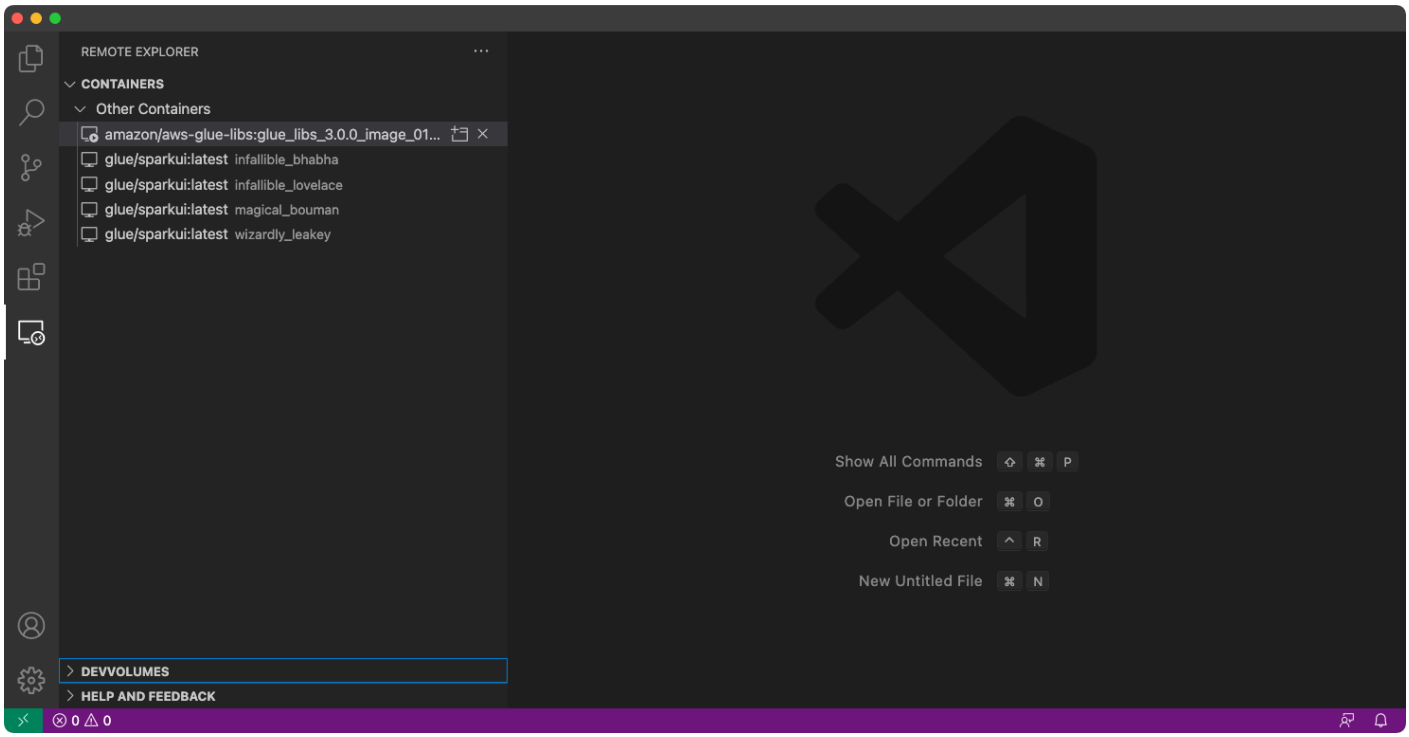
```
}
```

步驟：

1. 執行 Docker 容器。

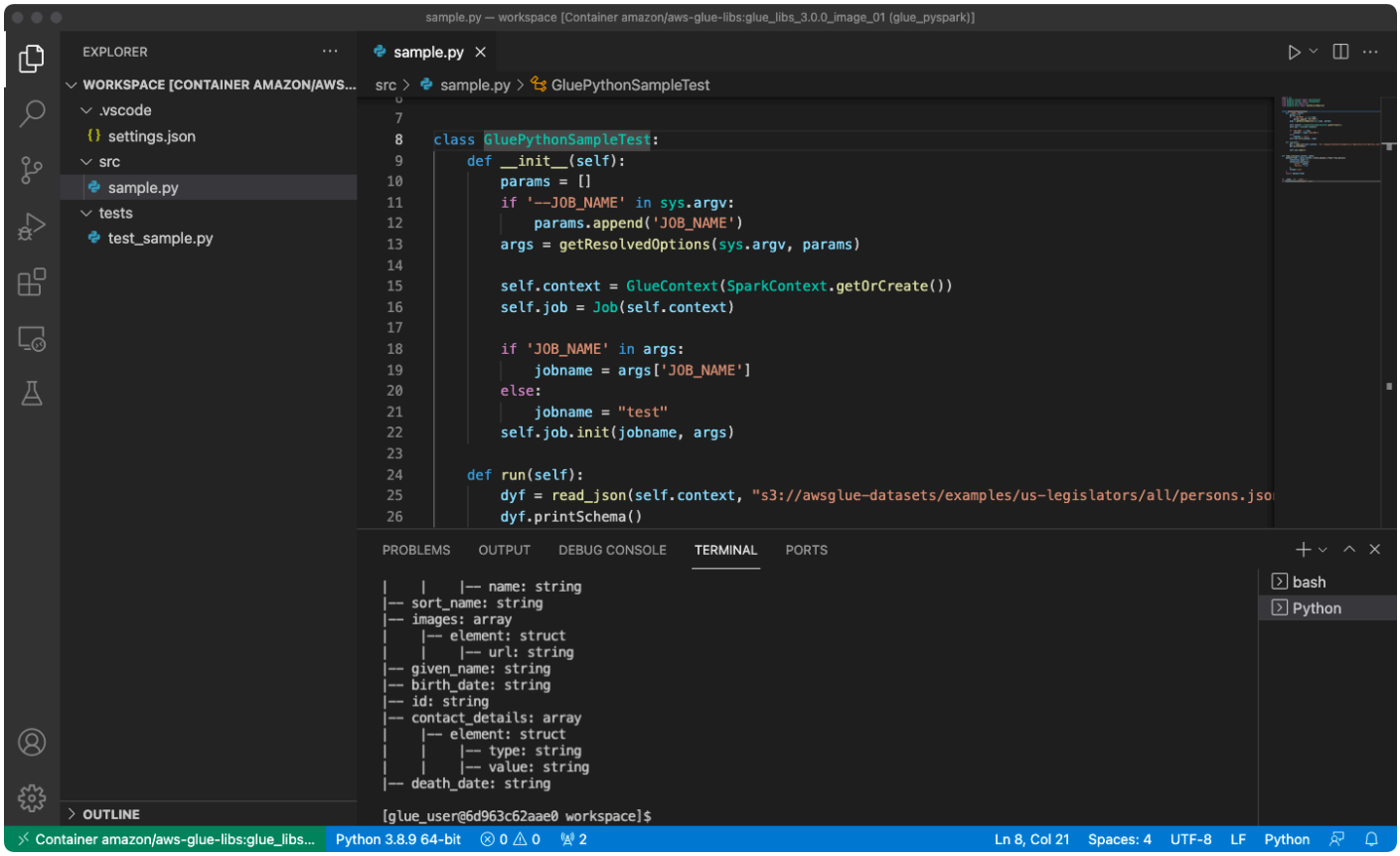
```
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v $WORKSPACE_LOCATION:/
home/glue_user/workspace/ -e AWS_PROFILE=$PROFILE_NAME -e DISABLE_SSL=true
 --rm -p 4040:4040 -p 18080:18080 --name glue_pyspark amazon/aws-glue-
libs:glue_libs_4.0.0_image_01 pyspark
```

2. 啟動 Visual Studio Code。
3. 選擇左側選單中的 Remote Explorer (遠端檔案總管)，然後選擇 `amazon/aws-glue-libs:glue_libs_4.0.0_image_01`。



4. 按一下滑鼠右鍵，選擇 Attach to Container (連接至容器)。若顯示對話方塊，請選擇 Got it (我知道了)。
5. 打開 `/home/glue_user/workspace/`.
6. 建立 Glue PySpark 指令碼，然後選擇 Run (執行)。

   您會看到指令碼成功執行。



附錄：測試用的 AWS Glue 任務範本程式碼

本附錄內含的 AWS Glue 任務範本程式碼為供您測試之用的指令碼。

sample.py：透過 Amazon S3 API 呼叫以使用 AWS Glue ETL 程式庫的範本程式碼

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions


class GluePythonSampleTest:
    def __init__(self):
        params = []
        if '--JOB_NAME' in sys.argv:
            params.append('JOB_NAME')
        args = getResolvedOptions(sys.argv, params)
```

```
        self.context = GlueContext(SparkContext.getOrCreate())
        self.job = Job(self.context)

        if 'JOB_NAME' in args:
            jobname = args['JOB_NAME']
        else:
            jobname = "test"
        self.job.init(jobname, args)

    def run(self):
        dyf = read_json(self.context, "s3://awsglue-datasets/examples/us-legislators/
all/persons.json")
        dyf.printSchema()

        self.job.commit()


def read_json(glue_context, path):
    dynamicframe = glue_context.create_dynamic_frame.from_options(
        connection_type='s3',
        connection_options={
            'paths': [path],
            'recurse': True
        },
        format='json'
    )
    return dynamicframe


if __name__ == '__main__':
    GluePythonSampleTest().run()
```

上述程式碼需有 AWS IAM 的 Amazon S3 許可。您需授予 IAM 受管政策 `arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess` 或 IAM 自訂政策，才能調用 `ListBucket` 和 `GetObject` 作為 Amazon S3 路徑。

test_sample.py：sample.py 單位測試用的範本程式碼。

```
import pytest
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions
```

```
import sys
from src import sample


@pytest.fixture(scope="module", autouse=True)
def glue_context():
    sys.argv.append('--JOB_NAME')
    sys.argv.append('test_count')

    args = getResolvedOptions(sys.argv, ['JOB_NAME'])
    context = GlueContext(SparkContext.getOrCreate())
    job = Job(context)
    job.init(args['JOB_NAME'], args)

    yield(context)

    job.commit()


def test_counts(glue_context):
    dyf = sample.read_json(glue_context, "s3://awsglue-datasets/examples/us-
legislators/all/persons.json")
    assert dyf.toDF().count() == 1961
```

使用 AWS Glue ETL 程式庫進行開發

您可在公有 Amazon S3 儲存貯體中使用 AWS Glue ETL 程式庫，並將其提供給 Apache Maven 建置系統使用。如此一來，您就能在本機開發及測試 Python 和 Scala 擷取、傳輸和載入 (ETL) 指令碼，而無需連線至網路。建議使用 Docker 映像檔進行本地開發，因為它提供了正確設定的環境以使用此程式庫。

本機開發適用於所有 AWS Glue 版本，包括 AWS Glue 0.9 版、1.0 版、2.0 版及更新版本。如需 AWS Glue 適用的 Python 和 Apache Spark 版本相關資訊，請參閱 Glue version job property。

該程式庫會隨 Amazon 軟體授權 (https://aws.amazon.com/asl) 一併發行。

本機開發限制

使用 AWS Glue Scala 程式庫在本機進行開發時，請謹記下列限制。

- 避免使用 AWS Glue 程式庫建立 Jar 組合 (「fat Jar」或「uber Jar」)，因為這可能導致下列功能遭停用：

- 任務書籤
- AWS Glue Parquet 寫入器 (在 AWS Glue 中使用 Parquet 格式)
- FillMissingValues 轉換 (Scala 或 Python)

上述功能只能在 AWS Glue 任務系統中使用。

- 本地開發不支援 FindMatches 轉換。
- 進行本機開發時，不支援向量化 SIMD CSV 讀取器。
- 進行本機開發時，不支援用於從 S3 路徑載入 JDBC 驅動程式的屬性 customJdbcDriverS3Path。或者，您可以在本機下載 JDBC 驅動程式並從中載入。
- 進行本機開發時，不支援 Glue Data Quality。

在本機開發 Python

您需要先完成一些事前準備步驟，才能接著使用 AWS Glue 公用程式來測試及提交 Python ETL 指令碼。

在本機開發 Python 的事前準備

請完成下列步驟以準備在本機開發 Python：

1. 複製 GitHub 的 AWS Glue Python 儲存庫 (https://github.com/awslabs/aws-glue-libs)。
2. 執行下列作業之一：
   - 若為 AWS Glue 0.9 版，請簽出分支 `glue-0.9`。
   - 若為 AWS Glue 1.0 版，請簽出分支 `glue-1.0`。AWS Glue 0.9 以上的所有版本都援 Python 3。
   - 若為 AWS Glue 2.0 版，請簽出分支 `glue-2.0`。
   - 若為 AWS Glue 3.0 版，請簽出分支 `glue-3.0`。
   - 若為 AWS Glue 4.0 版，請簽出 `master` 分支。
3. 從以下位置安裝 Apache Maven：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-common/apache-maven-3.6.0-bin.tar.gz。
4. 從以下位置安裝 Apache Spark 分發：
   - 若為 AWS Glue 0.9 版：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-0.9/spark-2.2.1-bin-hadoop2.7.tgz
   - 若為 AWS Glue 1.0 版：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-1.0/spark-2.4.3-bin-hadoop2.8.tgz

- 若為 AWS Glue 2.0 版：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-2.0/spark-2.4.3-bin-hadoop2.8.tgz
- 若為 AWS Glue 3.0 版：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-3.0/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3.tgz
- 對於 AWS Glue 4.0 版：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-4.0/spark-3.3.0-amzn-1-bin-3.3.3-amzn-0.tgz

5. 匯出 SPARK_HOME 環境變數，並以擷取自 Spark 存檔的根位置來設定該變數。例如：

- 若為 AWS Glue 0.9 版：`export SPARK_HOME=/home/$USER/spark-2.2.1-bin-hadoop2.7`
- 若為 AWS Glue 1.0 版和 2.0 版：`export SPARK_HOME=/home/$USER/spark-2.4.3-bin-spark-2.4.3-bin-hadoop2.8`
- 若為 AWS Glue 3.0 版：`export SPARK_HOME=/home/$USER/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3`
- 對於 AWS Glue 4.0 版：`export SPARK_HOME=/home/$USER/spark-3.3.0-amzn-1-bin-3.3.3-amzn-0`

執行 Python ETL 指令碼

透過適用於本機開發的 AWS Glue Jar 檔案，您即可在本機執行 AWS Glue Python 套件。

請使用下列公用程式和架構來測試 Python 指令碼，並予以執行。下表所列的命令皆需從 AWS Glue Python 套件的根目錄執行。

公用程式	命令	說明
AWS Glue Shell	`./bin/gluepyspark`	在與 AWS Glue ETL 程式庫整合的 shell 中輸入並執行 Python 指令碼。
AWS Glue Submit (提交)	`./bin/gluesparksubmit`	提交要執行的完整 Python 指令碼。
Pytest	`./bin/gluepytest`	撰寫並執行 Python 程式碼的單元測試。請務必在 PATH 中安裝並使用 pytest 模組。如需詳細資訊，請參閱 Pytest 文件。

在本機開發 Scala

您需要先完成一些事前準備步驟，然後發出 Maven 命令，才能在本機執行 Scala ETL 指令碼。

在本機開發 Scala 的事前準備

請完成下列步驟以準備在本機開發 Scala：

步驟 1：安裝軟體

在此步驟中，您需要安裝軟體並設定必要的環境變數。

1. 從以下位置安裝 Apache Maven：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-common/apache-maven-3.6.0-bin.tar.gz。
2. 從以下位置安裝 Apache Spark 分發：
   - 若為 AWS Glue 0.9 版：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-0.9/spark-2.2.1-bin-hadoop2.7.tgz
   - 若為 AWS Glue 1.0 版：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-1.0/spark-2.4.3-bin-hadoop2.8.tgz
   - 若為 AWS Glue 2.0 版：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-2.0/spark-2.4.3-bin-hadoop2.8.tgz
   - 若為 AWS Glue 3.0 版：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-3.0/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3.tgz
   - 對於 AWS Glue 4.0 版：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-4.0/spark-3.3.0-amzn-1-bin-3.3.3-amzn-0.tgz
3. 匯出 `SPARK_HOME` 環境變數，並以擷取自 Spark 存檔的根位置來設定該變數。例如：
   - 若為 AWS Glue 0.9 版：`export SPARK_HOME=/home/$USER/spark-2.2.1-bin-hadoop2.7`
   - 若為 AWS Glue 1.0 版和 2.0 版：`export SPARK_HOME=/home/$USER/spark-2.4.3-bin-spark-2.4.3-bin-hadoop2.8`
   - 若為 AWS Glue 3.0 版：`export SPARK_HOME=/home/$USER/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3`
   - 對於 AWS Glue 4.0 版：`export SPARK_HOME=/home/$USER/spark-3.3.0-amzn-1-bin-3.3.3-amzn-0`

步驟 2：設定 Maven 專案

您可使用以下 pom.xml 檔案做為 AWS Glue Scala 應用程式的範本。它包含必要的 dependencies、repositories 和 plugins 元素。以下列其中一項取代 Glue version 字串：

- 適用於 AWS Glue 4.0 版的 4.0.0
- 適用於 AWS Glue 3.0 版的 3.0.0
- 適用於 AWS Glue 1.0 或 2.0 版的 1.0.0
- 適用於 AWS Glue 0.9 版的 0.9.0

```
  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.amazonaws</groupId>
    <artifactId>AWSGlueApp</artifactId>
    <version>1.0-SNAPSHOT</version>
    <name>${project.artifactId}</name>
    <description>AWS ETL application</description>

        <properties>
            <scala.version>2.11.1 for AWS Glue 2.0 or below, 2.12.7 for AWS Glue 3.0
 and 4.0</scala.version>
            <glue.version>Glue version with three numbers (as mentioned earlier)</
glue.version>
        </properties>
    <dependencies>
        <dependency>
            <groupId>org.scala-lang</groupId>
            <artifactId>scala-library</artifactId>
            <version>${scala.version}</version>
   <!-- A "provided" dependency, this will be ignored when you package your application
 -->
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>AWSGlueETL</artifactId>
   <version>${glue.version}</version>
            <!-- A "provided" dependency, this will be ignored when you package your
 application -->
```

```
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <repositories>
        <repository>
            <id>aws-glue-etl-artifacts</id>
            <url>https://aws-glue-etl-artifacts.s3.amazonaws.com/release/</url>
        </repository>
    </repositories>
    <build>
        <sourceDirectory>src/main/scala</sourceDirectory>
        <plugins>
            <plugin>
                <!-- see http://davidb.github.com/scala-maven-plugin -->
                <groupId>net.alchim31.maven</groupId>
                <artifactId>scala-maven-plugin</artifactId>
                <version>3.4.0</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>compile</goal>
                            <goal>testCompile</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>1.6.0</version>
                <executions>
                    <execution>
                        <goals>
                        <goal>java</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                <systemProperties>
                    <systemProperty>
                        <key>spark.master</key>
                        <value>local[*]</value>
                    </systemProperty>
```

```
                    <systemProperty>
                        <key>spark.app.name</key>
                        <value>localrun</value>
                    </systemProperty>
                    <systemProperty>
                        <key>org.xerial.snappy.lib.name</key>
                        <value>libsnappyjava.jnilib</value>
                    </systemProperty>
                </systemProperties>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-enforcer-plugin</artifactId>
                <version>3.0.0-M2</version>
                <executions>
                    <execution>
                        <id>enforce-maven</id>
                        <goals>
                            <goal>enforce</goal>
                        </goals>
                        <configuration>
                            <rules>
                                <requireMavenVersion>
                                    <version>3.5.3</version>
                                </requireMavenVersion>
                            </rules>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
  <!-- The shade plugin will be helpful in building a uberjar or fatjar.
  You can use this jar in the AWS Glue runtime environment. For more information, see
https://maven.apache.org/plugins/maven-shade-plugin/ -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.2.4</version>
                <configuration>
                    <!-- any other shade configurations -->
                </configuration>
                <executions>
                    <execution>
                        <phase>package</phase>
```

```
                        <goals>
                            <goal>shade</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

執行 Scala ETL 指令碼

請從 Maven 專案的根目錄執行以下命令，藉此執行 Scala ETL 指令碼。

```
mvn exec:java -Dexec.mainClass="mainClass" -Dexec.args="--JOB-NAME jobName"
```

使用該指令碼主要類別的完全合格類別名稱來取代 *mainClass*，並將 *jobName* 替換成所需任務名稱。

設定測試環境

如需設定本機測試環境的範例，請參閱下列部落格文章：

- 在未使用 AWS 帳戶的情況下，在本機建置 AWS Glue ETL 管道
- 使用容器在本機開發 AWS Glue ETL 任務

如果您想要使用開發端點或筆記本來測試 ETL 指令碼，請參閱使用開發端點來開發指令碼。

> **Note**
>
> 開發端點不支援用於 AWS Glue 2.0 版任務。如需詳細資訊，請參閱以縮短的啟動時間執行 Spark ETL 任務。

開發端點

> **Note**
>
> 自 2023 年 3 月 31 日起，已經移除開發端點的主控台體驗。您仍可透過 開發端點 API 和 AWS Glue CLI 建立、更新和監控開發端點。

基於下列原因，我們強烈建議您從開發端點遷移到互動式工作階段。如需如何從開發端點遷移到互動式工作階段的所需動作，請參閱從開發端點遷移至互動式工作階段。

描述	開發端點	互動式工作階段
Glue 版本支援	支援 AWS Glue 0.9 版和 1.0 版	支援 AWS Glue 2.0 版和更高版本
開發端點在亞太區域 (雅加達) (ap-southeast-3)、中東 (阿拉伯聯合大公國) (me-central-1)、歐洲 (西班牙) (eu-south-2)、歐洲 (蘇黎世) (eu-central-2) 或今後推出服務的其他新區域中不可使用	互動式工作階段目前在中東 (阿拉伯聯合大公國) (me-central-1) 區域未提供使用，但稍後會提供。	
Spark 叢集的存取方法	支援 SSH、REPL Shell、Jupyter 筆記本、IDE (如 PyCharm)	支援 AWS Glue Studio 筆記本、Jupyter 筆記本、各種 IDE (例如 Visual Studio Code、PyCharm) 和 SageMaker 筆記本
至第一次查詢的時間	Spark 叢集需要 10-15 分鐘的設定時間	暫時性 Spark 叢集最長需要 1 分鐘的設定時間
價格模型	AWS 會根據佈建端點的時間和 DPU 數量對開發端點收費。開發端點不會逾時。每個佈建的開發端點有 10 分鐘的最短計費	AWS 依據工作階段處於作用中的時間和 DP 數目計費。互動式工作階段具有可設定的閒置逾時。 AWS Glue Studio 筆記

描述	開發端點	互動式工作階段
	持續時間。此外，如果您使用開發端點設定 Amazon EC2 執行個體上的 Jupyter 筆記本和 SageMaker 筆記本，AWS 會對這二者收費。	本提供互動式工作階段的內建介面，而且無需額外付費。每個互動工作階段都有 1 分鐘的最短計費持續時間。AWS Glue Studio 筆記本提供互動式工作階段的內建介面，且不收取其他費用。
主控台體驗	僅透過 CLI 和 API 提供	透過 AWS Glue 主控台、CLI 和 API 提供

從開發端點遷移至互動式工作階段

使用下列檢查清單決定要從開發端點遷移至互動式工作階段的合適方法。

您的指令碼是否仰賴 AWS Glue 0.9 或 1.0 的特定功能 (例如 HDFS、YARN)？

如果答案為是，請參閱將 AWS Glue 任務遷移至 AWS Glue 3.0 版，了解如何從 Glue 0.9 或 1.0 遷移至 Glue 3.0 及更高版本。

您使用哪種方法存取自己的開發端點？

如果使用此方法	則執行此操作
SageMaker 筆記本、Jupyter 筆記本或 JupyterLab	在 Jupyter 上下載 `.ipynb` 檔案以遷移到 AWS Glue Studio 筆記本，然後上傳 `.ipynb` 檔案以建立新的 AWS Glue Studio 筆記本任務。或者，您也可以使用 SageMaker Studio 並選擇 AWS Glue 內核。
Zeppelin 筆記本	透過複製並貼上程式碼手動將筆記本轉換成 Jupyter 筆記本，或使用第三方轉換器 (如 ze2nb) 自動轉換。然後在 AWS Glue Studio 筆記本或 SageMaker Studio 中使用筆記本。
IDE	請參閱使用 AWS Glue 互動式工作階段搭配 PyCharm 撰寫 AWS Glue 任務，或將互動式工

如果使用此方法	則執行此操作
	作階段與 Microsoft Visual Studio Code 搭配使用。
REPL	在本機安裝 aws-glue-session package，然後請執行下列命令： • 適用於 Python：jupyter console --kernal glue_pyspark • 適用於 Scala：jupyter console --kernal glue_spark
SSH	互動式工作階段中沒有對應的選項。或者，您可以使用 Docker 映像檔。如需進一步了解，請參閱使用 Docker 映像檔進行開發。

以下章節將提供在 AWS Glue 1.0 版使用開發端點來開發任務的資訊。

主題

- 使用開發端點來開發指令碼
- 管理筆記本

使用開發端點來開發指令碼

> **Note**
>
> 開發端點僅支援 2.0 以前的 AWS Glue 版本。針對可以撰寫和測試 ETL 指令碼的互動環境，請參閱 Notebooks on AWS Glue Studio。

AWS Glue 可以建立環境 (稱為開發端點)，以反覆開發和測試您的擷取、轉換和載入 (ETL) 指令碼。您可以使用 AWS Glue 主控台或 API 建立、編輯和刪除開發端點。

管理您的開發環境

當您建立開發端點時，必須提供組態值來佈建開發環境。這些值會告訴 AWS Glue 如何設定網路，讓您安全地存取您的端點，同時讓端點可以存取您的資料存放區。

然後，建立連接到端點的筆記本，並使用您的筆記本編寫和測試您的 ETL 指令碼。當您對於開發程序的結果感到滿意時，可建立 ETL 任務以執行您的指令碼。透過此程序，您可以用互動的方式新增函數和偵錯您的指令碼。

遵循本節中的教學課程，了解如何透過筆記本使用您的開發端點。

主題

- 開發端點工作流程
- AWS Glue 開發端點如何搭配 SageMaker 筆記本使用
- 新增開發端點
- 存取您的開發端點
- 教學課程：在 JupyterLab 中設定 Jupyter 筆記本以測試和偵錯 ETL 指令碼
- 教學課程：搭配開發端點使用 SageMaker 筆記本。
- 教學課程：使用 REPL shell 搭配開發端點
- 教學課程：設定 PyCharm professional 與開發端點
- 進階組態：在多個使用者之間共用開發端點

開發端點工作流程

若要使用 AWS Glue 開發端點，您可以遵循此工作流程：

1. 使用 API 建立開發端點。此端點將在虛擬私有雲端 (VPC) 中啟動，並包含您定義的安全群組。
2. API 會輪詢開發端點，直到其佈建完成且可供使用。準備就緒時，請使用下列其中一個方法連接到開發端點，以建立和測試 AWS Glue 指令碼。
   - 在您的帳戶中建立 SageMaker 筆記本。如需如何建立筆記本的詳細資訊，請參閱the section called “使用 AWS Glue Studio 筆記本編寫程式碼”。
   - 開啟終端機視窗直接連接到開發端點。
   - 如果您有專業版的 JetBrains PyCharm Python IDE，請將它連接到開發端點，並以互動方式將其用於開發作業。如果您將 pydevd 陳述式插入您的指令碼，PyCharm 就能支援遠端中斷點。
3. 當您完成偵錯和測試您的開發端點，您就可以將它刪除。

AWS Glue 開發端點如何搭配 SageMaker 筆記本使用

存取開發端點的常見方法之一是在 SageMaker 筆記本上使用 Jupyter。Jupyter 筆記本是一個開源 Web 應用程式，廣泛用於視覺化、分析、機器學習等。AWS Glue SageMaker 筆記本為您提供了一個搭配 AWS Glue 開發端點的 Jupyter 筆記本體驗。在 AWS Glue SageMaker 筆記本中，Jupyter 筆記本環境預先設定了 SparkMagic，這是一個開源的 Jupyter 外掛程式，可將 Spark 任務提交到遠端 Spark 叢集。Apache Livy 是一種允許透過 REST API 與遠端 Spark 叢集進行互動的服務。在 AWS Glue SageMaker 筆記本中，SparkMagic 被設定為針對在 AWS Glue 開發端點執行的 Livy 伺服器呼叫 REST API。

下列文字流排說明每個元件的運作方式：

AWS Glue SageMaker 筆記本：(Jupyter → SparkMagic) → (網路) → AWS Glue 開發端點：(Apache Livy → Apache Spark)

一旦您在 Jupyter 筆記本上執行在每個段落中編寫的 Spark 指令碼，Spark 程式碼就會透過 SparkMagic 提交到 Livy 伺服器，然後一個名為 "livy-session-N" 的 Spark 任務會在 Spark 叢集上執行。這個任務叫做 Livy 工作階段。Spark 任務將在筆記本工作階段處於活動狀態時執行。當您從筆記本關閉 Jupyter 核心或工作階段逾時時，Spark 任務將會終止。每個筆記本 (.ipynb) 檔案啟動一個 Spark 任務。

您可以使用單一 AWS Glue 開發端點來搭配多個 SageMaker 筆記本執行個體。您可以在每個 SageMaker 筆記本執行個體中建立多個筆記本檔案。當您開啟每個筆記本檔案並執行段落時，會透過 SparkMagic 在 Spark 叢集上每個筆記本檔案的 Livy 工作階段啟動。每個 Livy 工作階段對應於單個 Spark 任務。

AWS Glue 開發端點和 SageMaker 筆記本的預設行為

Spark 任務是根據 Spark 設定執行。有多種方法可以設定 Spark 組態 (例如 Spark 叢集組態、SparkMagic 的組態等)。

依預設，Spark 根據 Spark 叢集組態配置叢集資源給 Livy 工作階段。在 AWS Glue 開發端點中，叢集組態取決於工作者類型。下面是一個資料表，它解釋了每個工作者類型的常見組態。

	標準	G.1X	G.2X
`spark.driver.memory`	5G	10G	20G

	標準	G.1X	G.2X
spark.executor.memory	5G	10G	20G
spark.executor.cores	4	8	16
spark.dynamicAllocation.enabled	TRUE	TRUE	TRUE

Spark 執行器的最大數量由 DPU (或 NumberOfWorkers) 和工作者類型自動計算。

	標準	G.1X	G.2X
最大 Spark 執行器數量	(DPU - 1) * 2 - 1	(NumberOfWorkers - 1)	(NumberOfWorkers - 1)

例如，如果您的開發端點有 10 個工作者，並且工作者類型為 G.1X，那麼您將有 9 個 Spark 執行器，並且整個叢集將有 90G 的執行器記憶體，因為每個執行器都有 10G 的記憶體。

無論指定的工作者類型為何，Spark 動態資源配置都會開啟。如果資料集足夠大，Spark 可以將所有執行器配置給單個 Livy 工作階段，因為預設未設定 spark.dynamicAllocation.maxExecutors。這代表同一個開發端點上的其他 Livy 工作階段將等待啟動新的執行器。如果資料集很小，Spark 將能夠同時將執行器配置給多個 Livy 工作階段。

> **Note**
>
> 如需有關如何在不同的使用案例中配置資源，以及如何設定組態來修改行為的詳細資訊，請參閱 進階組態：在多個使用者之間共用開發端點。

新增開發端點

使用開發端點在 AWS Glue 中反覆開發和測試您的擷取、轉換和載入 (ETL) 指令碼。只能透過 AWS Command Line Interface 使用開發端點。

1. 在命令列視窗中，輸入類似如下的命令。

```
aws glue create-dev-endpoint --endpoint-name "endpoint1" --role-arn
 "arn:aws:iam::account-id:role/role-name" --number-of-nodes "3" --glue-version
 "1.0" --arguments '{"GLUE_PYTHON_VERSION": "3"}' --region "region-name"
```

   此命令指定 AWS Glue 1.0 版。由於此版本同時支援 Python 2 和 Python 3，您可以使用 `arguments` 參數來指示所需的 Python 版本。如果省略 `glue-version` 參數，將會假設 AWS Glue 0.9 版。如需 AWS Glue 版本的詳細資訊，請參閱 Glue version job property。

   如需其他命令列參數的相關資訊，請參閱 AWS CLI 命令參考中的 create-dev-endpoint。

2. (選用) 輸入下列命令來查看開發端點狀態。當狀態變更為 READY，開發端點已就緒可供使用。

```
aws glue get-dev-endpoint --endpoint-name "endpoint1"
```

存取您的開發端點

當您在虛擬私有雲端 (VPC) 中建立開發端點時，AWS Glue 只會傳回一個私有 IP 地址。不會填入公有 IP 地址欄位。建立非 VPC 開發端點時，AWS Glue 只會傳回一個公有 IP 地址。

如果您的開發端點具有公有地址，請確認可使用開發端點的 SSH 私有金鑰與其連接，如下列範例所示。

```
ssh -i dev-endpoint-private-key.pem glue@public-address
```

假設您的開發端點具有私有地址，VPC 子網路可從公有網際網路路由，且其安全群組允許來自用戶端的傳入存取。在此情況下，請依照以下步驟將彈性 IP 地址連接到開發端點，以允許來自網際網路的存取。

> **Note**
>
> 如果您想要使用彈性 IP 地址，使用的子網路需要透過與路由表相關聯的網際網路閘道。

透過連接彈性 IP 地址來存取開發端點

1. 開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格中，選擇 Dev endpoints (開發端點)，然後導覽至開發端點詳細資訊頁面。記錄在後續步驟中使用的私有地址。
3. 在 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
4. 在導覽窗格的 Network & Security (網路與安全) 中，選擇 Network Interfaces (網路界面)。
5. 在 AWS Glue 主控台開發端點詳細資訊頁面中搜尋與 Private address (私有地址) 相對應的 Private DNS (IPv4) (私有 DNS (IPv4))。

   您可能需要修改在 Amazon EC2 主控台中會顯示哪些資料欄。請注意此地址的 Network interface ID (網路界面 ID) (ENI) (例如 `eni-12345678`)。
6. 在 Amazon EC2 主控台的 Network & Security (網路與安全) 中，選擇 Elastic IPs (彈性 IP)。
7. 選擇 Allocate new address (配置新地址)，然後選擇 Allocate (配置) 以配置新的彈性 IP 地址。
8. 在 Elastic IP (彈性 IP) 頁面，選擇新配置的彈性 IP。然後，選擇 Actions (動作)、Associate address (關聯地址)。
9. 在 Associate address (關聯地址) 頁面上，執行下列動作：
   - 對於資源類型，選擇 Network interface (網路介面)。
   - 在 Network interface (網路界面) 方塊中，輸入私有地址的 Network interface ID (網路界面 ID) (ENI)。
   - 選擇 Associate (關聯)。
10. 確認與開發端點相關聯的 SSH 私有金鑰可存取新關聯的彈性 IP 地址，如下列範例所示。

```
ssh -i dev-endpoint-private-key.pem glue@elastic-ip
```

如需使用堡壘主機讓 SSH 存取開發端點私有地址的詳細資訊，請參閱 AWS 安全部落格文章安全地連接到執行於私有 Amazon VPC 的 Linux 執行個體。

教學課程：在 JupyterLab 中設定 Jupyter 筆記本以測試和偵錯 ETL 指令碼

在本教學課程中，您將在本機電腦上執行的 JupyterLab 中 Jupyter 筆記本連接到開發端點。您可以執行這項操作，在進行部署之前，先以互動方式執行、偵錯和測試 AWS Glue 擷取、轉換和載入 (ETL)

指令碼。此教學課程使用 Secure Shell (SSH) 連接埠轉送以連接您的本機至 AWS Glue 開發端點。如需詳細資訊，請參閱 Wikipedia 中的 Port forwarding。

步驟 1：安裝 JupyterLab 和 Sparkmagic

您使用 `conda` 或 `pip` 來安裝 JupyterLab。`conda` 是可在 Windows、macOS 和 Linux 上執行的開源套件管理系統以及環境管理系統。`pip` 是 Python 的套件安裝程式。

如果您在 macOS 上安裝，您必須先安裝 Xcode，才能安裝 Sparkmagic。

1. 安裝 JupyterLab、Sparkmagic 和相關的延伸項目。

```
$ conda install -c conda-forge jupyterlab
$ pip install sparkmagic
$ jupyter nbextension enable --py --sys-prefix widgetsnbextension
$ jupyter labextension install @jupyter-widgets/jupyterlab-manager
```

2. 檢查 `Location` 的 `sparkmagic` 目錄。

```
$ pip show sparkmagic | grep Location
Location: /Users/username/.pyenv/versions/anaconda3-5.3.1/lib/python3.7/site-
packages
```

3. 將您的目錄變更為 `Location` 傳回的目錄，並安裝 Scala 和 PySpark 的核心。

```
$ cd /Users/username/.pyenv/versions/anaconda3-5.3.1/lib/python3.7/site-packages
$ jupyter-kernelspec install sparkmagic/kernels/sparkkernel
$ jupyter-kernelspec install sparkmagic/kernels/pysparkkernel
```

4. 下載範例 `config` 檔案。

```
$ curl -o ~/.sparkmagic/config.json https://raw.githubusercontent.com/jupyter-
incubator/sparkmagic/master/sparkmagic/example_config.json
```

   在此組態檔案中，您可以設定 Spark 相關的參數，如 `driverMemory` 和 `executorCores`。

步驟 2：啟動 JupyterLab

當您啟動 JupyterLab 時，系統會自動開啟您的預設網頁瀏覽器，然後顯示 URL `http://localhost:8888/lab/workspaces/{workspace_name}`。

```
$ jupyter lab
```

步驟 3：啟動 SSH 連接埠轉送以連線到您的開發端點

然後，使用 SSH 本機連接埠轉送，將本機連接埠 (在這裡是 8998) 轉送至 AWS Glue (169.254.76.1:8998) 定義的遠端目的地。

1. 開啟單獨的終端機視窗，讓您存取 SSH。在 Microsoft Windows 上，您可以使用 Git for Windows 所提供的 BASH shell，或安裝 Cygwin。
2. 執行以下 SSH 命令，修改如下：

   - 以 .pem 檔案 (內含與您用於建立開發端點的公有金鑰相關聯的私有金鑰) 的路徑取代 *private-key-file-path*。
   - 如果您轉送不同於 8998 的連接埠，請以您實際在本機使用的連接埠號取代 8998。地址 169.254.76.1:8998 是遠端連接埠，且並非由您所變更。
   - 以您的開發端點的公有 DNS 位址取代 *dev-endpoint-public-dns*。若要尋找此地址，請在 AWS Glue 主控台瀏覽到您的開發端點並選擇其名稱，然後將 Endpoint details (端點詳細資訊) 頁面中列出的 Public address (公有地址)。

   ```
   ssh -i private-key-file-path -NTL 8998:169.254.76.1:8998 glue@dev-endpoint-public-dns
   ```

   您可能會看到類似如下的警告訊息：

   ```
   The authenticity of host 'ec2-xx-xxx-xxx-xx.us-west-2.compute.amazonaws.com
    (xx.xxx.xxx.xx)'
   can't be established.  ECDSA key fingerprint is SHA256:4e97875Brt+1wKzRko
   +JflSnp21X7aTP3BcFnHYLEts.
   Are you sure you want to continue connecting (yes/no)?
   ```

   輸入 **yes**，然後在使用 JupyterLab 時維持終端機視窗在開啟狀態。
3. 檢查 SSH 連接埠轉送是否正確與開發端點一起運作。

   ```
   $ curl localhost:8998/sessions
   {"from":0,"total":0,"sessions":[]}
   ```

步驟 4：在筆記本段落中執行簡單的指令碼片段

現在，您的 JupyterLab 中的筆記本應該可以與您的開發端點一起工作。將以下指令碼片段輸入筆記本並執行。

1. 檢查 Spark 是否成功執行。下面的命令會指示 Spark 計算 1，然後列印該值。

```
spark.sql("select 1").show()
```

2. 檢查 AWS Glue Data Catalog 整合是否正在運作。以下命令會列出 Data Catalog 中的資料表。

```
spark.sql("show tables").show()
```

3. 檢查使用 AWS Glue 程式庫的簡單指令碼片段是否有效。

   以下指令碼使用 AWS Glue Data Catalog 中的 `persons_json` 資料表中繼資料以從範例資料中建立 `DynamicFrame`。接著它會列印出資料項目數和此資料的結構描述。

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create a Glue context
glueContext = GlueContext(SparkContext.getOrCreate())

# Create a DynamicFrame using the 'persons_json' table
persons_DyF = glueContext.create_dynamic_frame.from_catalog(database="legislators",
 table_name="persons_json")

# Print out information about *this* data
print("Count:  ", persons_DyF.count())
persons_DyF.printSchema()
```

指令碼的輸出如下。

```
Count:  1961
root
|-- family_name: string
|-- name: string
|-- links: array
```

```
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- name: string
|    |    |-- lang: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string
```

疑難排解

- 在 JupyterLab 安裝期間，如果您的電腦位於公司代理伺服器或防火牆後方，您可能會因為公司 IT 部門管理的自訂安全性設定檔而遇到 HTTP 和 SSL 錯誤。

  以下是當 conda 無法連接到它自己的儲存庫時會發生的典型錯誤範例：

  ```
  CondaHTTPError: HTTP 000 CONNECTION FAILED for url <https://repo.anaconda.com/pkgs/
  main/win-64/current_repodata.json>
  ```

  這可能是因為您的公司封鎖了與 Python 和 JavaScript 社群中廣泛使用的儲存庫的連線。如需詳細資訊，請參閱 JupyterLab 網站上的安裝問題。
- 如果在嘗試連接到開發端點時遇到連線遭拒的錯誤，有可能您使用的開發端點已過期。請嘗試建立新的開發端點並重新連線。

教學課程：搭配開發端點使用 SageMaker 筆記本。

在 AWS Glue 中，您可以建立開發端點，然後建立 SageMaker 筆記本以協助開發 ETL 和機器學習指令碼。SageMaker 筆記本執行個體是全受管的機器學習運算執行個體，可執行 Jupyter 筆記本應用程式。

1. 在 AWS Glue 主控台，選擇 Dev endpoints (開發端點) 以導覽至開發端點清單。
2. 在您要使用的開發端點名稱旁選取核取方塊，然後在 Action (動作) 選單中，選擇 Create SageMaker notebook (建立 SageMaker 筆記本)。
3. 填寫 Create and configure a notebook (建立並設定筆記本) 頁面，如下所示：

   a. 輸入記事本名稱。

   b. 在 Attach to development endpoint (連接至開發端點)，驗證開發端點。

   c. 建立或選擇 AWS Identity and Access Management (IAM) 角色。

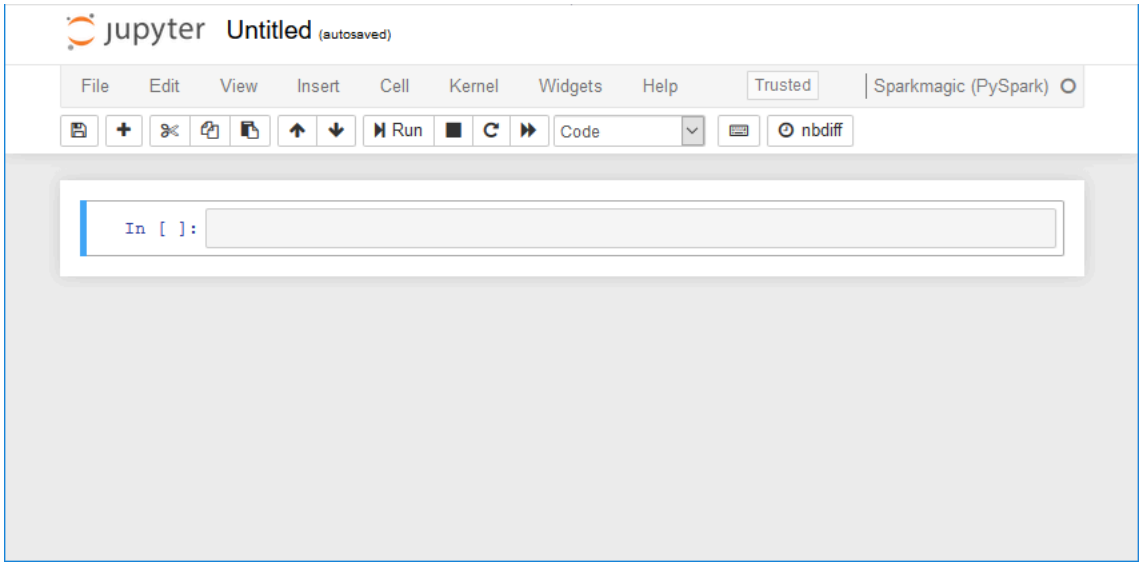
      建議您建立角色。如果您使用現有角色，請確定它具有必要的權限。如需更多詳細資訊，請參閱 the section called “步驟 6：建立適用於 SageMaker 筆記本的 IAM 政策”。

   d. (選用) 選擇 VPC、子網路以及一或多個安全群組。

   e. (選用) 選擇一個 AWS Key Management Service 加密金鑰。

   f. (選用) 為筆記本執行個體新增標籤。
4. 選擇 Create notebook (建立筆記本)。在 Notebooks (筆記本) 頁面中，選擇右上角的重新整理圖示，然後繼續操作，直到 Status (狀態) 顯示 `Ready` 為止。
5. 選取新筆記本名稱旁的核取方塊，然後選擇 Open notebook (開啟筆記本)。
6. 建立新的筆記本：在 jupyter 頁面中，選擇 New (新增)，然後選擇 Sparkmagic (PySpark)。

   您的螢幕畫面現在看起來應該與下列類似：



7. (選用) 在頁面頂端，選擇 Untitled (為命名)，然後為筆記本命名。
8. 若要啟動 Spark 應用程式，請在記事本中輸入下列指令，然後在工具列中選擇 Run (執行)。

```
spark
```

短暫的等待之後，您應可看到以下回應：

```
In [1]: spark
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1576209965005_0001	pyspark	idle	Link	Link	✓

```
SparkSession available as 'spark'.
<pyspark.sql.session.SparkSession object at 0x7f3d54913550>
```

9. 建立動態框架並針對其執行查詢：複製、貼上並執行下列程式碼，輸出 persons_json 資料表的計數和結構描述。

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import *
glueContext = GlueContext(SparkContext.getOrCreate())
persons_DyF = glueContext.create_dynamic_frame.from_catalog(database="legislators",
 table_name="persons_json")
print ("Count:  ", persons_DyF.count())
persons_DyF.printSchema()
```

教學課程：使用 REPL shell 搭配開發端點

您可在 AWS Glue 中建立開發端點，並呼叫 REPL (Read-Evaluate-Print Loop) shell，以增量執行 PySpark 程式碼，讓您在部署之前先用互動方式偵錯 ETL 指令碼。

為了在開發端點上使用 REPL，您需要取得端點 SSH 的授權。

1. 在本機電腦上開啟可執行 SSH 命令的終端視窗，並貼入編輯的 SSH 命令。執行命令。

   假設您已接受具備 Python 3 的 AWS Glue 1.0 版用於開發端點，輸出將如下所示：

```
Python 3.6.8 (default, Aug  2 2019, 17:42:44)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux
Type "help", "copyright", "credits" or "license" for more information.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/share/aws/glue/etl/jars/glue-assembly.jar!/
org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/spark/jars/slf4j-log4j12-1.7.16.jar!/
org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
 setLogLevel(newLevel).
2019-09-23 22:12:23,071 WARN  [Thread-5] yarn.Client (Logging.scala:logWarning(66))
 - Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading
 libraries under SPARK_HOME.
2019-09-23 22:12:26,562 WARN  [Thread-5] yarn.Client (Logging.scala:logWarning(66))
 - Same name resource file:/usr/lib/spark/python/lib/pyspark.zip added multiple
 times to distributed cache
2019-09-23 22:12:26,580 WARN  [Thread-5] yarn.Client (Logging.scala:logWarning(66))
 - Same path resource file:///usr/share/aws/glue/etl/python/PyGlue.zip added
 multiple times to distributed cache.
2019-09-23 22:12:26,581 WARN  [Thread-5] yarn.Client (Logging.scala:logWarning(66))
 - Same path resource file:///usr/lib/spark/python/lib/py4j-src.zip added multiple
 times to distributed cache.
2019-09-23 22:12:26,581 WARN  [Thread-5] yarn.Client (Logging.scala:logWarning(66))
 - Same path resource file:///usr/share/aws/glue/libs/pyspark.zip added multiple
 times to distributed cache.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
```

```
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.3
      /_/

Using Python version 3.6.8 (default, Aug  2 2019 17:42:44)
SparkSession available as 'spark'.
>>>
```

2. 輸入陳述式 `print(spark.version)`，測試 REPL shell 能否正常運作。只要顯示 Spark 版本，即表示 REPL 可以使用。
3. 現在您可以嘗試在 shell 中逐行執行下列簡易指令碼：

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import *
glueContext = GlueContext(SparkContext.getOrCreate())
persons_DyF = glueContext.create_dynamic_frame.from_catalog(database="legislators",
 table_name="persons_json")
print ("Count:  ", persons_DyF.count())
persons_DyF.printSchema()
```

教學課程：設定 PyCharm professional 與開發端點

本教學課程說明如何將您本機電腦上所執行的 PyCharm Professional Python IDE，連線到開發端點，以在部署 AWS Glue ETL (擷取、轉換及載入) 指令碼之前，先針對這些指令碼進行互動式的執行、除錯與測試作業。教學課程中的說明和螢幕截圖是根據 PyCharm Professional 2019.3 版。

若要以互動方式連線到開發端點，必須安裝 PyCharm Professional (專業版)。如果使用免費版，將無法執行這項動作。

> **Note**
>
> 教學課程會使用 Amazon S3 做為資料來源。如果要改為使用 JDBC 資料來源，您必須在虛擬私有雲端 (VPC) 中執行您的開發端點。若要在 VPC 中使用 SSH 連線至開發端點，您必須建立一個 SSH 通道。本教學課程不包括建立 SSH 通道的說明。如需使用 SSH 在 VPC 中連線至開發端點的資訊，請參閱 AWS 安全部落格中的安全連線置在 Private Amazon VPC 中執行的 Linux 執行個體。

主題

- 將 PyCharm professional 連線至開發端點
- 將指令碼部署到您的開發端點
- 設定遠端解譯器
- 在開發端點上執行您的指令碼

將 PyCharm professional 連線至開發端點

1. 在 PyCharm 中建立新的純 Python 專案，將其命名為 `legislators`。
2. 使用下列內容，在專案中建立名為 `get_person_schema.py` 的檔案：

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext


def main():
    # Create a Glue context
    glueContext = GlueContext(SparkContext.getOrCreate())

    # Create a DynamicFrame using the 'persons_json' table
    persons_DyF =
 glueContext.create_dynamic_frame.from_catalog(database="legislators",
 table_name="persons_json")

    # Print out information about this data
    print("Count:  ", persons_DyF.count())
    persons_DyF.printSchema()


if __name__ == "__main__":
    main()
```

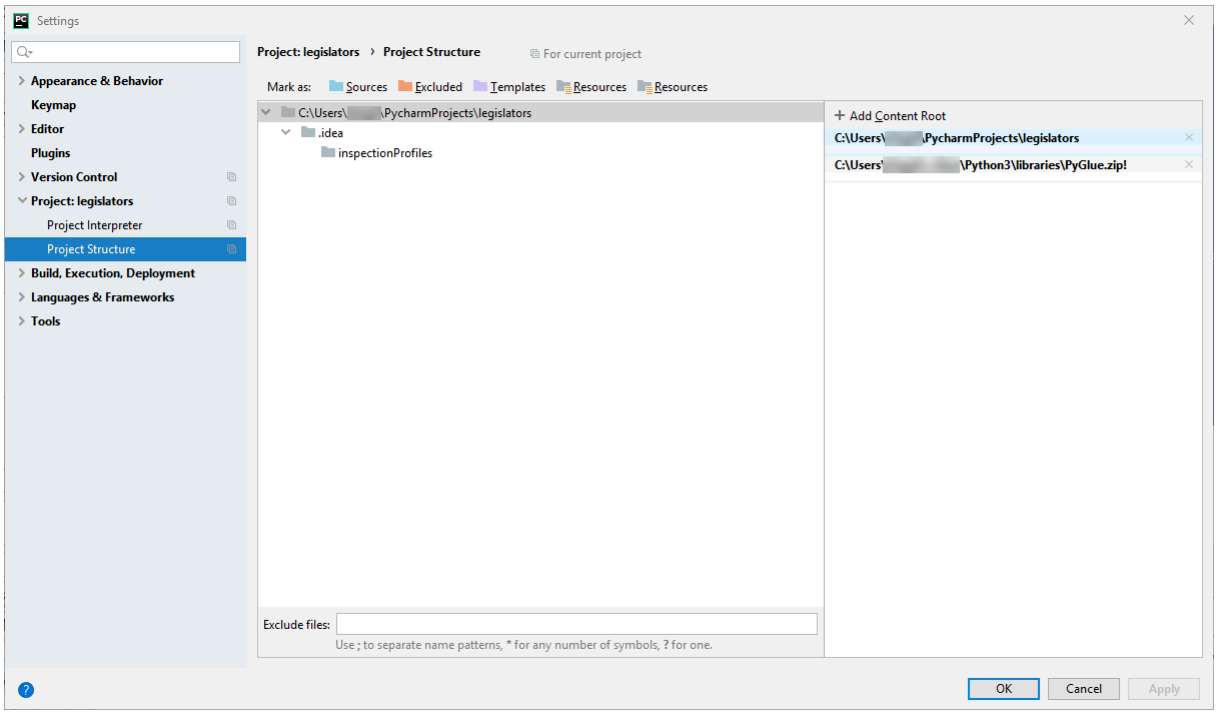
3. 執行下列任意一項：

   - 針對 AWS Glue 0.9 版，從 `https://s3.amazonaws.com/aws-glue-jes-prod-us-east-1-assets/etl/python/PyGlue.zip` 將 AWS Glue Python 程式庫檔案 `PyGlue.zip` 下載至您本機電腦上方便的位置。
   - 針對 AWS Glue 1.0 版和更新版本，從 `https://s3.amazonaws.com/aws-glue-jes-prod-us-east-1-assets/etl-1.0/python/PyGlue.zip` 將 AWS Glue Python 程式庫檔案 `PyGlue.zip` 下載至您本機電腦上方便的位置。

4. 在 PyCharm 中，新增 `PyGlue.zip` 做為您專案的內容根：

   - 在 PyCharm 中，依序選擇 File (檔案)、Settings (設定)，來開啟 Settings (設定) 對話方塊。(您也可以按 `Ctrl+Alt+S`。)
   - 展開 `legislators` 專案，然後選擇 Project Structure (專案架構)。然後在右側的窗格中，選擇 + Add Content Root (+ 新增內容根)。
   - 瀏覽至您儲存 `PyGlue.zip` 的位置，並選取此檔案，然後選擇 Apply (套用)。

   Settings (設定) 畫面的外觀應類似於下圖所示：



   在選擇 Apply (套用) 之後，讓 Settings (設定) 對話方塊保持開啟。

5. 設定部署選項，以使用 SFTP 來將本機指令碼上傳到您的開發端點 (此功能只在 PyCharm Professional 中提供)：

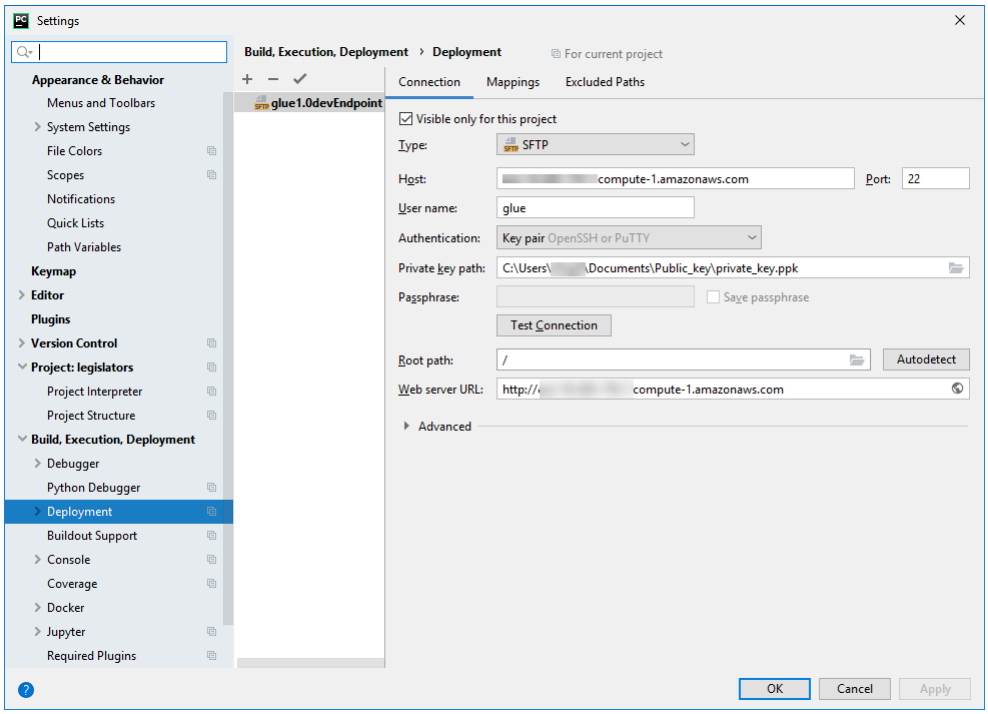
   - 在 Settings (設定) 對話方塊中，展開 Build, Execution, Deployment (建置、執行、部署) 區塊。選擇 Deployment (部署) 子區塊。
   - 選擇位於中間窗格頂端的 + 圖示，來新增伺服器。將其 Type (類型) 設為 SFTP 並命名。
   - 將 SFTP 主機設為開發端點的公用位址，如詳細資料頁面所列。(在 AWS Glue 主控台中選擇開發端點的名稱，以顯示詳細資料頁面)。如為在 VPC 中執行的開發端點，請將 SFTP 主機 設定為 SSH 通道的主機位址和開發端點的本機連接埠。
   - 將 User name (使用者名稱) 設定為 `glue`。

- 將 Auth type (驗證類型) 設定為 Key pair (OpenSSH or Putty) (金鑰對 (OpenSSH 或 Putty))。瀏覽至您開發端點私有金鑰檔案的所在位置，以設定 Private key file (私有金鑰檔案)。請注意，PyCharm 僅支援 DSA、RSA 和 ECDSA OpenSSH 金鑰類型，且不接受 Putty 私有格式的金鑰。您可以使用如下所示的語法，以最新版本的 `ssh-keygen` 來產生 PyCharm 可接受的金鑰對類型：

```
ssh-keygen -t rsa -f <key_file_name> -C "<your_email_address>"
```

- 選擇 Test connection (測試連線)，並允許連線以進行測試。如果連線成功，請選擇 Apply (套用)。

Settings (設定) 畫面的外觀現在應類似於下圖所示：

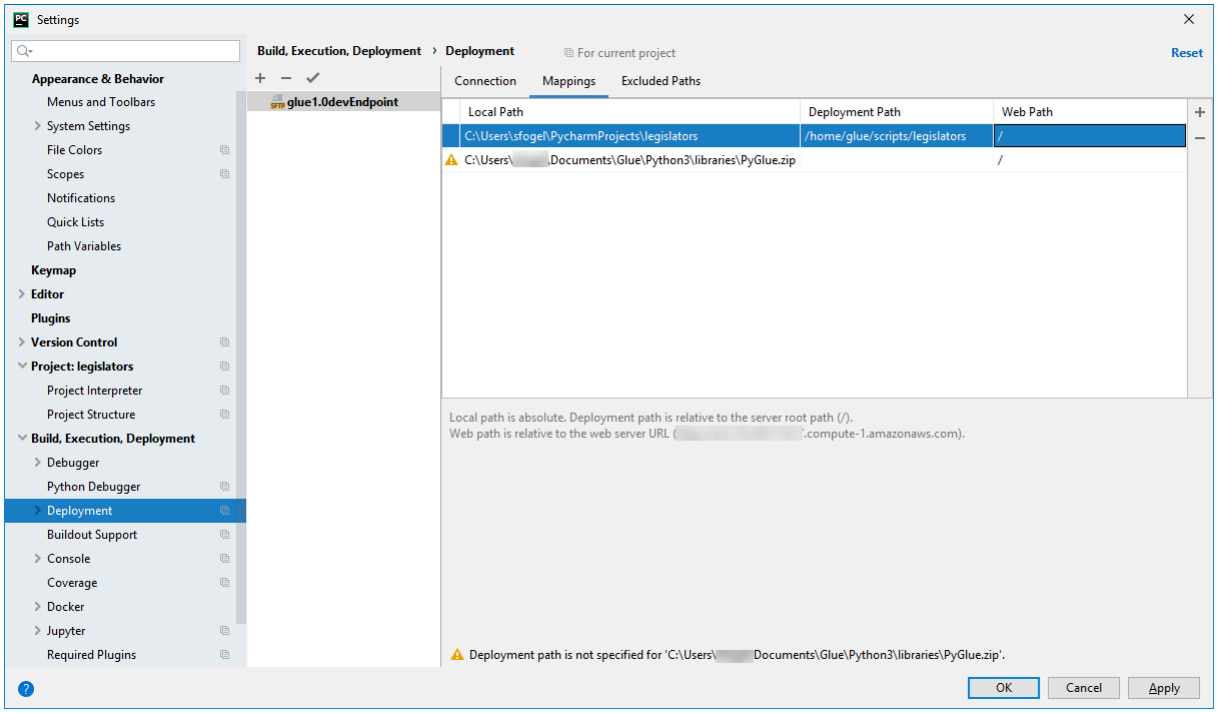


同樣地，在選擇 Apply (套用) 之後，讓 Settings (設定) 對話方塊保持開啟。

6. 將本機目錄映射至部署用的遠端目錄：

- 在 Deployment (部署) 頁面的右側窗格中，從位於頂端的索引標籤，選擇中間的 Mappings (映射) 索引標籤。
- 在 Deployment Path (部署路徑) 欄位中，輸入 `/home/glue/scripts/` 中的路徑，來做為部署您專案的路徑。例如：`/home/glue/scripts/legislators`。
- 選擇 Apply (套用)。

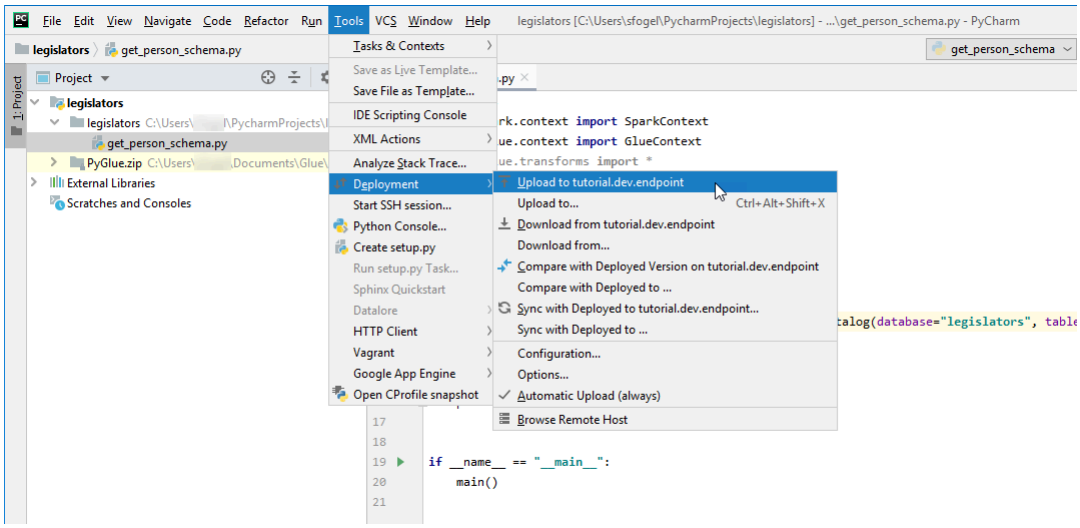
Settings (設定) 畫面的外觀現在應類似於下圖所示：



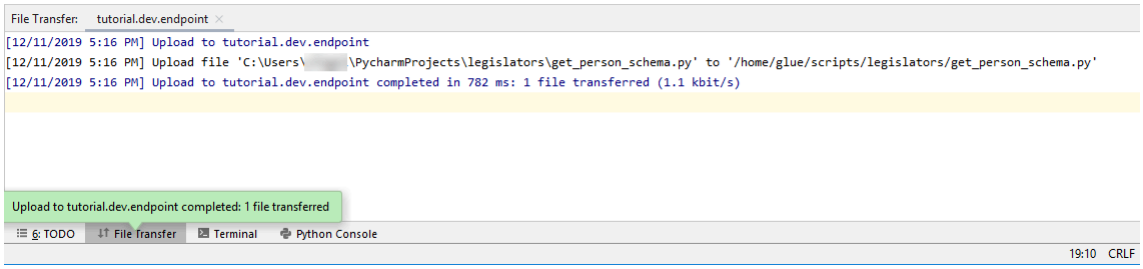
選擇 OK (確定) 以關閉 Settings (設定) 對話方塊。

將指令碼部署到您的開發端點

1. 選擇 Tools (工具)、Deployment (部署)，然後在您要設定開發端點使用的名稱，如下圖所示：



在部署指令碼之後，畫面的底部應類似於下圖所示：



2. 在功能表中，選擇 Tools (工具)、Deployment (部署)、Automatic Upload (always) (自動上傳 (永遠))。確認 Automatic Upload (always) (自動上傳 (永遠)) 旁出現核取記號。

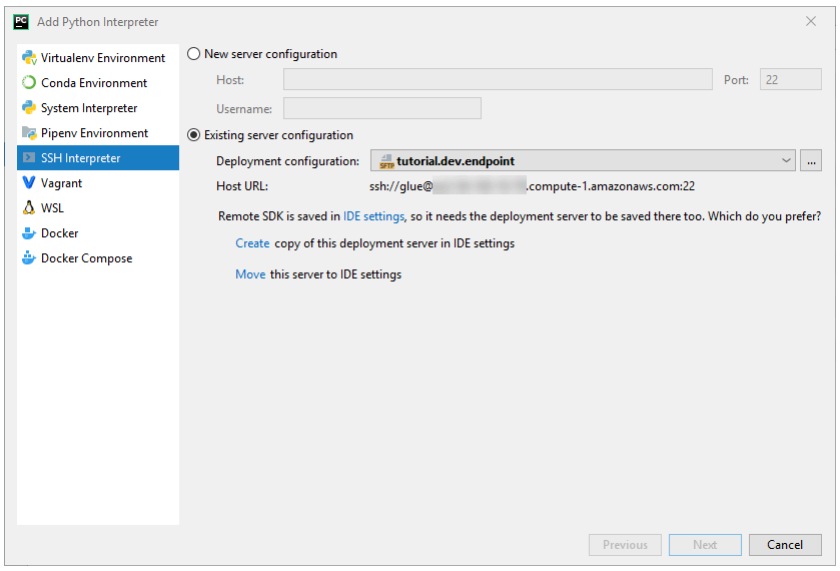
   啟用此選項時，PyCharm 會自動將變更的檔案上傳到開發端點。

設定遠端解譯器

設定 PyCharm 在開發端點上使用 Python 解釋器。

1. 在 File (檔案) 功能表中，選擇 Open (開啟)。
2. 展開專案 Legislators (國會議員)，然後選擇 Project Interpreter (專案解譯器)。
3. 選擇 Project Interpreter (專案解譯器) 清單旁的齒輪圖示，然後選擇 Add (新增)。
4. 在 Add Python Interpreter (新增 Python 解譯器) 對話方塊中，於左窗格中選擇 SSH Interpreter (SSH 解譯器)。
5. 選擇 Existing server configuration (現有伺服器組態)，然後在 Deployment configuration (部署組態) 清單中選擇您的組態。

   顯示的畫面應類似於下圖所示：



6. 選擇 Move this server to IDE settings (移動此伺服器至 IDE 設定)，然後選擇 Next (下一步)。
7. 在 Interpreter (解譯器) 欄位中，如果您使用 Python 2 則變更路徑為 `/usr/bin/gluepython`；如果您使用 Python 3 則變更路徑為 `/usr/bin/gluepython3`。然後選擇 Finish (完成)。

在開發端點上執行您的指令碼

執行指令碼：

- 在左窗格中，右鍵按一下檔案名稱，然後選擇 Run '***<filename>***' (執行 <檔案名稱>)。

  在顯示過一系列的訊息後，最終輸出應顯示計數和模式。

```
Count:  1961
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
```

```
|-- death_date: string


Process finished with exit code 0
```

設定已完成，您現在可以從遠端在您的開發端點上進行指令碼的除錯。

進階組態：在多個使用者之間共用開發端點

本節說明如何在典型使用案例中利用 SageMaker 筆記本搭配開發端點，在多個使用者之間共用開發端點。

單一租用戶組態

在單一租用戶使用案例中，為了簡化開發人員體驗並避免爭用資源，建議您讓每位開發人員針對他們正在處理的專案使用自己的開發端點大小。這也可以簡化工作者類型和 DPU 計數相關的決策，由開發人員根據他們正在處理的專案自行決定。

除非您同時執行多個筆記本檔案，否則不需要處理資源配置。如果您同時執行多個筆記本檔案中的程式碼，多個 Livy 工作階段將同時啟動。若要隔離 Spark 叢集設定，以便同時執行多個 Livy 工作階段，您可以遵循多租用戶使用案例中介紹的步驟。

例如，如果您的開發端點有 10 個工作者，並且工作者類型為 `G.1X`，那麼您將有 9 個 Spark 執行器，並且整個叢集將有 90G 的執行器記憶體，因為每個執行器都有 10G 的記憶體。

無論指定的工作者類型為何，Spark 動態資源配置都會開啟。如果資料集足夠大，Spark 可以將所有執行器配置給單個 Livy 工作階段，因為預設未設定 `spark.dynamicAllocation.maxExecutors`。這代表同一個開發端點上的其他 Livy 工作階段將等待啟動新的執行器。如果資料集很小，Spark 將能夠同時將執行器配置給多個 Livy 工作階段。

> **Note**
>
> 如需有關如何在不同的使用案例中配置資源，以及如何設定組態來修改行為的詳細資訊，請參閱 進階組態：在多個使用者之間共用開發端點。

多租用戶組態

> **Note**
>
> 請注意，開發端點旨在模擬 AWS Glue ETL 環境做為單一租用戶環境。雖然可以使用多租戶，但這是進階使用案例，仍建議大多數使用者為每個開發端點維護單一租用戶模式。

在多租用戶使用案例中，您可能需要處理資源的配置。關鍵因素是同時使用 Jupyter 筆記本的同時使用者數目。如果您的團隊在「跟隨太陽」工作流程中工作，並且每個時區只有一個 Jupyter 使用者，那麼同時使用者的數量只有一個，因此您不需要關心資源配置。但是，如果您的筆記本在多個使用者之間共用，並且每個使用者在臨機操作基礎上提交程式碼，那麼您將需要考慮以下幾點。

若要在多個使用者之間將 Spark 叢集資源分區，您可以使用 SparkMagic 組態。設定 SparkMagic 有兩個不同方式。

(A) 使用 %%configure -f 指令

如果您想從筆記本修改每個 Livy 工作階段的組態，則可以在筆記本段落執行 `%%configure -f` 指令。

例如，如果您想在 5 個執行器上執行 Spark 應用程式，則可以在筆記本段落執行下列命令。

```
%%configure -f
{"numExecutors":5}
```

然後，您將看到只有 5 個執行器在 Spark UI 上執行該任務。

我們建議限制動態資源配置的執行器的最大數量。

```
%%configure -f
{"conf":{"spark.dynamicAllocation.maxExecutors":"5"}}
```

(B) 修改 SparkMagic 組態檔

SparkMagic 是根據 Livy API 而運作。SparkMagic 使用組態建立 Livy 工作階段，如 `driverMemory`、 `driverCores`、`executorMemory`、`executorCores`、`numExecutors`、`conf` 等。這些都是決定從整個 Spark 叢集消耗多少資源的關鍵因素。SparkMagic 允許您提供組態檔來指定傳送到 Livy 的這些參數。您可以在 GitHub 儲存庫看到範例應用程式組態檔。

如果您想從筆記本中修改所有 Livy 工作階段的組態，則可以修改 /home/ec2-user/.sparkmagic/config.json 以新增 session_config。

若要修改 SageMaker 筆記本執行個體上的組態檔案，您可以依照下列步驟執行。

1. 開啟 SageMaker 筆記本。
2. 開啟終端機核心。
3. 執行下列命令：

```
sh-4.2$ cd .sparkmagic
sh-4.2$ ls
config.json logs
sh-4.2$ sudo vim config.json
```

例如，您可以將這些行新增至 /home/ec2-user/.sparkmagic/config.json，然後從筆記本重新啟動 Jupyter 核心。

```
  "session_configs": {
    "conf": {
      "spark.dynamicAllocation.maxExecutors":"5"
    }
  },
```

準則和最佳實務

為了避免這種資源衝突，您可以使用一些基本的方法，如：

- 透過提高 NumberOfWorkers (水平擴展) 以及升級 workerType (垂直擴展)，擁有更大的 Spark 叢集
- 每個使用者配置較少的資源 (每個 Livy 工作階段的資源較少)

您的方法將取決於您的使用案例。如果您有較大的開發端點，並且沒有大量的資料，則資源衝突的可能性將大幅減少，因為 Spark 可以根據動態配置策略來配置資源。

如上所述，Spark 執行器的數量可以根據 DPU (或 NumberOfWorkers) 和工作者類型自動計算。每個 Spark 應用程式會啟動一個驅動程式和多個執行器。若要計算，您將需要 NumberOfWorkers = NumberOfExecutors + 1。下面的矩陣根據同時使用者的數量，說明您在開發端點中需要多少容量。

同時筆記本使用者數量	要為每個使用者配置的 Spark 執行器的數量	開發端點的 NumberOfWorkers 總數
3	5	18
10	5	60
50	5	300

如果您想要為每個使用者配置較少的資源， `spark.dynamicAllocation.maxExecutors` (或 `numExecutors`) 將是設定為 Livy 工作階段參數的最簡單參數。如果您在 `/home/ec2-user/.sparkmagic/config.json` 設定以下組態，則 SparkMagic 會為每個 Livy 工作階段指派最多 5 個執行器。這將有助於隔離每個 Livy 工作階段的資源。

```
"session_configs": {
    "conf": {
      "spark.dynamicAllocation.maxExecutors":"5"
    }
  },
```

假設有 18 個工作者的開發端點 (G.1X)，並且同時有 3 個同時筆記本使用者。如果您的工作階段組態有 `spark.dynamicAllocation.maxExecutors=5`，那麼每個使用者可以使用 1 個驅動程式和 5 個執行器。即使您同時執行多個筆記本段落，也不會發生任何資源衝突。

取捨

使用此工作階段組態 `"spark.dynamicAllocation.maxExecutors":"5"`，您將能夠避免資源衝突錯誤，並且在存在同時使用者存取時不需要等待資源配置。但是，即使有許多免費資源 (例如，沒有其他同時使用者)，Spark 也不能為您的 Livy 工作階段指派超過 5 個執行器。

其他備註

當您停止使用筆記本時，停止 Jupyter 核心是一個很好的做法。這將釋放資源，其他筆記本使用者可以立即使用這些資源，而無需等待核心過期 (自動關機)。

常見問題

即使遵循指導方針，您可能會遇到某些問題。

找不到工作階段

當您嘗試執行筆記本段落，即使您的 Livy 工作階段已經終止，您會看到下面的訊息。若要啟用 Livy 工作階段，您需要透過選擇 Kernel (核心) > Restart (重新啟動)，然後再次執行筆記本段落。

```
An error was encountered:
Invalid status code '404' from http://localhost:8998/sessions/13 with error payload:
 "Session '13' not found."
```

YARN 資源不足

當您在 Spark 叢集沒有足夠的資源來啟動新的 Livy 工作階段時嘗試執行筆記本段落，您會看到下面的訊息。您通常可以透過遵循指導方針來避免此問題，但是，您仍可能會遇到此問題。要解決這個問題，您可以檢查是否有任何不需要的作用中 Livy 工作階段。如果有不需要的 Livy 工作階段，您將需要終止這些工作階段才能釋放叢集資源。如需詳細資訊，請參閱下節。

```
Warning: The Spark session does not have enough YARN resources to start.
The code failed because of a fatal error:
    Session 16 did not start up in 60 seconds..

Some things to try:
a) Make sure Spark has enough available resources for Jupyter to create a Spark
 context.
b) Contact your Jupyter administrator to make sure the Spark magics library is
 configured correctly.
c) Restart the kernel.
```

監控與除錯

本節說明監控資源和工作階段的技巧。

監控和除錯叢集資源配置

您可以觀看 Spark UI 來監控每個 Livy 工作階段配置了多少資源，以及任務上的有效 Spark 組態是什麼。若要啟用 Spark UI，請參閱為開發端點啟用 Apache Spark Web UI。

(選用) 如果您需要 Spark UI 的即時視圖，則可以針對 Spark 叢集上執行的 Spark 歷史記錄伺服器設定 SSH 通道。

```
ssh -i <private-key.pem> -N -L 8157:<development endpoint public address>:18080
 glue@<development endpoint public address>
```

然後您可以在瀏覽器中開啟 http://localhost:8157 以檢視 Spark UI。

釋放不需要的 Livy 工作階段

檢閱這些程序，以從筆記本或 Spark 叢集關閉任何不需要的 Livy 工作階段。

(a). 從筆記本終止 Livy 工作階段

您可以關閉 Jupyter 筆記本上的核心以終止不需要的 Livy 工作階段。

(b). 從 Spark 叢集終止 Livy 工作階段

如果不需要的 Livy 工作階段仍在執行，則可以關閉 Spark 叢集上的 Livy 工作階段。

作為執行此程序的先決條件，您需要為開發端點設定 SSH 公有金鑰。

若要登入到 Spark 叢集，您可以執行下列命令：

```
$ ssh -i <private-key.pem> glue@<development endpoint public address>
```

您可以執行下列命令來查看作用中 Livy 工作階段：

```
$ yarn application -list
20/09/25 06:22:21 INFO client.RMProxy: Connecting to ResourceManager at
 ip-255-1-106-206.ec2.internal/172.38.106.206:8032
Total number of applications (application-types: [] and states: [SUBMITTED, ACCEPTED,
 RUNNING]):2
Application-Id Application-Name Application-Type User Queue State Final-State Progress
 Tracking-URL
application_1601003432160_0005 livy-session-4 SPARK livy default RUNNING UNDEFINED 10%
 http://ip-255-1-4-130.ec2.internal:41867
application_1601003432160_0004 livy-session-3 SPARK livy default RUNNING UNDEFINED 10%
 http://ip-255-1-179-185.ec2.internal:33727
```

然後，您可以使用以下命令關閉 Livy 工作階段：

```
$ yarn application -kill application_1601003432160_0005
20/09/25 06:23:38 INFO client.RMProxy: Connecting to ResourceManager at
 ip-255-1-106-206.ec2.internal/255.1.106.206:8032
Killing application application_1601003432160_0005
20/09/25 06:23:39 INFO impl.YarnClientImpl: Killed application
 application_1601003432160_0005
```

管理筆記本

> **Note**
>
> 開發端點僅支援 2.0 以前的 AWS Glue 版本。針對可以撰寫和測試 ETL 指令碼的互動環境，請參閱 Notebooks on AWS Glue Studio。

筆記本可讓您在開發端點上進行互動開發並測試 ETL (擷取、轉換和載入) 指令碼。AWS Glue 提供 SageMaker Jupyter 筆記本的介面。您可以使用 AWS Glue 來建立和管理 SageMaker 筆記本。您也可以從 AWS Glue 主控台開啟 SageMaker 筆記本。

此外，您可以在支援 SageMaker 的 AWS Glue 開發端點上搭配使用 Apache Spark 與 SageMaker (而非 AWS Glue ETL 任務)。SageMaker Spark 是一種適用於 SageMaker 的開源 Apache Spark 程式庫。如需詳細資訊，請參閱 Using Apache Spark with Amazon SageMaker (搭配 Apache Spark 使用 Amazon SageMaker)。

> **Important**
>
> 您可以在下列 AWS 區域中使用 AWS Glue 開發端點來管理 SageMaker 筆記本：

區域	代碼
美國東部 (俄亥俄)	`us-east-2`
美國東部 (維吉尼亞北部)	`us-east-1`
美國西部 (加利佛尼亞北部)	`us-west-1`
美國西部 (奧勒岡)	`us-west-2`
亞太區域 (東京)	`ap-northeast-1`
亞太區域 (首爾)	`ap-northeast-2`
亞太區域 (孟買)	`ap-south-1`
亞太區域 (新加坡)	`ap-southeast-1`

區域	代碼
亞太區域 (雪梨)	ap-southeast-2
加拿大 (中部)	ca-central-1
歐洲 (法蘭克福)	eu-central-1
歐洲 (愛爾蘭)	eu-west-1
歐洲 (倫敦)	eu-west-2

使用 AWS Glue Studio 建立視覺化 ETL 任務

AWS Glue 任務封裝了一個指令碼，會連線至您的來源資料、處理資料，然後將它寫出至您的資料目標。一般而言，任務會執行擷取、轉換和載入 (ETL) 指令碼。任務可以執行專為 Apache Spark 和 Ray 執行期環境設計的指令碼。任務也可以執行一般用途的 Python 指令碼 (Python Shell 任務)。AWS Glue 觸發程序可以根據排程或事件，或隨需啟動任務。您可以監控任務執行以了解執行時間指標，例如完成狀態、持續時間和開始時間。

您可以使用 AWS Glue 產生的指令碼，也可以提供自己的指令碼。透過原始碼結構描述和目標位置或結構描述，程式AWS Glue Studio碼產生器可以自動建立 Apache Spark API (PySpark) 指令碼。您可以將此指令碼做為起點，編輯其內容以符合您的目標。

AWS Glue 可以寫入多種資料格式的輸出檔案。每種任務類型可支援不同的輸出格式。某些資料格式也可寫入常見的壓縮格式。

登入 AWS Glue 主控台

中的工作 AWS Glue 包含執行擷取、轉換和載入 (ETL) 工作的商務邏輯。您可以在 AWS Glue 主控台的 ETL 區塊中建立工作。

若要檢視現有工作，請登入 AWS Management Console 並開啟AWS Glue主控台，位於 https://console.aws.amazon.com/glue/。接著，請在 AWS Glue 中選擇 Jobs (工作) 索引標籤。Jobs (任務) 清單會顯示與每項任務相關指令碼的位置、任務最近修改的時間，以及目前的任務書籤選項。

建立新任務時，或儲存任務後，您可以使用 AWS Glue Studio 修改您的 ETL 任務。您可以在視覺化編輯器中編輯節點，或在開發人員模式中編輯任務指令碼來執行此動作。您也可以在視覺化編輯器中新增和移除節點，以建立更複雜的 ETL 任務。

在 AWS Glue Studio 中建立任務的後續步驟

您可以使用視覺化任務編輯器來設定任務的節點。每個節點代表一個動作，例如從源位置讀取資料或應用轉換到資料。您新增至任務的每個節點都具有提供資料位置或轉換相關資訊的屬性。

建立和管理任務的後續步驟如下：

- 使用 AWS Glue Studio 視覺化 ETL
- 檢視任務指令碼

- 修改任務屬性
- 儲存任務
- 開始任務執行
- 檢視最近任務執行的資訊
- 存取任務監控儀表板

使用 AWS Glue Studio 視覺化 ETL

您可以使用 AWS Glue Studio 中簡單的視覺介面來建立您的 ETL 任務。您使用 Jobs (任務) 頁面以建立新任務。您也可以使用指令碼編輯器或筆記本直接使用 AWS Glue Studio ETL 任務指令碼。

在 Jobs (任務) 頁面上，您可以查看您使用 AWS Glue Studio 或 AWS Glue 建立的所有任務。您可以在此頁面上檢視、管理和執行您的任務。

另請參閱部落格教學課程，了解關於如何使用 AWS Glue Studio 建立 ETL 任務的另一個範例。

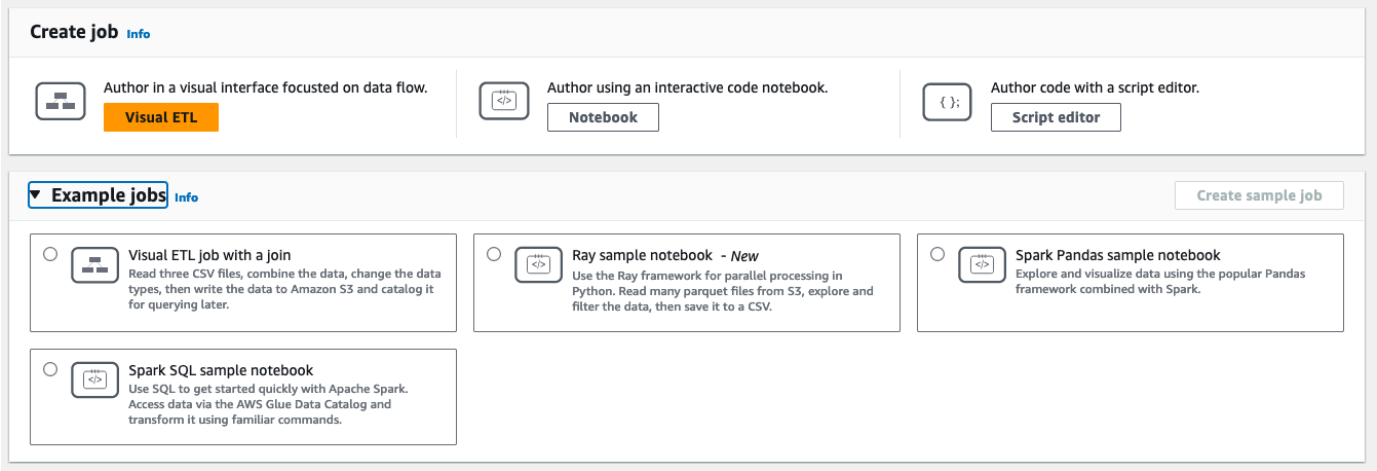
在 AWS Glue Studio 中啟動任務

AWS Glue 可讓您透過視覺化介面、互動式程式碼筆記本或指令碼編輯器來建立任務。您可以按一下任何選項來啟動任務，或根據範例任務建立新任務。

範例任務會使用您選擇的工具建立任務。例如，範例任務可讓您建立視覺化 ETL 任務以將 CSV 檔案聯結至型錄資料表，或是在使用 pandas 時，於互動式程式碼筆記本中使用 AWS Glue for Ray 或 AWS Glue for Spark 建立任務，或使用 SparkSQL 在互動式程式碼筆記本中建立任務。

AWS Glue Studio從頭開始建立工作

1. 請登入 AWS Management Console 並開啟AWS Glue Studio主控台，網址為 https://console.aws.amazon.com/gluestudio/。
2. 從導覽窗格中選擇 ETL 任務。
3. 在建立任務區段中，選取任務的組態選項。



用於從頭開始建立任務的選項：

- 視覺化 ETL：在專注於資料流程的視覺化介面中進行撰寫
- 使用互動式程式碼筆記本進行撰寫：在以 Jupyter 筆記本為基礎的筆記本介面中以互動方式撰寫任務

  選取此選項時，您必須在建立筆記本撰寫工作階段之前提供其他資訊。如需如何指定此資訊的詳細資訊，請參閱 AWS Glue Studio 中的筆記本入門。
- 使用指令碼編輯器撰寫程式碼：對於熟悉 ETL 指令碼程式設計和撰寫的人，選擇此選項來建立新的 Spark ETL 任務。選擇引擎 (Python shell、Ray、Spark (Python) 或 Spark (Scala))。然後，選擇重新開始或上傳指令碼，從本機檔案上傳現有的指令碼。如果您選擇使用指令碼編輯器，則無法使用視覺化任務編輯器設計或編輯任務。

  Spark 任務在由 AWS Glue 管理的 Apache Spark 環境中執行。預設情況下，新指令碼以 Python 編碼。若要編寫新的 Scala 指令碼，請參閱在 AWS Glue Studio 中建立和編輯 Scala 指令碼。

從範例工AWS Glue Studio作建立工作

您可以選擇從範例任務建立任務。在範例任務區段中，選擇範例任務，然後選擇建立範例任務。從其中一個選項建立範例任務會提供您可以使用的快速範本。

1. 請登入 AWS Management Console 並開啟AWS Glue Studio主控台，網址為 https://console.aws.amazon.com/gluestudio/。
2. 從導覽窗格中選擇 ETL 任務。
3. 選取用於從範例任務建立任務的選項：

- 聯結多個來源的視覺化 ETL 任務：讀取三個 CSV 檔案、合併資料、變更資料類型，然後將資料寫入 Amazon S3 並對其進行編目以供日後查詢。
- 使用 Pandas 的 Spark 筆記本：使用與 Spark 相結合的流行 Pandas 架構，探索和視覺化資料。
- 使用 SQL 的 Spark 筆記本：透過 SQL 快速開始使用 Apache Spark。透過 AWS Glue Data Catalog 存取資料，並使用熟悉的命令轉換資料。

4. 選擇建立範例任務。

任務編輯器功能

任務編輯器提供下列功能，用以建立和編輯任務。

- 任務的視覺化圖表，每個任務任務都有一個節點：用於讀取資料的資料來源節點；用於修改資料的轉換節點；用於寫入資料的資料目標節點。

  您可以在任務圖表中檢視和設定每個節點的屬性。您也可以檢視任務圖表中每個節點的結構描述和範例資料。這些功能可協助您驗證任務是否正在以正確的方式修改和轉換資料，而不必執行任務。
- [Script viewing and editing (指令碼檢視和編輯)] 索引標籤，您可以在其中修改為任務產生的程式碼。
- [Job details (任務詳細資訊)] 索引標籤，您可以在其中設定各種設定，以自訂 AWS Glue ETL 任務執行的環境。
- 「[Runs (執行)] 索引標籤，您可以在此檢視任務的目前和先前執行、檢視任務執行的狀態，以及存取任務執行的記錄。
- 「資料品質」標籤，您可在其中將資料品質規則套用至任務。
- [Schedules (排程)] 索引標籤，您可以在其中設定任務的開始時間，或設定週期性任務執行。
- 「版本控制」標籤，您可在其中設定 Git 服務，以搭配任務使用。

在視覺化任務編輯器中使用結構描述預覽

建立或編輯任務時，您可以使用 Output schema (輸出結構描述) 索引標籤來檢視您資料的結構描述。

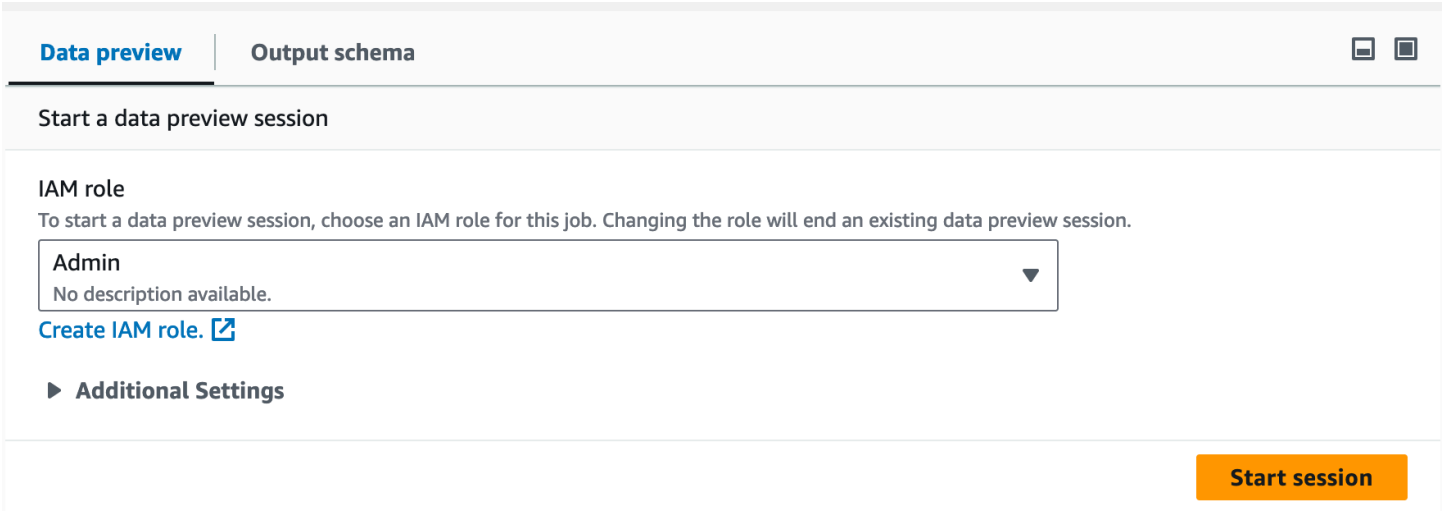
在查看結構描述之前，任務編輯器需要存取資料來源的許可。您可以在編輯器的 [Job details (任務詳細資訊)] 索引標籤或節點的 Output schema (輸出結構描述) 索引標籤中指定 IAM 角色。如果 IAM 角色具有存取資料來源的所有必要許可，則您可以在節點的 Output schema (輸出結構描述) 索引標籤檢視結構描述。

在視覺化任務編輯器中使用資料預覽

資料預覽可協助您使用資料範例建立和測試任務，而不必重複執行任務。透過使用資料預覽，您可以：

- 測試 IAM 角色，以確保您可以存取資料來源或資料目標。
- 檢查轉換是否以預期的方式修改資料。例如，如果您使用篩選器轉換，您可以確定篩選器選取正確的資料子集。
- 檢查資料。如果您的資料集包含具有多種類型值的欄，則資料預覽會顯示這些欄的元組清單。每個元組會包含資料類型及其值。

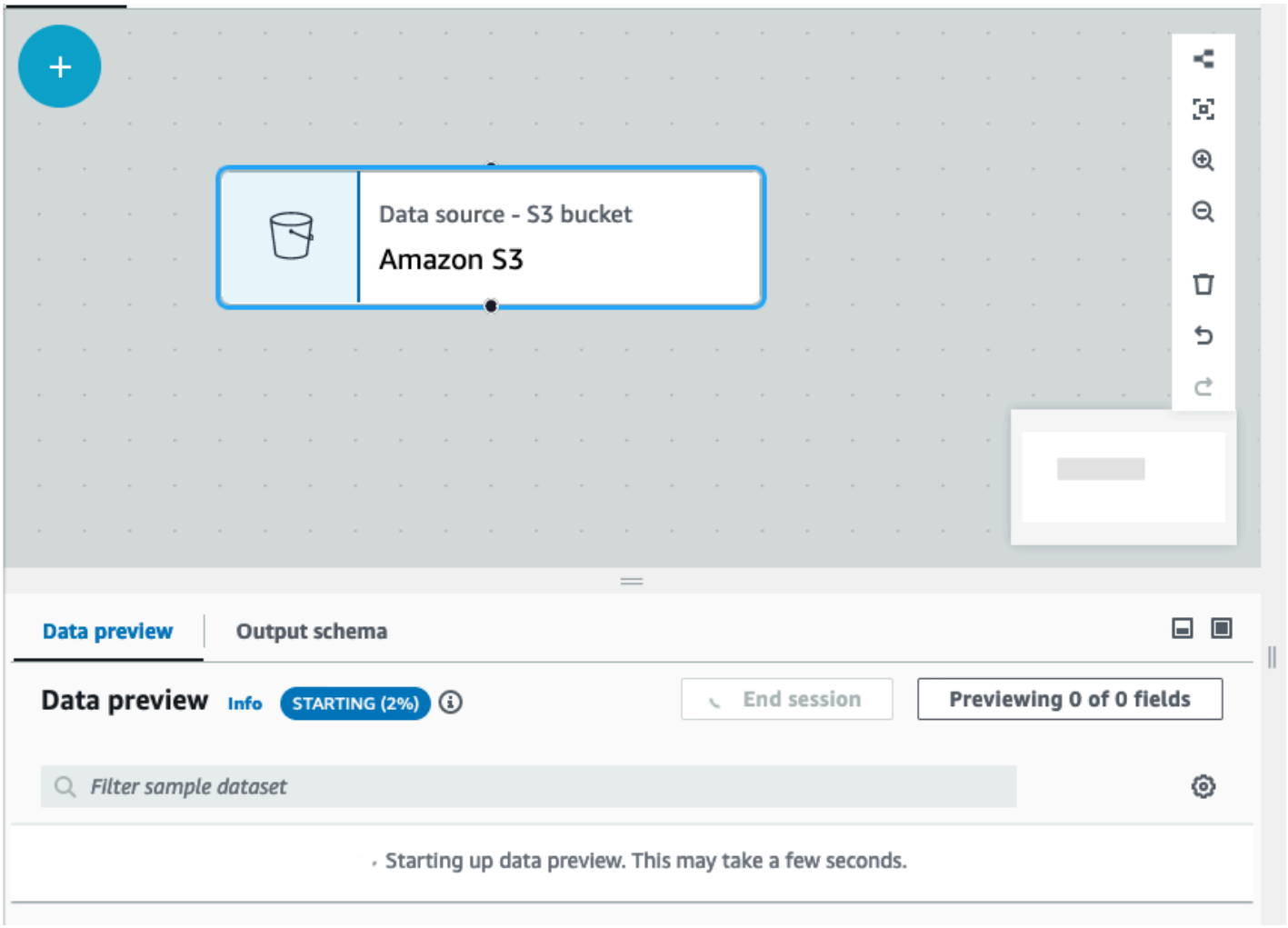
建立或編輯任務時，您可以使用任務畫布下方的資料預覽標籤來檢視您的資料範例。當任務上已設定角色或帳戶中已設定預設 IAM 角色時，新的資料預覽任務階段將自動啟動。如果先前尚未設定角色，您可以透過選取角色來啟動工作階段。



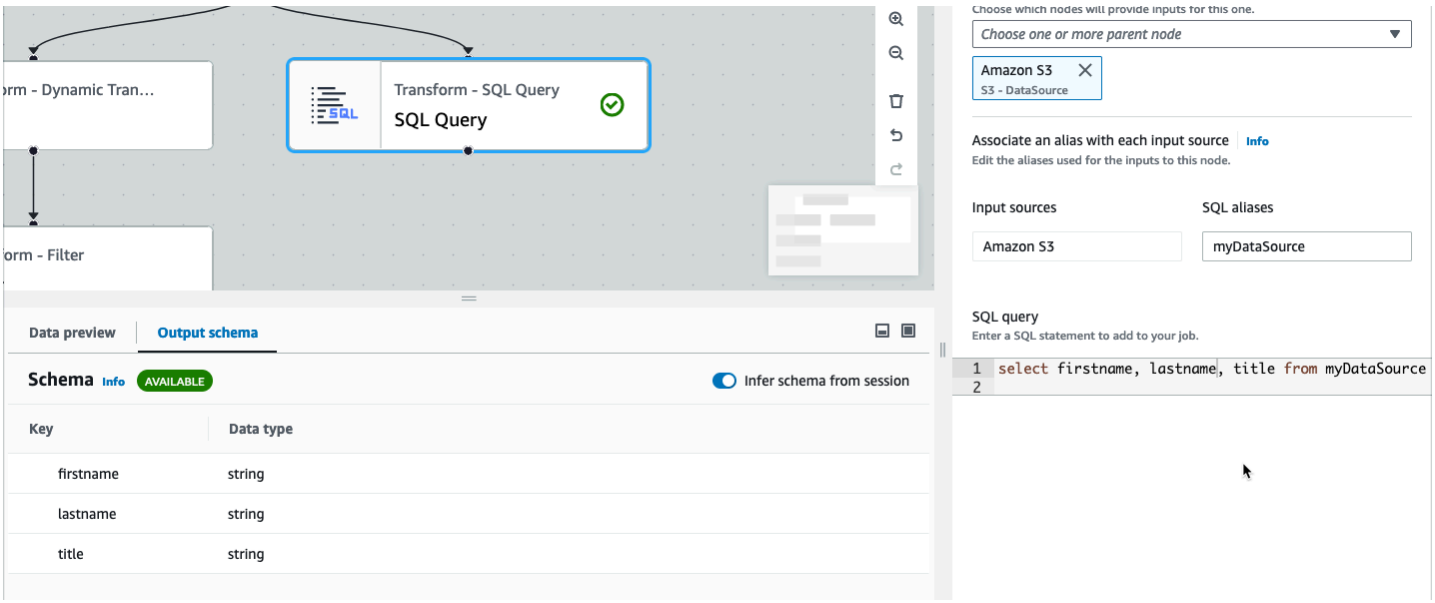
> **Note**
>
> 您針對資料預覽任務階段所選擇的角色亦將用於任務。

您可以按一下資訊圖示，查看工作階段的狀態和進度，以及工作階段詳細資料。

當工作階段準備就緒時，AWS Glue Studio 將會載入所選節點的資料。您可以在進行時檢視完成百分比。



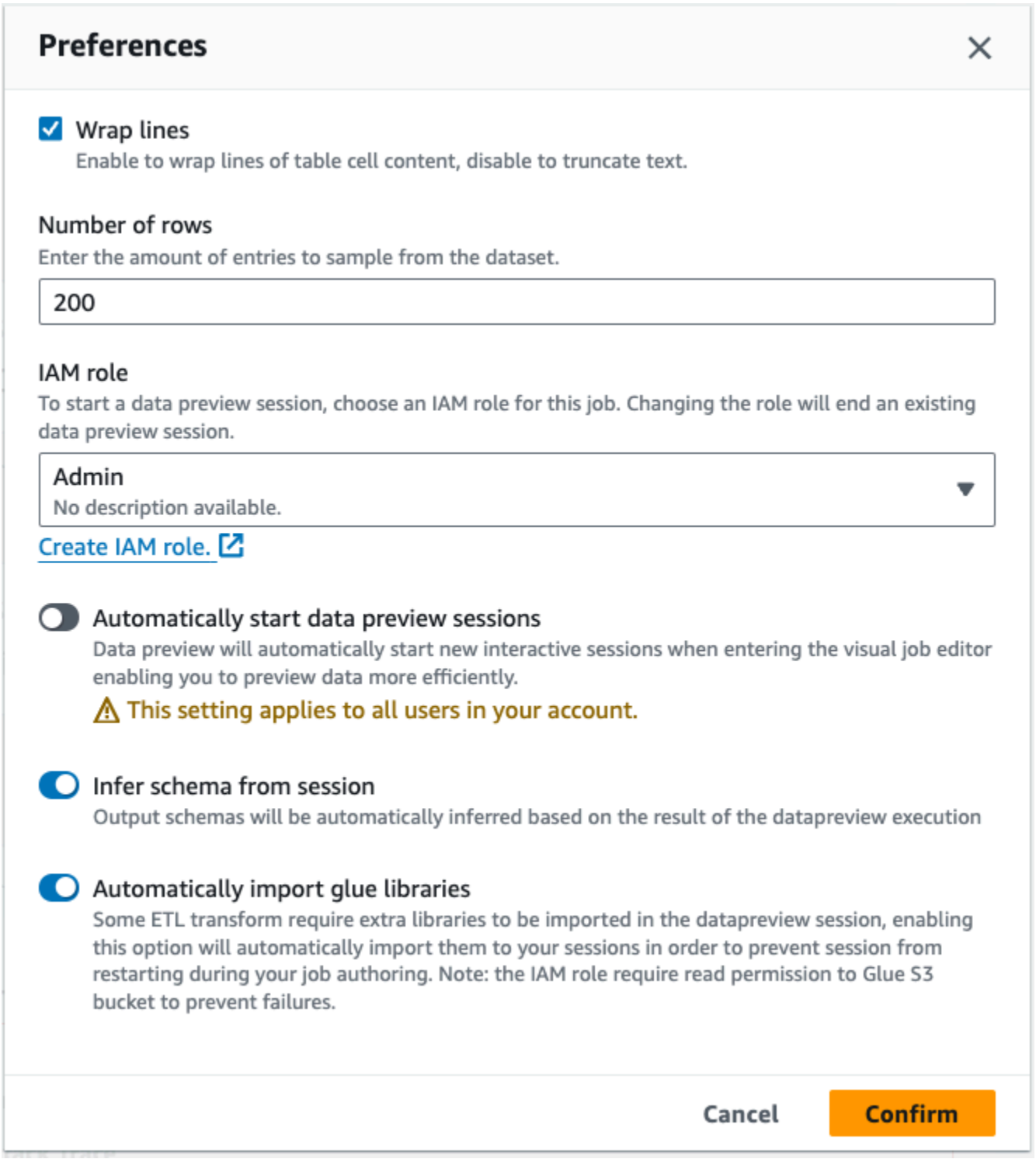
當您撰寫視覺化任務時，AWS Glue Studio 將會在您切換輸出結構描述索引標籤中的推論任務階段的結構描述時，自動更新選取之節點的結構描述。



設定資料預覽偏好設定：

選擇設定圖示 (齒輪符號)，以設定資料預覽的偏好設定。這些設定適用於任務圖表中的所有節點。您可以：

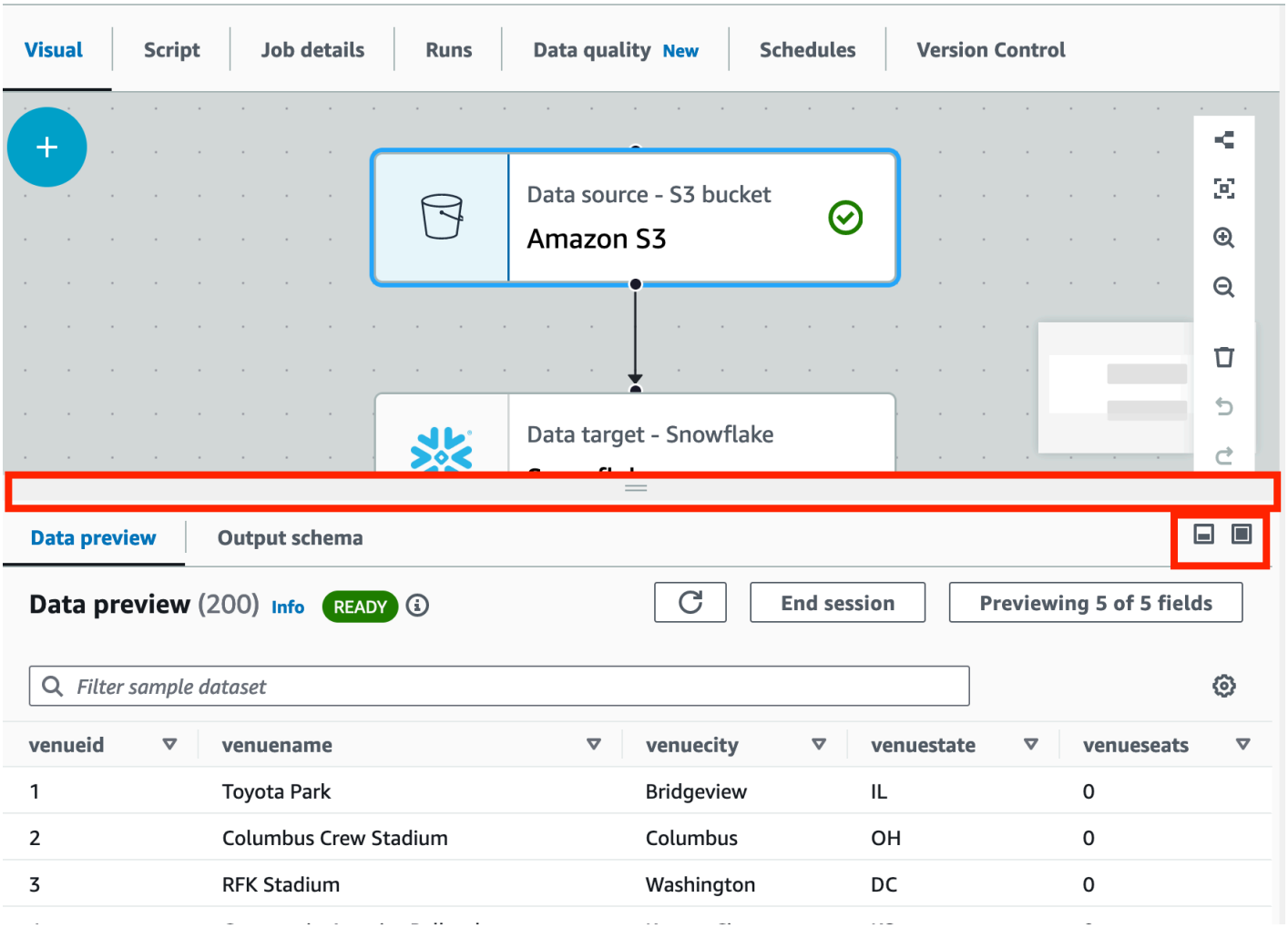
- 選擇將文字從一行換至下一行。此選項預設為啟用。
- 變更列數 (預設為 200)
- 選擇 IAM 角色或視需要建立 IAM 角色
- 選擇在您撰寫任務時自動開始新的任務階段。這會在撰寫任務時佈建新的互動式工作階段。此設定會在帳戶層級套用。完成設定後，便會在編輯任何任務時套用至您帳戶中的所有使用者。
- 選擇自動推論結構描述。系統會針對選取的節點自動推論輸出結構描述
- 選擇自動匯入 AWS Glue 程式庫。此功能相當實用，可在新增需要工作階段重新啟動的轉換時，防止資料預覽重新啟動新的工作階段



其他功能包括：

- 選擇 Previewing x of y fields (預覽 y 欄位中的 x) 按鈕以選取要預覽的欄 (欄位)。當您使用預設設定來預覽資料時，任務編輯器會顯示資料集的前 5 欄。您可以將此變更為全部顯示或全不顯示 (不建議使用)。

- 水平和垂直捲動資料預覽視窗。
- 使用最大化按鈕，將「資料預覽」標籤展開至覆蓋任務圖表，以便進一步檢視資料和資料結構。同樣地，請使用最小化按鈕將「資料預覽」標籤最小化。您也可以抓取控點窗格並向上拖曳，以展開資料預覽標籤。



- 使用結束工作階段來停止資料預覽。在您停止工作階段時，可以選擇新的 IAM 角色，並設定其他設定 (例如開啟或關閉設定以自動啟動新工作階段、推斷結構描述，或匯入 AWS Glue 程式庫)，然後再次啟動工作階段。

使用資料預覽時的限制

當您使用資料預覽時，可能會遇到下列限制。

- 第一次選擇 [Data preview (資料預覽)] 索引標籤時，您必須選擇 IAM 角色。此角色必須具有必要的許可，才能存取建立資料預覽所需的資料和其他資源。

- 提供 IAM 角色之後，需要一段時間才能檢視資料。對於資料少於 1 GB 的資料集，最多可能需要一分鐘的時間。如果您有大型資料集，您應該使用分割區來改善載入時間。直接從 Amazon S3 載入資料具有最佳效能。
- 如果您有非常大的資料集，而且查詢用於資料預覽的資料需要超過 15 分鐘，請求將會逾時。資料預覽有 30 分鐘的閒置逾時時間。若要減少此情況，請縮減資料集大小以使用資料預覽。
- 依預設，您會在「資料預覽」標籤中看到前 50 個資料欄。如果欄沒有資料值，您會收到一則訊息，指出沒有資料可顯示。您可以增加取樣的列數，或選取不同的欄以查看資料值。
- 資料預覽目前不支援串流資料來源或使用自訂連接器的資料來源。
- 一個節點上的錯誤會影響整個任務。如果任何一個節點在資料預覽中發生錯誤，則錯誤將會顯示在所有節點上，直到您修正為止。
- 如果您變更任務的資料來源，則可能需要更新該資料來源的子節點以符合新的結構描述。例如，如果您有可修改欄的 ApplyMapping 節點，而該欄不存在於取代資料來源中，則需要更新 ApplyMapping 轉換節點。
- 如果您檢視 SQL 查詢轉換節點的 [Data preview (資料預覽)] 索引標籤，且 SQL 查詢使用不正確的欄位名稱，則 [Data preview (資料預覽)] 索引標籤會顯示錯誤。

指令碼程式碼產生

使用視覺化編輯器建立任務時，會自動為您產生 ETL 程式碼。AWS Glue Studio 會建立功能完整的任務指令碼，並將其儲存在 Amazon S3 位置。

AWS Glue Studio 產生兩種形式的程式碼：原始版本或經典版本，以及更新的簡化版本。依預設，使用新的程式碼產生器建立任務指令碼。您可以使用 Script (指令碼) 索引標籤上的經典程式碼產生器產生任務指令碼，方法是選擇 Generate classic script (產生經典指令碼) 切換按鈕。

在新版本的產生程式碼中，一些差異包括：

- 大型註解區塊不再新增至指令碼
- 程式碼中的輸出結構會使用您在視覺化編輯器中指定的節點名稱。
  在類別指令碼中，輸出結構只是命名為 `DataSource0`、`DataSource1`、`Transform0`、`Transform1`、`DataSink0`、`DataSink1`，以此類推。
- 長命令會分割成多行，以避免為檢視完整命令而捲動頁面。

AWS Glue Studio 中的新功能需要新版本的程式碼產生，並且不適用於經典程式碼指令碼。當您嘗試執行這些任務時，系統會提示您更新任務。

編輯 AWS Glue 受管資料轉換節點

AWS Glue Studio 會提供兩種類型的轉換：

- AWS Glue 原生轉換 – 適用於所有使用者，並由 AWS Glue 管理。
- 自訂視覺化轉換 – 允許您上傳自己的轉換以在 AWS Glue Studio 中使用

AWS Glue 受管資料轉換節點

AWS Glue Studio 提供一組內建轉換，讓您用於處理資料。您的資料從任務圖表中的一個節點傳遞到稱為 DynamicFrame 的資料結構中的另一個節點，這是 Apache Spark SQL DataFrame 的擴展。

在任務的預先填入圖表中，資料來源和資料目標節點之間是變更結構描述轉換節點。您可以設定此轉換節點來修改資料，也可以使用其他轉換。

AWS Glue Studio 提供以下內建轉換：

- ChangeSchema：將資料來源中的資料屬性索引鍵映射至資料目標中的資料屬性索引鍵。您可以重新命名索引鍵、修改索引鍵的資料類型，以及選擇要從資料集中捨棄哪些索引鍵。
- SelectFields：選擇您要保留的資料屬性索引鍵。
- DropFields：選擇您要捨棄的資料屬性索引鍵。
- RenameField：重新命名單一資料屬性索引鍵。
- Spigot：將資料範例寫入 Amazon S3 儲存貯體。
- Join：使用指定資料屬性索引鍵上的比較片語，將兩個資料集聯結為一個資料集。可以使用內、外、左、右、左半、左反聯結。
- 聯集：合併多個具有相同結構描述之資料來源的資料列。
- SplitFields：將資料屬性索引鍵分成兩個 DynamicFrames。輸出是 DynamicFrames 的集合：一個具有所選資料屬性索引鍵，另一個具有其餘資料屬性索引鍵。
- SelectFromCollection：請從 DynamicFrames 集合選擇一個 DynamicFrame。輸出為所選的 DynamicFrame。
- FillMissingValues：尋找遺失值之資料集中的記錄，並新增具有由插補決定建議值的新欄位。
- Filter (篩選條件)：根據篩選條件，將資料集分割成兩個。
- 刪除 null 欄位：如果資料行中的所有值都為 "null"，則從資料集中刪除此行。
- 刪除重複項：選擇符合整個資料列或指定索引鍵，從資料來源中移除資料列。

- SQL：在文字輸入欄位中輸入 SparkSQL 程式碼，以使用 SQL 查詢來轉換資料。輸出是一個單一 DynamicFrame。
- 彙總：在選定的欄位和列上執行計算 (例如平均值、總和、最小值、最大值)，並建立一個新欄位來包含新計算的值。
- 壓平合併：將結構內的欄位擷取到頂層欄位。
- UUID：針對每個資料列新增具有通用不重複識別碼的資料欄。
- 識別符：針對每個資料列新增含有數字識別符的資料欄。
- 時間戳記：將資料欄轉換為時間戳記類型。
- 格式化時間戳記：將時間戳記資料欄轉換為格式化字串。
- 條件式路由器轉換：對傳入資料套用多個條件。傳入資料的每個資料列均依據群組篩選條件進行評估，並將其處理到其對應的群組。
- 串連資料欄轉換：使用具有選用間隔符號之其他資料欄的值來構建新的字串資料欄。
- 分割字串轉換：使用規則運算式將字串分解為字符陣列，以定義分割方式。
- 陣列至資料欄轉換：將陣列類型之資料欄的部分或全部元素擷取到新資料欄中。
- 新增目前時間戳記轉換：以處理資料的時間來標記資料列。這對於稽核目的或追蹤資料管道中的延遲非常實用。
- 樞紐資料列至資料欄轉換：透過旋轉成為新資料欄的所選資料欄上的唯一值來彙總數值。如果選取多資料欄，則會串連這些值以命名新資料欄。
- 取消樞紐資料欄至資料列：將資料欄轉換為新資料欄的值，並為每個唯一值產生一個資料列。
- 自動平衡處理轉換：在工作者之間更好地重新分發資料。這對資料不平衡或其來源不允許進行足夠平行處理的情況非常實用。
- 衍生資料欄轉換：根據數學公式或 SQL 運算式定義新資料欄，您可以在其中使用資料中的其他資料欄，以及常數和常值。
- 查詢轉換：當索引鍵符合資料中定義的查詢資料欄時，從已定義的型錄資料表新增資料欄。
- 分解陣列或映射成資料列轉換：將巢狀結構中的值擷取到更容易操作的個別資料列中。
- 記錄比對轉換：調用現有的記錄比對機器學習資料分類轉換。
- 移除 Null 資料列轉換：從所有資料欄為 Null 或空白的資料集資料列移除。
- 解析 JSON 資料欄轉換：解析包含 JSON 資料的字串資料欄，並將其轉換為結構或陣列資料欄，具體取決於 JSON 是物件還是陣列。
- 擷取 JSON 路徑轉換：從 JSON 字串資料欄擷取新資料欄。

- 從規則運算式中擷取字串片段：使用規則運算式擷取字串片段，並從中建立新資料欄，或在使用規則運算式組時建立多個資料欄。
- Custom transform (自訂轉換)：在文字輸入欄位中輸入程式碼，以使用自訂轉換。輸出是 `DynamicFrames` 的集合。

在 AWS Glue Studio 中使用資料準備配方

AWS Glue Studio 可讓您在視覺化工作流程中使用 AWS Glue DataBrew 配方。這允許客戶的 AWS Glue DataBrew 配方與其他 AWS Glue Studio 節點一起在 AWS Glue 任務中執行。

在 DataBrew 中，配方是指一組資料轉換步驟。Databrew 配方規定如何轉換已經讀取的資料，而且不會描述在何處以及如何讀取資料，以及如何和在何處寫入資料。此項在 AWS Glue Studio 的來源和目標節點中設定。如需有關配方的詳細資訊，請參閱建立和使用 AWS Glue DataBrew 配方。

您可以從「資源」面板取得資料準備配方節點。您可以將資料準備配方節點連線至視覺化工作流程中的另一個節點，無論它是資料來源節點還是其他轉換節點。選擇 AWS Glue DataBrew 配方和版本之後，配方中套用的步驟就會顯示在節點屬性索引標籤中。

先決條件

- 您已在 AWS Glue DataBrew 中建立 AWS Glue DataBrew 配方。
- 您擁有如下所述的所需 IAM 許可。

適用於 AWS Glue DataBrew 的 IAM 許可

本主題提供的資訊將協助了解身為 IAM 管理員的您可在資料準備配方轉換的 AWS Identity and Access Management (IAM) 政策中使用的動作和資源。

如需 AWS Glue 中有關安全的額外資訊，請參閱 Access Management。

下表列出使用者執行特定操作以使用「資料準備配方」轉換所需的許可。

「資料準備配方」轉換動作

動作	描述
`databrew:ListRecipes`	授予擷取 AWS Glue DataBrew 配方的許可。
`databrew:ListRecipeVersions`	授予擷取 AWS Glue DataBrew 配方版本的許可。

動作	描述
databrew:DescribeRecipe	授予擷取 AWS Glue DataBrew 配方描述的許可。

您用來存取此功能的角色應具有允許多個 AWS Glue DataBrew 的政策。您可以使用包含必要動作的 AWSGlueConsoleFullAccess 政策，或將下列內嵌政策新增至您的角色，以達成此目的：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "databrew:ListRecipes",
                "databrew:ListRecipeVersions",
                "databrew:DescribeRecipe"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

若要使用「資料準備配方」轉換，您必須將 IAM:PassRole 動作新增至許可政策。

其他必要許可

動作	描述
iam:PassRole	授予 IAM 許可，以允許使用者傳遞核准的角色。

如果沒有這些許可，會發生下列錯誤：

```
"errorCode": "AccessDenied"
```

```
"errorMessage": "User: arn:aws:sts::account_id:assumed-role/AWSGlueServiceRole is not
authorized to perform: iam:PassRole on resource: arn:aws:iam::account_id:role/service-
role/AWSGlueServiceRole
because no identity-based policy allows the iam:PassRole action"
```

限制

- 並非所有 AWS Glue DataBrew 配方都受到 AWS Glue 支援。某些配方無法在 AWS Glue Studio 中執行。
  - 不支援具有 UNION 和 JOIN 轉換的配方，但是，AWS Glue Studio 已經具有「聯結」和「聯集」轉換節點，可用於「資料準備配方」節點之前或之後。
- 從 AWS Glue 4.0 版開始的任務支援資料準備配方節點。將資料準備配方節點新增至任務後，系統會自動選取此版本。
- 資料準備配方節點需要使用 Python。當資料準備配方節點新增至任務時，會自動設定此選項。
- 使用資料預覽時，您需要在將資料準備配方節點新增至任務後重新啟動資料預覽工作階段。

如何在 AWS Glue Studio 中使用 AWS Glue DataBrew 配方

若要在 AWS Glue Studio 中使用 AWS Glue DataBrew 配方，請先在 AWS Glue DataBrew 中建立配方。如果您已擁有要使用的配方，則可略過此步驟。

若要在 AWS Glue DataBrew 中建立 AWS Glue DataBrew 配方：

1. 在 AWS Glue DataBrew 中撰寫配方。如需詳細資訊，請參閱 AWS Glue DataBrew 入門。
2. 儲存您的配方。
3. 發布您的配方。這會將您的配方作為 1.0 版發布。

若要在 AWS Glue Studio 中使用資料準備配方節點：

您可以在視覺化 ETL 任務中使用多個資料準備配方節點。若要這麼做，請依照下列步驟新增資料準備配方節點，並將另一個資料準備配方節點新增至任務。例如，工作流程可能會遵循以下模式：

- 資料來源 1 > 配方 1 > 輸出 1
- 資料來源 2 > 配方 2 > 輸出 2
- 輸出 1、輸出 2 > 聯結

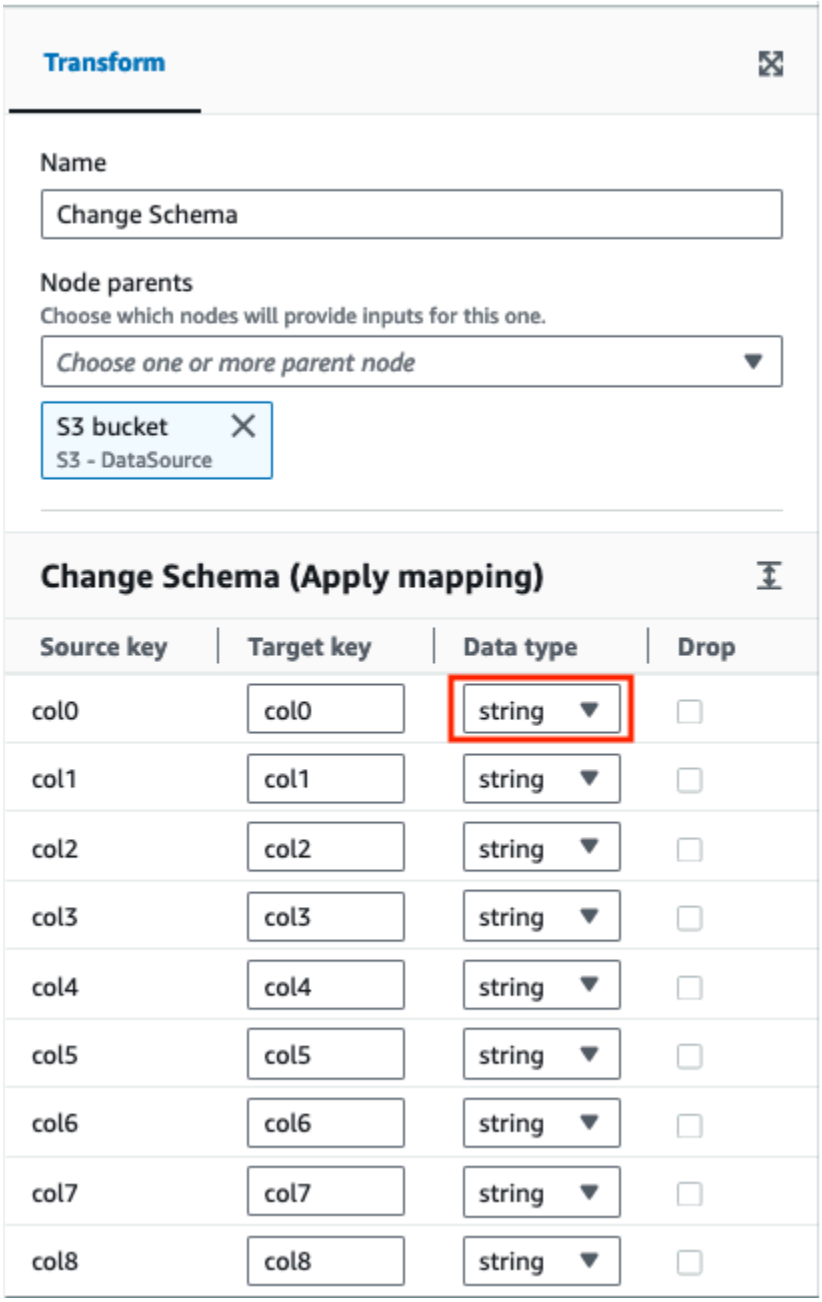
1. 使用資料來源啟動 AWS Glue Studio 中的 AWS Glue 任務。
2. 將資料準備配方節點新增至您的資料來源。
3. 在搜尋欄位中輸入配方名稱，依名稱篩選配方。
4. 選擇已發布的版本。只有已發布的版本可用。
5. 視需要新增其他轉換節點來完成任務的撰寫，並新增資料目標節點以儲存任務輸出。
6. 在任務詳細資訊索引標籤中進行必要的組態變更，例如命名任務和視需要調整配置的容量，然後儲存任務。
7. 從動作下拉式選單中選擇執行來執行任務。

若要在資料來源為 Amazon S3 且資料格式為 CSV 時變更結構描述：

如果 CSV 檔案中的所有資料欄一開始都以字串資料類型的形式載入 AWS Glue Studio，您需要確定資料欄資料類型與 AWS Glue DataBrew 配方中的其餘步驟相容。

AWS Glue DataBrew 配方只規定如何轉換已經讀取的資料。它不會描述在何處以及如何讀取資料。

1. 在多步驟配方節點之前新增變更結構描述節點。
2. 按一下變更結構描述節點，視需要在資料欄的轉換中選取新資料類型，以將結構描述變更為與 AWS Glue DataBrew 中的資料欄資料類型相同。



若要在資料來源無標頭時變更結構描述：

AWS Glue DataBrew 配方只規定如何轉換已經讀取的資料。它不會描述在何處以及如何讀取資料。

在 AWS Glue Studio 中載入無標頭的資料集時，預設標頭名稱與 AWS Glue DataBrew 中載入的資料集不同。

1. 在 ETL 任務中，在資料準備配方節點之前新增變更結構描述節點。

2. 選擇變更結構描述節點，並將資料欄名稱變更為 AWS Glue DataBrew 配方中使用的相同名稱。

使用「變更結構描述」重新映射資料屬性索引鍵

變更結構描述轉換會將來源資料屬性索引鍵重新映射到目標資料所需的設定。在「變更結構描述」轉換節點中，您可以：

- 變更多個資料屬性索引鍵的名稱。
- 變更資料屬性索引鍵的資料類型 (如果支援新的資料類型，且兩種資料類型之間有轉換路徑)。
- 透過指示要捨棄的資料屬性索引鍵來選擇資料屬性索引鍵的子集。

您也可以視需要將其他「變更結構描述」節點新增至工作圖表，例如修改其他資料來源或在「聯結」轉換之後。

使用具有十進制數據類型的更改模式

使用具有十進位資料類型的變更結構描述轉換時，「變更結構描述」轉換會將精確度修改為預設值 (10,2)。若要修改此項並設定使用案例的精確度，您可以使用 SQL Query 轉換，並以特定精確度轉換資料行。

例如，如果您有一個類型為 Decimal 的名稱為 DecimalCol "" 的輸入資料行，並且想要將其重新對應至名為 "OutputDecimalCol" 的輸出資料行，並且特定精確度為 (18,6)，您可以：

1. 在「變更結構描述」轉換之後新增後續的 SQL 查詢轉換。
2. 在 SQL 查詢轉換中，使用 SQL 查詢將重新對應的資料行轉換為想要的有效位數。SQL 查詢看起來像這樣：

```
SELECT col1, col2, CAST(DecimalCol AS DECIMAL(18,6)) AS OutputDecimalCol
FROM __THIS__
```

在上面的 SQL 查詢中：

- `col1` 和 `col2` 是您資料中的其他欄，您想要在不修改的情況下通過。
- `DecimalCol` 是輸入資料中的原始欄名稱。
- 「轉換（DecimalCol 如十進制（18,6））」將 `` 轉換為十進制類型，精度為 18 位數和 6 位小數。DecimalCol
- `AS OutputDecimalCol `將鑄造的列重命名為` `。OutputDecimalCol

藉由使用 SQL 查詢轉換，您可以覆寫 [變更結構描述] 轉換所設定的預設有效位數，並將 Decimal 資料行明確轉換為想要的有效位數。此方法可讓您利用變更結構描述轉換來重新命名和重新建構資料，同時透過後續的 SQL 查詢轉換處理 Decimal 資料行的精確度需求。

將變更結構描述轉換新增至您的工作

> **Note**
>
> 變更結構描述轉換不區分大小寫。

將「變更結構描述」轉換節點新增至您的任務圖表

1. (選用) 根據需要開啟資源面板，然後選擇變更結構描述，將新轉換新增至您的任務圖表。
2. 在節點屬性面板中，輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 選擇節點屬性面板中的轉換索引標籤。
4. 修改輸入結構描述：
   - 若要重新命名資料屬性索引鍵，請在目標索引鍵欄位中輸入索引鍵的新名稱。
   - 若要變更資料屬性索引鍵的資料類型，請從資料類型清單中為索引鍵選擇新的資料類型。
   - 若要從目標結構描述移除資料屬性索引鍵，請選擇該索引鍵的捨棄核取方塊。
5. (選用) 設定轉換節點屬性之後，您可以選擇節點詳細資訊面板中的輸出結構描述索引標籤來檢視資料的修改後結構描述。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。如果您尚未在任務詳細資訊索引標籤上指定 IAM 角色，系統會提示您在此輸入 IAM 角色。
6. (選用) 設定節點屬性和轉換屬性之後，您可以選擇節點詳細資訊面板中的資料預覽索引標籤來預覽修改後的資料集。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。使用此功能需支付相關費用，並且在您提供 IAM 角色後立即開始計費。

使用「刪除重複項」

「刪除重複項」轉換會為您提供兩個選項，以從資料來源中移除資料列。您可以選擇移除完全相同的重複資料列，也可以選擇要比對的欄位，并根据所選欄位僅移除這些資料列。

例如，在此資料集中，您有重複的資料列，其中某些資料列中的所有值與另一資料列完全相同，而資料列中的某些值則相同或不同。

Row	Name	Email	Age	State	注意
1	Joy	joy@gmail	33	NY	
2	Tim	tim@gmail	45	OH	
3	Rose	rose@gmail	23	NJ	
4	Tim	tim@gmail	42	OH	
5	Rose	rose@gmail	23	NJ	
6	Tim	tim@gmail	42	OH	這是一個重複的資料列，並與第 4 資料列的所有值完全相符
7	Rose	rose@gmail	23	NJ	這是一個重複的資料列，並與第 5 資料列的所有值完全相符

如果您選擇符合整個資料列，則會從資料集中移除第 6 資料列和第 7 資料列。資料集現在為：

Row	Name	Email	Age	State
1	Joy	joy@gmail	33	NY
2	Tim	tim@gmail	45	OH
3	Rose	rose@gmail	23	NJ
4	Tim	tim@gmail	42	OH
5	Rose	rose@gmail	23	NJ

如果您選擇指定索引鍵，則可以選擇移除與 'name' 和 'email' 相符的資料列。這讓您可以更好地控制資料集的「重複資料列」。透過指定 'name' 和 'email'，資料集現在為：

Row	Name	Email	Age	State
1	Joy	joy@gmail	33	NY
2	Tim	tim@gmail	45	OH
3	Rose	rose@gmail	23	NJ

需要謹記的一些事項：

- 為了將資料列識別為重複項，值需區分大小寫。資料列中的所有值都需具有相同的大小寫，這適用於您選擇的任一選項 (符合整個資料列或指定索引鍵)。
- 所有值都會以字串形式讀入。
- 刪除重複項轉換使用 Spark dropDuplicates 命令。
- 使用刪除重複項轉換時，會保留第一個資料列，並刪除其他資料列。
- 刪除重複項轉換不會變更資料框的結構描述。如果您選擇指定索引鍵，則所有欄位都保留在產生的資料框中。

使用 SelectFields 移除大多數資料屬性索引鍵

您可以使用 SelectFields 轉換，從資料集建立資料屬性索引鍵的子集。您可以指定要保留的資料屬性索引鍵，其餘的項目會從資料集中移除。

> **Note**
>
> SelectFields 轉換區分大小寫。如果您需要不區分大小寫的方式來選擇欄位，請使用 ApplyMapping。

將 SelectFields 轉換節點新增至任務圖表

1. (選用) 根據需要開啟資源面板，然後選擇 SelectFields，將新轉換新增至您的任務圖表。

2. 在 Node properties (節點屬性) 索引標籤上，輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 選擇節點詳細資訊面板中的轉換索引標籤。
4. 在 SelectFields 標題下，選擇您想要保留的資料集中的資料屬性索引鍵。任何未選取的資料屬性索引鍵都會從資料集中捨棄。

   您也可以選擇欄標題欄位旁的核取方塊，自動選擇資料集中的所有資料屬性索引鍵。然後，您可以取消選取個別資料屬性索引鍵，將其從資料集中移除。
5. (選用) 設定轉換節點屬性之後，您可以選擇節點詳細資訊面板中的輸出結構描述索引標籤來檢視資料的修改後結構描述。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。如果您尚未在任務詳細資訊索引標籤上指定 IAM 角色，系統會提示您在此輸入 IAM 角色。
6. (選用) 設定節點屬性和轉換屬性之後，您可以選擇節點詳細資訊面板中的資料預覽索引標籤來預覽修改後的資料集。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。使用此功能需支付相關費用，並且在您提供 IAM 角色後立即開始計費。

使用 DropFields 保留大部分的資料屬性索引鍵

您可以使用 DropFields 轉換，從資料集建立資料屬性索引鍵的子集。您可以指定要從資料集中移除的資料屬性索引鍵，並保留其餘的索引鍵。

> **Note**
>
> DropFields 轉換區分大小寫。如果您需要不區分大小寫的方式來選取欄位，請使用變更結構描述。

將 DropFields 轉換節點新增到您的任務圖表

1. (選用) 根據需要開啟資源面板，然後選擇 DropFields，將新轉換新增至您的任務圖表。
2. 在節點屬性索引標籤上，輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 選擇節點詳細資訊面板中的轉換索引標籤。
4. 在 DropFields 標題下，選擇要從資料來源捨棄的資料屬性索引鍵。

   您也可以選擇欄標題欄位旁的核取方塊，自動選擇資料集中的所有資料屬性索引鍵。然後，您可以取消選取個別資料屬性索引鍵，以便它們保留在資料集中。

5. (選用) 設定轉換節點屬性之後，您可以選擇節點詳細資訊面板中的輸出結構描述索引標籤來檢視資料的修改後結構描述。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。如果您尚未在任務詳細資訊索引標籤上指定 IAM 角色，系統會提示您在此輸入 IAM 角色。
6. (選用) 設定節點屬性和轉換屬性之後，您可以選擇節點詳細資訊面板中的資料預覽索引標籤來預覽修改後的資料集。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。使用此功能需支付相關費用，並且在您提供 IAM 角色後立即開始計費。

重新命名資料集中的欄位

您可以使用 RenameField 轉換來變更資料集中個別屬性索引鍵的名稱。

> **Note**
>
> RenameField 轉換區分大小寫。如果您需要不區分大小寫的轉換，請使用 ApplyMapping。

> **Tip**
>
> 如果您使用變更結構描述轉換，您可以使用單一轉換重新命名資料集中的多個資料屬性索引鍵。

將 RenameField 轉換節點新增至任務圖表

1. (選用) 根據需要開啟資源面板，然後選擇 RenameField，將新轉換新增至您的任務圖表。
2. 在節點屬性索引標籤上，輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 選擇轉換索引標籤。
4. 在資料欄位標題下，從來源資料中選擇屬性索引鍵，然後在新欄位名稱欄位中輸入新名稱。
5. (選用) 設定轉換節點屬性之後，您可以選擇節點詳細資訊面板中的輸出結構描述索引標籤來檢視資料的修改後結構描述。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。如果您尚未在任務詳細資訊索引標籤上指定 IAM 角色，系統會提示您在此輸入 IAM 角色。

6. (選用) 設定節點屬性和轉換屬性之後，您可以選擇節點詳細資訊面板中的資料預覽索引標籤來預覽修改後的資料集。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。使用此功能需支付相關費用，並且在您提供 IAM 角色後立即開始計費。

使用 Spigot 對您的資料集進行取樣

若要測試任務所執行的轉換，您可能需要取得資料樣本，以檢查轉換是否如預期運作。Spigot 轉換會將記錄子集從資料集寫入 Amazon S3 儲存貯體中的 JSON 檔案。資料取樣方法可以是從檔案開頭的特定記錄數目，或是用於挑選記錄的機率因素。

將 Spigot 轉換節點新增到您任務圖表

1. (選用) 根據需要開啟資源面板，然後選擇 Spigot，將新轉換新增至您的任務圖表。
2. 在節點屬性索引標籤上，輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 選擇節點詳細資訊面板中的轉換索引標籤。
4. 輸入 Amazon S3 路徑或選擇 Amazon S3 路徑瀏覽 S3在 Amazon S3 中選擇位置。這是任務寫入包含資料範例之 JSON 檔案的位置。
5. 輸入取樣方式的資訊。您可指定從資料集的開頭開始寫入的記錄數的值，以及選擇任何給定記錄的機率閾值 (以十進制值輸入，最大值為 1)。

   例如，若要從資料集寫入前 50 筆記錄，您可以將記錄數設定為 50，以及機率閾值 設定為 1 (100%)。

聯結資料集

Join 轉換允許您將兩個資料集合成一個。您可以在要比較的每個資料集的結構描述中指定索引鍵名稱。DynamicFrame 的輸出包含索引鍵符合聯結條件的列。符合聯結條件的每個資料集中的列會合併成輸出 DynamicFrame 中的單一列，其中包含任何一個資料集中找到的所有欄。

將 Join 轉換節點新增到您的任務圖表

1. 如果只有一個可用的資料來源，您必須將新的資料來源節點新增至任務圖表。
2. 選擇聯結的其中一個來源節點。開啟資源面板，然後選擇聯結，將新轉換新增至您的任務圖表。
3. 在 Node properties (節點屬性) 索引標籤上，輸入任務圖表中節點的名稱。
4. 在節點屬性索引標籤的節點父項標題下，新增一個節點父項，以便有兩個資料集為聯結提供輸入。父項可以是資料來源節點或轉換節點。

> **Note**
>
> 一個聯結只能有兩個父節點。

5. 選擇轉換索引標籤。

   如果您看到一則訊息，指出有衝突的索引鍵名稱，您可以：

   - 選擇解決它以自動將 ApplyMapping 轉換節點新增到您的任務圖表。ApplyMapping 節點會將字首新增至資料集中與其他資料集中的索引鍵名稱相同的任何索引鍵。例如，如果您使用預設值 **right**，那麼右側資料集中任何具有與左側資料集索引鍵相同名稱的索引鍵將被重命名為 `(right)`*key name*。
   - 在任務圖表中的前面手動新增轉換節點，以移除或重新命名衝突的索引鍵。

6. 在聯結類型清單中選擇聯結的類型。

   - 內部聯結：根據聯結條件，從兩個資料集為每個相符項目傳回欄的列。不會傳回不符合聯結條件的列。
   - 左聯結：左側資料集的所有列，以及右側資料集中符合聯結條件的列。
   - 右聯結：右側資料集的所有列，以及左側資料集中符合聯結條件的列。
   - 外聯結：來自兩個資料集的所有列。
   - 左半聯結：根據聯結條件，在右側資料集內具有相符項目的左側資料集中所有列。
   - 左反聯結：根據聯結條件，在右側資料集內沒有相符項目的左側資料集中所有列。

7. 在轉換索引標籤的聯結條件標題下，選擇新增條件。從每個資料集中選擇要比較的屬性索引鍵。比較運算子左側的屬性索引鍵稱為左側資料集，右側的屬性索引鍵稱為右側資料集。

   對於更複雜的聯結條件，您可以多次選擇新增條件，新增其他比對索引鍵。如果您不小心新增條件，您可以選擇刪除圖示
   (🗑)
   將其移除。

8. (選用) 設定轉換節點屬性之後，您可以選擇節點詳細資訊面板中的輸出結構描述索引標籤來檢視資料的修改後結構描述。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。如果您尚未在任務詳細資訊索引標籤上指定 IAM 角色，系統會提示您在此輸入 IAM 角色。

9. (選用) 設定節點屬性和轉換屬性之後，您可以選擇節點詳細資訊面板中的資料預覽索引標籤來預覽修改後的資料集。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。使用此功能需支付相關費用，並且在您提供 IAM 角色後立即開始計費。

如需聯結輸出結構描述的範例，請試想在具有下列屬性索引鍵的兩個資料集之間使用聯結：

```
Left: {id, dept, hire_date, salary, employment_status}
Right: {id, first_name, last_name, hire_date, title}
```

聯結被設定為使用 = 比較運算子比對 `id` 和 `hire_date` 索引鍵。

因為這兩個資料集都包含 `id` 和 `hire_date` 索引鍵，您選擇解決它以自動新增字首 **`right`** 到右側資料集的索引鍵。

輸出結構描述中的索引鍵將是：

```
{id, dept, hire_date, salary, employment_status,
(right)id, first_name, last_name, (right)hire_date, title}
```

使用「聯集」來合併資料列

如果您想要合併具有相同結構描述的多個資料來源的資料列，可以使用「聯集」轉換節點。

「聯集」轉換有兩種類型：

1. ALL：套用 ALL 時，產生的聯集不會移除重複的資料列。
2. DISTINCT：套用 DISTINCT 時，產生的聯集會移除重複的資料列。

聯集與聯結

您可以使用「聯集」來合併資料列。您可以使用「聯結」來合併資料欄。

在視覺化 ETL 畫布中使用「聯集」轉換

1. 新增多個資料來源以執行聯集轉換。若要新增資料來源，請開啟資源面板，然後從「來源」索引標籤中選擇資料來源。在使用「聯集」轉換之前，您必須確保聯集中涉及的所有資料來源都具有相同的結構描述和結構。

2. 當您至少有兩個要使用「聯集」轉換合併的資料來源時，請將其新增至畫布以建立「聯集」轉換。開啟畫布上的資源面板並搜尋 'Union'。您也可以選擇資源面板中的「轉換」索引標籤，向下捲動直到找到「聯集」轉換，然後選擇聯集。
3. 選取任務畫布上的「聯集」節點。在「節點屬性」視窗中，選擇要連線至「聯集」轉換的父節點。
4. AWS Glue 檢查相容性，以確保「聯集」轉換可套用至所有資料來源。如果資料來源的結構描述相同，則會允許執行此操作。如果資料來源沒有相同的結構描述，則會顯示無效的錯誤訊息：“此聯集的輸入結構描述不相同。請考慮使用 ApplyMapping 來符合結構描述。」 若要修正此問題，請選擇使用 ApplyMapping。
5. 選擇聯集類型。
    1. ALL：依預設會選取「所有聯集」類型；如果資料組合中有任何資料列，則會產生重複的資料列。
    2. DISTINCT：如果您想要從產生的資料組合中移除重複資料列，請選擇 DISTINCT。

使用 SplitFields 將資料集分成兩個

SplitFields 轉換可讓您選擇輸入資料集中的某些資料屬性索引鍵，並將其放入一個資料集中，並將未選取的索引鍵放入個別的資料集中。這個轉換的輸出是 `DynamicFrames` 的集合。

> **Note**
>
> 您必須使用 SelectFromCollection 轉換以將 `DynamicFrames` 的集合轉換成單一 `DynamicFrame`，然後才能將輸出傳送到目標位置。

SplitFields 轉換區分大小寫。如果您需要不區分大小寫的屬性索引鍵名稱，請新增 ApplyMapping 轉換做為父節點。

將 SplitFields 轉換節點新增到您的任務圖表

1. (選用) 根據需要開啟資源面板，然後選擇 SplitFields，將新轉換新增至您的任務圖表。
2. 在節點屬性索引標籤上，輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 選擇轉換索引標籤。
4. 選擇要放入第一個資料集的屬性索引鍵。您沒有選擇的索引鍵會放置在第二個資料集中。

5. (選用) 設定轉換節點屬性之後，您可以選擇節點詳細資訊面板中的輸出結構描述索引標籤來檢視資料的修改後結構描述。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。如果您尚未在任務詳細資訊索引標籤上指定 IAM 角色，系統會提示您在此輸入 IAM 角色。
6. (選用) 設定節點屬性和轉換屬性之後，您可以選擇節點詳細資訊面板中的資料預覽索引標籤來預覽修改後的資料集。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。使用此功能需支付相關費用，並且在您提供 IAM 角色後立即開始計費。
7. 設定 SelectFromCollection 轉換節點來處理產生的資料集。

SelectFromCollection 轉換概觀

某些轉換有多個資料集作為其輸出，而不是單個資料集，例如 SplitFields。SelectFromCollection 轉換從資料集的集合 (DynamicFrames 的陣列) 選擇一個資料集 (DynamicFrame)。轉換的輸出是選定的 DynamicFrame。

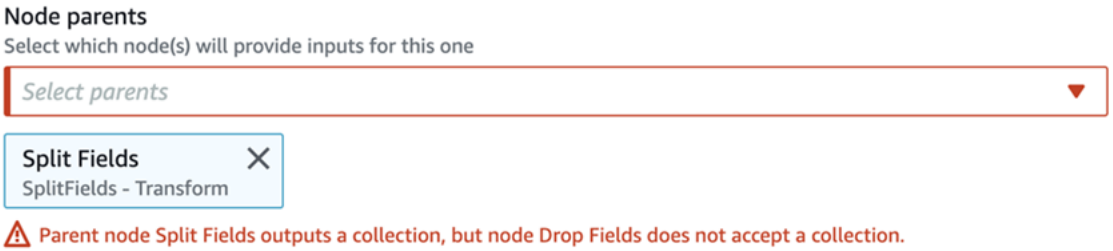
您必須在使用建立 DynamicFrames 集合的轉換後使用此轉換，例如：

- 自訂程式碼轉換
- SplitFields

如果在任何這些轉換之後您沒有新增 SelectFromCollection 至您的任務圖表，您會收到任務錯誤。

此轉換的父節點必須是傳回 DynamicFrames 集合的節點。如果您為傳回單一 DynamicFrame 的此轉換節點選擇父項，例如 Join 轉換，您的任務會傳回錯誤。

同樣地，如果您在任務圖表中使用 SelectFromCollection 節點作為需要單一 DynamicFrame 作為輸入之轉換的父項，您的任務也會傳回錯誤。



使用 SelectFromCollection 選擇要保留的資料集

使用 SelectFromCollection 轉換以將 DynamicFrames 集合轉換成單一 DynamicFrame。

將 SelectFromCollection 轉換節點新增到您的任務圖表

1. (選用) 根據需要開啟資源面板，然後選擇 SelectFromCollection，將新轉換新增至您的任務圖表。
2. 在節點屬性索引標籤上，輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 選擇轉換索引標籤。
4. 在影格索引標題下，選擇對應於您想從 DynamicFrames 集合中選取之 DynamicFrame 的陣列索引號碼。

   例如，如果此轉換的父節點是 SplitFields 轉換，在輸出結構描述索引標籤，您可以看到每個 DynamicFrame 的結構描述。如果您想要保留 DynamicFrame 與 輸出 2 的結構描述相關聯，您需要為影格索引的值選取 **1**，這是清單中的第二個值。

   只有 DynamicFrame 會包含在輸出中。
5. (選用) 設定轉換節點屬性之後，您可以選擇節點詳細資訊面板中的輸出結構描述索引標籤來檢視資料的修改後結構描述。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。如果您尚未在任務詳細資訊索引標籤上指定 IAM 角色，系統會提示您在此輸入 IAM 角色。
6. (選用) 設定節點屬性和轉換屬性之後，您可以選擇節點詳細資訊面板中的資料預覽索引標籤來預覽修改後的資料集。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。使用此功能需支付相關費用，並且在您提供 IAM 角色後立即開始計費。

尋找並填入資料集中遺失的值

您可以使用 FillMissingValues 轉換以尋找遺失值的資料集中的記錄，並新增具有由插補決定值的新欄位。輸入資料集用於訓練機器學習 (ML) 模型，以決定遺失值應該是什麼。如果您使用增量資料集，則會使用每個增量集作為 ML 模型的訓練資料，因此結果可能不會那麼準確。

在任務圖表中使用 FillMissingValues 轉換節點

1. (選用) 根據需要開啟資源面板，然後選擇 FillMissingValues，將新轉換新增至您的任務圖表。
2. 在 Node properties (節點屬性) 索引標籤上，輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 選擇轉換索引標籤。
4. 對於資料欄位，從您要分析遺失值的來源資料中選擇欄或欄位名稱。

5. (選用) 在新增欄位名稱欄位中，輸入新增至每筆記錄的欄位名稱，該欄位將保留已分析欄位的估計取代值。如果已分析的欄位沒有遺失值，則分析欄位中的值會複製到新欄位中。

   如果未指定新欄位的名稱，則預設名稱是已分析欄位的名稱，附加上 `_filled`。例如，如果您為資料欄位輸入 **Age**，並且不指定新欄位名稱的值，則會將名為 **Age_filled** 的新欄位新增至每個記錄。

6. (選用) 設定轉換節點屬性之後，您可以選擇節點詳細資訊面板中的輸出結構描述索引標籤來檢視資料的修改後結構描述。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。如果您尚未在任務詳細資訊索引標籤上指定 IAM 角色，系統會提示您在此輸入 IAM 角色。
7. (選用) 設定節點屬性和轉換屬性之後，您可以選擇節點詳細資訊面板中的資料預覽索引標籤來預覽修改後的資料集。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。使用此功能需支付相關費用，並且在您提供 IAM 角色後立即開始計費。

篩選資料集中的索引鍵

使用 Filter 轉換，藉由根據規則表達式篩選輸入資料集中的記錄來建立新的資料集。輸出中會移除不符合篩選條件的列。

- 對於字串資料類型，您可以篩選索引鍵值與指定字串相符的列。
- 對於數值資料類型，您可以使用比較運算子 <、>、=、!=、<= 及 >= 將索引鍵值與指定值進行比較，以篩選列。

如果您指定多個篩選條件，則預設會使用 AND 運算子來結合結果，但您可以改為選擇 OR。

Filter 轉換區分大小寫。如果您需要不區分大小寫的屬性索引鍵名稱，請新增 ApplyMapping 轉換做為父節點。

將 Filter 轉換節點新增到您的任務圖表

1. (選用) 根據需要開啟資源面板，然後選擇篩選，將新轉換新增至您的任務圖表。
2. 在 Node properties (節點屬性) 索引標籤上，輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 選擇轉換索引標籤。
4. 選擇全域和 或 全域或。這會決定多個篩選條件的組合方式。所有條件都使用 AND 或 OR 操作來結合。如果您只有一個篩選條件，則可以選擇任一個。

5. 選擇篩選條件區段的新增條件按鈕以新增篩選條件。

   在索引鍵欄位中，從資料集中選擇屬性索引鍵名稱。在操作欄位中，選擇比較運算子。在數值欄位中，輸入比較值。以下是篩選條件的一些範例：

   - `year >= 2018`
   - `State matches 'CA*'`

   當您篩選字串值時，請確定比較值使用的規則表達式格式符合任務屬性中選取的指令碼語言 (Python 或 Scala)。

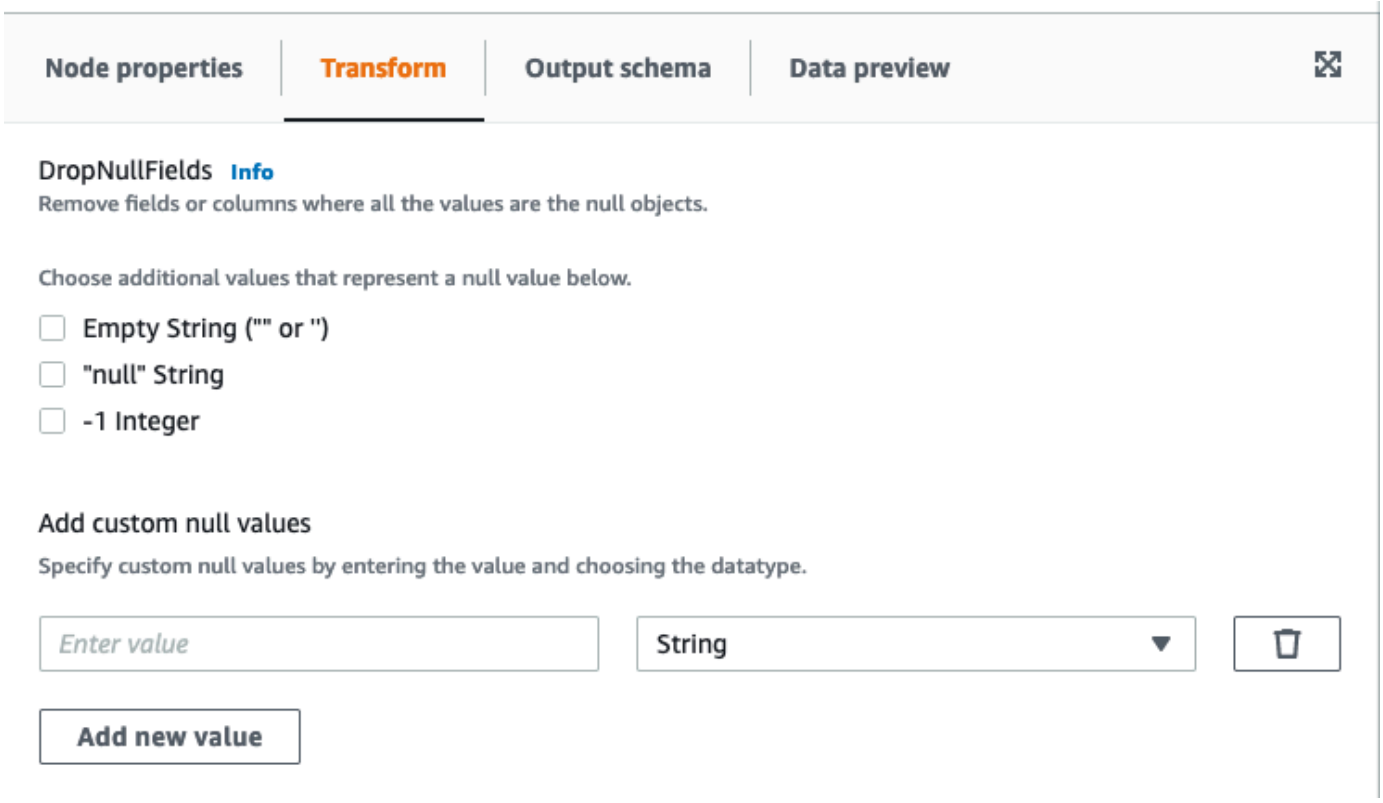
6. 視需要加入其他篩選條件。
7. (選用) 設定轉換節點屬性之後，您可以選擇節點詳細資訊面板中的輸出結構描述索引標籤來檢視資料的修改後結構描述。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。如果您尚未在任務詳細資訊索引標籤上指定 IAM 角色，系統會提示您在此輸入 IAM 角色。
8. (選用) 設定節點屬性和轉換屬性之後，您可以選擇節點詳細資訊面板中的資料預覽索引標籤來預覽修改後的資料集。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。使用此功能需支付相關費用，並且在您提供 IAM 角色後立即開始計費。

使用 DropNullFields 刪除具有空值的欄位

如果欄位中的所有值都為 ‘null’，使用 DropNullFields 轉換以從資料集中刪除此欄位。依預設，AWS Glue Studio 將識別空物件，但某些值，如空字串、值為 “null” 的字串、-1 整數或其他預留位置 (如零) 不會自動識別為空值。

使用 DropNullFields

1. 新增一個 DropNullFields 節點到任務圖表。
2. 在節點屬性索引標籤上，選擇代表空值的其他值。您可以選擇不選取任何值或選取所有值：



- 空字串 ("" 或 '') – 包含空字串的欄位將被刪除
- "null string" – 包含帶有字 'null' 的字串的欄位將被刪除
- -1 整數 – 包含 -1 整數的欄位將被刪除

3. 如果需要，您也可以指定自訂空值。這些是資料集中可能唯一的空值。若要新增自訂空值，請選擇新增值。
4. 輸入自訂空值。例如，這可以為 0，或用來表示資料集中之 null 的任何值。
5. 在下拉式清單欄位中選擇資料類型。資料類型可以是字串或整數。

> **Note**
>
> 自訂空值及其資料類型必須完全相符，才能將欄位識別為空值並移除此欄位。部分相符，其中只有自訂 Null 值相符，但資料類型不會導致移除欄位。

使用 SQL 查詢轉換資料

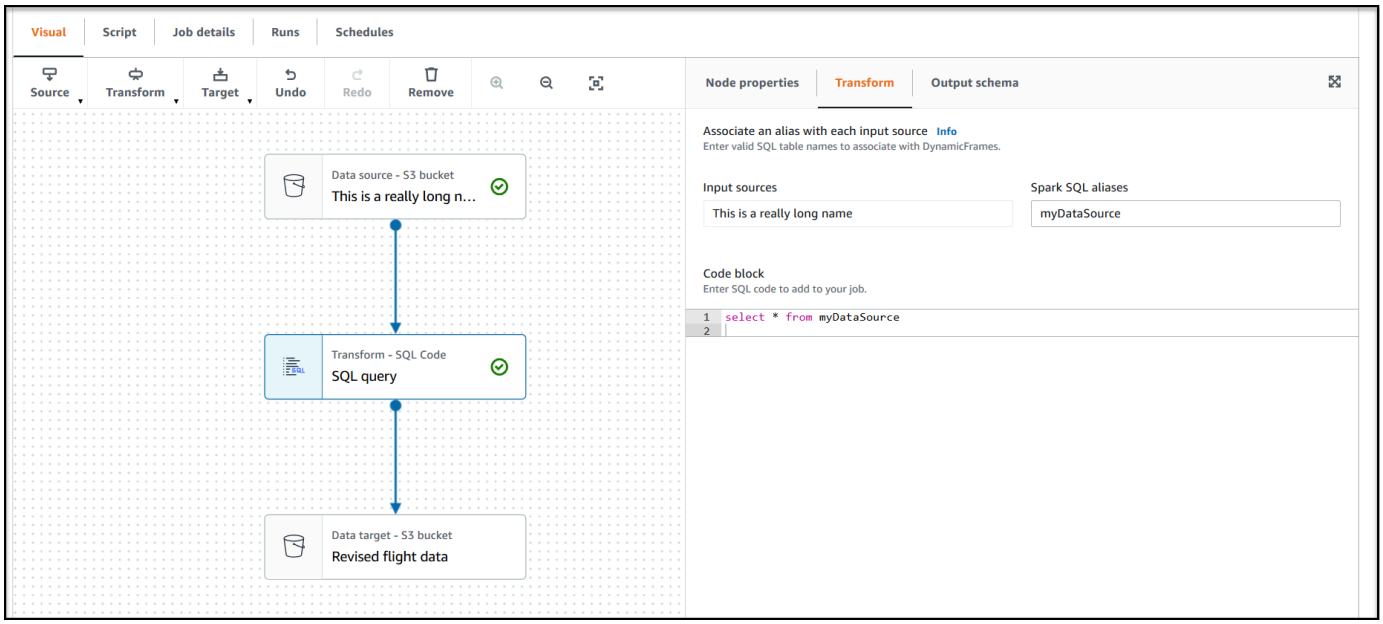
您可以使用 SQL 轉換來以 SQL 查詢的形式編寫自己的轉換。

SQL 轉換節點可以有多個資料集作為輸入，但只產生一個資料集作為輸出。包含一個文字欄位，您可以在其中輸入 Apache SparkSQL 查詢。您可以將別名指派給每個用作輸入的資料集，以協助簡化 SQL 查詢。如需 SQL 語法的詳細資訊，請參閱 Spark SQL 文件。

> **Note**
>
> 如果您使用 Spark SQL 轉換搭配位於 VPC 中的資料來源，請新增 AWS Glue VPC 端點至包含資料來源的 VPC。如需設定開發端點的詳細資訊，請參閱 AWS Glue 開發人員指南中的新增開發端點、設定適用於開發端點的環境以及存取您的開發端點。

在任務圖表中使用 SQL 轉換節點

1. (選用) 視需要將轉換節點新增至任務圖表。選擇 Spark SQL 節點類型。
2. 在 Node properties (節點屬性) 索引標籤上，輸入任務圖表中節點的名稱。如果尚未選取節點父項，或者如果您想要進行 SQL 轉換的多個輸入，請從 Node parents (節點父項) 清單選擇用作轉換之輸入來源的節點。視需要新增其他父節點。
3. 選擇節點詳細資訊面板中的 Transform (轉換) 索引標籤。
4. SQL 查詢的來源資料集是由您在每個節點欄位的 Name (名稱) 中指定的名稱來識別。如果您不想使用這些名稱，或者名稱不適合 SQL 查詢，您可以將名稱關聯到每個資料集。主控台提供預設別名，例如 `MyDataSource`。



例如，如果 SQL 轉換節點的父節點名為 `Rename Org PK field`，您可以將名稱 `org_table` 與此資料集相關聯。然後可以在 SQL 查詢中使用此別名來代替節點名稱。

5. 在 Code block (程式碼區塊) 標題下的文字輸入欄位，貼上或輸入 SQL 查詢。文字欄位會顯示 SQL 語法反白顯示和關鍵字建議。
6. 選取 SQL 轉換節點後，選擇 Output schema (輸出結構描述) 索引標籤，然後選擇 Edit (編輯)。提供描述 SQL 查詢輸出欄位的欄和資料類型。

   使用頁面的 Output schema (輸出結構描述) 區段中的以下動作指定結構描述：

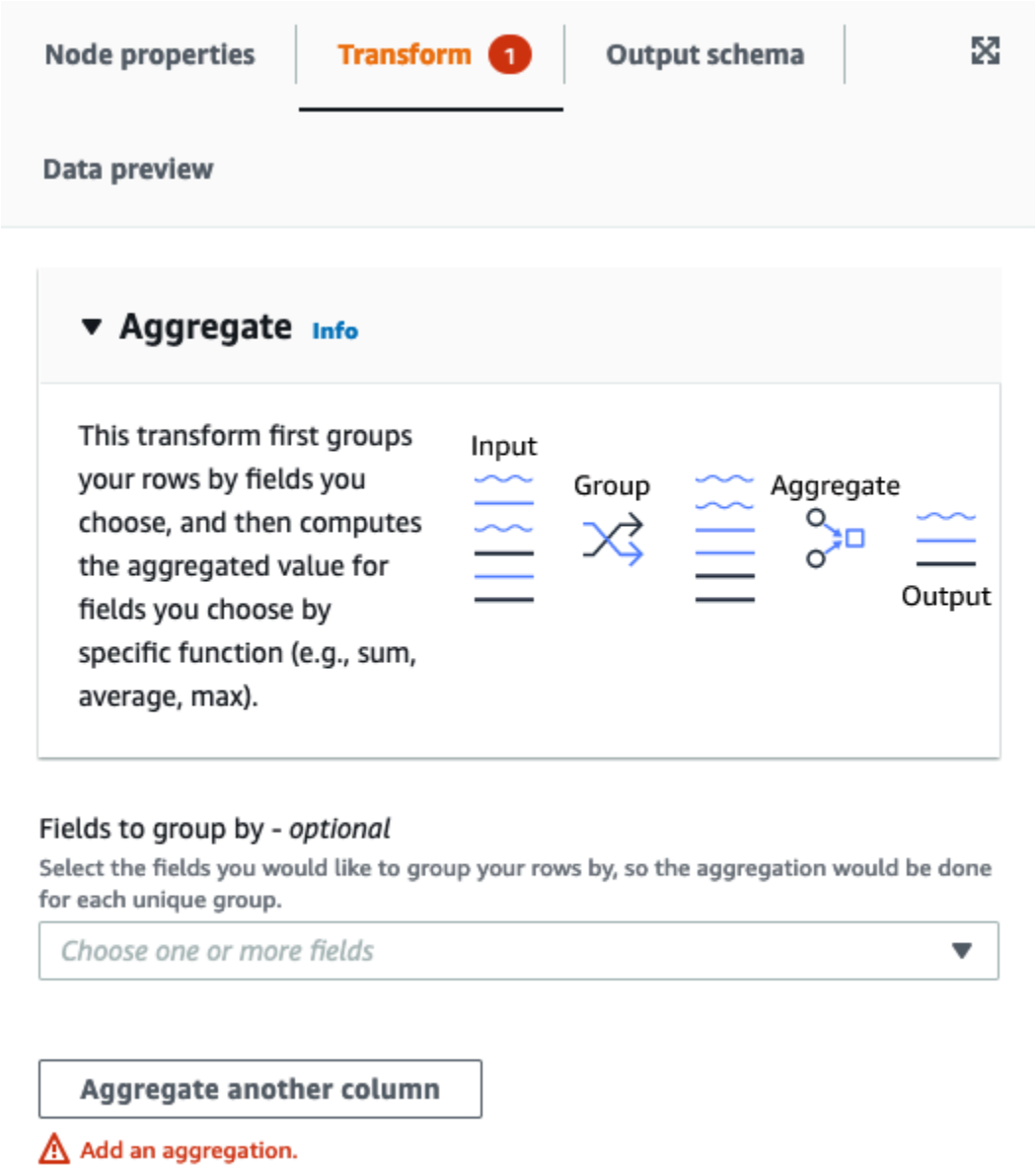
   - 若要重新命名欄，請將游標放在欄的 Key (索引鍵) 文字方塊 (也稱為欄位或屬性索引鍵)，然後輸入新名稱。
   - 若要變更欄的資料類型，請從下拉式清單中選取欄的新資料類型。
   - 若要將新的頂層欄新增至結構描述，請選擇 [Overflow (溢位)] (⋯) 按鈕，然後選擇 Add root key (新增根索引鍵)。新的欄會加入在結構描述的頂端。
   - 若要從結構描述移除欄，請選擇索引鍵名稱最右側的刪除圖示 (🗑)。
7. 當您完成指定輸出結構描述時，請選擇 Apply (套用) 儲存您的變更，並結束結構描述編輯器。如果您不想儲存變更，請選擇 Cancel (取消) 以編輯結構描述編輯器。
8. (選用) 設定節點屬性和轉換屬性之後，您可以選擇節點詳細資訊面板中的 Data preview (資料預覽) 索引標籤來預覽修改後的資料集。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。使用此功能需支付相關費用，並且在您提供 IAM 角色後立即開始計費。

使用彙總在選取的欄位上執行加總計算

若要使用彙總轉換

1. 將彙總節點新增至任務圖表。
2. 在節點屬性索引標籤上，透過選擇下拉式清單欄位 (選用) 來選擇要分組在一起的欄位。您可以一次選取多個欄位，或在搜尋列中輸入來搜尋欄位名稱。

   選取欄位時，會顯示名稱和資料類型。若要移除欄位，請選擇欄位上的 'X'。



3. 選擇彙總另一個資料欄。至少需要選取一個欄位。

4. 在要彙總的欄位中選擇一個欄位。
5. 選擇要套用至所選欄位的彙總函數：

- avg – 計算平均值
- countDistinct – 計算唯一非空值的數量
- count – 計算非空值的數量
- first – 傳回滿足 'group by' 條件的第一個值
- last – 傳回滿足 'group by' 條件的最後一個值
- kurtosis – 計算頻率分佈曲線峰值的清晰度
- max – 傳回滿足 'group by' 條件的最大值
- min – 傳回滿足 'group by' 條件的最小值
- skewness – 測量正態分佈的概率分佈的不對稱性
- stddev_pop – 計算人口標準差，並返回人口方差的平方根
- sum – 群組中所有值的總和
- sumDistinct – 群組中不同值的總和
- var_samp – 群組的樣本變異數 (忽略空值)
- var_pop – 群組的母體變異數 (忽略空值)

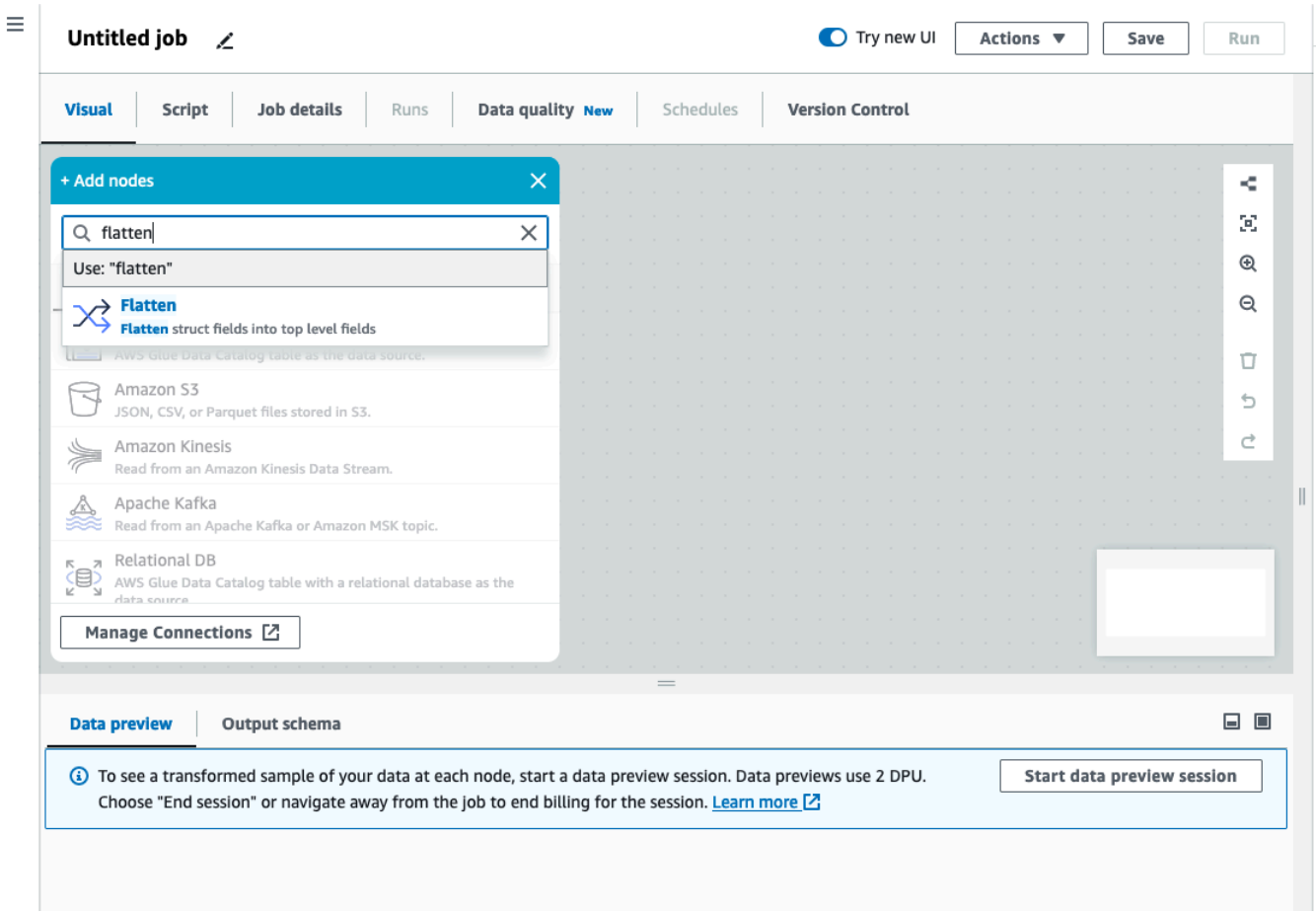
壓平合併巢狀結構

壓平合併資料中巢狀結構的欄位，使其成為頂層欄位。新欄位會使用欄位名稱來命名，其字首為與之關聯的結構欄位的名稱，並以點分隔。

例如，如果資料具有名稱為 "phone_numbers" 之結構類型的欄位，則其中包含一個名稱為 "home_phone" 的 “Struct” 類型的欄位，以及 "country_code" 和 "number" 兩個欄位。經壓平合併後，這兩個欄位將成為頂層欄位，名稱分別為："phone_numbers.home_phone.country_code" 和 "phone_numbers.home_phone.number"。

在您的任務圖表中新增壓平合併轉換節點

1. 開啟資源面板，然後選擇轉換索引標籤，然後選擇壓平合併，將新轉換新增至您的任務圖表。您也可以在搜尋列中輸入「壓平合併」，然後按一下「壓平合併」節點。新增節點時選取的節點將成為其父節點。



2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. (選用) 在轉換索引頁籤上，您可以限制要壓平合併的最大巢狀層級。例如，將該值設定為 1，意味著只會壓平合併頂層結構。將最大值設定為 2，將會直接壓平合併頂層及其下方的結構。

新增 UUID 資料欄

當您新增 UUID (通用不重複識別符) 資料欄時，會為每個資料列指派不重複的 36 字元字串。

在您的任務圖表中新增 UUID 轉換節點

1. 開啟資源面板，然後選擇 UUID，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. (選用) 在轉換索引標籤上，您可以自訂新資料欄的名稱。依預設，該名稱會命名為 "uuid"。

新增識別符資料欄

為資料集中的每個資料列指派數值識別符。

在您的任務圖表中新增識別符轉換節點

1. 開啟資源面板，然後選擇識別符，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. (選用) 在轉換索引標籤上，您可以自訂新資料欄的名稱。依預設，該名稱會命名為 "id"。
4. (選用) 如果任務以遞增方式處理和存放資料，您會想要避免在任務執行之間重複使用相同的 ID。

   在轉換索引標籤上，標記不重複核取方塊選項。這會在識別符中包含任務時間戳記，使其在多次執行之間是不重複的。若要允許更大的數字，資料欄而不是類型 long 將會是十進位。

將資料欄轉換為時間戳記類型

您可以使用轉換截止時間戳記，將數值或字串資料欄的資料類型變更為時間戳記，以便使用該資料類型存放，或套用至需要時間戳記的其他轉換。

在您的任務圖表中新增截止時間戳記轉換節點

1. 開啟資源面板，然後選擇截止時間戳記，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 在轉換索引標籤上，輸入要轉換的資料欄名稱。
4. 在轉換索引標籤上，定義如何透過選擇類型來剖析選取的資料欄。

   如果該值是一個數字，則可以用秒 (Unix/Python 時間戳記)、毫秒或微秒來表示，選擇相應的選項。

   如果該值是格式化字串，則選擇 "iso" 類型，該字串需要符合 ISO 格式的其中一個變體，例如："2022-11-02T14:40:59.915Z"。

   如果您不知道目前的類型，或者不同的資料列使用不同的類型，則可以選擇 ”autodetect“，系統會以較小的效能成本來做出最佳猜測。

5. (選用) 在轉換索引標籤上，您可以建立新資料欄並輸入新資料欄名稱來保留原始資料欄，而不是轉換選取的資料欄。

將時間戳記資料欄轉換為格式化字串

根據模式將時間戳記資料欄格式化為字串。您可以使用格式化時間戳記，作為所需格式的字串來取得日期和時間。您可以使用 Spark 日期語法，以及大多數 Python 日期碼來定義格式。

例如，如果您想要日期字串格式化為 "2023-01-01 00:00"，則可以使用 Spark 語法將這種格式定義為 "yyyy-MM-dd HH:mm" 或使用等效的 Python 日期碼將其定義為 "%Y-%m-%d %H:%M"

在您的任務圖表中新增格式化時間戳記轉換節點

1. 開啟資源面板，然後選擇格式化時間戳記，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 在轉換索引標籤上，輸入要轉換的資料欄名稱。
4. 在轉換索引標籤上，輸入要使用的時間戳記格式模式，使用 Spark 日期語法或 Python 日期碼來表示。
5. (選用) 在轉換索引標籤上，您可以建立新資料欄並輸入新資料欄名稱來保留原始資料欄，而不是轉換選取的資料欄。

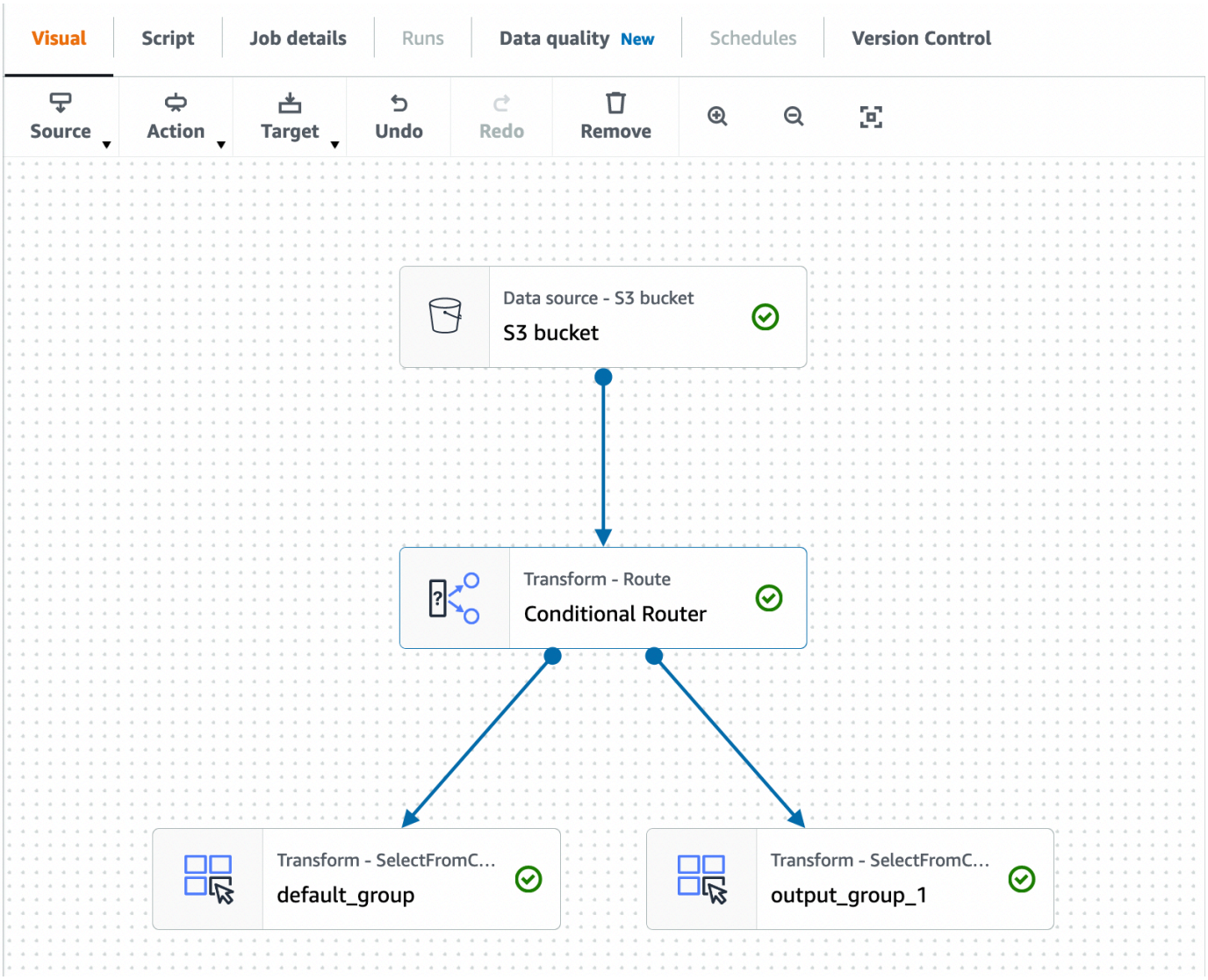
建立條件式路由器轉換

條件式路由器轉換可讓您對傳入資料套用多個條件。傳入資料的每個資料列均依據群組篩選條件進行評估，並將其處理到其對應的群組。如果資料列符合多個群組篩選條件，則轉換會將資料列傳遞給多個群組。如果資料列不符合任何條件，則可以捨棄或路由至預設輸出群組。

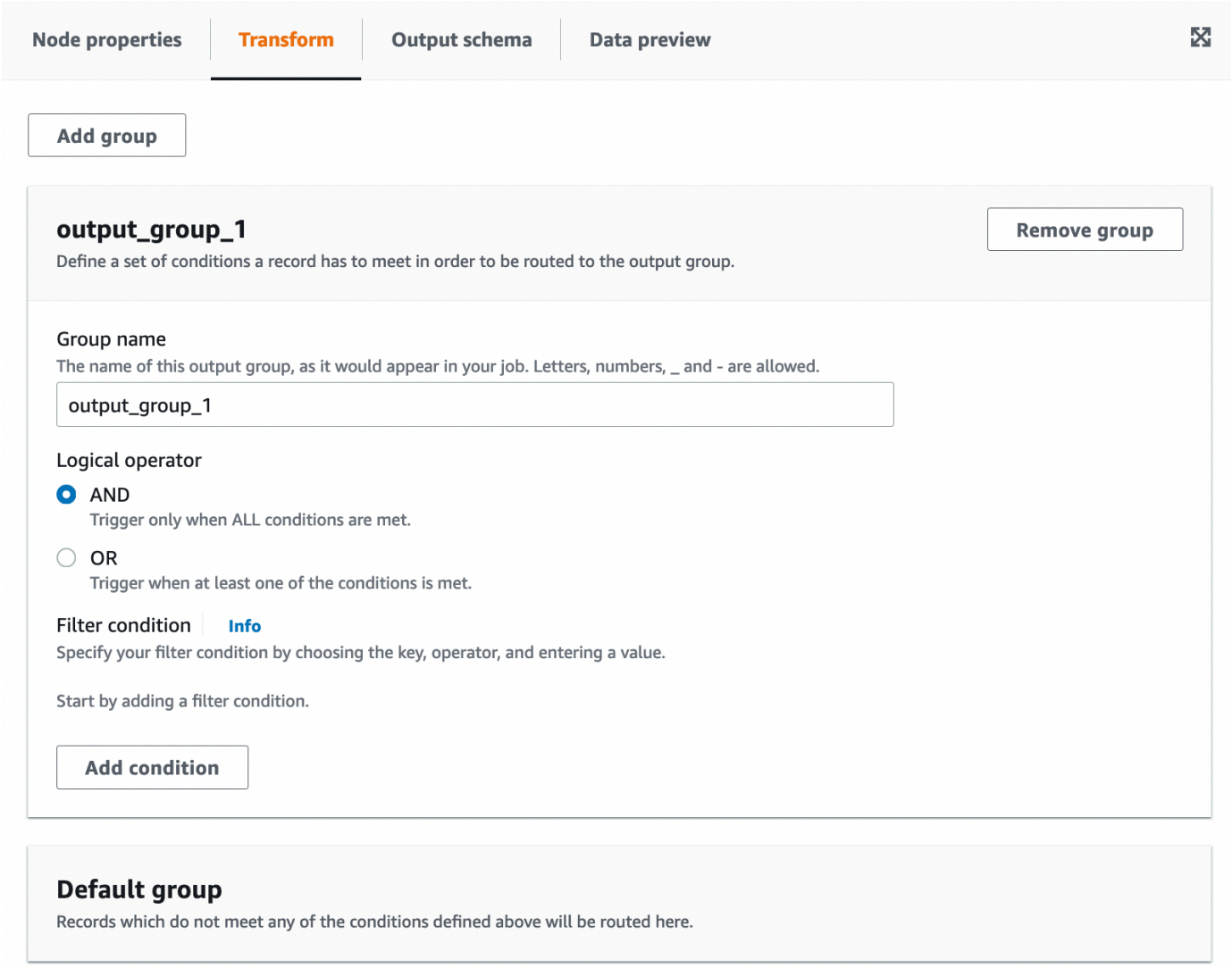
此轉換與篩選條件轉換類似，但對於想要在多個條件下測試相同輸入資料的使用者很有用。

新增條件式路由器轉換：

1. 選擇要執行條件式路由器轉換的節點。這可以是來源節點或其他轉換。
2. 選擇操作，然後使用搜尋列來尋找並選擇 'Conditional Router'。條件式路由器轉換與兩個輸出節點一起新增。一個輸出節點，即 'Default group'，包含的記錄與其他輸出節點中定義的任何條件均不相符。無法編輯預設群組。

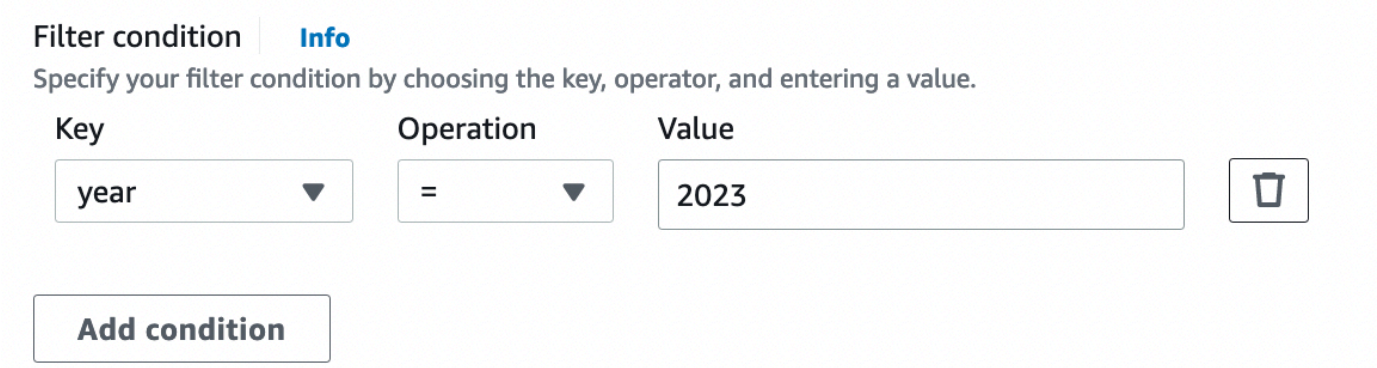


您可以選擇新增群組來新增其他輸出群組。針對每個輸出群組，您可以為群組命名，並新增篩選條件和邏輯運算子。



3. 輸入群組的新名稱，以對輸出群組名稱重新命名。AWS Glue Studio 會自動為您對群組命名 (例如，'output_group_1')。
4. 選擇邏輯運算子 (AND、OR)，並透過指定索引鍵、操作和值來新增篩選條件。邏輯運算子可讓您實作多個篩選條件，並在您指定的每個篩選條件上執行邏輯運算子。

   指定索引鍵後，您可以從結構描述的可用索引鍵中進行選擇。然後，您可以根據選取的索引鍵類型來選擇可用的操作。例如，如果索引鍵類型為 'string'，則可從中選擇的可用操作為 'matches'。



5. 在值欄位中，輸入值。若要新增其他篩選條件，請選擇新增條件。若要移除篩選條件，請選擇垃圾桶圖示。

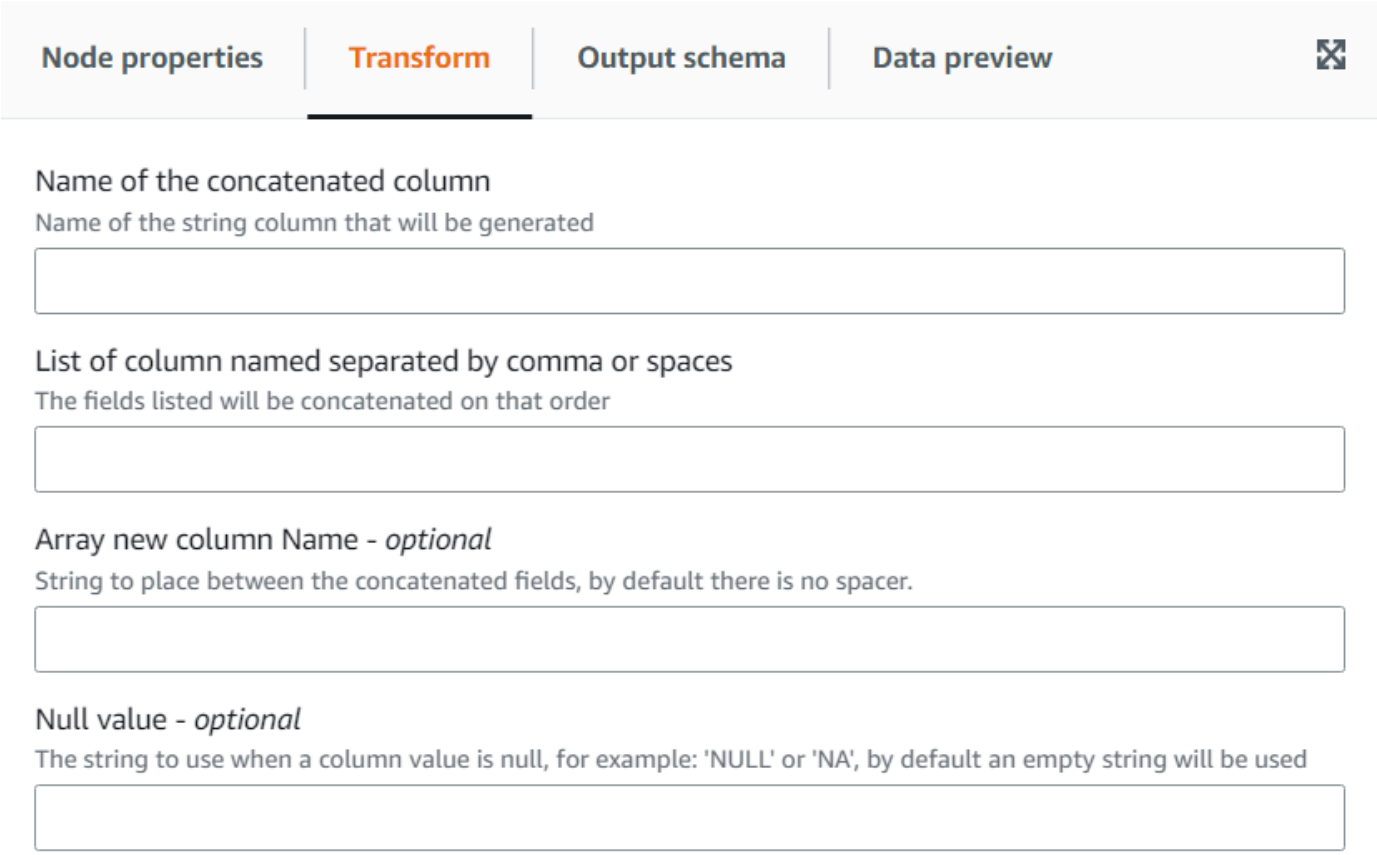
使用「串連資料欄」轉換附加資料欄

「串連」轉換允許您使用具有選用間隔符號之其他資料欄的值來構建新的字串資料欄。例如，如果我們將串連資料欄 “date” 定義為 “year”、“month”、“day”(按該順序) 的串連，並以 “-” 作為間隔符號，我們將得到：

day	month	year	日期
01	01	2020	2020-11-01
02	01	2020	2020-07-02
03	01	2020	2020-06-03
04	01	2020	2020-11-04

若要新增「串連」轉換：

1. 開啟資源面板。然後選擇串連資料欄，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 在轉換索引標籤上，輸入將保留串連字串的資料欄名稱，以及要串連的資料欄。您在下拉式選單中檢查資料欄的順序將是使用的順序。



4. 分隔符號 – 選用：輸入要在串連欄位之間放置的字串。預設沒有分隔符號。
5. Null 值 – 選用：輸入資料欄值為 Null 時要使用的字串。依預設，在資料欄具有 'NULL' 或 'NA' 值的情況下，使用空字串。

使用「分割字串」轉換來分解字串資料欄

「分割字串」轉換允許使用規則運算式將字串分解為字符陣列，以定義分割方式。然後，您可以將資料欄保留為陣列類型，或在此之後套用陣列至資料欄轉換，以將陣列值擷取到頂層欄位中，並假設每個字符都有我們事先知道的含義。此外，如果字符的順序不重要 (例如一組類別)，則可以使用分解轉換為每個值產生一個單獨的資料列。

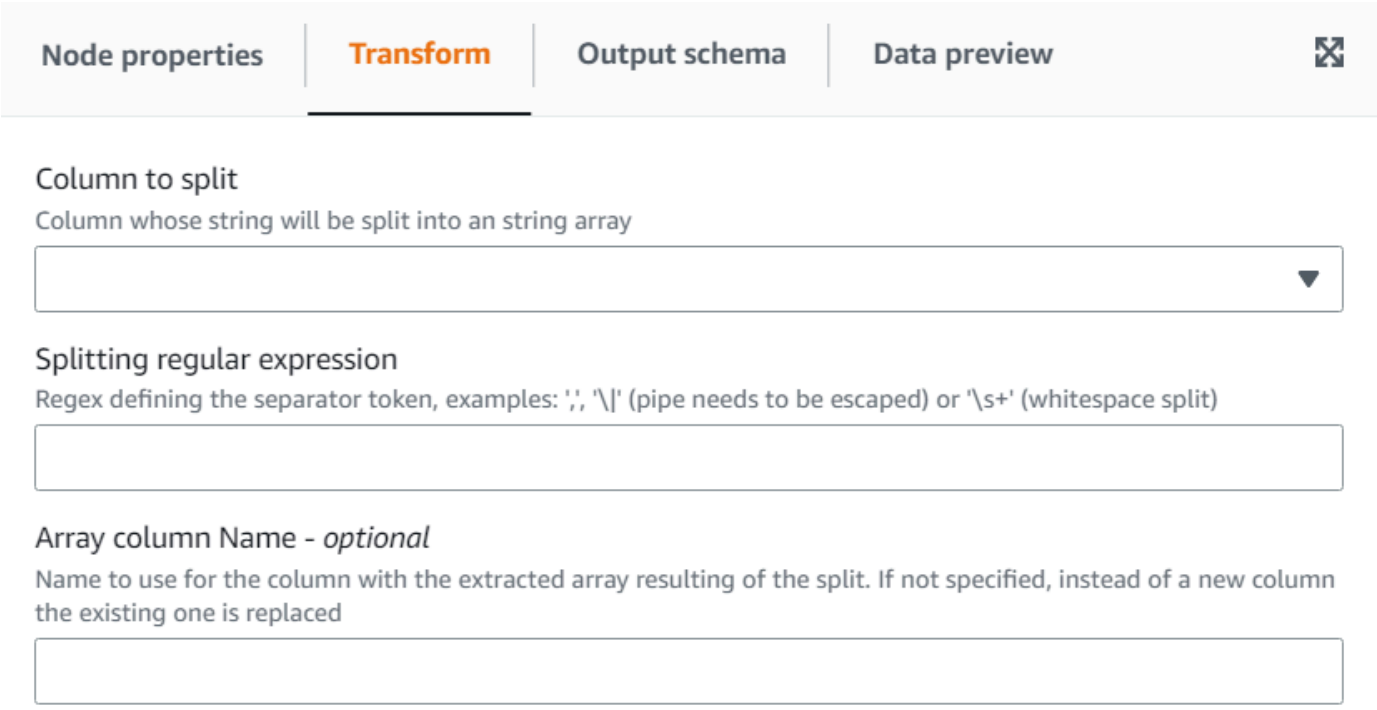
例如，您可以使用逗號作為模式來分割 “categories” 資料欄，以新增 “categories_arr” 資料欄。

product_id	categories	categories_arr
1	sports,winter	[sports, winter]
2	garden,tools	[garden, tools]

product_id	categories	categories_arr
3	videogames	[videogames]
4	game,boardgame,social	[game, boardgame, social]

若要新增「分割字串」轉換：

1. 開啟資源面板，然後選擇「分割字串」，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 在轉換索引標籤上，選擇要分割的資料欄，然後輸入要用於分割字串的模式。在大多數情況下，您只需輸入字元，除非它作為規則運算式具有特殊含義並需要逸出。需要逸出的字元為 `\.[]{}()<>*+-=!?^$|`，字元前面加上反斜線。例如，如果您想透過一個點 ('.') 分隔，則需要輸入 `\.`。但是，逗號沒有特殊含義，可以按以下方式指定：`,`。



4. (選用) 如果要保留原始字串資料欄，則可以為新陣列資料欄輸入名稱，這樣可以同時保留原始字串資料欄和新的記號化陣列資料欄。

使用「陣列至資料欄」轉換，將陣列的元素擷取到頂層資料欄中

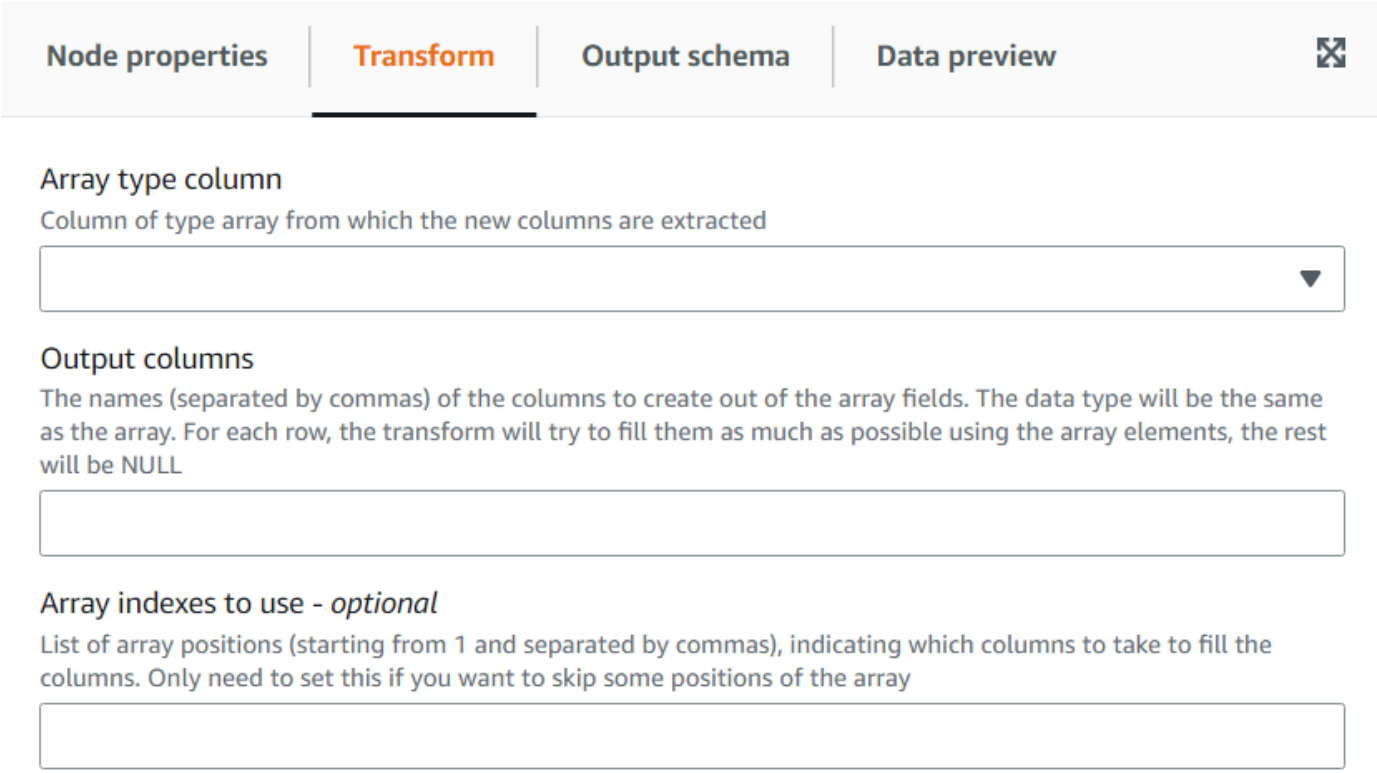
「陣列至資料欄」轉換可讓您將陣列類型之資料欄的部分或全部元素擷取到新資料欄中。如果陣列有足夠的值來擷取，則該轉換將盡可能地填充新資料欄，並且您可以選擇將元素放在指定位置。

例如，如果您有一個陣列資料欄 “subnet”，是在 ip v4 子網路上套用「分割字串」轉換的結果，則可以將第一個和第四個位置擷取到新資料欄 “first_octect” 和 “forth_octect” 中。在此範例中，轉換的輸出將是 (注意，最後兩個資料列的陣列比預期的短)：

子網	first_octect	fourth_octect
[54, 240, 197, 238]	54	238
[192, 168, 0, 1]	192	1
[192, 168]	192	
[]		

若要新增「陣列至資料欄」轉換：

1. 開啟資源面板，然後選擇陣列至資料欄，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 在轉換索引標籤上，選擇要擷取的陣列資料欄，然後輸入所擷取字符的新資料欄清單。



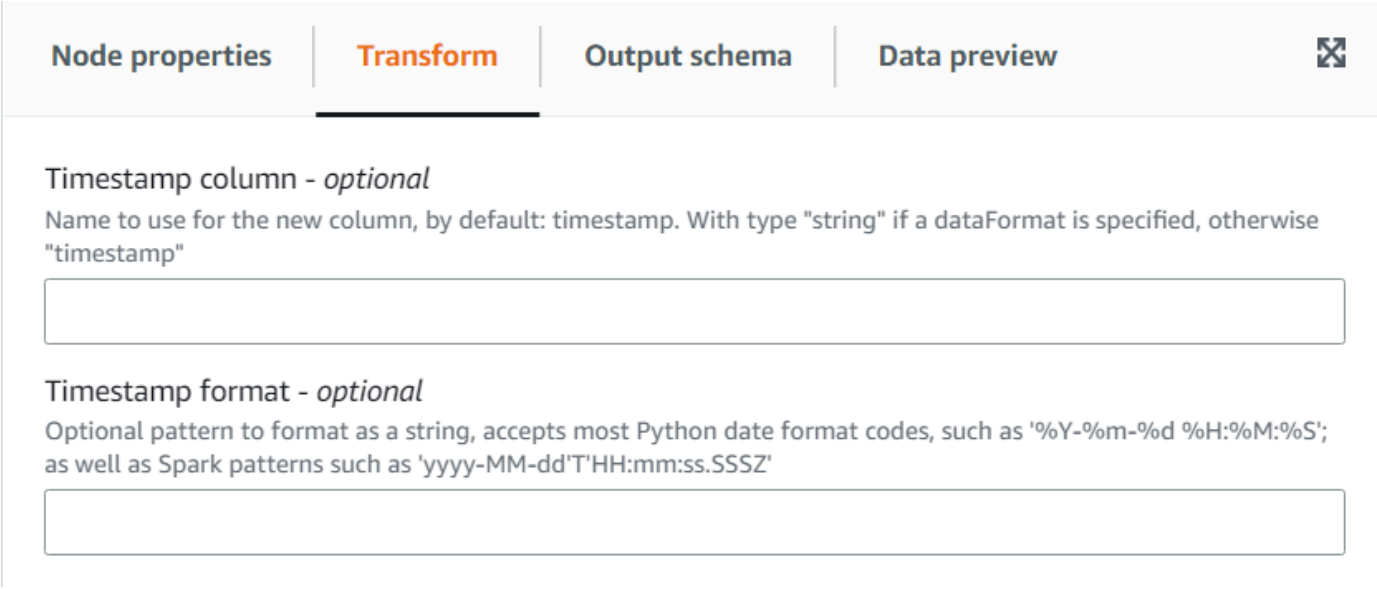
4. (選用) 如果您不想採用陣列字符以指派給資料欄，則可以指定要採用的索引，這些索引將按照指定順序指派給資料欄清單。例如，如果輸出資料欄是 “column1, column2, column3” 和索引 “4, 1, 3”，則陣列的第四個元素將轉到 column1，第一個元素轉到 column2，第三個元素轉到 column3 (如果陣列比索引編號短，則將設定 NULL 值)。

使用新增目前時間戳記轉換

新增目前時間戳記轉換可讓您以處理資料的時間來標記資料列。這對於稽核目的或追蹤資料管道中的延遲非常實用。您可以將此新資料欄新增為時間戳記資料類型或格式化字串。

若要新增「新增目前時間戳記」轉換：

1. 開啟資源面板，然後選擇新增目前時間戳記，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。



3. (選用) 在轉換索引標籤上，輸入新資料欄的自訂名稱，如果您希望將資料欄設定為格式化日期字串，請輸入格式。

使用「樞紐資料列至資料欄」轉換

樞紐資料列至資料欄轉換可讓您透過旋轉將變為新資料欄之所選新資料欄上的唯一值來彙總數值資料欄 (如果選取多個資料欄，則會串連這些值以命名新資料欄)。如此就可以合併資料列，同時為每個唯一值提供包含部分彙總的更多資料欄。例如，如果您有按月份和國家/地區劃分的以下銷售額資料集 (經過排序以便說明)：

year	month	國家/地區	amount
2020	Jan	uk	32
2020	Jan	de	42
2020	Jan	us	64
2020	Feb	uk	67
2020	Feb	de	4
2020	Feb	de	7
2020	Feb	us	6

year	month	國家/地區	amount
2020	Feb	us	12
2020	Jan	us	90

如果您將金額和國家樞紐操作為彙總資料欄，則會從原始國家資料欄建立新資料欄。在下表中，您會看到新資料欄 de、uk 和 us，而非國家資料欄。

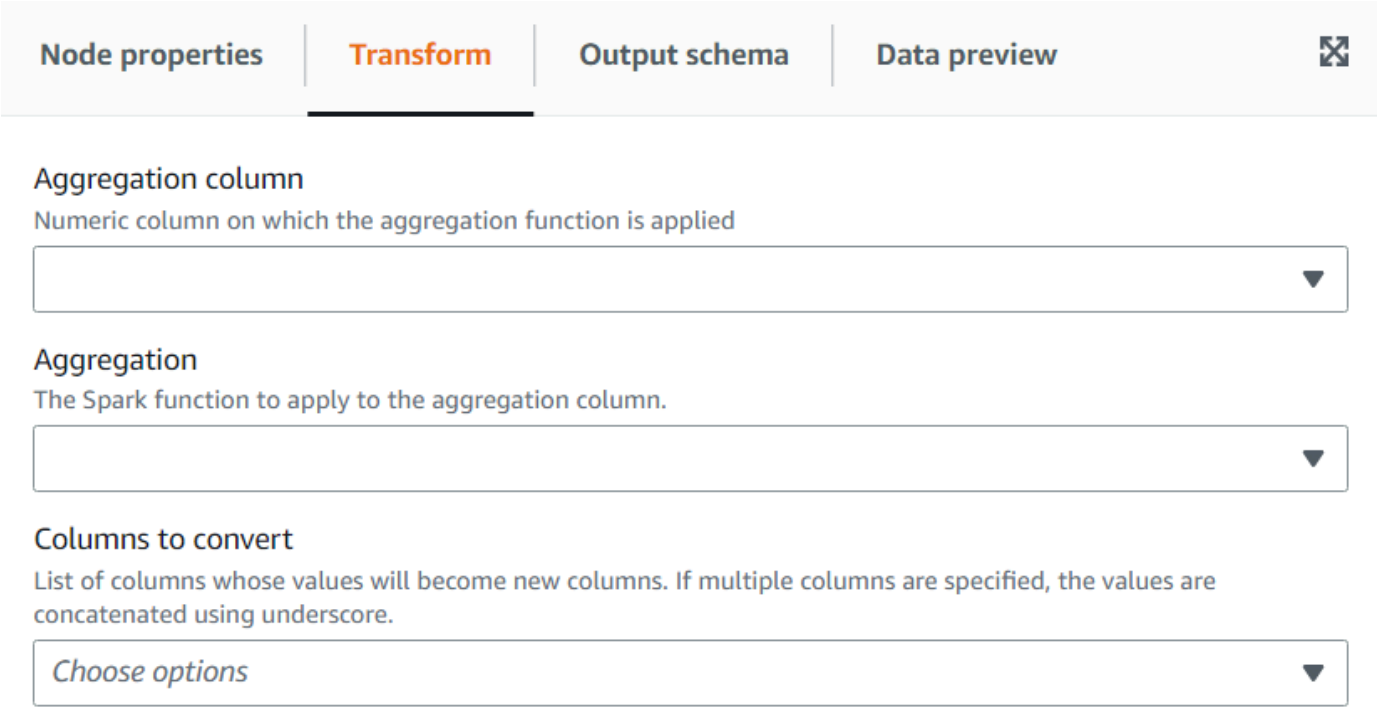
year	month	de	uk	us
2020	Jan	42	32	64
2020	Jan	11	67	18
2021	Jan			90

如果您想要對月份和縣市進行樞紐操作，則會為這些資料欄的每個值組合取得一個資料欄：

year	Jan_de	Jan_uk	Jan_us	Feb_de	Feb_uk	Feb_us
2020	42	32	64	11	67	18
2021			90			

若要新增「樞紐資料列至資料欄」轉換：

1. 開啟資源面板，然後選擇樞紐資料列至資料欄，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 在轉換索引標籤上，選擇要彙總以產生新資料欄值的數值資料欄、要套用的彙總函數，以及將其唯一值轉換為新資料欄的資料欄。



使用「取消樞紐資料欄至資料列」轉換

取消樞紐轉換允許您將資料欄轉換為新資料欄的值，並為每個唯一值產生一個資料列。這與樞紐相反，但請注意，它不是等效的，因為它不能將具有彙總的相同值之資料列分開，也不能將組合拆分為原始資料欄 (您可以稍後使用「分割」轉換來執行此操作)。例如，如果您有下列資料表：

year	month	de	uk	us
2020	Jan	42	32	64
2020	Feb	11	67	18
2021	Jan			90

您可以取消資料欄樞紐：“de”、“uk” 和 “us” 至值為 “amount” 的資料欄 “country” 中，並取得以下內容 (為了說明目的而在此處排序)：

year	month	country	amount
2020	Jan	uk	32

year	month	country	amount
2020	Jan	de	42
2020	Jan	us	64
2020	Feb	uk	67
2020	Feb	de	11
2020	Feb	us	18
2021	Jan	us	90

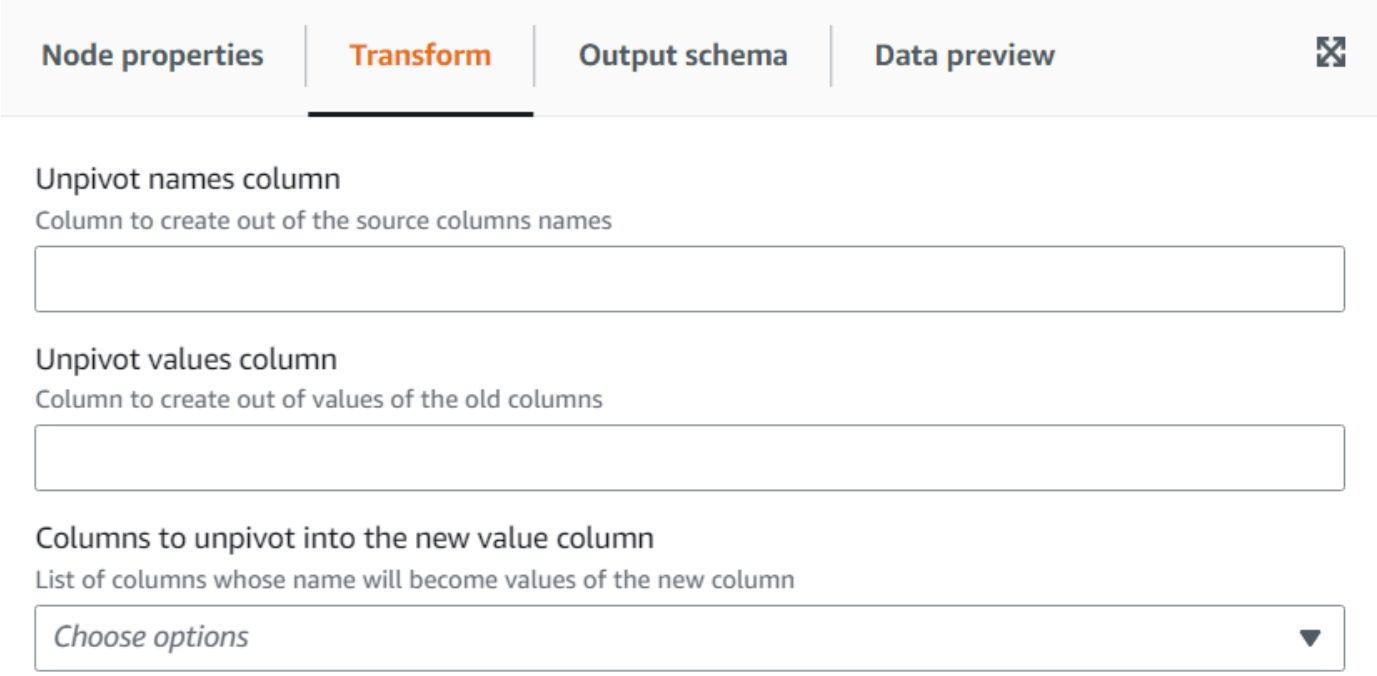
請注意，預設不會產生具有 NULL 值的資料欄 (“de” 和 “uk of Jan 2021”)。您可以啟用該選項以取得：

year	month	country	amount
2020	Jan	uk	32
2020	Jan	de	42
2020	Jan	us	64
2020	Feb	uk	67
2020	Feb	de	11
2020	Feb	us	18
2021	Jan	us	90
2021	Jan	de	
2021	Jan	uk	

若要新增「取消樞紐資料欄至資料列」轉換：

1. 開啟資源面板，然後選擇取消樞紐資料欄至資料列，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。

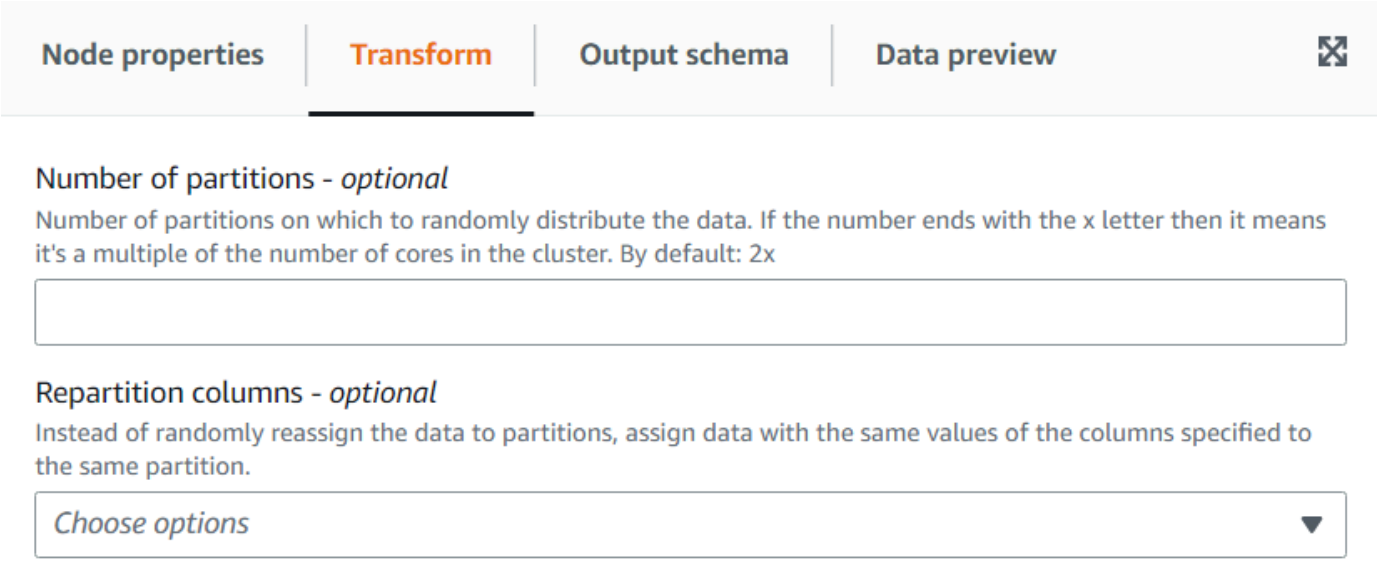
2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 在轉換索引標籤上，輸入要建立的新資料欄，以保留要取消樞紐之所選資料欄的名稱和值。



使用「自動平衡處理」轉換來最佳化您的執行期

自動平衡處理轉換會在工作者之間重新分配資料，以達到更好的效能。這對資料不平衡或其來源不允許進行足夠平行處理的情況很有幫助。在來源被壓縮或為 JDBC 的情況下，這很常見。資料的重新分配具有適度的效能成本，因此如果資料已經很平衡，最佳化可能無法減省工作量。在下方，該轉換使用 Apache Spark 重新分割，在多個最適合叢集容量的分割區之間隨機重新指派資料。高級使用者可以手動輸入多個分割區。此外，它還可以根據指定的資料欄重新整理資料，以最佳化分割資料表的寫入。這會產生更緊湊的輸出檔案。

1. 開啟資源面板，然後選擇自動平衡處理，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. (選用) 在轉換索引標籤上，您可以輸入數個分割區。一般而言，建議讓系統決定此值，但是如果您需要控制此值，可以調整乘數或輸入特定值。如果要儲存依資料欄分割的資料，您可以選擇與重新分割資料欄相同的資料欄。這樣一來，它將最大限度地減少每個分割區上的檔案數量，並避免在每個分割區內留下很多檔案，這將影響查詢該資料之工具的效能。



使用「衍生資料欄」轉換合併其他資料欄

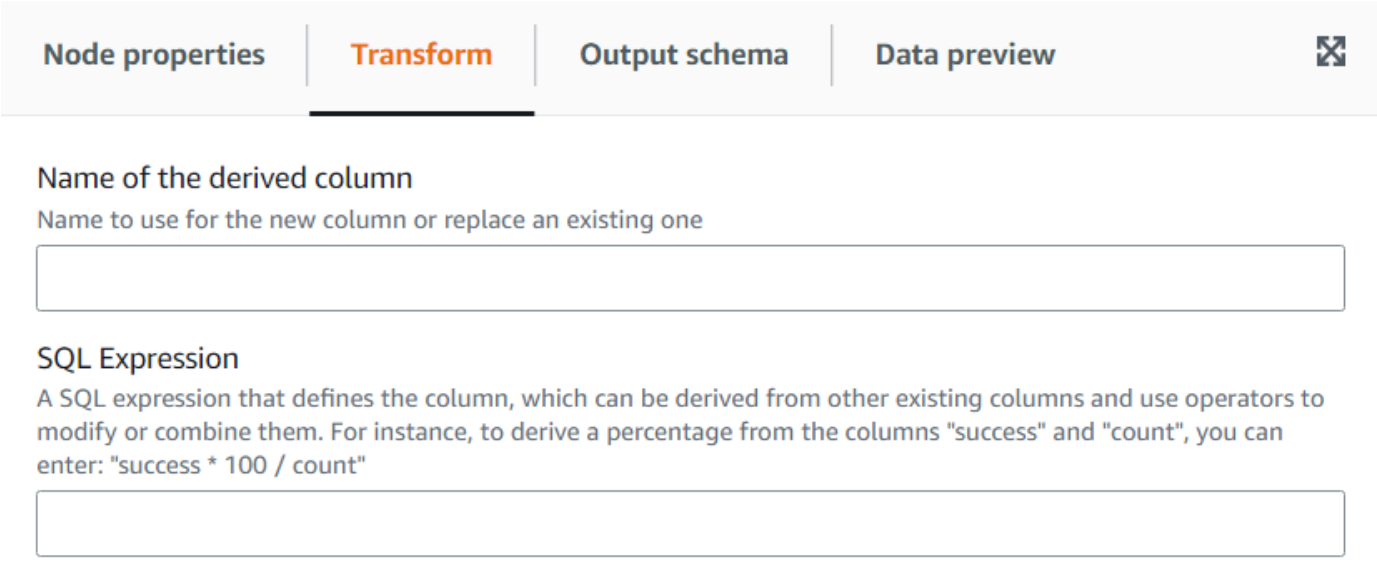
衍生資料欄轉換可讓您根據數學公式或 SQL 運算式定義新資料欄，您可以在其中使用資料中的其他資料欄以及常數和常值。例如，若要從 "success" 和 "count" 資料欄衍生 “percentage” 資料欄，您可以輸入 SQL 運算式："success * 100 / count || '%'"。

範例結果：

success	count	百分比
14	100	14%
6	20	3%
3	40	7.5%

若要新增「衍生資料欄」轉換：

1. 開啟資源面板，然後選擇 衍生資料欄，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 在轉換索引標籤上，輸入資料欄的名稱及其內容的運算式。



使用「查詢」轉換從型錄資料表新增相符資料

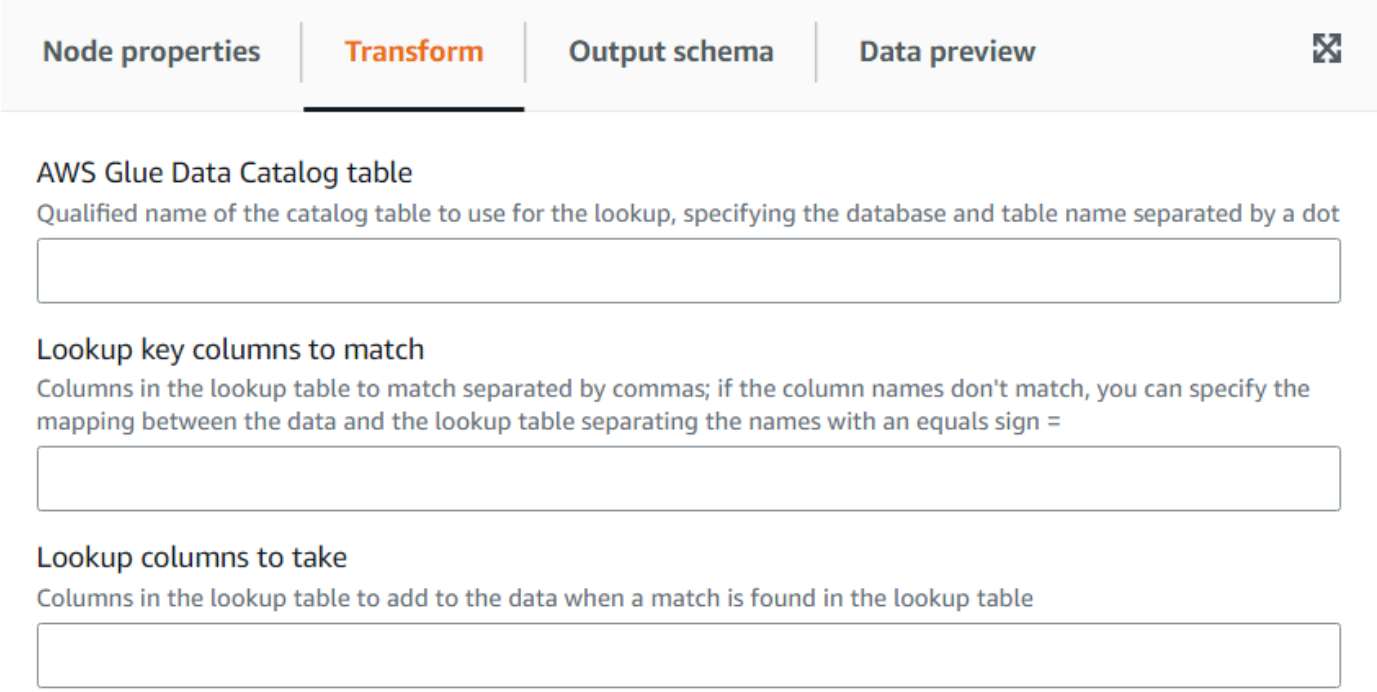
當索引鍵符合資料中定義的查詢資料欄時，查詢轉換可讓您從定義的型錄資料表新增資料欄。這等同於在資料與用作條件相符資料欄的查詢資料表之間進行左外部聯結。

若要新增「查詢」轉換：

1. 開啟資源面板，然後選擇查詢，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 在轉換索引標籤上，輸入用於執行查詢的完整型錄資料表名稱。例如，如果您的資料庫是 “mydb”，資料表是 “mytable”，則輸入 “mydb.mytable”。然後輸入條件以在查詢資料表中尋找相符項目 (如果組成了查詢索引鍵)。輸入以逗號分隔的索引鍵資料欄清單。如果一個或多個索引鍵資料欄的名稱不相同，則需要定義相符映射。

   例如，如果資料的資料欄是 “user_id” 和 “region”，並且在使用者資料表中，對應的資料欄名為 “id” 和 “region”，則在要相符的資料欄欄位中輸入：“user_id=id, region”。您可以執行 region=region 的動作，但並非必要，因為它們是相同的。
4. 最後，輸入要從查詢資料表中相符的資料列引入的資料欄，以將它們合併到資料中。如果沒有找到相符項目，這些資料欄將設定為 NULL。

> **Note**
>
> 在查詢轉換下，它使用左聯結以提高效率。如果查詢資料表具有複合索引鍵，請確保將要相符的資料欄設定為與所有索引鍵資料欄相符，以便僅產生一個相符項目。否則，將有多個查詢資料列相符，這將導致為每個相符項目新增額外的資料列。



使用「將陣列或映射分解成資料列」轉換

分解轉換可讓您將巢狀結構中的值擷取到更容易操作的個別資料列。對於陣列，該轉換會為陣列的每個值產生一個資料列，並複寫資料列中其他資料欄的值。對於映射，該轉換將為每個條目產生一個資料列，其中索引鍵和值作為資料欄加上該資料列中的任何其他資料欄。

例如，如果此資料集中的「類別」陣列資料欄具有多個值。

product_id	category
1	[sports, winter]
2	[garden, tools]

product_id	category
3	[videogames]
4	[game, boardgame, social]
5	[]

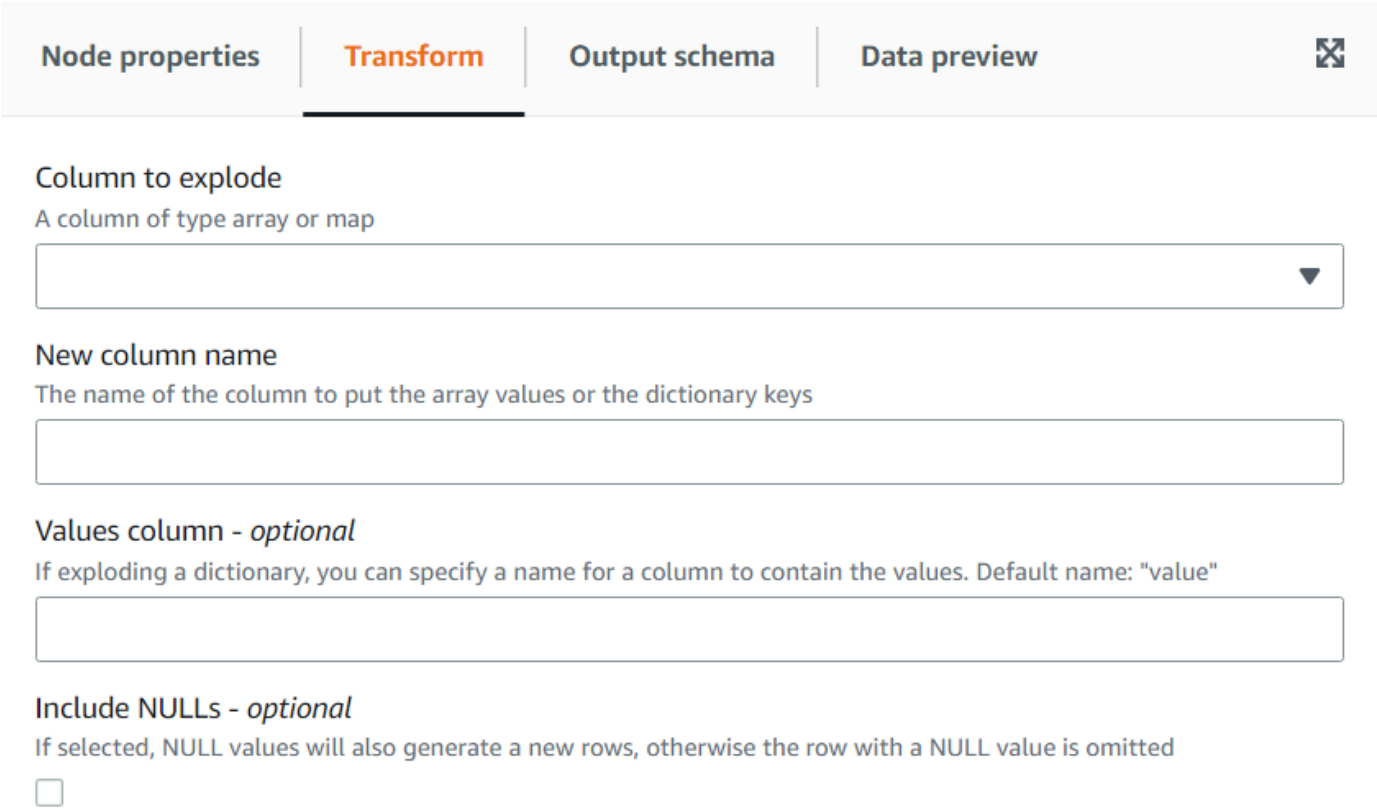
如果將「類別」資料欄分解為具有相同名稱的資料欄，則會覆寫該資料欄。您可以選取要包含 NULL 以取得下列項目 (為了說明目的而排序)：

product_id	category
1	sports
1	winter
2	garden
2	tool
3	videogames
4	game
4	boardgame
4	social
5	

若要新增「將陣列或映射分解成資料列」轉換：

1. 開啟資源面板，然後選擇將陣列或映射分解成資料列，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. (選用) 在節點屬性索引標籤上，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。

3. 在轉換索引標籤上，選擇要分解的資料欄 (它必須是陣列或映射類型)。然後輸入陣列項目的資料欄名稱，或輸入索引鍵和值的資料欄名稱 (如果您要分解映射)。
4. (選用) 在轉換索引標籤上，依預設，如果要分解的資料欄為 NULL 或具有空白結構，則在分解的資料集中會省略它。如果您想保留資料列 (新資料欄為 NULL)，請勾選「包含 NULL」。

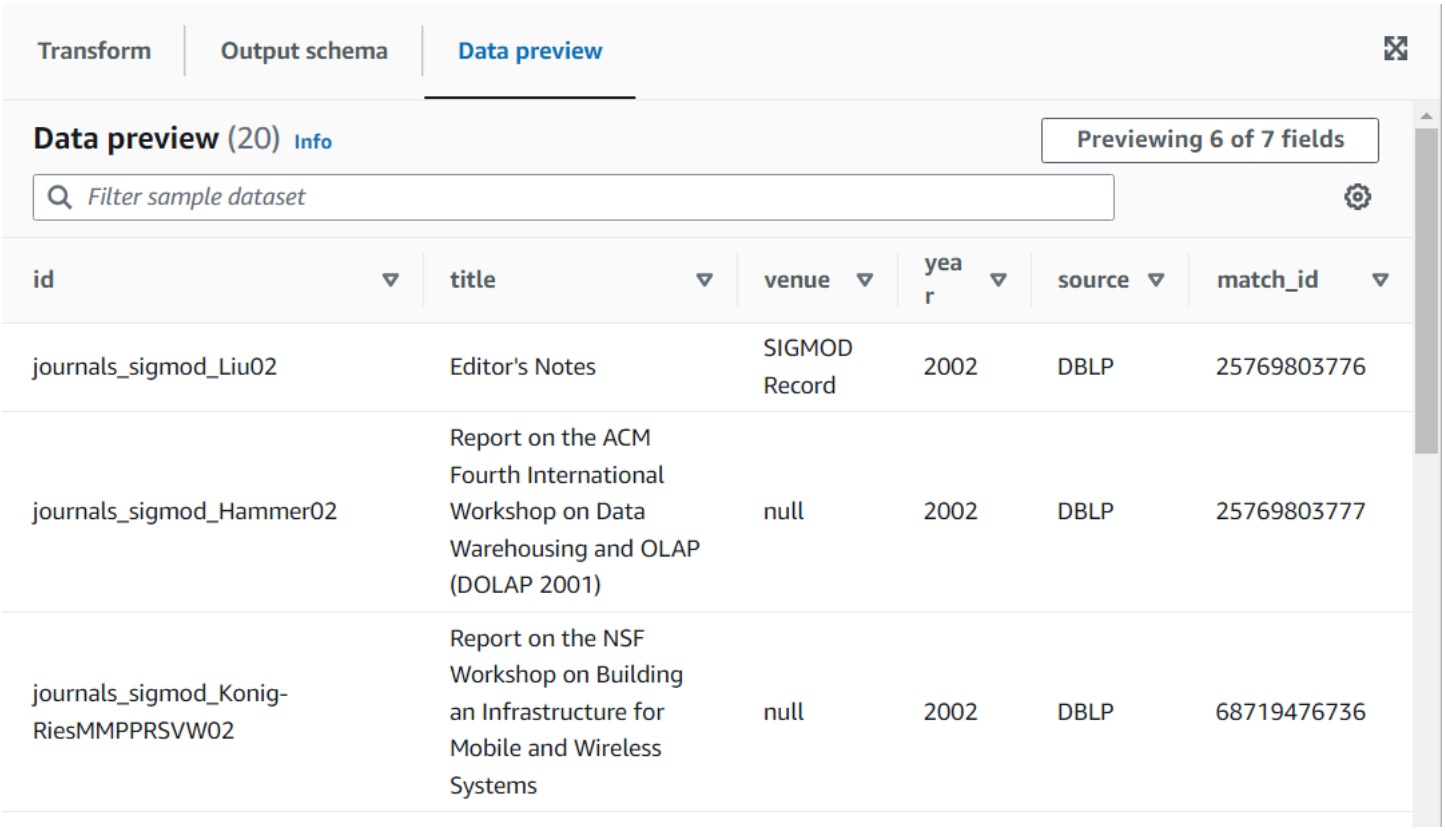


使用「記錄比對」轉換來調用現有的資料分類轉換

此轉換會調用現有的記錄比對機器學習資料分類轉換。

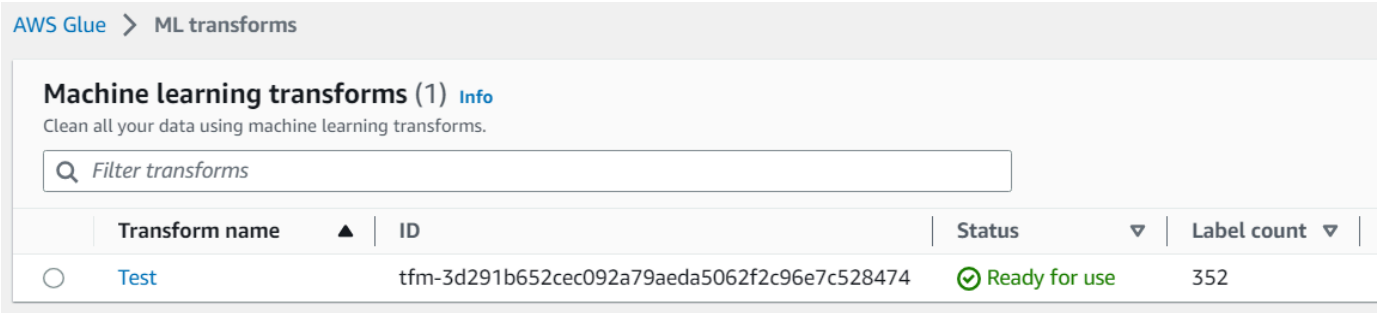
轉換會根據標籤，針對訓練過的模型來評估目前的資料。新增資料欄 "match_id"，以根據演算法訓練將每個資料列指派給一組視為等效的項目。如需詳細資訊，請參閱 Record matching with Lake Formation FindMatches。

> **Note**
>
> 視覺化任務使用的 AWS Glue 版本必須與 AWS Glue 用來建立「記錄比對」轉換的版本相符。



將「記錄比對」轉換節點新增至您的任務圖表

1. 開啟資源面板，然後選擇記錄比對，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. 在節點屬性面板中，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 在轉換索引標籤上，輸入從機器學習轉換頁面擷取的 ID：



4. (選用) 在轉換索引標籤上，您可以勾選新增可信度分數的選項。以額外運算為代價，該模型將估計每個相符項目的可信度分數，並將其作為附加資料欄。

移除 Null 資料列

此轉換會移除所有資料欄皆為 Null 的資料集資料列。此外，您可以擴展此條件以包含空白欄位，以便在至少有一個非空白資料欄的情況下保留資料列。

將「移除 Null 資料列」轉換節點新增至您的任務圖表

1. 開啟資源面板，然後選擇移除 Null 資料列，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. 在節點屬性面板中，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. (選用) 如果您想要求非 Null 值且非空白的資料列，可在轉換索引標籤上勾選擴展選項，這樣就會在此轉換中將空字串、陣列或映射視為 Null 值。

剖析包含 JSON 資料的字串資料欄

此轉換會剖析包含 JSON 資料的字串資料欄，並將其轉換為結構或陣列資料欄，具體取決於 JSON 是物件還是陣列。(選用) 您可以保留已剖析資料欄和原始資料欄。

您可以使用選擇性的取樣來提供或推斷 JSON 結構描述 (若為 JSON 物件)。

將「剖析 JSON 資料欄」轉換節點新增至您的任務圖表

1. 開啟資源面板，然後選擇剖析 JSON 資料欄，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. 在節點屬性面板中，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 在轉換索引標籤上，選取包含 JSON 字串的資料欄。
4. (選用) 在轉換索引標籤上，使用 SQL 語法輸入 JSON 資料所遵循的結構描述，例如：如果是物件，則為 "field1 STRING, field2 INT"；如果是陣列，則為 "ARRAY<STRING>"。

   如果是陣列就需要結構描述，但如果是物件，若未指定結構描述，則將使用資料來推斷結構描述。若要減少推斷結構描述的影響 (特別是在大型資料集上)，您可以輸入用於推斷結構描述的範例比例，以避免讀取整筆資料兩次。如果值小於 1，則會使用隨機範例的對應比例來推斷結構描述。如果資料可靠且物件在資料列之間保持一致，則可以使用小比例 (例如 0.1) 來改善效能。
5. (選用) 如果要同時保留原始字串資料欄和已剖析的資料欄，您可以在轉換索引標籤上輸入新的資料欄名稱。

擷取 JSON 路徑

此轉換會從 JSON 字串資料欄擷取新資料欄。當您只需要幾個資料元素，而且不想將整個 JSON 內容匯入資料表結構描述時，此轉換非常有用。

將「擷取 JSON 路徑」轉換節點新增至您的任務圖表

1. 開啟資源面板，然後選擇擷取 JSON 路徑，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. 在節點屬性面板中，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 在轉換索引標籤上，選取包含 JSON 字串的資料欄。輸入一個或多個以逗號分隔的 JSON 路徑運算式，每個運算式都會參考如何從 JSON 陣列或物件中擷取值。例如，如果 JSON 資料欄包含具有屬性 "prop_1" 和 "prop2" 的物件，則可同時擷取兩者，並指定它們的名稱 "prop_1, prop_2"。

   如果 JSON 欄位具有特殊字元，若要從此 JSON `{"a. a": 1}` 中擷取屬性，可以使用 `$['a. a']` 路徑。例外狀況為逗號，因為它被保留用來分隔路徑。然後輸入每個路徑的對應資料欄名稱，並以逗號分隔。
4. (選用) 在轉換索引標籤上，您可以選擇在擷取後刪除 JSON 資料欄，這在擷取所需的部分後不需要其餘 JSON 資料時很有意義。

使用規則運算式擷取字串片段

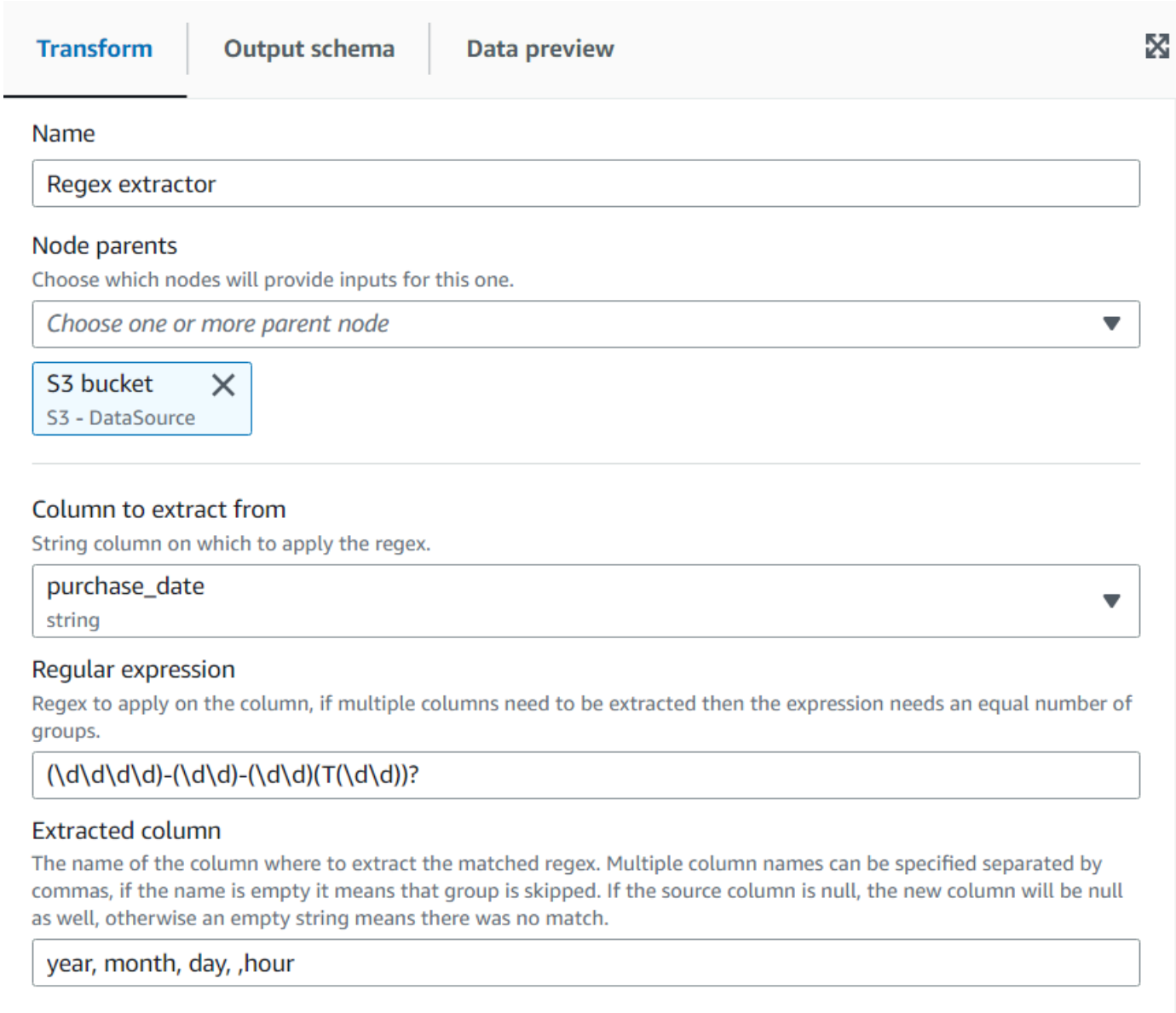
此轉換使用規則運算式擷取字串片段，並從中建立新資料欄，或使用 Regex 群組建立多個資料欄。

將「Regex 擷取器」轉換節點新增至您的任務圖表

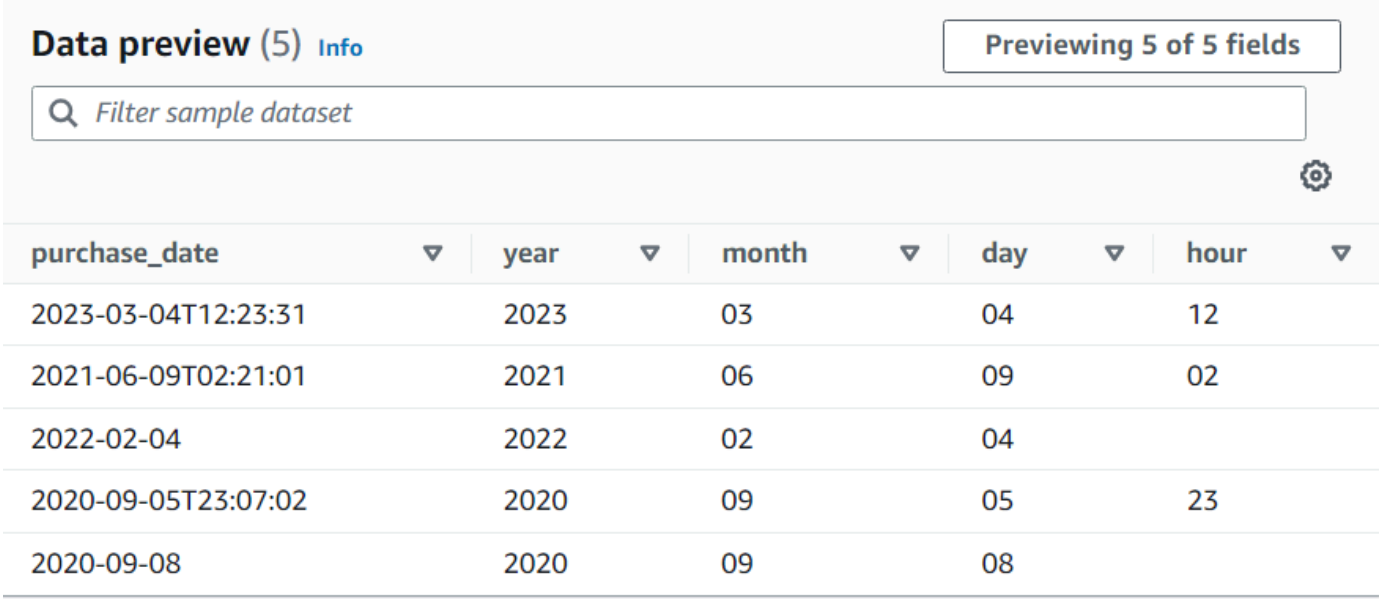
1. 開啟資源面板，然後選擇 Regex 擷取器，將新轉換新增至您的任務圖表。新增節點時選取的節點將成為其父節點。
2. 在節點屬性面板中，您可以輸入任務圖表中節點的名稱。如果尚未選取節點父項，請從節點父項清單選擇用作轉換之輸入來源的節點。
3. 在轉換索引標籤上，輸入規則運算式及其需要套用的資料欄。然後輸入要存放相符字串的新資料欄名稱。只有當來源資料欄為 Null 時，新資料欄才會是 Null。如果 Regex 不相符，則資料欄將為空白。

   如果 Regex 使用群組，則會有一個用逗號分隔的對應資料欄名稱，但是您可以透過將資料欄名稱保留為空白來跳過群組。

例如，如果您有一個資料欄為 "purchase_date"，其中包含使用長和短 ISO 日期格式的字串，然後您想要擷取年、月、日和小時 (如果可用)。請注意小時群組為選用，否則在不可用的資料列中，擷取的所有群組都將是空白字串 (因為 Regex 不相符)。在這種情況下，我們不希望群組使時間可選，而是讓其成為內部時間，因此我們將名稱留空，使其不會被擷取 (該群組將包括 T 字元)。



產生資料預覽：



建立自訂轉換

如果您需要對資料執行更複雜的轉換，或是想要將資料屬性索引鍵新增至資料集，您可以將自訂程式碼轉換新增至您的任務圖表。自訂程式碼節點可讓您輸入執行轉換的指令碼。

使用自訂程式碼時，您必須使用結構描述編輯器來指出透過自訂程式碼對輸出所做的變更。編輯結構描述時，您可以執行下列動作：

- 新增或移除資料屬性索引鍵
- 變更資料屬性索引鍵的資料類型
- 變更資料屬性索引鍵的名稱
- 重新建構巢狀屬性索引鍵

您必須使用 SelectFromCollection 轉換以從自訂轉換節點的結果選擇單一 `DynamicFrame`，然後才能將輸出傳送到目標位置。

使用下列任務，將自訂轉換節點新增到您的任務圖表。

將自訂程式碼轉換節點新增至任務圖表

將自訂轉換節點新增到您的任務圖表

1. (選用) 根據需要開啟資源面板，然後選擇自訂轉換，將自訂轉換新增至您的任務圖表。

2. 在 Node properties (節點屬性) 索引標籤上，輸入任務圖表中節點的名稱。如果尚未選取節點父項，或者如果您想要進行自訂轉換的多個輸入，請從節點父項清單選擇用作轉換之輸入來源的節點。

輸入自訂轉換節點的程式碼

您可以在輸入欄位中輸入或複製程式碼。任務會使用此程式碼來執行資料轉換。您可以提供 Python 或 Scala 的程式碼片段。程式碼應該需要一個或多個 DynamicFrames 作為輸入，並會傳回 DynamicFrames 的集合。

輸入自訂轉換節點的指令碼

1. 在任務圖表中選取自訂轉換節點後，選擇轉換索引標籤。
2. 在程式碼區塊標題下的文字輸入欄位，貼上或輸入轉換的程式碼。您使用的程式碼必須符合任務詳細資訊索引標籤指定的語言。

   在程式碼中引用輸入節點時，AWS Glue Studio 會根據建立的順序循序命名任務圖表節點傳回的 DynamicFrames。在程式碼中使用下列其中一種命名方法：

   - 經典程式碼產生 – 使用功能性名稱來引用任務圖表中的節點。
     - 資料來源節點：DataSource0、DataSource1、DataSource2，以此類推。
     - 轉換節點：Transform0、Transform1、Transform2，以此類推。
   - 新程式碼產生 – 使用節點的節點屬性索引標籤，附加 '_node1'、'_node2'，以此類推。例如，S3bucket_node1、ApplyMapping_node2、S3bucket_node2、MyCustomNodeName_node

   如需新程式碼產生器的詳細資訊，請參閱 指令碼程式碼產生。

下列範例展示要在程式碼方塊中輸入的程式碼格式：

Python

下面的範例採用第一個接收的 DynamicFrame、將其轉換成 DataFrame 來套用原生篩選方法 (只保留超過 1000 個投票的記錄)，然後將其轉換回 DynamicFrame，然後再傳回它。

```
def FilterHighVoteCounts (glueContext, dfc) -> DynamicFrameCollection:
    df = dfc.select(list(dfc.keys())[0]).toDF()
    df_filtered = df.filter(df["vote_count"] > 1000)
    dyf_filtered = DynamicFrame.fromDF(df_filtered, glueContext, "filter_votes")
```

```
    return(DynamicFrameCollection({"CustomTransform0": dyf_filtered}, glueContext))
```

Scala

下面的範例採用第一個接收的 DynamicFrame、將其轉換成 DataFrame 來套用原生篩選方法 (只保留超過 1000 個投票的記錄)，然後將其轉換回 DynamicFrame，然後再傳回它。

```
object FilterHighVoteCounts {
  def execute(glueContext : GlueContext, input : Seq[DynamicFrame]) :
 Seq[DynamicFrame] = {
    val frame = input(0).toDF()
    val filtered = DynamicFrame(frame.filter(frame("vote_count") > 1000),
 glueContext)
    Seq(filtered)
  }
}
```

編輯自訂轉換節點的結構描述

當您使用自訂轉換節點時，AWS Glue Studio 不會自動推斷由轉換建立的輸出結構描述。您可以使用結構描述編輯器來描述由自訂轉換程式碼所實作的結構描述變更。

自訂程式碼節點可以有任意數量的父節點，每個節點都會提供 DynamicFrame 作為自訂程式碼的輸入。自訂程式碼節點會傳回 DynamicFrames 的集合。用作輸入的每個 DynamicFrame 會有相關聯的結構描述。您必須新增用於描述每個由自訂程式碼節點傳回之 DynamicFrame 的結構描述。

> **Note**
>
> 當您在自訂轉換上設定自己的結構描述時，AWS Glue Studio 不會從以前的節點繼承結構描述。若要更新結構描述，請選擇自訂轉換節點，然後選擇資料預覽索引標籤。產生預覽後，選擇 'Use Preview Schema'。然後，結構描述將會由使用預覽資料的結構描述替換。

編輯自訂轉換節點的輸出結構描述

1. 在任務圖表中選取自訂轉換節點後，在節點詳細資訊面板中，選擇輸出結構描述索引標籤。
2. 選擇編輯以對結構描述進行變更。

   如果您有巢狀資料屬性索引鍵 (例如陣列或物件)，您可以選擇每個結構描述面板右上方的展開列圖示

()，
以展開子資料屬性索引鍵的清單。選擇此圖示後，它會變更為摺疊列圖示
()，
您可以選擇摺疊子屬性索引鍵清單。

3. 使用頁面右側區段中的下列動作來修改結構描述：

   - 若要重新命名屬性索引鍵，請將游標放在屬性索引鍵的索引鍵文字方塊，然後輸入新名稱。
   - 若要變更屬性索引鍵的資料類型，請使用清單來選擇屬性索引鍵的新資料類型。
   - 若要將新的頂層屬性索引鍵新增至結構描述，請選擇位於取消按鈕左側的溢位
     ()
     圖示，然後選擇新增根索引鍵。
   - 若要將子屬性索引鍵新增至結構描述，請選擇與父索引鍵相關聯新增索引鍵圖示
      。
     輸入子索引鍵的名稱，然後選擇資料類型。
   - 若要從結構描述移除屬性索引鍵，請選擇索引鍵名稱最右側的移除圖示
     ()。

4. 如果您的自訂轉換程式碼使用多個 DynamicFrames，您可以新增其他輸出結構描述。

   - 若要新增空白結構描述，請選擇溢位
     ()
     圖示，然後選擇新增輸出結構描述。
   - 若要將現有結構描述複製到新的輸出結構描述，請確定要複
     製的結構描述顯示在結構描述選取器中。選擇溢位
     ()
     圖示，然後選擇複製。

   如果要移除輸出結構描述，請確定要複製的結構描述顯示在結構描述選取器中。選擇溢位
   ()
   圖示，然後選擇刪除。

5. 新增根索引鍵至新的結構描述或編輯重複的索引鍵。

6. 當您修改輸出結構描述時，請選擇套用按鈕以儲存您的變更，並結束結構描述編輯器。

   如果您不想儲存您的變更，請選擇取消按鈕。

設定自訂轉換輸出

自訂程式碼轉換會傳回 DynamicFrames 集合，即使結果集內只有一個 DynamicFrame。

處理自訂轉換節點的輸出

1. 新增SelectFromCollection轉換節點，該節點具有自訂轉換節點做為其父節點。更新此轉換以指示您想要使用的資料集。如需詳細資訊，請參閱使用 SelectFromCollection 選擇要保留的資料集。
2. 如果您想使用自訂轉換節點產生的其他 DynamicFrames，請新增其他 SelectFromCollection 轉換至任務圖表。

   請試想下列場景：您新增自訂轉換節點，將航班資料集分割成多個資料集，但在每個輸出結構描述中複製某些識別屬性索引鍵，例如航班日期或航班號碼。您可以為每個輸出結構描述新增 SelectFromCollection 轉換節點，並將自訂轉換節點做為其父項。
3. (選用) 然後，您可以使用每個 SelectFromCollection 將節點轉換為任務中其他節點的輸入，或作為資料目標節點的父節點。

AWS Glue 自訂視覺化轉換

自訂視覺化轉換允許您建立轉換並使其可用於 AWS Glue Studio 任務。自訂視覺化轉換讓可能不熟悉編寫程式碼的 ETL 開發人員，使用 AWS Glue Studio 介面搜尋和使用不斷增長的轉換程式庫。

您可以建立自訂視覺化轉換，然後將其上傳到 Amazon S3，以便透過 AWS Glue Studio 中的視覺化編輯器加以使用，從而處理這些任務。

主題

- 開始使用自訂視覺化轉換
- 步驟 1. 建立 JSON 組態檔案
- 步驟 2. 實作轉換邏輯
- 步驟 3. 在 AWS Glue Studio 中驗證自訂視覺化轉換並進行疑難排解
- 步驟 4. 視需要更新自訂視覺化轉換
- 步驟 5. 在 AWS Glue Studio 中使用自訂視覺化轉換
- 使用範例
- 自訂視覺化指令碼範例
- 影片

開始使用自訂視覺化轉換

若要建立自訂視覺化轉換，請執行以下步驟。

- 步驟 1. 建立 JSON 組態檔案
- 步驟 2. 實作轉換邏輯
- 步驟 3. 驗證自訂視覺化轉換
- 步驟 4. 視需要更新自訂視覺化轉換
- 步驟 5. 在 AWS Glue Studio 中使用自訂視覺化轉換

透過設定 Amazon S3 儲存貯體開始使用，然後繼續步驟 1. 建立 JSON 組態檔案。

先決條件

客戶提供的轉換資料位於客戶 AWS 帳戶內。該帳戶擁有轉換，因此擁有檢視 (搜尋和使用)、編輯或刪除轉換的所有許可。

為了在 AWS Glue Studio 中使用自訂轉換，您需要建立兩個檔案並將其上傳到該 AWS 帳戶中的 Amazon S3 資產儲存貯體：

- Python 檔案 - 包含轉換函數
- JSON 檔案 – 描述轉換。這也稱為定義轉換所需的組態檔案。

為了將檔案配對在一起，請對兩者使用相同名稱。例如：

- myTransform.json
- myTransform.py

或者，您可以透過提供包含圖示的 SVG 檔案，為自訂視覺化轉換指定自訂圖示。為了將檔案配對在一起，請對圖示使用相同名稱：

- myTransform.svg

AWS Glue Studio 將使用各自的檔案名稱自動比對它們。任何現有模組的檔案名稱皆不能相同。

轉換檔案名稱的建議慣例

AWS Glue Studio 會將檔案作為模組 (例如 `import myTransform`) 匯入任務指令碼中。因此，您的檔案名必須遵循為 Python 變數名稱 (識別符) 設定的相同命名規則。具體來說，名稱必須以字母或底線開頭，且完全由字母、數字和/或底線組成。

> **Note**
>
> 確保轉換檔案名稱不會與現有的載入 Python 模組 (例如 `sys, array, copy` 等) 衝突，避免出現未預期的執行時間問題。

設定 Amazon S3 儲存貯體

您建立的轉換會存放在 Amazon S3 中，且由您的 AWS 帳戶擁有。只要將檔案 (json 和 py) 上傳到目前存放所有任務指令碼的 Amazon S3 資產資料夾 (例如 `s3://aws-glue-assets-<accountid>-<region>/transforms`)，即可建立新的自訂視覺化轉換。如果使用自訂圖示，也可以將其上傳。按預設，AWS Glue Studio 將從相同 S3 儲存貯體的 /transforms 資料夾中讀取所有 .json 檔案。

步驟 1. 建立 JSON 組態檔案

需要 JSON 組態檔案才能定義和描述您的自訂視覺化轉換。組態檔案的結構描述如下。

JSON 檔案結構

欄位

- `name: string` – (必要) 用來識別轉換的轉換系統名稱。遵循為 Python 變數名稱 (識別符) 設定的相同命名規則。具體來說，名稱必須以字母或底線開頭，且完全由字母、數字和/或底線組成。
- `displayName: string` – (選用) AWS Glue Studio 視覺化任務編輯器中顯示的轉換名稱。如果未指定 `displayName`，則 `name` 會用作 AWS Glue Studio 中的轉換名稱。
- `description: string` – (選用) 轉換描述會顯示在 AWS Glue Studio 中，且可供搜尋。
- `functionName: string` – (必要) Python 函數名稱用於識別要在 Python 指令碼中呼叫的函數。
- `path: string` – (選用) Python 來源檔案的完整 Amazon S3 路徑。如果未指定，則 AWS Glue 使用檔案名稱比對功能將 .json 和 .py 檔案配對在一起。例如，JSON 檔案的名稱 `myTransform.json` 會與位於同一個 Amazon S3 位置的 Python 檔案 `myTransform.py` 配對。
- `parameters: Array of TransformParameter object` – (選用) 在 AWS Glue Studio 視覺化編輯器中設定參數時顯示的參數清單。

TransformParameter 欄位

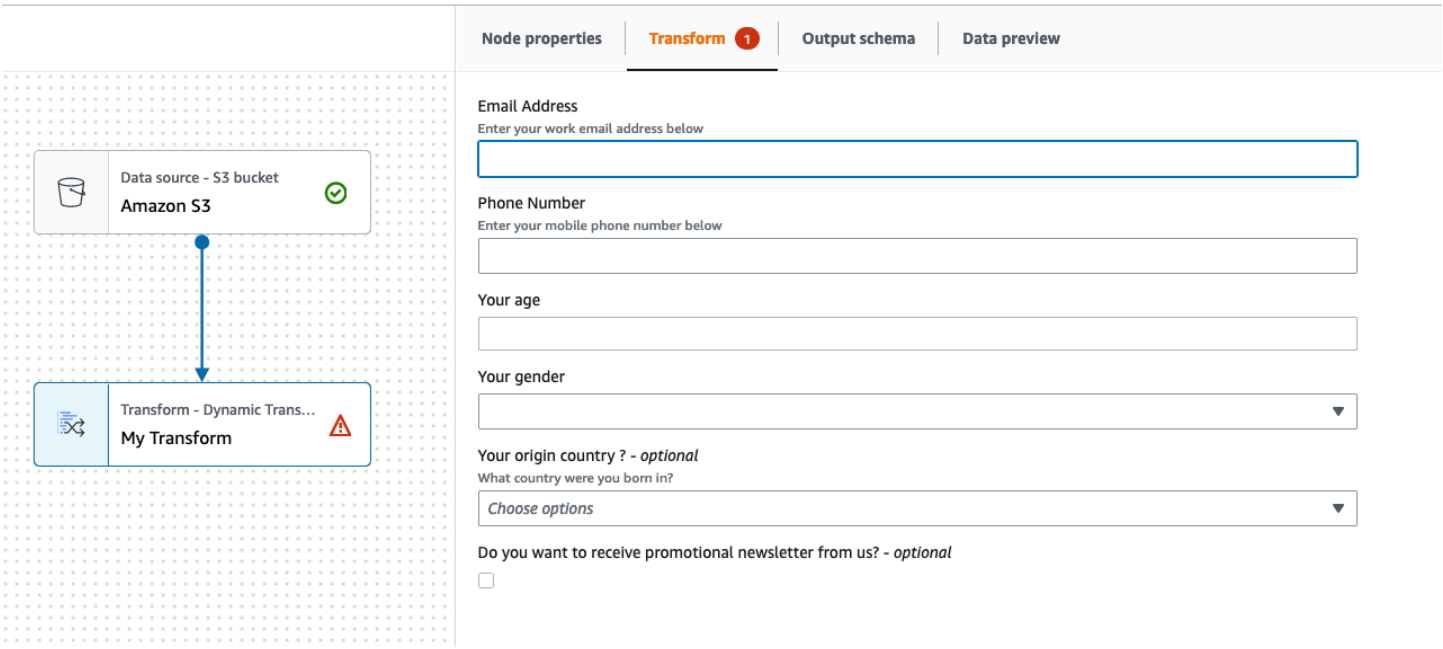
- `name: string` – (必要) 將作為任務指令碼中的具名引數傳遞至 Python 函數的參數名稱。遵循為 Python 變數名稱 (識別符) 設定的相同命名規則。具體來說，名稱必須以字母或底線開頭，且完全由字母、數字和/或底線組成。
- `displayName: string` – (選用) AWS Glue Studio 視覺化任務編輯器中顯示的轉換名稱。如果未指定 `displayName`，則 `name` 會用作 AWS Glue Studio 中的轉換名稱。
- `type: string` – (必要) 接受常見 Python 資料類型的參數類型。有效值： 'str' | 'int' | 'float' | 'list' | 'bool'。
- `isOptional: boolean`：(選用) 決定參數是否為選用參數。依預設，所有參數皆為必要參數。
- `description: string` – (選用) 描述會顯示在 AWS Glue Studio 中，可協助使用者設定轉換參數。
- `validationType: string` – (選用) 定義驗證此參數的方式。目前僅支援規則運算式。依預設，系統會將驗證類型設定為 `RegularExpression`。
- `validationRule: string` – (選用) 在 `validationType` 設定為 `RegularExpression` 時，用於在提交之前驗證表單輸入的規則運算式。規則運算式語法必須相容於 RegExp Ecmascript 規格。
- `validationMessage: string` – (選用) 驗證失敗時顯示的訊息。
- `listOptions: An array of TransformParameterListOption object` 或 `string` 或字串值 "column"：(選用) 用於在單選或複選 UI 控制中顯示的選項。接受逗號分隔值清單或 `TransformParameterListOption` 類型的強型別 JSON 物件。您也可以透過指定字串值 "column" 動態填入父節點結構描述中的資料欄清單。
- `listType: string` – (選用) 定義類型 = 'list' 的選項類型。有效值： 'str' | 'int' | 'float' | 'list' | 'bool'。接受常見 Python 資料類型的參數類型。

TransformParameterListOption 欄位

- `value: string | int | float | bool` – (必要) 選項值。
- `label: string` - (可選) 在選取下拉式選單中顯示的選項標籤。

AWS Glue Studio 中的轉換參數

依預設，除非參數在 .json 檔案中標記為 `isOptional`，否則參數為必要項。在 AWS Glue Studio 中，參數會顯示在 Transform (轉換) 索引標籤中。範例顯示使用者定義的參數，例如電子郵件地址、電話號碼、年齡、性別和原籍國家/地區。

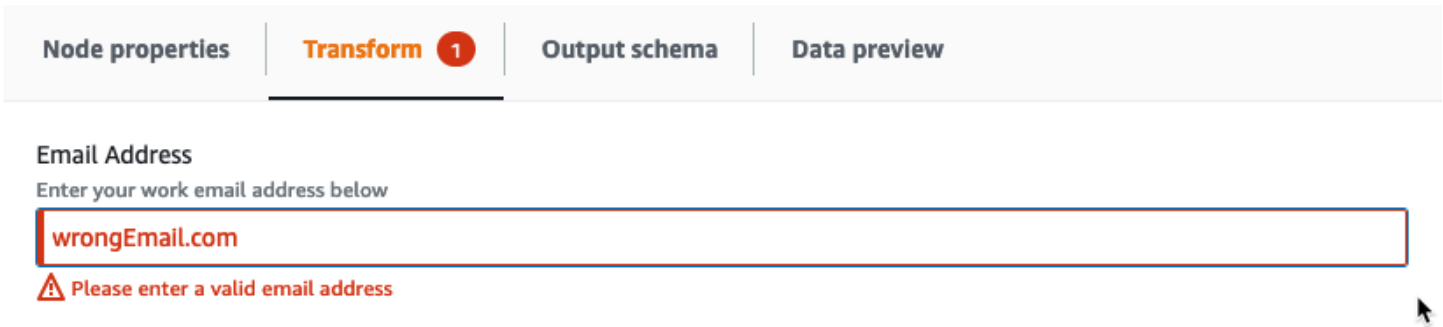


您可以在 `validationMessage` 中指定 `validationRule` 參數並指定驗證訊息，使用 json 檔案中的規則運算式強制執行 AWS Glue Studio 中的某些驗證。

```
    "validationRule": "^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})$",
    "validationMessage": "Please enter a valid US number"
```

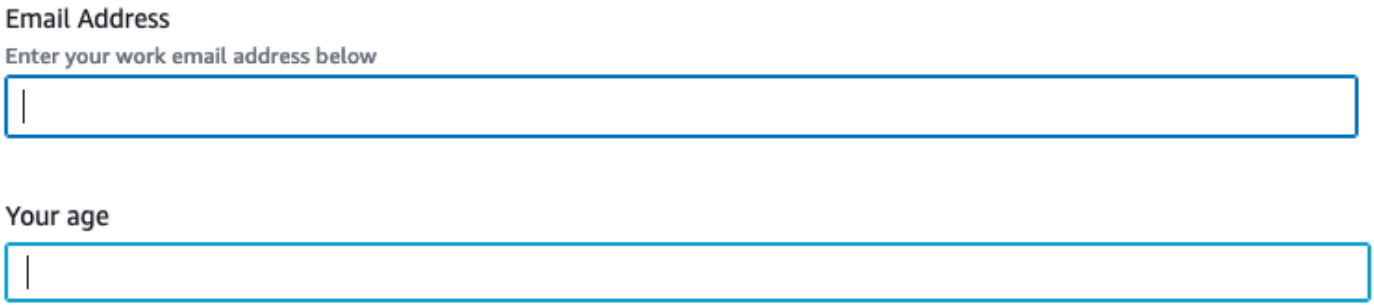
> **Note**
>
> 由於驗證發生在瀏覽器中，您的規則運算式語法必須相容於 RegExp Ecmascript 規格。這些規則運算式不支援 Python 語法。

新增驗證可防止使用者儲存具有錯誤之使用者輸入的任務。AWS Glue Studio 顯示驗證訊息，如範例所示：

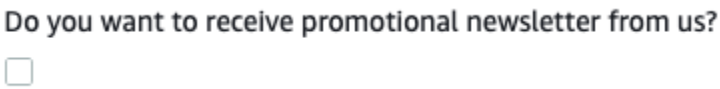


系統會根據參數組態在 AWS Glue Studio 中顯示參數。

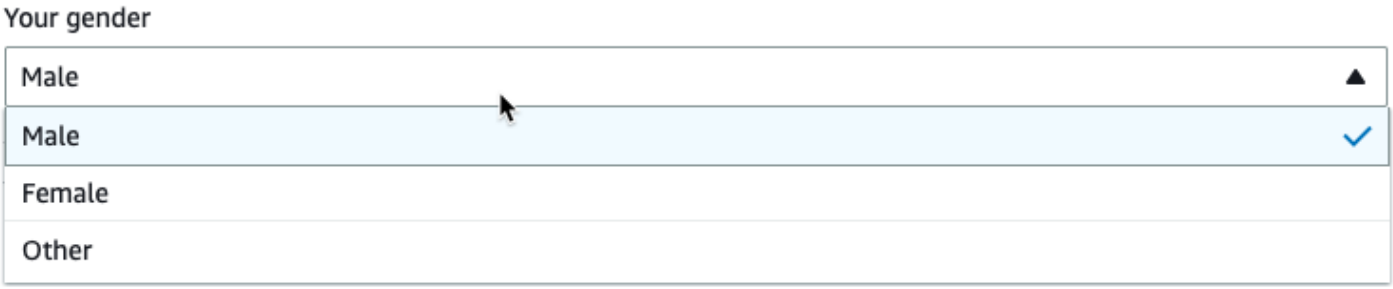
- 如果 `type` 是 `str`、`int` 或 `float` 中的任何一項，則會顯示文字輸入欄位。例如，螢幕擷取畫面會顯示「電子郵件地址」和「您的年齡」參數的輸入欄位。



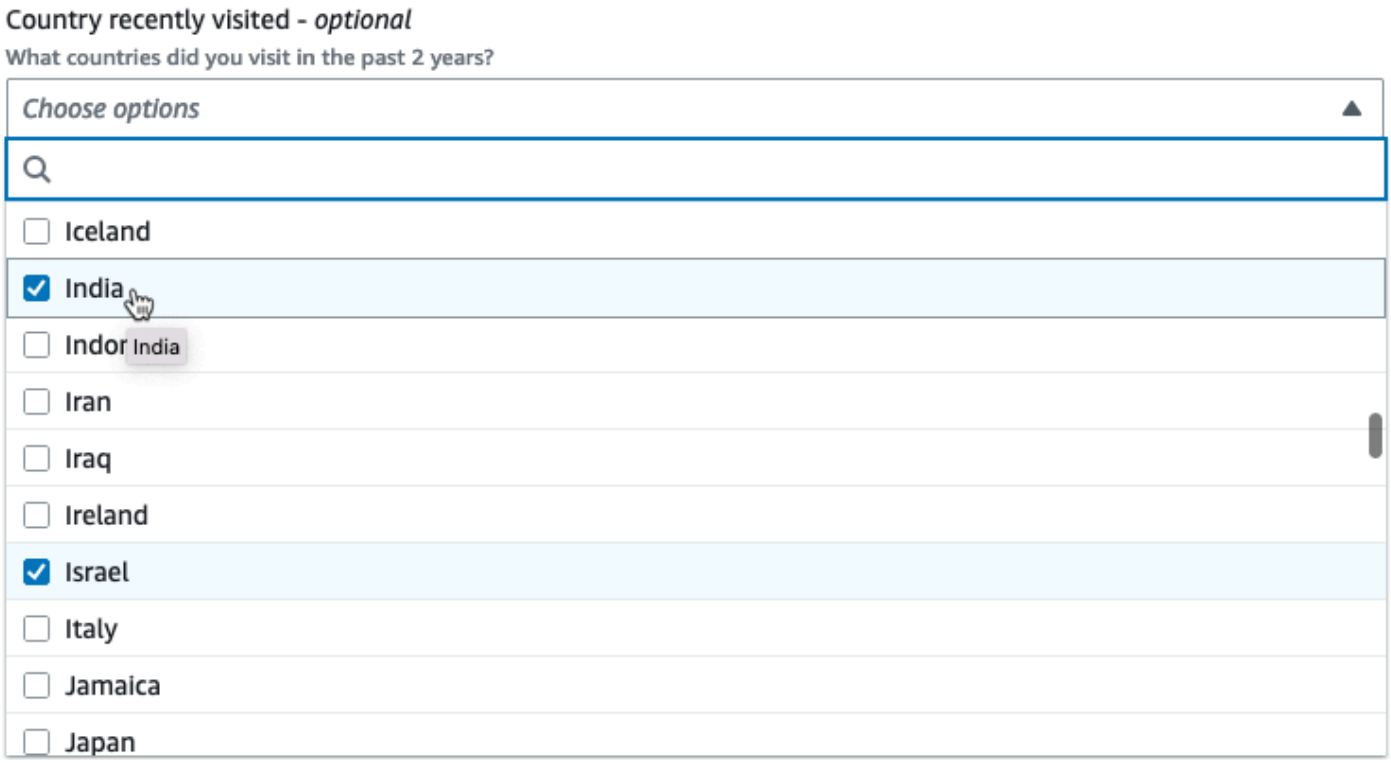
- 如果 `type` 為 `bool`，則會顯示核取方塊。



- 如果 `type` 為 `str` 且提供 `listOptions`，則會顯示單一選取清單。



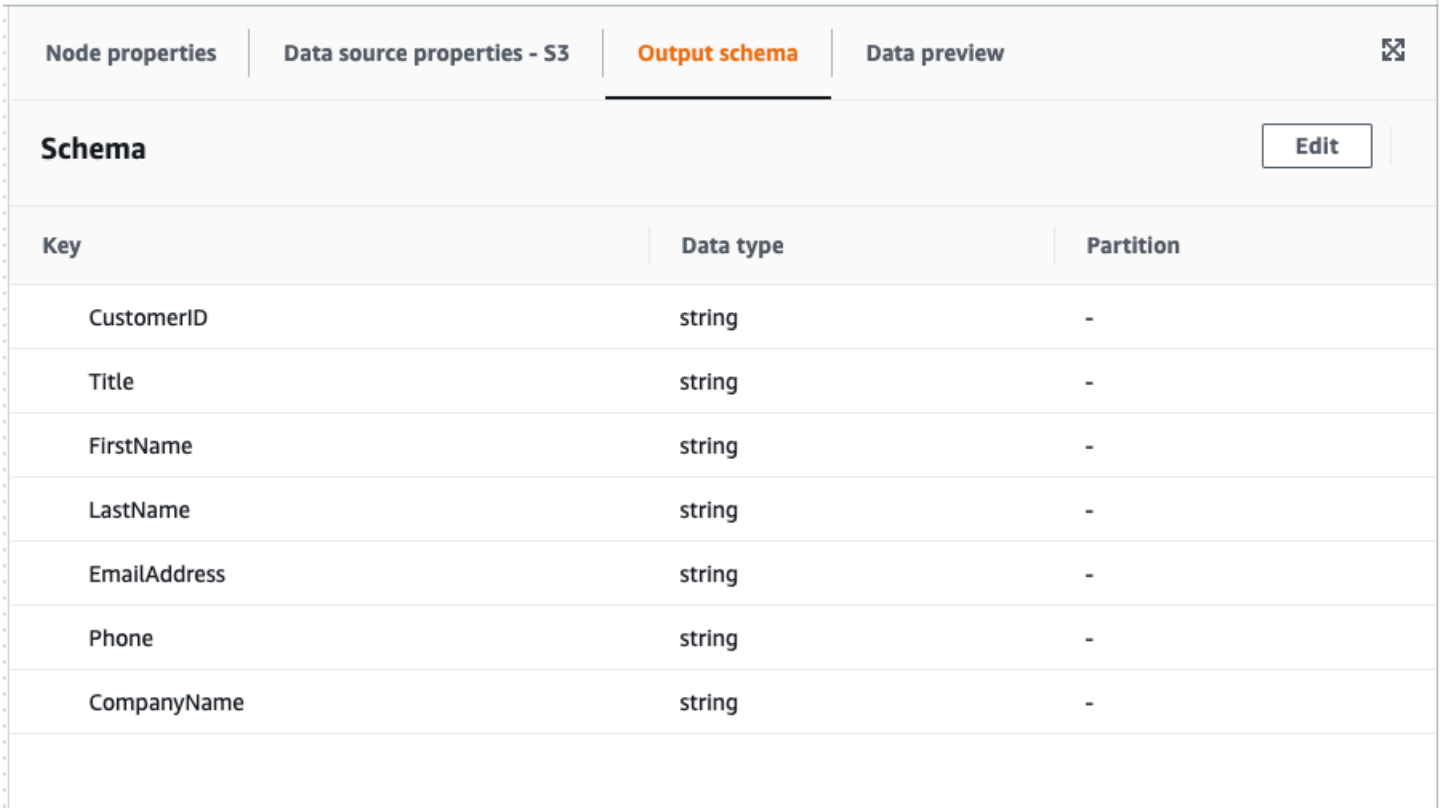
- 如果 `type` 為 `list` 和 `listOptions` 且提供 `listType`，則會顯示多選清單。



顯示資料欄選取器作為參數

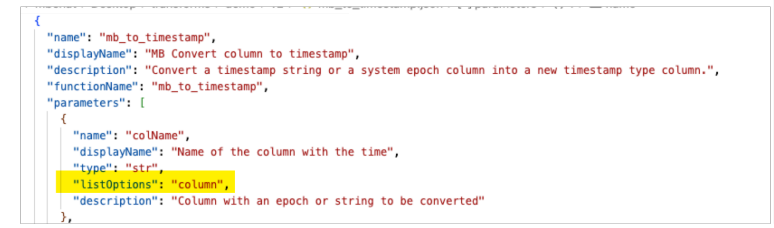
如果組態要求使用者從結構描述選擇資料欄，您可以顯示資料欄選取器，讓使用者不需要輸入資料欄名稱。透過將 `listOptions` 欄位設定為 "column"，AWS Glue Studio 會根據父節點輸出結構描述動態顯示資料欄選取器。AWS Glue Studio 可以顯示單個或多個資料欄選取器。

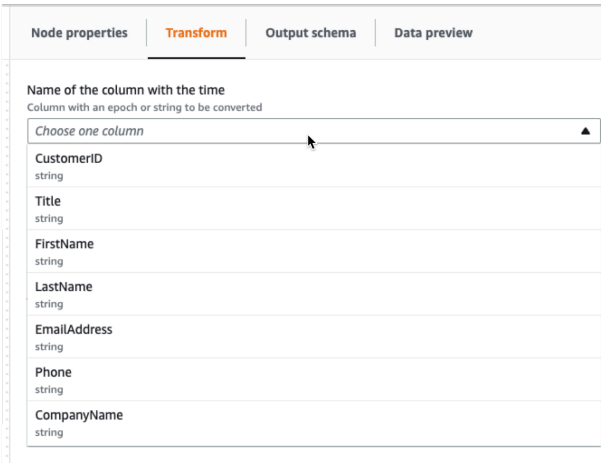
下列範例使用結構描述：



若要定義「自訂視覺化轉換」參數以顯示單一資料欄：

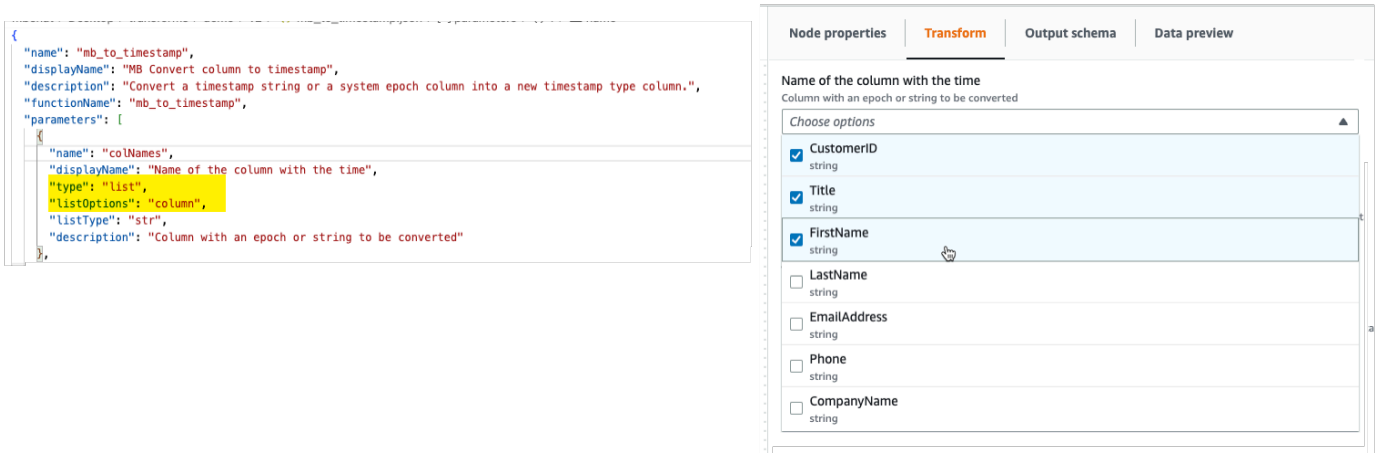
1. 在您的 JSON 檔案中，對於 `parameters` 物件，將 `listOptions` 值設為 "column"。這可讓使用者從 AWS Glue Studio 中的點選清單中選擇資料欄。





2. 您也可以透過將參數定義為以下項目來允許選取多個資料欄：

   - `listOptions: "column"`
   - `type: "list"`



步驟 2. 實作轉換邏輯

> **Note**
>
> 自訂視覺化轉換僅支援 Python 指令碼。不支援 Scala。

若要新增實作由 .json 組態檔案定義之函數的程式碼，建議將 Python 檔案放置在與 .json 檔案相同的位置，具有相同的名稱，但副檔名為 ".py"。AWS Glue Studio 會自動配對 .json 和 .py 檔案，讓您免於在組態檔案中指定 Python 檔案的路徑。

在 Python 檔案中，新增宣告的函數，設定具名參數並註冊用於 `DynamicFrame`。以下是 Python 檔案的範例：

```
from awsglue import DynamicFrame

# self refers to the DynamicFrame to transform,
# the parameter names must match the ones defined in the config
# if it's optional, need to provide a default value
def myTransform(self, email, phone, age=None, gender="",
                      country="", promotion=False):
   resulting_dynf = # do some transformation on self
   return resulting_dynf

DynamicFrame.myTransform = myTransform
```

建議使用 AWS Glue 筆記本，以最快的方式開發和測試 Python 程式碼。請參閱 AWS Glue Studio 中的筆記本入門。

為了說明如何實作轉換邏輯，下列範例中的自訂視覺化轉換是用來篩選傳入資料的轉換，以便僅保留與美國特定州相關的資料。.json 檔案包含 `functionName` 的參數 `custom_filter_state` 和兩個引數 (類型為 "str" 的 "state" 和 "colName")。

範例組態 .json 檔案是：

```
{
"name": "custom_filter_state",
"displayName": "Filter State",
"description": "A simple example to filter the data to keep only the state indicated.",
"functionName": "custom_filter_state",
"parameters": [
   {
    "name": "colName",
    "displayName": "Column name",
    "type": "str",
    "description": "Name of the column in the data that holds the state postal code"
   },
   {
    "name": "state",
    "displayName": "State postal code",
    "type": "str",
    "description": "The postal code of the state whole rows to keep"
   }
  ]
}
```

在 Python 中實作隨附指令碼

1. 啟動 AWS Glue 筆記本並執行提供給要啟動之工作階段的初始儲存格。執行初始儲存格會建立必要的基本元件。
2. 建立執行篩選的函數 (如範例中所述)，並在 `DynamicFrame` 上註冊。複製下方的程式碼並貼入 AWS Glue 筆記本的儲存格中。

```
from awsglue import DynamicFrame

def custom_filter_state(self, colName, state):
    return self.filter(lambda row: row[colName] == state)
```

```
DynamicFrame.custom_filter_state = custom_filter_state
```

3. 建立或載入範例資料，以測試相同儲存格或新儲存格中的程式碼。如果在新儲存格中新增範例資料，請不要忘記執行該儲存格。例如：

```
# A few of rows of sample data to test
data_sample = [
    {"state": "CA", "count": 4},
    {"state": "NY", "count": 2},
    {"state": "WA", "count": 3}
]
df1 = glueContext.sparkSession.sparkContext.parallelize(data_sample).toDF()
dynf1 = DynamicFrame.fromDF(df1, glueContext, None)
```

4. 測試以使用不同的引數驗證 "custom_filter_state"：

```
[14]: dynf1.custom_filter_state("state", "NY").show()
      {"count": 2, "state": "NY"}
```

5. 執行多個測試後，使用 .py 副檔名儲存程式碼，並使用與 .json 檔案名稱相同的名稱命名 .py 檔案。.py 和 .json 檔案應該位於相同的轉換資料夾中。

   複製下方程式碼並貼入檔案中，再使用 .py 檔案副檔名重新命名。

```
from awsglue import DynamicFrame

def custom_filter_state(self, colName, state):
    return self.filter(lambda row: row[colName] == state)

DynamicFrame.custom_filter_state = custom_filter_state
```

6. 在 AWS Glue Studio 中，開啟視覺化任務，然後從可用的 Transforms (轉換) 清單中選取轉換，以將該轉換新增至任務。

   若要在 Python 指令碼程式碼中重複使用此轉換，請將 Amazon S3 路徑新增至「參考檔案路徑」下任務中的 .py 檔案，然後在指令碼中匯入 Python 檔案的名稱 (不含副檔名)，方法為將 Python 檔案的名稱新增至檔案頂端。例如：`import` <檔案名稱 (不含副檔名)>

步驟 3. 在 AWS Glue Studio 中驗證自訂視覺化轉換並進行疑難排解

在將自訂視覺化轉換載入 AWS Glue Studio 之前，AWS Glue Studio 會先驗證 JSON 組態檔案。驗證包括：

- 存在必要欄位
- JSON 格式驗證
- 參數不正確或無效
- 在同一個 Amazon S3 路徑中存在 .py 和 .json 檔案
- 比對 .py 和 .json 的檔案名稱

如果驗證成功，轉換會列在視覺化編輯器的可用 Actions (動作) 清單中。如果提供了自訂圖示，則該圖示應顯示在動作旁。

如果驗證失敗，AWS Glue Studio 不會載入自訂視覺化轉換。

步驟 4. 視需要更新自訂視覺化轉換

一旦建立和使用，只要轉換遵循相應的 json 定義，就可以更新轉換指令碼：

- 指派給 DynamicFrame 時使用的名稱與 json `functionName` 非常相符。
- 函數引數必須在 json 檔案中定義，如 步驟 1. 建立 JSON 組態檔案 中所述。
- Python 檔案的 Amazon S3 路徑無法變更，因為這些任務直接依存於該路徑。

> **Note**
>
> 如果需要進行任何更新，請確保指令碼和 .json 檔案一致地更新，而且透過新轉換再次正確儲存任何視覺化任務。如果在進行更新之後未儲存視覺化任務，將不會套用和驗證更新。如果 Python 指令碼檔案被重新命名或未放置在 .json 檔案旁，則需要您在 .json 檔案中指定完整路徑。

自訂圖示

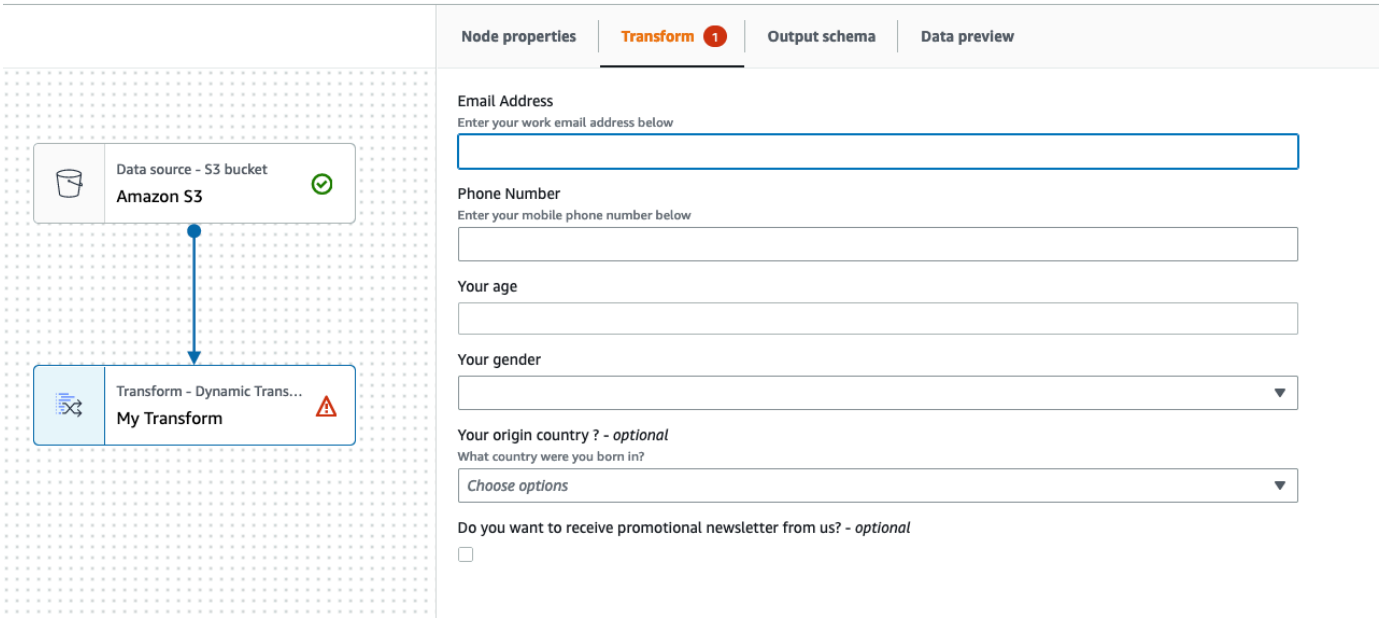
如果您確定動作的預設圖示在視覺上無法將其區分為工作流程的一部分，您可以提供自訂圖示，如 the section called “ 開始使用自訂視覺化轉換 ” 中所述。您可以透過更新 Amazon S3 中託管的對應 SVG 來更新圖示。

為了獲得最佳效果，請依照 Cloudscape 設計系統的指引將影像設計為以 32x32 像素檢視。如需有關 Cloudscape 指引的詳細資訊，請參閱 Cloudscape 文件。

步驟 5. 在 AWS Glue Studio 中使用自訂視覺化轉換

若要在 AWS Glue Studio 中使用自訂視覺化轉換，請上傳組態檔案和來源檔案，然後從 Action (動作) 選單中選取轉換。任何需要值或輸入的參數都可以在 Transform (轉換) 索引標籤中使用。

1. 將兩個檔案 (Python 來源檔案和 JSON 組態檔案) 上傳至存放任務指令碼的 Amazon S3 資產資料夾。依預設，AWS Glue 會從同一個 Amazon S3 儲存貯體內的 /transforms 資料夾中提取所有 JSON 檔案。
2. 從 Action (動作) 選單中選擇自訂視覺化轉換。其會以您在 .json 組態檔案中指定的轉換 `displayName` 或名稱來命名。
3. 為在組態檔案中設定的任何參數輸入值。



使用範例

以下是 .json 組態檔案中所有可能參數的範例。

```
{
  "name": "MyTransform",
  "displayName": "My Transform",
  "description": "This transform description will be displayed in UI",
  "functionName": "myTransform",
  "parameters": [
```

```
      {
      "name": "email",
      "displayName": "Email Address",
      "type": "str",
      "description": "Enter your work email address below",
      "validationType": "RegularExpression",
      "validationRule": "^\\w+([\\.-]?\\w+)*@\\w+([\\.-]?\\w+)*(\\.\\w{2,3})+$",
      "validationMessage": "Please enter a valid email address"
    },
    {
      "name": "phone",
      "displayName": "Phone Number",
      "type": "str",
      "description": "Enter your mobile phone number below",
      "validationRule": "^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})$",
      "validationMessage": "Please enter a valid US number"
    },
    {
      "name": "age",
      "displayName": "Your age",
      "type": "int",
      "isOptional": true
    },
    {
      "name": "gender",
      "displayName": "Your gender",
      "type": "str",
      "listOptions": [
            {"label": "Male", "value": "male"},
            {"label": "Female", "value": "female"},
            {"label": "Other", "value": "other"}
        ],
      "isOptional": true
    },
    {
      "name": "country",
      "displayName": "Your origin country ?",
      "type": "list",
      "listOptions": "Afghanistan,Albania,Algeria,American
Samoa,Andorra,Angola,Anguilla,Antarctica,Antigua and
Barbuda,Argentina,Armenia,Aruba,Australia,Austria,Azerbaijan,Bahamas,Bahrain,Bangladesh,Barbad
and Herzegovina,Botswana,Bouvet Island,Brazil,British Indian Ocean Territory,Brunei
Darussalam,Bulgaria,Burkina Faso,Burundi,Cambodia,Cameroon,Canada,Cape
Verde,Cayman Islands,Central African Republic,Chad,Chile,China,Christmas
```

```
 Island,Cocos (Keeling Islands),Colombia,Comoros,Congo,Cook Islands,Costa
 Rica,Cote D'Ivoire (Ivory Coast),Croatia (Hrvatska,Cuba,Cyprus,Czech
 Republic,Denmark,Djibouti,Dominica,Dominican Republic,East Timor,Ecuador,Egypt,El
 Salvador,Equatorial Guinea,Eritrea,Estonia,Ethiopia,Falkland Islands (Malvinas),Faroe
 Islands,Fiji,Finland,France,France,Metropolitan,French Guiana,French Polynesia,French
 Southern
 Territories,Gabon,Gambia,Georgia,Germany,Ghana,Gibraltar,Greece,Greenland,Grenada,Guadeloupe,G
Bissau,Guyana,Haiti,Heard and McDonald Islands,Honduras,Hong
 Kong,Hungary,Iceland,India,Indonesia,Iran,Iraq,Ireland,Israel,Italy,Jamaica,Japan,Jordan,Kazak
 (North),Korea
 (South),Kuwait,Kyrgyzstan,Laos,Latvia,Lebanon,Lesotho,Liberia,Libya,Liechtenstein,Lithuania,Lu
 Islands,Martinique,Mauritania,Mauritius,Mayotte,Mexico,Micronesia,Moldova,Monaco,Mongolia,Mont
 Antilles,New Caledonia,New Zealand,Nicaragua,Niger,Nigeria,Niue,Norfolk
 Island,Northern Mariana Islands,Norway,Oman,Pakistan,Palau,Panama,Papua
 New Guinea,Paraguay,Peru,Philippines,Pitcairn,Poland,Portugal,Puerto
 Rico,Qatar,Reunion,Romania,Russian Federation,Rwanda,Saint Kitts and Nevis,Saint
 Lucia,Saint Vincent and The Grenadines,Samoa,San Marino,Sao Tome and Principe,Saudi
 Arabia,Senegal,Seychelles,Sierra Leone,Singapore,Slovak Republic,Slovenia,Solomon
 Islands,Somalia,South Africa,S. Georgia and S. Sandwich Isls.,Spain,Sri
 Lanka,St. Helena,St. Pierre and Miquelon,Sudan,Suriname,Svalbard and Jan Mayen
 Islands,Swaziland,Sweden,Switzerland,Syria,Tajikistan,Tanzania,Thailand,Togo,Tokelau,Tonga,Tri
 and Tobago,Tunisia,Turkey,Turkmenistan,Turks and Caicos
 Islands,Tuvalu,Uganda,Ukraine,United Arab Emirates,United Kingdom
 (Britain / UK),United States of America (USA),US Minor Outlying
 Islands,Uruguay,Uzbekistan,Vanuatu,Vatican City State (Holy See),Venezuela,Viet
 Nam,Virgin Islands (British),Virgin Islands (US),Wallis and Futuna Islands,Western
 Sahara,Yemen,Yugoslavia,Zaire,Zambia,Zimbabwe",
      "description": "What country were you born in?",
      "listType": "str",
      "isOptional": true
    },
    {
      "name": "promotion",
      "displayName": "Do you want to receive promotional newsletter from us?",
      "type": "bool",
      "isOptional": true
    }
  ]
}
```

自訂視覺化指令碼範例

下列範例會執行對等的轉換。不過，第二個範例 (SparkSQL) 最簡潔且最有效率，其次是 pandas UDF，最後是第一個範例中的低階映射。下列範例是將兩欄加總的簡單轉換範例的完整範例：

```
from awsglue import DynamicFrame

# You can have other auxiliary variables, functions or classes on this file, it won't
 affect the runtime
def record_sum(rec, col1, col2, resultCol):
    rec[resultCol] = rec[col1] + rec[col2]
    return rec


# The number and name of arguments must match the definition on json config file
# (expect self which is the current DynamicFrame to transform
# If an argument is optional, you need to define a default value here
#  (resultCol in this example is an optional argument)
def custom_add_columns(self, col1, col2, resultCol="result"):
    # The mapping will alter the columns order, which could be important
    fields = [field.name for field in self.schema()]
    if resultCol not in fields:
        # If it's a new column put it at the end
        fields.append(resultCol)
    return self.map(lambda record: record_sum(record, col1, col2,
 resultCol)).select_fields(paths=fields)


# The name we assign on DynamicFrame must match the configured "functionName"
DynamicFrame.custom_add_columns = custom_add_columns
```

下列範例是利用 SparkSQL API 的對等轉換。

```
from awsglue import DynamicFrame

# The number and name of arguments must match the definition on json config file
# (expect self which is the current DynamicFrame to transform
# If an argument is optional, you need to define a default value here
#  (resultCol in this example is an optional argument)
def custom_add_columns(self, col1, col2, resultCol="result"):
    df = self.toDF()
    return DynamicFrame.fromDF(
```

```
        df.withColumn(resultCol, df[col1] + df[col2]) # This is the conversion logic
        , self.glue_ctx, self.name)


# The name we assign on DynamicFrame must match the configured "functionName"
DynamicFrame.custom_add_columns = custom_add_columns
```

下列範例使用相同的轉換，但使用 pandas UDF，這比使用普通 UDF 更有效率。如需有關撰寫 pandas UDF 的詳細資訊，請參閱 Apache Spark SQL 文件。

```
from awsglue import DynamicFrame
import pandas as pd
from pyspark.sql.functions import pandas_udf

# The number and name of arguments must match the definition on json config file
# (expect self which is the current DynamicFrame to transform
# If an argument is optional, you need to define a default value here
#  (resultCol in this example is an optional argument)
def custom_add_columns(self, col1, col2, resultCol="result"):
    @pandas_udf("integer")  # We need to declare the type of the result column
    def add_columns(value1: pd.Series, value2: pd.Series) # pd.Series:
        return value1 + value2

    df = self.toDF()
    return DynamicFrame.fromDF(
        df.withColumn(resultCol, add_columns(col1, col2)) # This is the conversion
 logic
        , self.glue_ctx, self.name)

# The name we assign on DynamicFrame must match the configured "functionName"
DynamicFrame.custom_add_columns = custom_add_columns
```

影片

下列影片提供視覺化自訂轉換的簡介，並示範如何使用這些轉換。

將資料湖架構與 AWS Glue Studio 搭配使用

概要

若檔案存放於在 Amazon S3 上建置的資料湖中，開放原始碼資料湖架構可以簡化這些檔案的增量資料處理。AWS Glue 3.0 及更高版本支援下列開放原始碼資料湖儲存架構：

- Apache Hudi
- Linux Foundation Delta Lake
- Apache Iceberg

從 AWS Glue 4.0 開始，AWS Glue 會為這些架構提供原生支援，讓您能夠以交易一致的方式讀取和寫入存放在 Amazon S3 中的資料。您不需要安裝個別的連接器或完成額外的設定步驟，就能在 AWS Glue 任務中使用這些架構。

透過 Spark 指令碼編輯器任務，資料湖架構可以用作 AWS Glue Studio 內的來源或目標。如需使用 Apache Hudi、Apache Iceberg 和 Delta Lake 的詳細資訊，請參閱搭配使用資料湖架構與 AWS Glue ETL 任務。

從 AWS Glue 串流來源建立開放資料表格式

AWS Glue 串流 ETL 工作會持續使用串流來源的資料、清理和轉換傳輸中資料，並可在數秒內完成分析。

AWS 提供廣泛的服務選項，可滿足您的需求。資料庫複寫服務 (例如，AWS Database Migration Service) 可從來源系統將資料複寫至 Amazon S3 (通常會託管資料湖的儲存層)。雖然在支援線上來源應用程式之關聯式資料庫管理系統 (RDBMS) 上套用更新相當簡單，但在資料湖上套用此 CDC 程序卻相當困難。開放原始碼資料管理架構可簡化增量資料處理與資料管道開發，是解決此問題的絕佳選擇。

如需詳細資訊，請參閱：

- 使用 AWS Glue 串流建立以 Apache Hudi 為基礎的近乎即時交易資料湖
- 建置即時 GDPR 對齊的 Apache Iceberg 資料湖

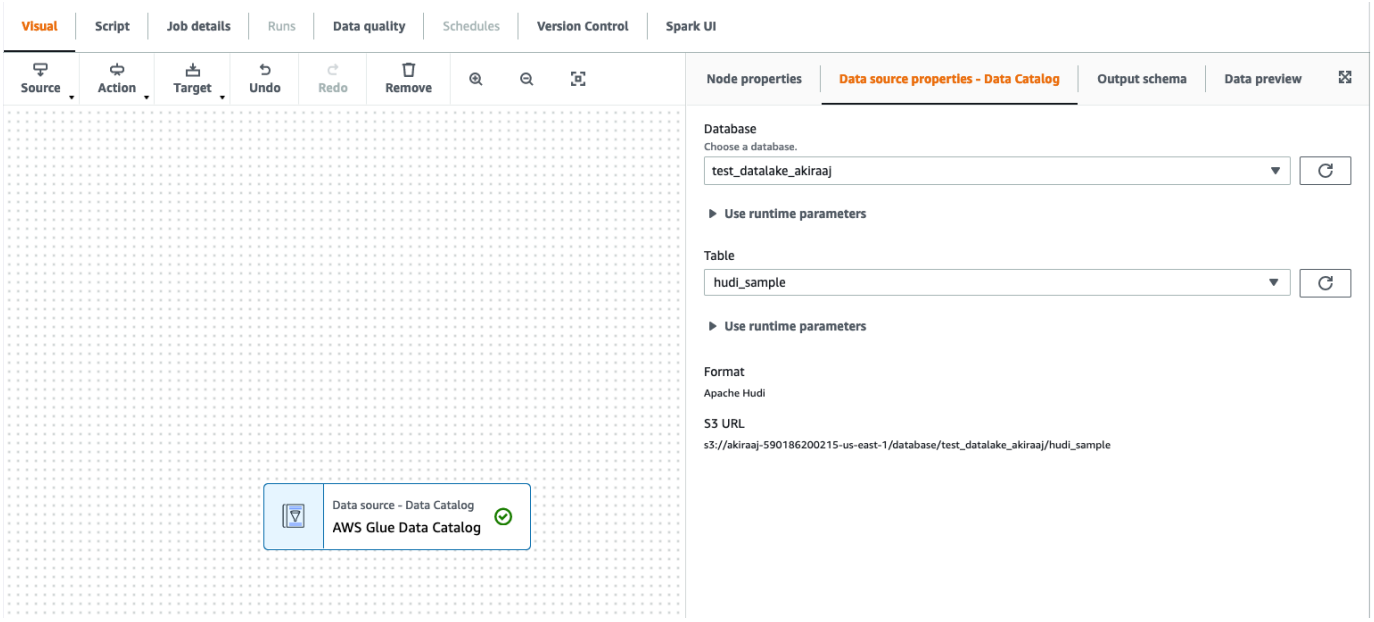
在 AWS Glue Studio 中使用 Hudi 架構

建立或編輯任務時，AWS Glue Studio 會視乎您使用的版本，自動為您新增對應的 AWS Glue Hudi 程式庫。如需詳細資訊，請參閱使用 AWS Glue 中的 Hudi 架構。

在 Data Catalog 資料來源中使用 Apache Hudi 架構

若要將 Hudi 資料來源格式新增至任務，請執行下列操作：

1. 從「來源」選單中，選擇 AWS Glue Studio Data Catalog。
2. 在資料來源屬性索引標籤中，選擇資料庫和資料表。
3. AWS Glue Studio 顯示格式類型為 Apache Hudi 和 Amazon S3 URL。



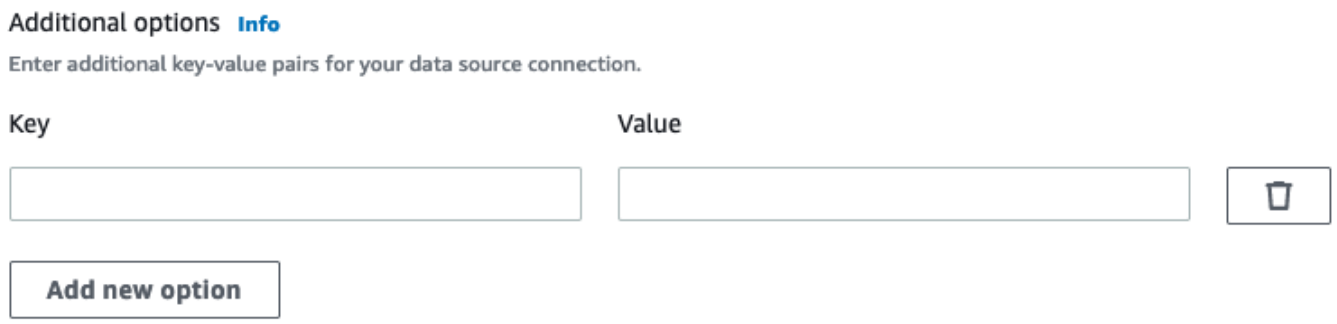
在 Amazon S3 資料來源中使用 Hudi 架構

1. 從「來源」選單中，選擇 Amazon S3。
2. 如果您選擇 Data Catalog 資料表做為 Amazon S3 來源類型，請選擇資料庫和資料表。
3. AWS Glue Studio 顯示格式為 Apache Hudi 和 Amazon S3 URL。
4. 如果您選擇 Amazon S3 位置做為 Amazon S3 來源類型，請按一下瀏覽 Amazon S3 來選擇 Amazon S3 URL。
5. 在資料格式中，選取 Apache Hudi。

> **Note**
>
> 如果 AWS Glue Studio 無法從您選取的 Amazon S3 資料夾或檔案推斷結構描述，請選擇其他選項，以選取新的資料夾或檔案。
> 在其他選項中，選擇結構描述推論下的下列選項：

- 讓 AWS Glue Studio 自動選擇一個範例檔案 – AWS Glue Studio 會在 Amazon S3 位置選擇一個範例檔案，以便推斷結構描述。在自動取樣檔案欄位中，您可以檢視自動選取的檔案。
- 從 Amazon S3 中選擇一個範例檔案 – 按一下瀏覽 Amazon S3，選擇要使用的 Amazon S3 檔案。

6. 按一下推斷結構描述。然後，您可以按一下輸出結構描述索引標籤，來檢視輸出結構描述。
7. 選擇其他選項，以輸入鍵值對。



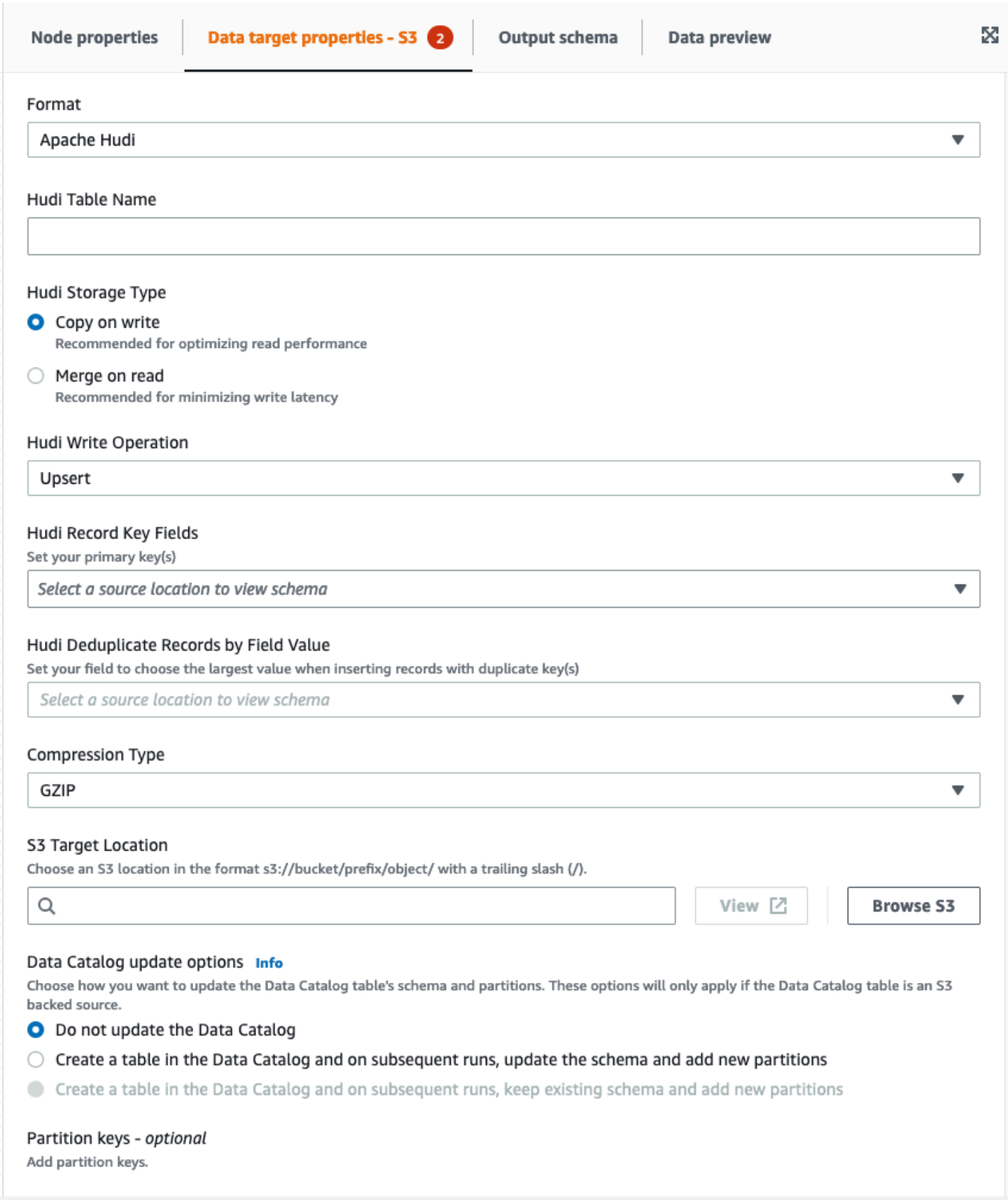
在資料目標中使用 Apache Hudi 架構

在 Data Catalog 資料目標中使用 Apache Hudi 架構

1. 從目標選單中，選擇 AWS Glue Studio Data Catalog。
2. 在資料來源屬性索引標籤中，選擇資料庫和資料表。
3. AWS Glue Studio 顯示格式類型為 Apache Hudi 和 Amazon S3 URL。

在 Amazon S3 資料目標中使用 Apache Hudi 架構

輸入值，或者從可用選項中選取，以設定 Apache Hudi 格式。如需有關 Apache Hudi 的詳細資訊，請參閱 Apache Hudi 文件。


Node properties
Data target properties - S3 2
Output schema
Data preview
Format
Apache Hudi
Hudi Table Name
Hudi Storage Type
Copy on write
Recommended for optimizing read performance
Merge on read
Recommended for minimizing write latency
Hudi Write Operation
Upsert
Hudi Record Key Fields
Set your primary key(s)
Select a source location to view schema
Hudi Deduplicate Records by Field Value
Set your field to choose the largest value when inserting records with duplicate key(s)
Select a source location to view schema
Compression Type
GZIP
S3 Target Location
Choose an S3 location in the format s3://bucket/prefix/object/ with a trailing slash (/).
View
Browse S3
Data Catalog update options Info
Choose how you want to update the Data Catalog table's schema and partitions. These options will only apply if the Data Catalog table is an S3 backed source.
Do not update the Data Catalog
Create a table in the Data Catalog and on subsequent runs, update the schema and add new partitions
Create a table in the Data Catalog and on subsequent runs, keep existing schema and add new partitions
Partition keys - optional
Add partition keys.

- Hudi 資料表名稱 – 這是 hudi 資料表的名稱。
- Hudi 儲存類型 – 從兩個選項中選擇：
  - 寫入時複製 – 建議用於最佳化讀取效能。這是預設的 Hudi 儲存類型。每次更新都會在寫入期間建立新版檔案。
  - 讀取時合併 – 建議將寫入延遲降至最低。更新會記錄到以資料列為基礎的 delta 檔案，並視需要壓縮以建立新版本的直欄式檔案。
- Hudi 寫入操作 – 請從下列選項中選擇：
  - Upsert – 這是輸入記錄首先透過查詢索引標記為插入或更新的預設操作。建議您更新現有資料的位置。
  - 插入 – 這會插入記錄，但不檢查現有記錄，並且可能會導致重複。
  - 大量插入 – 這會插入記錄，建議用於大量資料。
- Hudi 記錄關鍵字欄位 – 使用搜尋列來搜尋並選擇主要記錄索引鍵。Hudi 中的記錄由主索引鍵標識，這是記錄所屬的一對記錄索引鍵和分區路徑。
- Hudi 預結合欄位 – 這是在實際寫入之前預結合中使用的欄位。當兩個記錄具有相同的索引鍵值時，AWS Glue Studio 會選擇預先組合欄位中值最大的記錄。設定增量值 (例如 updated_at) 所屬的欄位。
- 壓縮類型 – 從壓縮類型選項中選擇：未壓縮、GZIP、LZO 或 Snappy。
- Amazon S3 目標位置 – 按一下瀏覽 S3 來選擇 Amazon S3 目標位置。
- Data Catalog 更新選項 – 請從下列選項中選擇：
  - 不更新資料目錄：(預設值) 如果您不希望任務更新資料目錄 (即使結構描述變更或新增分割區)，請選擇此選項。
  - 在 Data Catalog 中建立資料表，並在後續執行時，更新結構描述並新增分區：如果選擇此選項，任務會在第一次執行任務時，在 Data Catalog 中建立資料表。在後續任務執行中，如果結構描述變更或新增分割區，任務會更新 Data Catalog 資料表。

    您還必須從 Data Catalog 中選取資料庫，然後輸入資料表名稱。
  - 在資料目錄和後續執行中建立資料表，保留現有的結構描述並新增分割區：如果選擇此選項，任務會在第一次執行任務時，在資料目錄中建立資料表。在後續的任務執行中，任務只會更新 Data Catalog 資料表以新增新的分割區。

    您還必須從 Data Catalog 中選取資料庫，然後輸入資料表名稱。
- 分割區索引鍵：選擇要在輸出中用作分割索引鍵的欄。若要新增更多分割區索引鍵，請選擇新增分割區索引鍵。
- 其他選項 – 根據需要輸入鍵值對。

透過 AWS Glue Studio 產生程式碼

儲存任務後，如果偵測到 Hudi 來源或目標，則會將下列任務參數新增至任務：

- `--datalake-formats` – 在視覺化任務中偵測到不同的資料湖格式清單 (可直接選擇 “Format” 或間接選取由資料湖支援的目錄資料表)。
- `--conf` – 根據 `--datalake-formats` 的值產生。例如，如果 `--datalake-formats` 的值為 'hudi'，則 AWS Glue 會針對此參數產生 `spark.serializer=org.apache.spark.serializer.KryoSerializer –conf spark.sql.hive.convertMetastoreParquet=false` 的值。

覆寫 AWS Glue 提供的程式庫

若要使用 AWS Glue 不支援的 Hudi 版本，您可以指定自己的 Hudi 程式庫 JAR 檔案。若要使用您自己的 JAR 檔案：

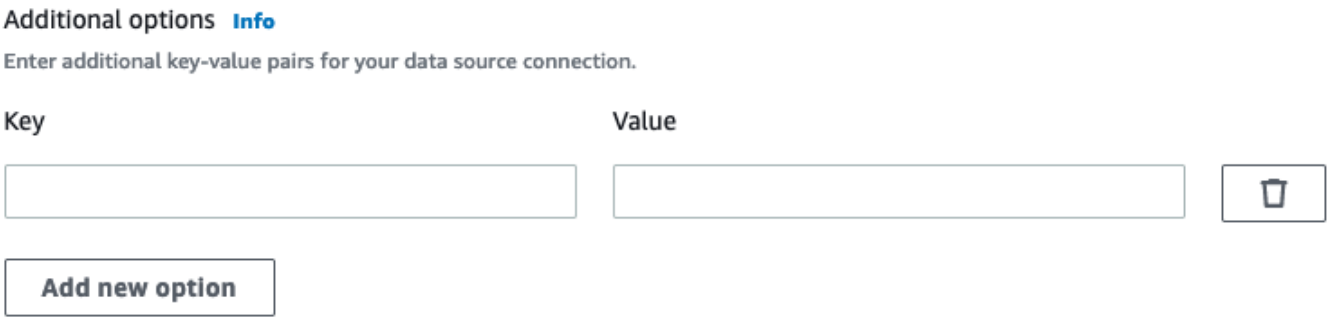
- 請使用 `--extra-jars` 任務參數。例如 `'--extra-jars': 's3pathtojarfile.jar'`。如需詳細資訊，請參閱 AWS Glue 任務參數。
- 請勿包括 `hudi` 作為 `--datalake-formats` 任務參數的值。輸入空白字串做為值，可確保 AWS Glue 不會自動為您提供任何資料湖程式庫。如需詳細資訊，請參閱使用 AWS Glue 中的 Hudi 架構。

在 AWS Glue Studio 中使用 Delta Lake 架構

在資料來源中使用 Delta Lake 架構

在 Amazon S3 資料來源中使用 Delta Lake 架構

1. 從「來源」選單中，選擇 Amazon S3。
2. 如果您選擇 Data Catalog 資料表做為 Amazon S3 來源類型，請選擇資料庫和資料表。
3. AWS Glue Studio 顯示格式為 Delta Lake 和 Amazon S3 URL。
4. 選擇其他選項，以輸入鍵值對。例如，鍵值對可能是：索引鍵：timestampAsOf 和值：2023-02-24 14:16:18。



5. 如果您選擇 Amazon S3 位置做為 Amazon S3 來源類型，請按一下瀏覽 Amazon S3 來選擇 Amazon S3 URL。
6. 在資料格式中，選擇 Delta Lake。

> **Note**
>
> 如果 AWS Glue Studio 無法從您選取的 Amazon S3 資料夾或檔案推斷結構描述，請選擇其他選項，以選取新的資料夾或檔案。
> 在其他選項中，選擇結構描述推論下的下列選項：
>
> - 讓 AWS Glue Studio 自動選擇一個範例檔案 – AWS Glue Studio 會在 Amazon S3 位置選擇一個範例檔案，以便推斷結構描述。在自動取樣檔案欄位中，您可以檢視自動選取的檔案。
> - 從 Amazon S3 中選擇一個範例檔案 – 按一下瀏覽 Amazon S3，選擇要使用的 Amazon S3 檔案。

7. 按一下推斷結構描述。然後，您可以按一下輸出結構描述索引標籤，來檢視輸出結構描述。

在 Data Catalog 資料來源中使用 Delta Lake 架構

1. 從來源選單中，選擇 AWS Glue Studio Data Catalog。
2. 在資料來源屬性索引標籤中，選擇資料庫和資料表。
3. AWS Glue Studio 顯示格式類型為 Delta Lake 和 Amazon S3 URL。

> **Note**
>
> 如果您的 Delta Lake 來源尚未註冊為 AWS Glue Data Catalog 資料表，則會有兩個選項：

1. 針對 Delta Lake 資料存放區建立 AWS Glue 爬蟲程式。如需詳細資訊，請參閱如何為 Delta Lake 的資料儲存指定配置選項。
2. 使用 Amazon S3 資料來源，以選取您的 Delta Lake 資料來源。請參閱 在 Amazon S3 資料來源中使用 Delta Lake 架構 。

在資料目標中使用 Delta Lake 格式

在 Data Catalog 資料目標中使用 Delta Lake 格式

1. 從目標選單中，選擇 AWS Glue Studio Data Catalog。
2. 在資料來源屬性索引標籤中，選擇資料庫和資料表。
3. AWS Glue Studio 顯示格式類型為 Delta Lake 和 Amazon S3 URL。

在 Amazon S3 資料來源中使用 Delta Lake 格式

輸入值，或者從可用選項中選取，以設定 Delta Lake 格式。

- 壓縮類型 – 從壓縮類型選項中選擇：未壓縮或 Snappy。
- Amazon S3 目標位置 – 按一下瀏覽 S3 來選擇 Amazon S3 目標位置。
- Data Catalog 更新選項 – Glue Studio 視覺化編輯器中不支援更新此格式的 Data Catalog。
  - 不更新資料目錄：(預設值) 如果您不希望任務更新資料目錄 (即使結構描述變更或新增分割區)，請選擇此選項。
  - 若要在 AWS Glue 任務執行後更新 Data Catalog，請執行或排程 AWS Glue 爬蟲程式。如需詳細資訊，請參閱如何為 Delta Lake 的資料儲存指定配置選項。
- 分割區索引鍵 – 選擇要在輸出中用作分割索引鍵的欄。若要新增更多分割區索引鍵，請選擇 Add a partition key (新增分割區索引鍵)。
- 或者，選擇其他選項，以輸入鍵值對。例如，鍵值對可能是：索引鍵：timestampAsOf 和值：2023-02-24 14:16:18。

在 AWS Glue Studio 中使用 Apache Iceberg 架構

在資料目標中使用 Apache Iceberg 架構

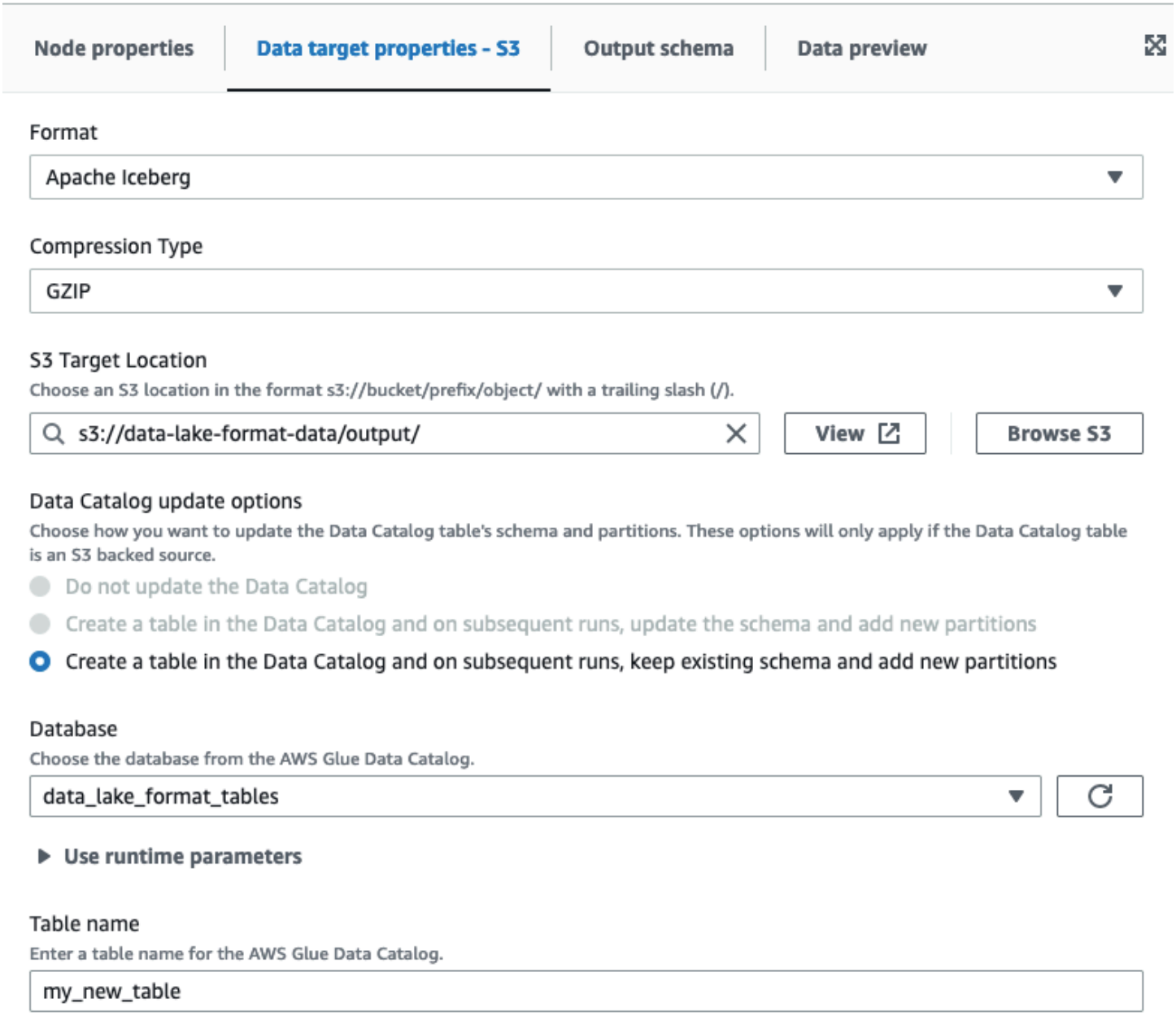
在資料型錄資料目標中使用 Apache Iceberg 架構

1. 從目標選單中，選擇 AWS Glue Studio Data Catalog。
2. 在資料來源屬性索引標籤中，選擇資料庫和資料表。
3. AWS Glue Studio 將格式類型顯示為 Apache Iceberg 和 Amazon S3 URL。

在 Amazon S3 資料目標中使用 Apache Iceberg 架構

輸入值或從可用選項中選取，以設定 Apache Iceberg 格式。

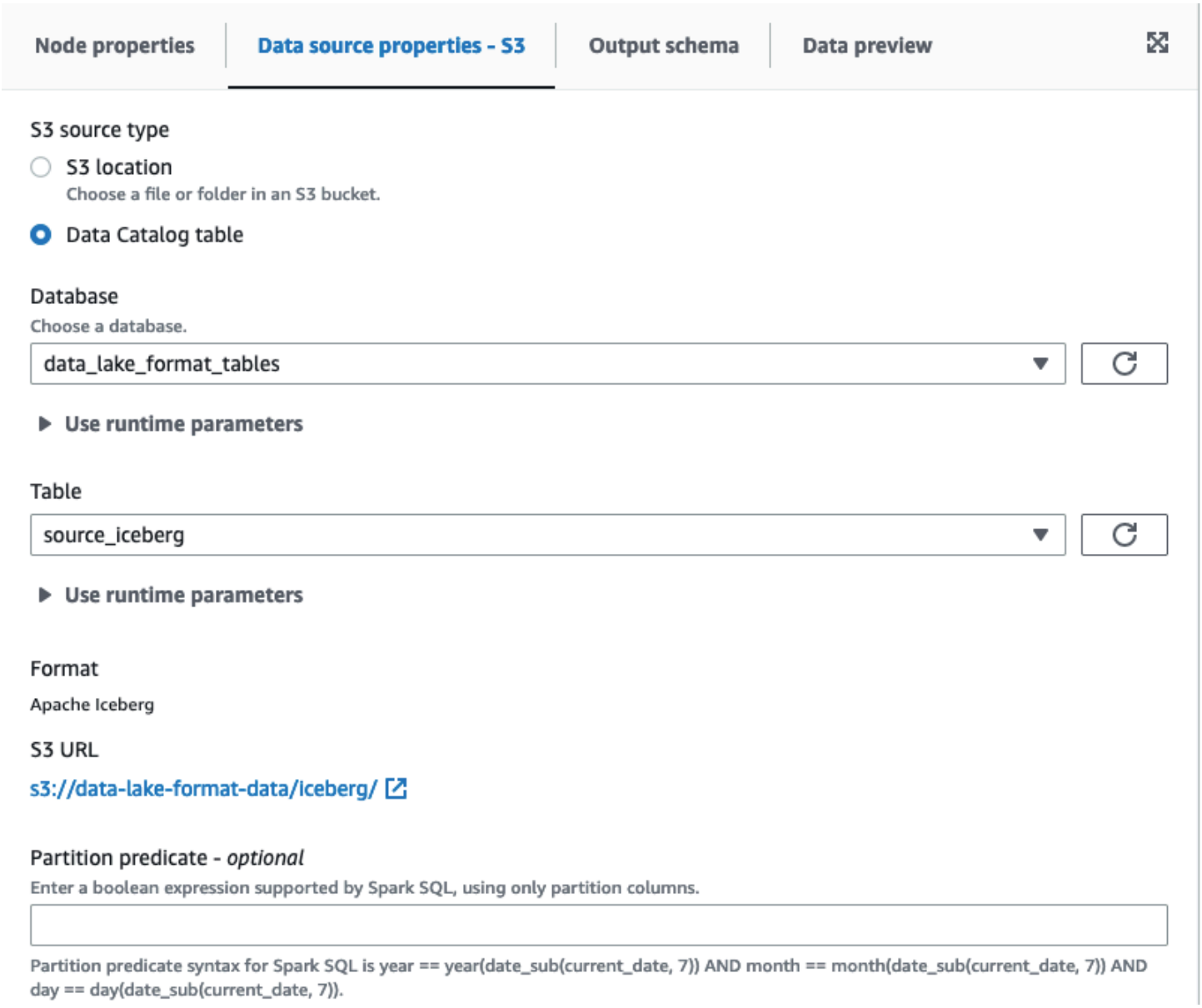
- 格式：從下拉式選單中選擇 Apache Iceberg。
- Amazon S3 目標位置：按一下瀏覽 S3 來選擇 Amazon S3 目標位置。
- 資料型錄更新選項：必須選取在資料型錄中建立資料表，並在後續執行時保留現有結構描述和新增分割區，才能繼續執行。使用 AWS Glue 撰寫新的 Iceberg 資料表時，需要將 Data Catalog 設定為 Iceberg 資料表的型錄。若要更新已在 Data Catalog 中註冊的現有 Iceberg 資料表，請選擇 Data Catalog 作為目標。
  - 資料庫：從 Data Catalog 中選擇資料庫。
  - 資料表名稱：輸入資料表名稱的值。Apache Iceberg 資料表名稱必須完全使用小寫。因為不允許使用空格，如有需要請使用底線。例如 "data_lake_format_tables"。



在 Amazon S3 資料來源中使用 Apache Iceberg 架構

在資料型錄資料來源中使用 Apache Iceberg 架構

1. 從來源選單中，選擇 AWS Glue Studio Data Catalog。
2. 在資料來源屬性索引標籤中，選擇資料庫和資料表。
3. AWS Glue Studio 將格式類型顯示為 Apache Iceberg 和 Amazon S3 URL。



在 Amazon S3 資料來源中使用 Apache Iceberg 架構

Apache Iceberg 無法作為 AWS Glue Studio 中 Amazon S3 來源節點的資料選項使用。

設定資料目標節點

資料目標是任務寫入轉換後資料的位置。

資料目標選項概觀

您的資料目標 (也稱為資料接收) 可以是：

- S3 — 任務會以您選擇的 Amazon S3 位置和您指定的格式將資料寫入檔案中。

如果您為資料目標設定分割區欄，則任務會根據分割區索引鍵將資料集寫入 Amazon S3 的目錄中。

- AWS Glue Data Catalog - 任務會使用與 Data Catalog 中資料表相關聯的資訊，將輸出資料寫入目標位置。

  您可以手動建立資料表，或使用爬蟲程式建立資料表。您也可以使用 AWS CloudFormation 範本，在 Data Catalog 中建立資料表。
- 連接器 - 連接器是一段程式碼，可促進資料存放區和 AWS Glue 之間的通訊。任務會使用連接器和關聯的連線，將輸出資料寫入目標位置。您可以訂閱 AWS Marketplace 提供的連接器，也可以自行建立自訂連接器。如需詳細資訊，請參閱 新增連接器至 AWS Glue Studio

您可以選擇在任務寫入 Amazon S3 資料目標時更新 Data Catalog 。在結構描述或分割區變更時，無須爬蟲程式更新 Data Catalog ，此選項可讓您輕鬆保持資料表在最新狀態。此選項可簡化讓資料可供分析的程序，方法是選擇性地將新資料表新增至 Data Catalog 、更新資料表分割區，以及直接從任務更新資料表結構描述。

編輯資料目標節點

資料目標是任務寫入轉換後資料的位置。

在任務圖表中新增或設定資料目標節點

1. (選用) 如果您需要新增目標節點，請選擇視覺化編輯器頂端工具列中的 Target (目標)，然後選擇 S3 或 Glue Data Catalog (Glue 資料目錄)。

   - 如果選擇 S3，則任務會將資料集寫入您指定的 Amazon S3 位置中的一個或多個檔案。
   - 如果選擇 AWS Glue Data Catalog，則任務會寫入至由從 Data Catalog 選取的資料表所描述的位置。
2. 選擇任務圖表中的資料目標節點。當您選擇節點時，節點詳細資訊面板會出現在頁面右側。
3. 選擇 Node properties (節點屬性) 索引標籤，然後輸入下列資訊：

   - Name (名稱)：輸入要與任務圖表中節點產生關聯的名稱。
   - Node type (節點類型)：應該已經選取值，但您可以視需要變更它。
   - Node parents (節點父項)：父節點是任務圖表中的節點，提供您要寫入目標位置的輸出資料。對於預先填入的任務圖表，目標節點應該已經選取父節點。如果沒有顯示父節點，請從清單中選擇父節點。

     目標節點有單一的父節點。

4. 設定 Data target properties (資料目標屬性) 資訊。如需詳細資訊，請參閱下列章節：

   - 將 Amazon S3 用於資料目標
   - 使用 Data Catalog 資料表做為資料目標
   - 將連接器用於資料目標

5. (選用) 設定資料目標節點屬性之後，您可以在節點詳細資訊面板中選擇 Output schema (輸出結構描述) 索引標籤來檢視資料的輸出結構描述。當您第一次針對任務中的任何節點選擇此索引標籤時，系統會提示您提供 IAM 角色以存取資料。如果您尚未在 Job details (任務詳細資訊) 索引標籤上指定 IAM 角色，系統會提示您在此輸入 IAM 角色。

將 Amazon S3 用於資料目標

對於除 Amazon S3 和連接器以外的所有資料來源，資料表必須存在於 AWS Glue Data Catalog，以取得您選擇的來源類型。AWS Glue Studio 不會建立 Data Catalog 資料表。

設定寫入 Amazon S3 的資料目標節點

1. 前往新任務或已儲存任務的視覺化編輯器。
2. 在任務圖表中選擇資料來源節點。
3. 選擇 Data source properties (資料來源屬性) 索引標籤，然後輸入下列資訊：

   - Format (格式)：從清單中選擇格式。資料結果的可用格式類型為：
     - JSON：JavaScript 物件標記法。
     - CSV：逗號分隔值。
     - Avro：Apache Avro JSON 二進位。
     - Parquet：Apache Parquet 單欄式儲存。
     - Glue Parquet：已針對 `DynamicFrames` 做為資料格式而最佳化的自訂 Phquet 寫入器類型。它不需要預先計算的資料結構描述，而是動態地運算並修改結構描述。
     - ORC：Apache 最佳化資料列單欄式 (ORC) 格式。

     若要進一步了解這些格式選項，請參閱 AWS Glue 開發人員指南中的在 AWS Glue 中的 ETL 輸入與輸出格式選項。
   - Compression Type (壓縮類型)：您可以選擇使用 `gzip` 或 `bzip2` 格式壓縮資料。預設為沒有壓縮，或 None (無)。

- S3 Target Location (S3 目標位置)：資料輸出的 Amazon S3 儲存貯體和位置。您可以選擇 Browse S3 (瀏覽 S3) 按鈕以查看您有權存取的 Amazon S3 儲存貯體，並選擇其中一個作為目標目的地。
- Data Catalog 更新選項
  - Do not update the Data Catalog (不更新 Data Catalog)：(預設值) 如果您不希望任務更新 Data Catalog (即使結構描述變更或新增分割區)，請選擇此選項。
  - Create a table in the Data Catalog and on subsequent runs, update the schema and add new partitions (在 Data Catalog 中建立資料表，並在後續執行時，更新結構描述並新增分割區)：如果選擇此選項，任務會在第一次執行任務時，在 Data Catalog 中建立資料表。在後續任務執行中，如果結構描述變更或新增分割區，任務會更新 Data Catalog 資料表。

    您還必須從 Data Catalog 中選取資料庫，然後輸入資料表名稱。
  - Create a table in the Data Catalog and on subsequent runs, keep existing schema and add new partitions (在 Data Catalog 和後續執行中建立資料表，保留現有的結構描述並新增分割區)：如果選擇此選項，任務會在第一次執行任務時，在 Data Catalog 中建立資料表。在後續的任務執行中，任務只會更新 Data Catalog 資料表以新增新的分割區。

    您還必須從 Data Catalog 中選取資料庫，然後輸入資料表名稱。
  - Partition keys (分割區索引鍵)：選擇要在輸出中用作分割索引鍵的欄。若要新增更多分割區索引鍵，請選擇 Add a partition key (新增分割區索引鍵)。

使用 Data Catalog 資料表做為資料目標

對於除 Amazon S3 和連接器以外的所有資料來源，資料表必須存在於 AWS Glue Data Catalog，以取得您選擇的目標類型。AWS Glue Studio 不會建立 Data Catalog 資料表。

為使用 Data Catalog 資料表的目標設定資料屬性

1. 前往新任務或已儲存任務的視覺化編輯器。
2. 選擇任務圖表中的資料目標節點。
3. 選擇 Data target properties (資料目標屬性) 索引標籤，然後輸入下列資訊：
   - Database (資料庫)：從清單中選擇包含要用作目標之資料表的資料庫。此資料庫必須存在於 Data Catalog 中。
   - Table (資料表)：從清單中選擇定義輸出資料之結構描述的資料表。此資料表必須已存在於 Data Catalog 中。

Data Catalog 中的資料表包含欄名稱、資料類型定義、分割區資訊，以及目標資料集的其他中繼資料。您的任務會寫入至此 Data Catalog 中所述的位置。

如需在 Data Catalog 中建立資料表的詳細資訊，請參閱 AWS Glue 開發人員指南中的在 Data Catalog 中定義資料表。

- Data Catalog 更新選項
  - Do not change table definition (不變更資料表定義)：(預設值) 如果您不希望任務更新 Data Catalog (即使結構描述變更或新增分割區)，請選擇此選項。
  - Update schema and add new partitions (更新結構描述並新增分割區)：如果您選擇此選項，則如果結構描述變更或新增了新的分割區，任務就會更新 Data Catalog 資料表。
  - Keep existing schema and add new partitions (保留現有的結構描述並新增分割區)：如果您選擇此選項，任務只會更新 Data Catalog 資料表以新增新的分割區。
  - Partition keys (分割區索引鍵)：選擇要在輸出中用作分割索引鍵的欄。若要新增更多分割區索引鍵，請選擇 Add a partition key (新增分割區索引鍵)。

將連接器用於資料目標

如果您選取 Node type (節點類型) 的連接器，請遵循使用自訂連接器編寫任務中的說明以完成設定資料目標屬性。

編輯或上傳任務指令碼

使用 AWS Glue Studio 視覺化編輯器來編輯任務指令碼或上傳自己的指令碼。

只有當任務是使用 AWS Glue Studio 建立時，您才可以使用視覺化編輯器來編輯任務節點。如果任務是使用 AWS Glue 主控台、透過 API 命令或命令列介面 (CLI) 建立的，則您可以使用 AWS Glue Studio 中的指令碼編輯器來編輯任務指令碼、參數和排程。您也可以透過將任務轉換為僅限指令碼模式，編輯在 AWS Glue Studio 中建立之任務的指令碼。

編輯任務指令碼或上傳您自己的指令碼

1. 如果建立新任務，請在 Jobs (任務) 頁面上，選擇 Spark script editor (Spark 指令碼編輯器) 選項來建立 Spark 任務或選擇 Python Shell script editor (Python Shell 指令碼編輯器) 來建立 Python Shell 任務。您可以撰寫新指令碼，也可以上傳現有指令碼。如果選擇 Spark script editor (Spark 指令碼編輯器)，您可以編寫或上傳 Scala 或 Python 指令碼。如果選擇 Python Shell script editor (Python Shell 指令碼編輯器)，您只能編寫或上傳 Python 指令碼。

選擇建立新任務的選項後，在出現的 Options (選項) 區段中，您可以選擇以入門指令碼開始 (Create a new script with boilerplate code (使用樣板程式碼建立新指令碼))，或者您可以上傳本機檔案以作為任務指令碼使用。

如果您選擇 Spark script editor (Spark 指令碼編輯器)，您可以上傳 Python 或 Scala 指令碼檔案。Scala 指令碼的副檔名必須為 `.scala`。Python 指令碼必須被識別為 Python 類型的檔案。如果您選擇 Python Shell script editor (Python Shell 指令碼編輯器)，您只能上傳 Python 指令碼檔案。

完成選擇後，請選擇 Create (建立) 以建立任務並開啟視覺化編輯器。

2. 前往新任務或儲存任務的視覺化任務編輯器，然後選擇 Script (指令碼) 索引標籤。
3. 如果您沒有使用其中一個指令碼編輯器選項建立新任務，而且您從未編輯現有任務的指令碼，Script (指令碼) 索引標籤會顯示標題 Script (Locked) (指令碼 (已鎖定))。這表示指令碼編輯器處於唯讀模式。請選擇 Edit script (編輯指令碼) 解除鎖定指令碼以進行編輯。

   為了讓指令碼變成可編輯的，AWS Glue Studio 會將您的任務從視覺化任務轉換為僅限指令碼任務。如果您將指令碼解除鎖定以進行編輯，儲存之後就無法再使用視覺化編輯器來執行此任務。

   在確認視窗中，選擇 Confirm (確認) 繼續，或 Cancel (取消) 以保持任務可供視覺編輯。

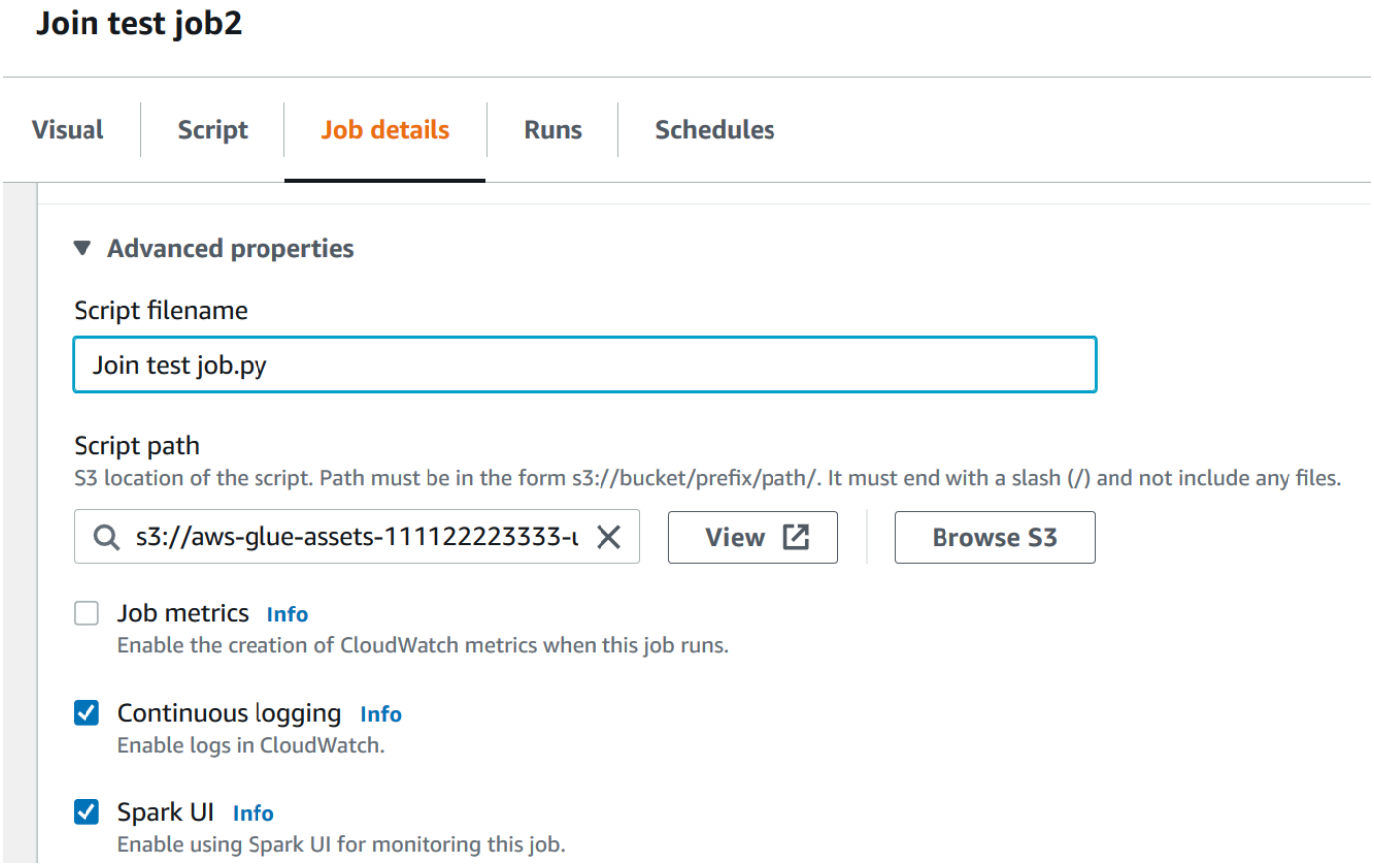
   如果選擇 Confirm (確認)，Visual (視覺效果) 索引標籤不會再出現在編輯器中。您可以使用 AWS Glue Studio 以使用指令碼編輯器來修改指令碼、修改任務詳細資訊或排程，或查看任務執行。

   > **Note**
   >
   > 在您儲存任務之前，轉換為僅限指令碼任務並不是永久性的。如果您重新整理主控台網頁，或在儲存任務之前先關閉任務，然後在視覺化編輯器中重新開啟任務，您仍然可以在視覺化編輯器中編輯個別節點。

4. 視需要編輯指令碼。

   當您完成編輯指令碼時，選擇 Save (儲存) 儲存任務，並將任務從視覺效果永久轉換為僅限指令碼。

5. (選用) 您可以選擇 Script (指令碼) 索引標籤上的 Download (下載) 按鈕，從 AWS Glue Studio 主控台下載指令碼。當您選擇此按鈕時，會開啟新的瀏覽器視窗，顯示其在 Amazon S3 中位置的指令碼。任務的 Job details (任務詳細資訊) 索引標籤中的 Script filename (指令碼檔案名稱) 和 Script path (指令碼路徑) 參數會決定指令碼檔案在 Amazon S3 中的名稱和位置。



當您儲存任務時，AWS Glue 會將任務指令碼儲存在這些欄位指定的位置。如果在 Amazon S3 中修改此位置的指令碼檔案，則您下次編輯任務時，AWS Glue Studio 會載入修改後的指令碼。

在 AWS Glue Studio 中建立和編輯 Scala 指令碼

當您選擇指令碼編輯器來建立任務時，依預設，任務程式設計語言被設定為 Python 3。如果您選擇編寫新的指令碼而不是上傳指令碼，則 AWS Glue Studio 會啟動一個新的指令碼，其中包含以 Python 編寫的樣板文字。如果您想改為編寫 Scala 指令碼，您必須先將指令碼編輯器設定為使用 Scala。

> **Note**
>
> 如果您選擇 Scala 作為任務的程式設計語言，並使用視覺化編輯器來設計任務，則產生的任務指令碼將用 Scala 編寫，並且不需要進一步的操作。

在 AWS Glue Studio 中編寫新的 Scala 指令碼

1. 建立新任務，方法是選擇 Spark script editor (Spark 指令碼編輯器) 選項。

2. 在 Options (選項) 下方，選擇 Create a new script with boilerplate code (使用樣板程式碼建立新指令碼)。
3. 選擇 Job details (任務詳細資訊) 索引標籤並設定 Language (語言) 為 `Scala` (而不是 `Python 3`)。

> **Note**
>
> 當您選擇 Spark script editor (Spark 指令碼編輯器) 選項建立任務，任務的 Type (類型) 屬性會自動設定為 `Spark`。

4. 選擇 Script (指令碼) 索引標籤。
5. 移除 Python 樣板文字。您可以用以下 Scala 樣板文字替換它。

```
import com.amazonaws.services.glue.{DynamicRecord, GlueContext}
import org.apache.spark.SparkContext
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job

object MyScript {
  def main(args: Array[String]): Unit = {
    val sc: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(sc)

    }
}
```

6. 在編輯器中編寫您的 Scala 任務指令碼。視需要新增其他 `import` 陳述式。

在 AWS Glue Studio 中建立和編輯 Python Shell 任務

當您選擇 Python shell 指令碼編輯器來建立任務時，您可以上傳現有的 Python 指令碼，或者編寫新的指令碼。如果您選擇編寫新的指令碼，樣板程式碼將新增到新的 Python 任務指令碼中。

建立新的 Python Shell 任務

請參閱在 AWS Glue Studio 中啟動任務中的指示。

Python Shell 任務支援的任務屬性與 Spark 任務支援的任務屬性不相同。下列清單說明 Job details (任務詳細資訊) 索引標籤上 Python Shell 任務的可用任務參數的變更。

- 任務的 Type (類型) 屬性會自動設定為 `Python Shell`，且無法變更。
- 會有一個任務的 Python version (Python 版本) 屬性，而不是 Language (語言)。目前，在 AWS Glue Studio 中建立的 Python Shell 任務使用 Python 3.6。
- Glue version (Glue 版本) 屬性無法使用，因為它不適用於 Python Shell 任務。
- 會改為顯示 Data processing units (資料處理單元) 屬性，取代 Worker type (工作者類型) 和 Number of workers (工作者數目)。此任務屬性決定執行任務時，Python Shell 會使用多少資料處理單元 (DPU)。
- Job bookmark (任務書籤) 屬性無法使用，因為它不支援 Python Shell 任務。
- 在 Advanced properties (進階屬性) 下，下列屬性不適用於 Python Shell 任務。
  - 任務指標
  - 連續記錄
  - Spark UI 和 Spark UI logs path (Spark UI 日誌路徑)
  - Dependent jars path (相依的 jar 路徑)，在 Libraries (程式庫) 標題下

變更任務圖表中節點的父節點

您可以變更節點的父項，以移動任務圖表中的節點或變更節點的資料來源。

變更父節點

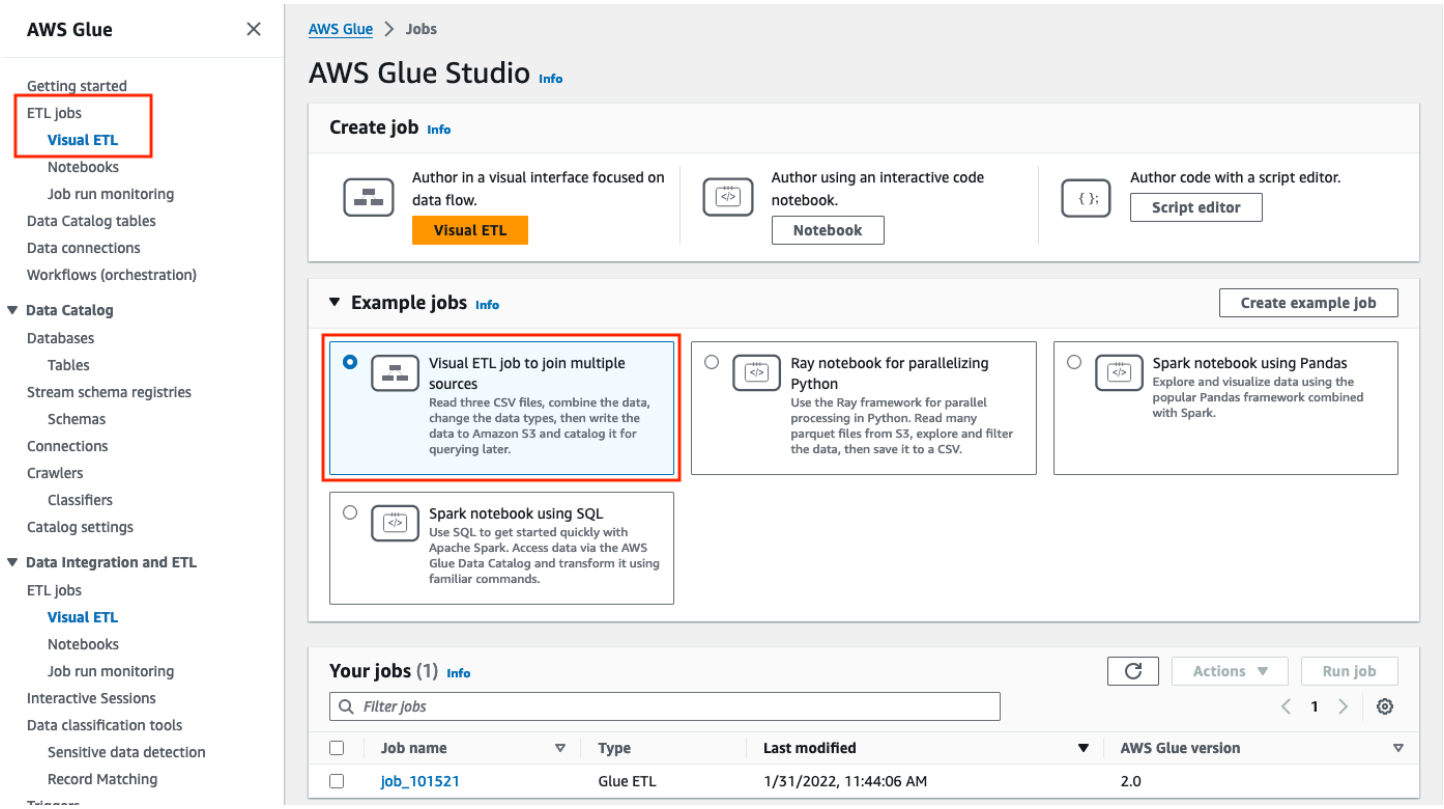
1. 在任務圖表中選擇您要修改的節點。
2. 在節點詳細資訊面板中，在 Node properties (節點屬性) 索引標籤的 Node parents (節點父項) 標題下移除節點的目前父項。
3. 從清單中選擇一個新的父節點。
4. 視需要修改節點的其他屬性，以符合新選取的父節點。

如果您不小心修改了節點，您可以使用 Undo (復原) 按鈕來反轉動作。

從任務圖表中刪除節點

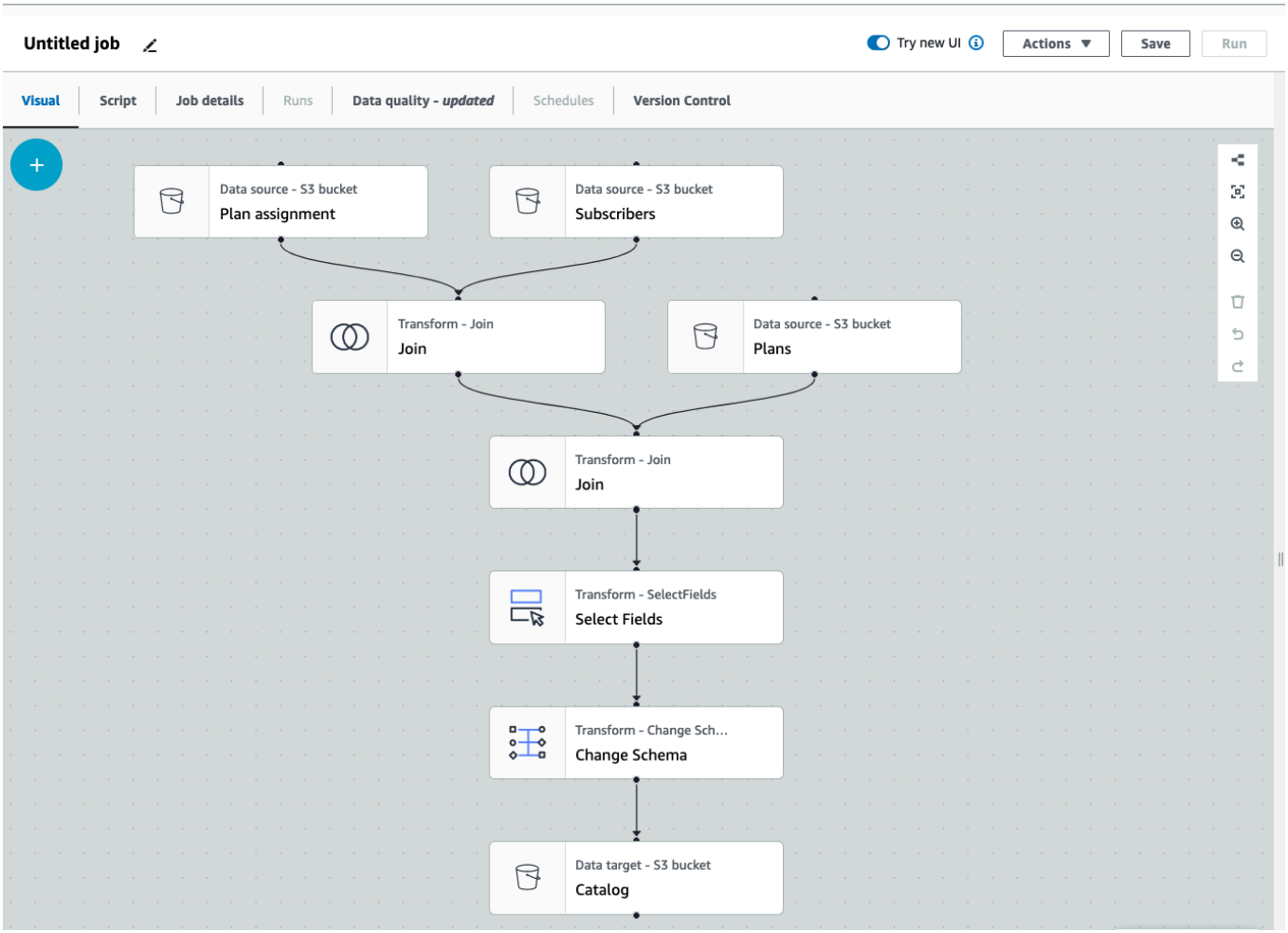
使用 Visual ETL 工作時，您可以從畫布中移除節點，而不必重新新增或重新建構連接至已移除節點的任何節點。

在以下範例中，您可以依照選擇 ETL 工作 > 視覺化 ETL，然後在範例工作中，選擇 Visual ETL 工作來連接多個來源。選擇 [建立範例工作] 以建立工作，並依照下列步驟執行。



若要從畫布中移除節點

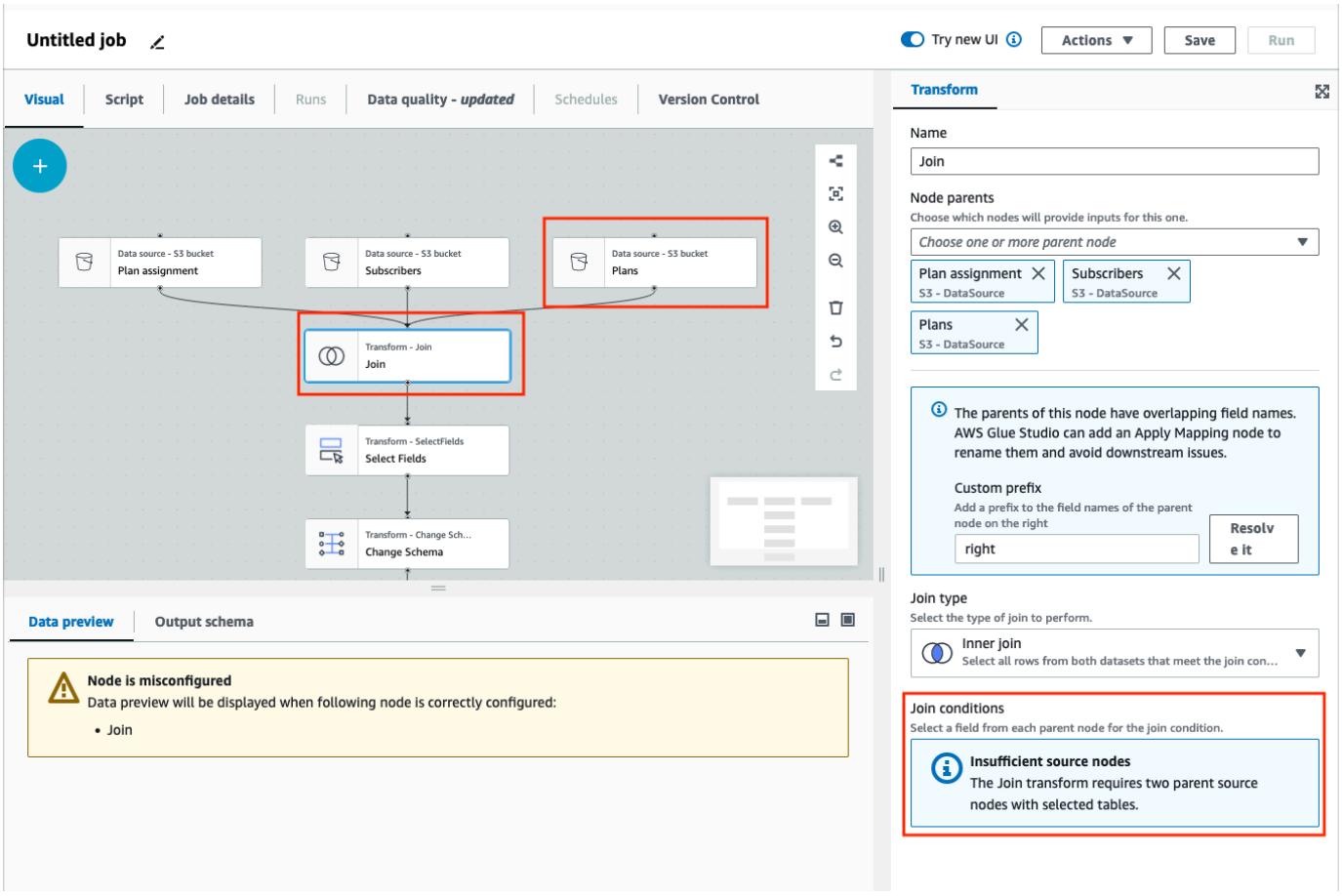
1. 從 AWS Glue 主控台中，從導覽功能表選擇 Visual ETL，然後選擇現有的工作。工作畫布顯示範例工作，如下所示。



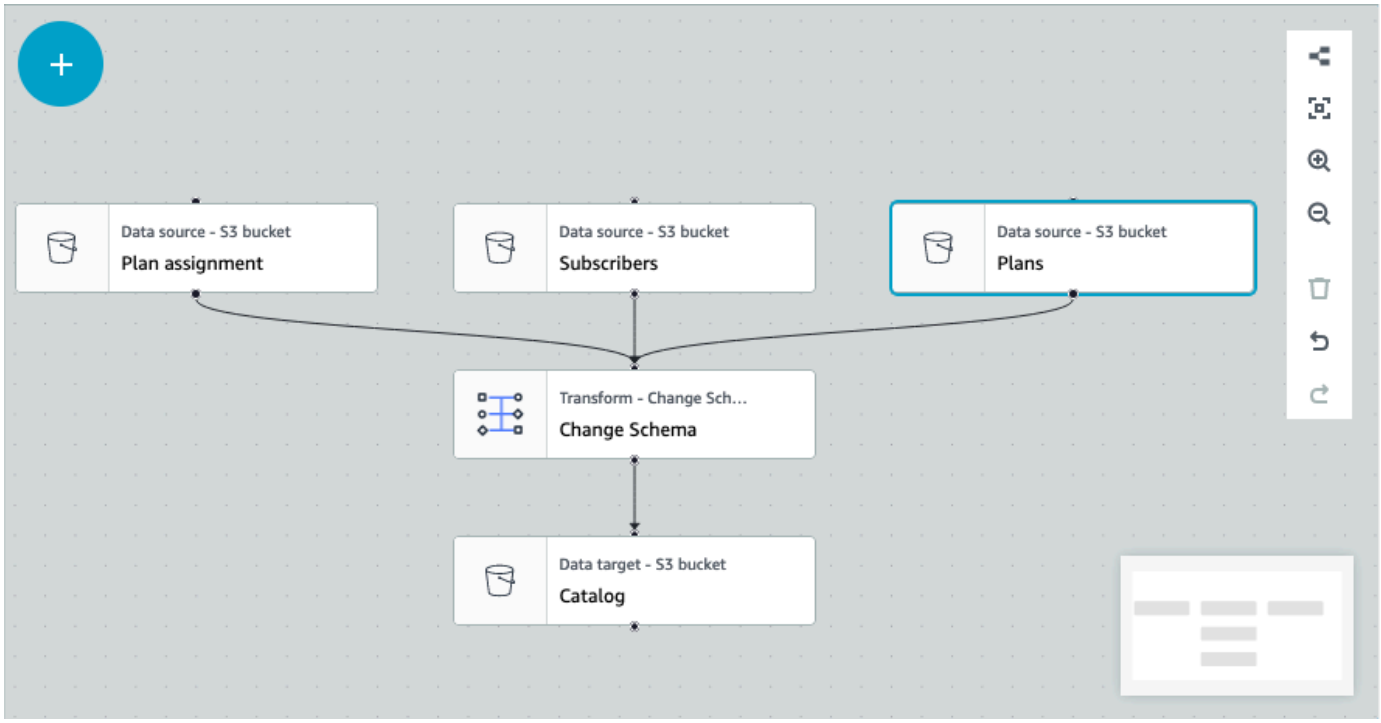
2. 選擇您要移除的節點。畫布將放大到節點。在畫布右側的工具列中，選擇「垃圾桶」圖示。這會移除節點，且連接到該節點的任何節點都會移動到工作流程中。在此範例中，第一個聯結節點已從畫布中刪除。

   如果您刪除工作流程中的節點，則 AWS Glue 會重新排列節點，使其以不會導致工作流程無效的方式進行組織。您可能仍然需要更正節點的配置。

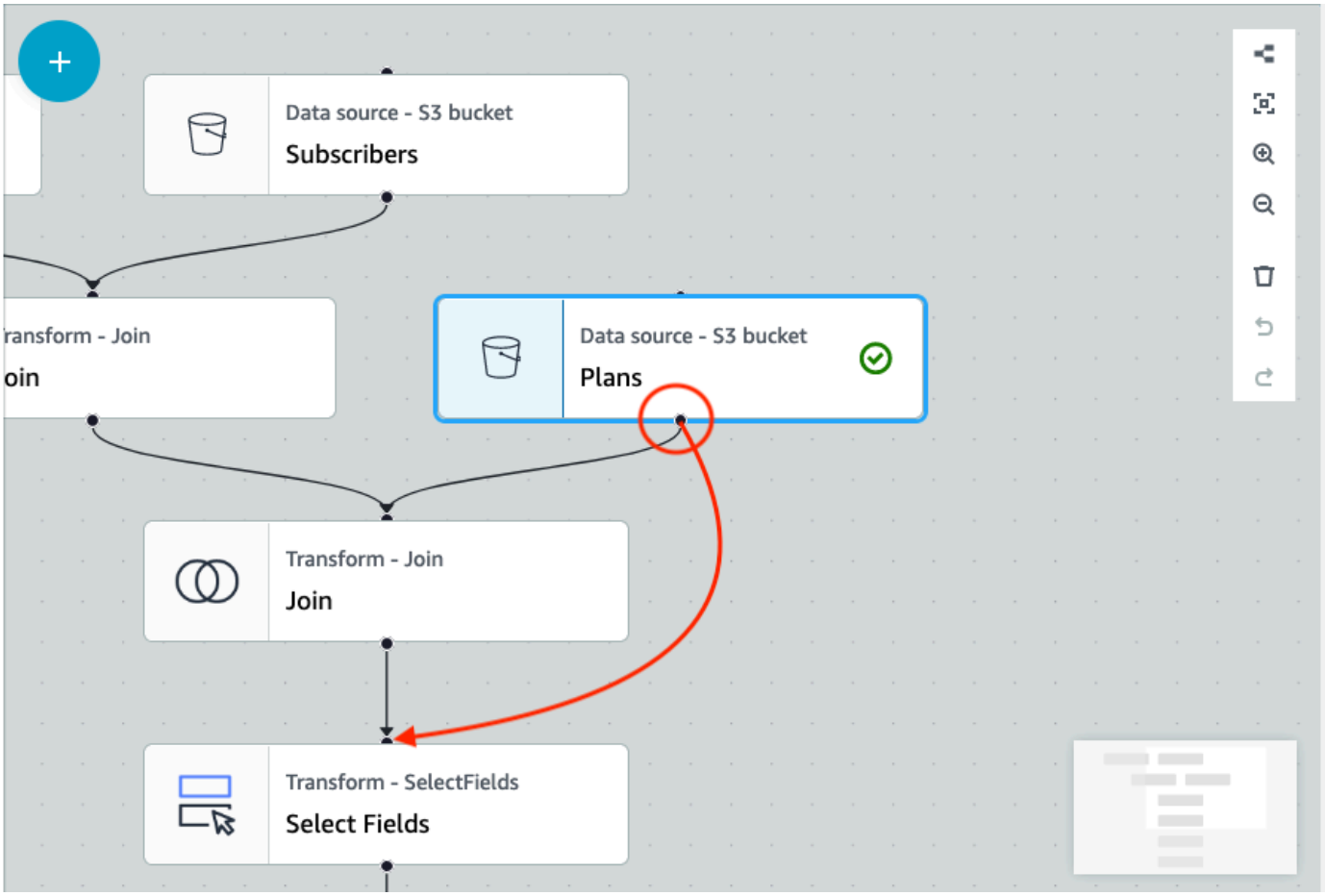
   在該示例中，「訂戶」節點下的「加入」節點已被刪除。因此，[計劃] 來源節點已移至頂層，並且仍連接至子 [聯結] 節點。「連接」節點現在需要額外的配置，因為 Join 需要兩個帶有選定表的父源節點。畫布右側的「轉換」索引標籤會在「聯結」條件下顯示遺失的需求。



3. 刪除第二個「連接」節點和 「選擇字段」 節點。刪除節點後，工作流程將如下所示。



4. 若要修改節點連接，請按一下節點的操作框，然後將連接對拖曳至新節點。這將允許您刪除節點並重新排列邏輯流程中的節點。在此範例中，透過按一下「計劃」(Plans) 節點上的操作框，然後將連接拖曳至「連接」(Join) 節點 (如紅色箭頭所示)，即可建立新連接。



5. 如果您需要還原任何動作，請選擇畫布右側工具列中 「垃圾桶」圖示下方的「復原」圖示。

將來源和目標參數新增至 AWS Glue 資料目錄節點

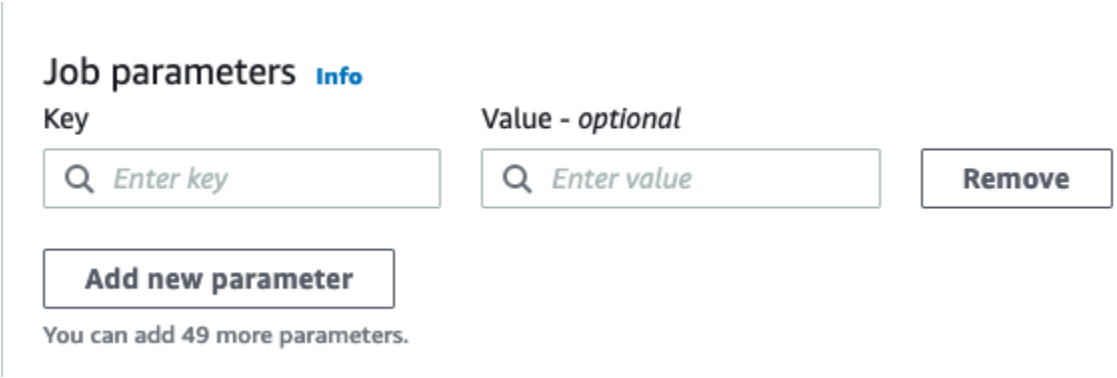
AWS Glue Studio 可讓您將視覺化任務參數化。生產環境和開發環境中的目錄資料表名稱可能不同，因此您可以為任務執行時要執行的資料庫和資料表定義及選取執行時間參數。

任務參數化可讓您將來源和目標參數化，並在使用 AWS Glue 資料目錄節點時，將這些參數儲存至任務。將來源和目標指定為參數，等同於任務可以重複使用，尤其可在多個環境中使用相同的任務。如要節省管理來源和目標所需的時間和心力，在不同部署環境中廣泛使用程式碼，會是相當實用的作法。此外，您指定的自訂參數將覆寫 AWS Glue 任務特定執行的任何預設引數。

新增來源和目標參數

無論您是使用 AWS Glue 資料目錄節點作為來源或目標，您都可以在 Job details (任務詳細資訊) 索引標籤的 Advanced properties (進階屬性) 區段定義執行時間參數。

1. 選擇 AWS Glue 資料目錄節點作為來源節點或目標節點。
2. 選擇 Job details (任務詳細資訊) 索引標籤。
3. 選擇 Advanced properties (進階屬性)。
4. 在「Job parameters」(任務參數) 區段中，輸入索引鍵值，例如資料庫來源的參數可以是 `--db.source`。您可以為索引鍵輸入任何名稱，只要索引鍵名稱後面加上兩個英文破折號即可。

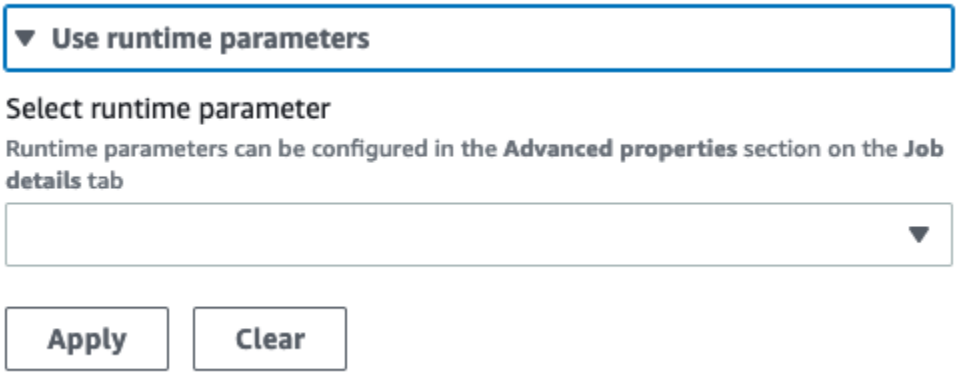


5. 輸入值，例如資料庫參數化的值可以是 `databasename`。
6. 如果您要新增更多參數，請選擇 Add new parameter (新增參數)。最多可新增 50 個參數。定義索引鍵值組之後，您就可以在 AWS Glue 資料目錄節點使用參數。

選取執行時間參數

> **Note**
>
> 無論 AWS Glue 資料目錄節點是來源或目標，為資料庫和資料表選取執行時間參數的程序並無二致。

1. 選擇 AWS Glue 資料目錄節點作為來源節點或目標節點。
2. 在 Data source properties - Data Catalog (資料來源屬性 - 資料目錄) 索引標籤的 Database (資料庫) 底下，選擇 Use runtime parameters (使用執行時間參數)。



3. 從下拉式選單中選擇參數。例如，如果您選取您為來源資料庫定義的參數，當您選擇 Apply (套用) 後，資料庫會自動填入資料庫下拉式選單。
4. 在「Table」(資料表) 區段中，選擇您已定義為來源資料表的參數。一旦您選擇 Apply (套用)，資料表就會自動填入，作為您要使用的資料表。
5. 當您儲存並執行任務，AWS Glue Studio 就會在任務執行期間參照您選取的參數。

在 AWS Glue 中使用 Git 版本控制系統

> **Note**
>
> 筆記本目前不支援 AWS Glue Studio 中的版本控制。然而，可支援 AWS Glue 任務指令碼和視覺化 ETL 任務的版本控制。

如果您有遠端儲存庫，並且想要使用儲存庫來管理AWS Glue工作，您可以使用AWS Glue Studio或將變更同步 AWS CLI 到儲存庫和中的工作AWS Glue。以這種方式同步變更，等於是將任務從 AWS Glue Studio 推送到您的儲存庫，或從儲存庫提取到 AWS Glue Studio。

在 AWS Glue Studio 中完成 Git 整合後，您可以：

- 與 Git 版本控制系統集成，例如 AWS CodeCommit， GitHub GitLab，和比特桶
- 無論您是使用視覺化任務或指令碼任務，都能在 AWS Glue Studio 中編輯 AWS Glue 任務，並將任務同步到儲存庫
- 將任務中的來源和目標參數化
- 從儲存庫提取任務，並在 AWS Glue Studio 中編輯
- 利用 AWS Glue Studio 的多分支工作流程，從分支提取任務及/或將任務推送到分支，藉以測試任務

- 從儲存庫下載檔案並將任務上傳至 AWS Glue Studio，以便建立跨帳戶任務
- 使用您選擇的自動化工具（例如詹金斯 AWS CodeDeploy等）

此影片示範如何將 G AWS lue 與 Git 整合，並建立連續且協同合作的程式碼管線。

IAM 許可

確保任務具有下列其中一個 IAM 許可。如需有關設定 IAM 許可的詳細資訊，請參閱設定 AWS Glue Studio 的 IAM 許可。

- `AWSGlueServiceRole`
- `AWSGlueConsoleFullAccess`

整合 Git 至少需要執行下列動作：

- `glue:UpdateJobFromSourceControl` — 以便能使用版本控制系統中的任務更新 AWS Glue
- `glue:UpdateSourceControlFromJob` — 以便能使用儲存於 AWS Glue 的任務更新版本控制系統
- `s3:GetObject` — 以便能擷取任務的指令碼，同時將指令碼推送至版本控制系統
- `s3:PutObject` — 以便能在從來源控制系統提取任務時更新指令碼

必要條件

為了將任務推送至來源控制儲存庫，您將需要：

- 已由您管理員建立的儲存庫
- 儲存庫中的分支
- 個人存取權杖 (如果是 Bitbucket，此為儲存庫存取權杖)
- 儲存庫擁有者的使用者名稱
- 在儲存庫中設定許可，以允許 AWS Glue Studio 讀取和寫入儲存庫
  - GitLab-將令牌範圍設置為 API，讀取存儲庫和寫入存儲庫
  - Bitbucket – 將許可設定為：
    - 工作區成員資格 – 讀取、寫入
    - 專案 – 寫入、管理員讀取
    - 儲存庫 – 讀取、寫入、管理員、刪除

> **Note**
>
> 使用時 AWS CodeCommit，不需要個人訪問令牌和存儲庫所有者。請參閱 Getting started with Git and AWS CodeCommit。

使用來自 AWS Glue Studio 中的來源控制儲存庫的任務

為了從來源控制儲存庫提取不在 AWS Glue Studio 中的任務，並在 AWS Glue Studio 使用該任務，不同任務類型會有不同的先決條件。

對於視覺化任務：

- 您需要一個資料夾和任務定義 (需與任務名稱相符) 的 JSON 檔案

  如需範例，請參閱下方的任務定義。儲存庫中的分支應包含路徑 `my-visual-job/my-visual-job.json`，其中資料夾和 JSON 檔案都需與任務名稱相符

```
{
  "name" : "my-visual-job",
  "description" : "",
  "role" : "arn:aws:iam::aws_account_id:role/Rolename",
  "command" : {
    "name" : "glueetl",
    "scriptLocation" : "s3://foldername/scripts/my-visual-job.py",
    "pythonVersion" : "3"
  },
  "codeGenConfigurationNodes" : "{\"node-nodeID\":{\"S3CsvSource\":
{\"AdditionalOptions\":{\"EnableSamplePath\":false,\"SamplePath\":\"s3://notebook-
test-input/netflix_titles.csv\"},\"Escaper\":\"\",\"Exclusions\":[],\"Name\":\"Amazon
 S3\",\"OptimizePerformance\":false,\"OutputSchemas\":[{\"Columns\":[{\"Name\":
\"show_id\",\"Type\":\"string\"},{\"Name\":\"type\",\"Type\":\"string\"},{\"Name\":
\"title\",\"Type\":\"choice\"},{\"Name\":\"director\",\"Type\":\"string\"},{\"Name\":
\"cast\",\"Type\":\"string\"},{\"Name\":\"country\",\"Type\":\"string\"},{\"Name\":
\"date_added\",\"Type\":\"string\"},{\"Name\":\"release_year\",\"Type\":\"bigint\"},
{\"Name\":\"rating\",\"Type\":\"string\"},{\"Name\":\"duration\",\"Type\":\"string
\"},{\"Name\":\"listed_in\",\"Type\":\"string\"},{\"Name\":\"description\",\"Type
\":\"string\"}]}],\"Paths\":[\"s3://dalamgir-notebook-test-input/netflix_titles.csv
\"],\"QuoteChar\":\"quote\",\"Recurse\":true,\"Separator\":\"comma\",\"WithHeader
\":true}}}"
}
```

對於指令碼任務：

- 您需要一個資料夾、內含任務定義的 JSON 檔案和指令碼
- 資料夾和 JSON 檔案應與任務名稱相符。指令碼名稱必須與任務定義中的 `scriptLocation` 及檔案附檔名相符

  例如在下方的任務定義中，儲存庫中的分支應包含路徑 `my-script-job/my-script-job.json` 和 `my-script-job/my-script-job.py`。指令碼名稱應與 `scriptLocation` 中的名稱相符，包括指令碼的附檔名

```
{
  "name" : "my-script-job",
  "description" : "",
  "role" : "arn:aws:iam::aws_account_id:role/Rolename",
  "command" : {
    "name" : "glueetl",
    "scriptLocation" : "s3://foldername/scripts/my-script-job.py",
    "pythonVersion" : "3"
  }
}
```

限制

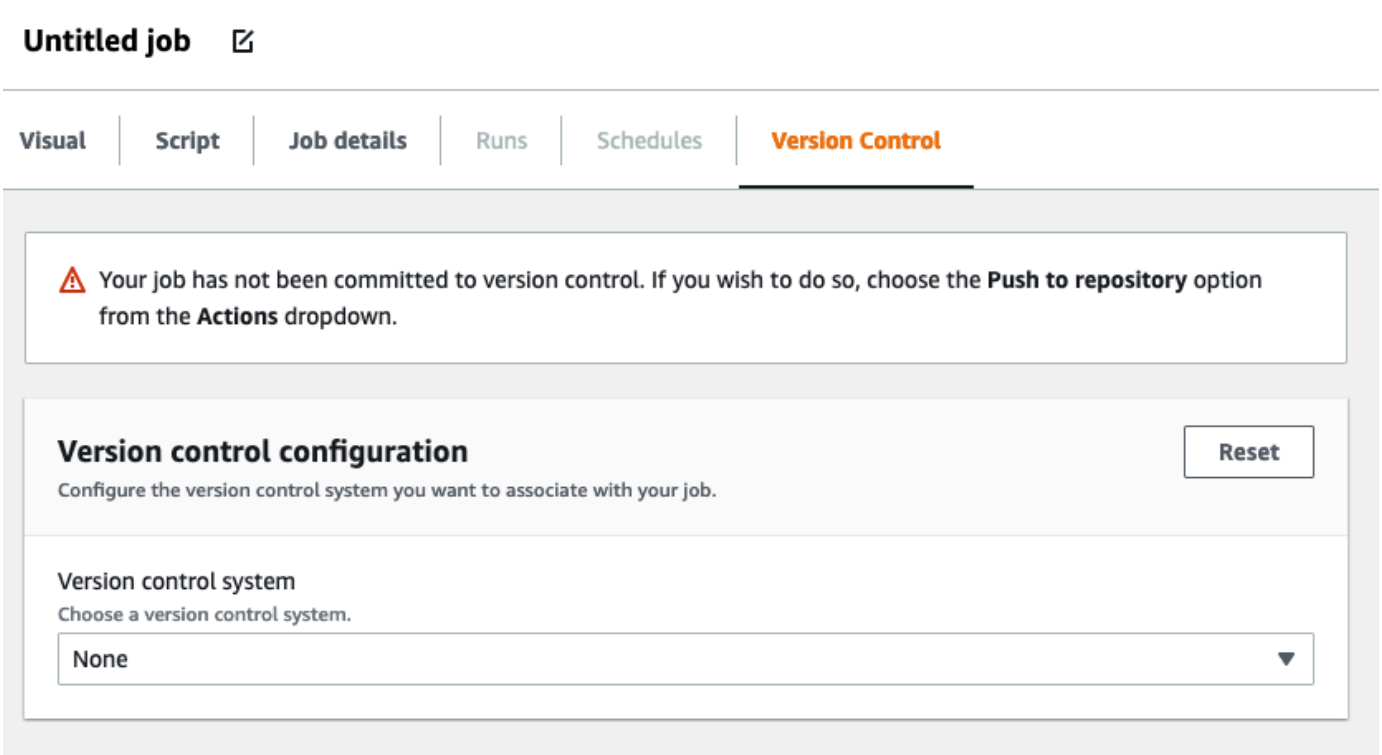
- AWS Glue目前不支持從-組推/拉GitLab。

將版本控制儲存庫與 AWS Glue 連線

您可以輸入您的版本控制儲存庫詳細資訊，並在 AWS Glue Studio 任務編輯器的 Version Control (版本控制) 索引標籤中管理。若要與您的 Git 儲存庫整合，則必須在每次登入 AWS Glue Studio 時連線至您的儲存庫。

連接 Git 版本控制系統：

1. 在 AWS Glue Studio 中，開始新任務並選擇 Version Control (版本控制) 索引標籤。



2. 在版本控制系統中按一下下拉式選單，從可用選項中選擇 Git 服務。

   - AWS CodeCommit
   - GitHub
   - GitLab
   - Bitbucket

3. 根據您選擇的 Git 版本控制系統，您會需要完成不同的欄位。

   對於 AWS CodeCommit：

   選取任務的儲存庫和分支，完成儲存庫組態：

   - 存放庫 — 如果您已在中設定存放庫 AWS CodeCommit，請從下拉式功能表中選取存放庫。您的儲存庫會自動填入清單中
   - 分支 – 從下拉式選單中選取分支
   - 資料夾 – 選用 – 輸入要儲存任務的資料夾名稱。如果留空，系統會為您自動建立資料夾。資料夾名稱會預設為任務名稱

   對於 GitHub：

完成下列欄位以完成 GitHub 設定：

- 個人訪問令牌-這是存 GitHub 儲庫提供的令牌。有關個人訪問令牌的更多信息，請參閱 GitHub 文檔
- 存放庫擁有者 — 這是 GitHub 存放庫的擁有者。

從中選取儲存區域和分支，以完成儲存區域組態 GitHub。

- 存放庫 — 如果您已在中設定存放庫 GitHub，請從下拉式功能表中選取存放庫。您的儲存庫會自動填入清單中
- 分支 – 從下拉式選單中選取分支
- 資料夾 – 選用 – 輸入要儲存任務的資料夾名稱。如果留空，系統會為您自動建立資料夾。資料夾名稱會預設為任務名稱

對於 GitLab：

> **Note**
>
> AWS Glue目前不支持從-組推/拉GitLab。

- 個人訪問令牌-這是存 GitLab 儲庫提供的令牌。有關個人訪問令牌的更多信息，請參閱 GitLab 個人訪問令牌
- 存放庫擁有者 — 這是 GitLab 存放庫的擁有者。

從中選取儲存區域和分支，以完成儲存區域組態 GitLab。

- 存放庫 — 如果您已在中設定存放庫 GitLab，請從下拉式功能表中選取存放庫。您的儲存庫會自動填入清單中
- 分支 – 從下拉式選單中選取分支
- 資料夾 – 選用 – 輸入要儲存任務的資料夾名稱。如果留空，系統會為您自動建立資料夾。資料夾名稱會預設為任務名稱

對於 Bitbucket：

- 應用程序密碼 — Bitbucket 使用應用程序密碼，而不是存儲庫訪問令牌。如需應用程式密碼的詳細資訊，請參閱應用程式密碼
- 儲存庫擁有者 – 這是 Bitbucket 儲存庫的擁有者。在 Bitbucket 中，擁有者是儲存庫的建立者。

選取 Bitbucket 的工作空間、儲存庫、分支和資料夾，完成儲存庫組態。

- 工作空間 – 如果您在 Bitbucket 中設定工作空間，請從下拉式選單中選取該工作空間。系統會自動填入您的工作空間
- 儲存庫 – 如果您在 Bitbucket 中設定儲存庫，請從下拉式選單中選取該儲存庫。系統會自動填入您的儲存庫
- 分支 – 從下拉式選單中選取分支。系統會自動填入您的分支
- 資料夾 – 選用 – 輸入要儲存任務的資料夾名稱。如果留空，系統會使用任務名稱為您自動建立資料夾。

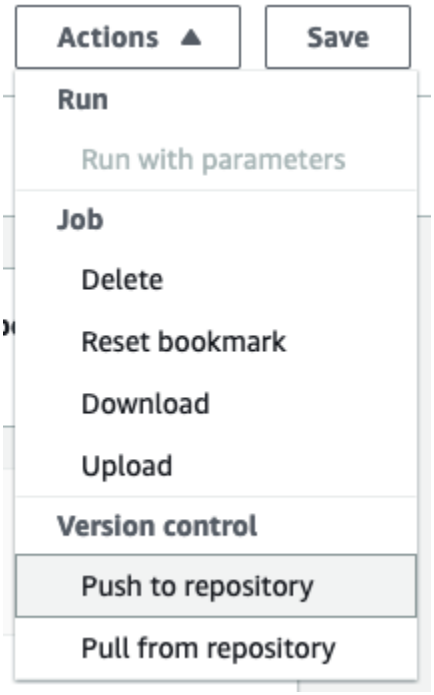
4. 在 AWS Glue Studio 任務的頂端選擇 Save (儲存)

將 AWS Glue 任務推送到來源儲存庫

輸入版本控制系統的詳細資訊後，您就可以在 AWS Glue Studio 編輯任務，並將任務推送到您的來源儲存庫。如果您對 Git 概念 (例如推送和提取) 不甚熟悉，請參閱開始使用 Git 和 AWS CodeCommit 教學。

如要將任務推送到儲存庫，您需要輸入版本控制系統的詳細資訊並儲存任務。

1. 在 AWS Glue Studio 任務中，選擇 Actions (動作)。其他選單選項會隨即開啟。



2. 選擇 Push to repository (推送到儲存庫)。

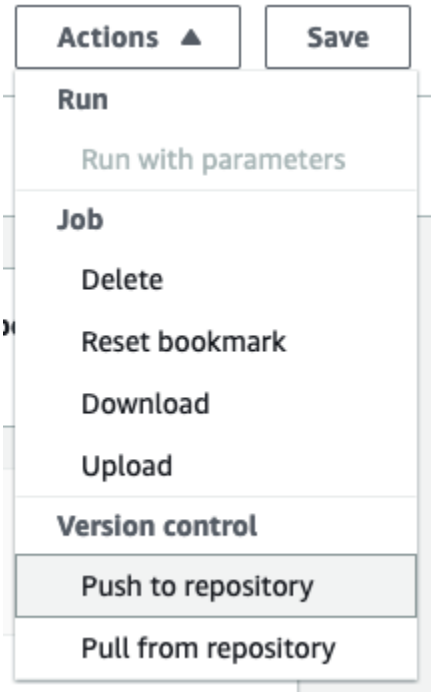
   此動作會儲存任務。將任務推送到儲存庫時，AWS Glue Studio 會推送最近一次儲存的變更。如果儲存庫中的任務已由您或其他使用者修改，而且與 AWS Glue Studio 中的任務不同步，儲存庫中的任務就會由您從 AWS Glue Studio 推送且儲存於 AWS Glue Studio 的任務覆寫。

3. 選擇 Confirm (確認) 以完成動作。系統會隨即在儲存庫中建立新的遞交。如果您使用的是 AWS CodeCommit，確認訊息會顯示最新提交的連結 AWS CodeCommit。

從來源儲存庫提取 AWS Glue 任務

只要您在 Version control (版本控制) 索引標籤中輸入 Git 儲存庫的詳細資訊，您也可以從儲存庫提取任務，並在 AWS Glue Studio 中編輯。

1. 在 AWS Glue Studio 任務中，選擇 Actions (動作)。其他選單選項會隨即開啟。



2. 選擇 Pull from repository (從儲存庫提取)。
3. 選擇 Confirm (確認)。如要這麼做，您必須先從儲存庫取得最新的遞交，並在 AWS Glue Studio 中更新任務。
4. 在 AWS Glue Studio 中編輯任務。完成變更後，您可以從 Actions (動作) 選單中選擇 Push to repository (推送到儲存庫)，將任務同步至儲存庫。

使用 AWS Glue Studio 筆記本編寫程式碼

使用 AWS Glue Studio 中的互動式筆記本介面或 AWS Glue 中的互動式工作階段，資料工程師可以比之前更快、更輕鬆地撰寫 AWS Glue 任務。

主題

- 使用筆記本概觀
- 使用 AWS Glue Studio 中的筆記本建立 ETL 任務
- 筆記本編輯器元件
- 儲存您的筆記本和任務指令碼
- 管理筆記本工作階段
- CodeWhisperer 搭配使用 AWS Glue Studio notebooks

使用筆記本概觀

AWS Glue Studio 允許您在基於 Jupyter 筆記本的筆記本介面中以互動方式編寫任務。透過 AWS Glue Studio 中的筆記本，您可以編輯任務指令碼和檢視輸出，而無需執行完整任務；可以編輯資料整合程式碼和檢視輸出，而無需執行完整任務；還可以新增 Markdown 並將筆記本儲存為 .ipynb 檔案和任務指令碼。您可以直接啟動筆記型，而無需在本機安裝軟體或管理伺服器。對程式碼感到滿意之後，只要按一下按鈕，AWS Glue Studio 就可將您的筆記本轉換為 Glue 任務。

使用筆記本的若干優點包括：

- 無需佈建或管理叢集
- 沒有閒置叢集要支付費用
- 無需預先設定
- 無需安裝 Jupyter 筆記本
- 與 AWS Glue ETL 相同的執行時間/平台

透過 AWS Glue Studio 啟動筆記本，所有的設定步驟都會自動完成，以便您在幾秒鐘之後探索資料並開始開發任務指令碼。AWS Glue Studio 設定帶有 AWS Glue Jupyter 核心的 Jupyter 筆記本。您不需要設定 VPC、網路連線或開發端點，即可使用此筆記本。

若要使用筆記本介面建立任務：

- 設定必要的 IAM 許可。
- 啟動筆記本工作階段以建立任務
- 在筆記本的儲存格中編寫程式碼
- 執行並測試程式碼以檢視輸出
- 儲存任務

儲存您的筆記本之後，此筆記本就是完整的 AWS Glue 任務。您可以管理任務的所有方面，例如排程任務執行、設定任務參數，以及直接在筆記本旁檢視任務執行歷史記錄。

使用 AWS Glue Studio 中的筆記本建立 ETL 任務

開始在 AWS Glue Studio 主控台中使用筆記本

1. 連接 AWS Identity and Access Management 政策至 AWS Glue Studio 使用者，並為您的 ETL 任務和筆記本建立 IAM 角色。

2. 按 授予 IAM 角色的許可 中所述，為筆記本設定其他 IAM 安全性。
3. 開啟位於 https://console.aws.amazon.com/gluestudio/ 的 AWS Glue Studio 主控台。

> **Note**
>
> 檢查您的瀏覽器是否未封鎖第三方 Cookie。任何因預設或使用者啟用設定而封鎖第三方 Cookie 的瀏覽器，將使筆記本無法啟動。如需管理 Cookie 的詳細資訊，請參閱：
>
> - Chrome
> - Firefox
> - Safari

4. 選擇左側導覽選單中的 Jobs (任務) 連結。
5. 選擇 Jupyter 筆記本，然後選擇 Create (建立) 開始新的筆記本工作階段。
6. 在 Create job in Jupyter notebook (在 Jupyter 筆記本中建立任務) 頁面上，提供任務名稱並選擇要使用的 IAM 角色。選擇 Create job (建立任務)。

   短時間後，會出現筆記本編輯器。
7. 新增程式碼後，必須執行儲存格以啟動工作階段。可透過多種方式執行儲存格：
   - 按下執行按鈕。
   - 使用鍵盤快速鍵：
     - 在 MacOS 上，使用 Command + Enter 來執行儲存格。
     - 在 Windows 上，使用 Shift + Enter 來執行儲存格。

   如需使用 Jupyter 筆記本介面撰寫程式碼的相關資訊，請參閱 Jupyter 筆記本使用者文件。
8. 若要測試指令碼，請執行整個指令碼或個別儲存格。任何命令輸出將顯示在儲存格下方的區域中。
9. 在您完成開發筆記本之後，您可以儲存任務，然後執行任務。您可以在 Script (指令碼) 索引標籤中找到指令碼。您新增至筆記本的任何魔術命令都會遭到去除，且不會儲存為已產生之 AWS Glue 任務指令碼的一部分。AWS Glue Studio 會將 `job.commit()` 自動新增至已從筆記本內容產生之指令碼的末尾。

   如需有關執行任務的詳細資訊，請參閱 開始任務執行。

筆記本編輯器元件

筆記本編輯器介面包含下列主要區段。

- 筆記本介面 (主面板) 與工具列
- 任務編輯索引標籤

筆記本編輯器

AWS Glue Studio 筆記本編輯器是基於 Jupyter 筆記本應用程式。AWS Glue Studio 筆記本介面與 Juypter Notebook 所提供的介面類似，請參閱筆記本使用者介面。互動式工作階段使用的筆記本是 Jupyter 筆記本。

雖然 AWS Glue Studio 筆記本與 Juptyer Notebook 類似，但它在幾個關鍵方面有所不同：

- 目前 AWS Glue Studio 筆記本無法安裝擴充
- 無法使用多個索引標籤；任務與筆記本之間存在 1:1 的關係
- AWS Glue Studio 筆記本沒有 Jupyter 筆記本中存在的相同頂部檔案選單
- AWS Glue Studio 筆記本目前僅能使用 AWS Glue 核心執行。請注意，您無法自行更新核心。

AWS Glue Studio 任務編輯索引標籤

用來與 ETL 任務互動的索引標籤位於筆記本頁面的頂端。它們類似於顯示在 AWS Glue Studio 的視覺化任務編輯器中的索引標籤，並且執行相同的動作。

- Notebook (筆記本) – 使用此索引標籤檢視使用筆記本介面的任務指令碼。
- Job details (任務詳細資訊) – 設定任務執行的環境和屬性。
- Runs (執行) – 檢視此任務先前執行的相關資訊。
- Schedules (排程) – 設定在特定時間執行任務的排程。

儲存您的筆記本和任務指令碼

您可以隨時儲存您的筆記本和正在建立的任務指令碼。只要選擇右上角的 Save (儲存) 按鈕，就像使用視覺化編輯器或指令碼編輯器一樣。

選擇 Save (儲存) 後，筆記本檔案將儲存在預設位置：

- 依預設，任務指令碼會儲存至 Job Details (任務詳細資訊) 索引標籤 Advanced properties (進階屬性) 下的任務詳細資訊屬性 Script path (指令碼路徑) 中指示的 Amazon S3 位置。任務指令碼會儲存在名為 `Scripts` 的子資料夾內。
- 依預設，筆記本檔案 (`.ipynb`) 會儲存至 Job Details (任務詳細資訊) 索引標籤 Advanced properties (進階屬性) 下的任務詳細資訊 Script path (指令碼路徑) 中指示的 Amazon S3 位置。筆記本檔案會儲存在名為 `Notebooks` 的子資料夾內。

> **Note**
>
> 當您儲存任務時，任務指令碼僅包含來自筆記本的程式碼儲存格。任務指令碼中不包括 Markdown 儲存格和魔術命令。但是，`.ipynb` 檔案將包含任何 Markdown 和魔術命令。

儲存任務之後，您就可以使用在筆記本中建立的指令碼來執行任務。

管理筆記本工作階段

AWS Glue Studio 中的筆記本以 AWS Glue 的互動式工作階段功能為基礎。使用互動式工作階段需要支付費用。為了協助管理成本，您可以監控為帳戶建立的工作階段，並設定所有工作階段的預設值。

變更所有筆記本工作階段的預設逾時

依預設，如果筆記本已啟動並且沒有執行任何儲存格，佈建的 AWS Glue Studio 筆記本會在 12 小時後逾時。不存在相關成本，且無法設定逾時。

一旦您執行儲存格，這便會啟動互動式工作階段。此工作階段的預設逾時時間為 48 小時。此逾時可以透過在執行儲存格之前傳遞 `%idle_timeout` 魔術命令來設定。

若要修改 AWS Glue Studio 中筆記本的預設工作階段逾時

1. 在筆記本中，於儲存格中輸入 `%idle_timeout` 魔術命令，並以分鐘為單位指定逾時值。
2. 例如：`%idle_timeout 15` 會將預設逾時變更為 15 分鐘。如果 15 分鐘內未使用工作階段，則工作階段會自動停止。

安裝其他 Python 模組

如果您想使用 pip 將其他模組安裝到工作階段中，則可以使用 `%additional_python_modules` 將其新增至您的工作階段中：

```
%additional_python_modules awswrangler, s3://mybucket/mymodule.whl
```

additional_python_modules 的所有引數都傳遞給 `pip3 install -m <>`

如需可用 Python 模組的清單，請參閱 Using Python libraries with AWS Glue。

變更 AWS Glue 組態

您可以使用魔術命令來控制 AWS Glue 任務組態值。如果您要變更任務組態值，則必須在筆記本中使用適當的魔術命令。請參閱 Magics supported by AWS Glue interactive sessions for Jupyter。

> **Note**
>
> 無法再覆寫執行中工作階段的屬性。若要變更工作階段的組態，您可以停止工作階段、設定新的組態，然後啟動新的工作階段。

AWS Glue 支援各種工作者類型。您可以使用 `%worker_type` 設定工作者類型。例如：`%worker_type G.2X` 。預設值是 G.1X。

您還可以使用 `%number_of_workers` 指定工作者數量。例如，若要指定 40 個工作者：`%number_of_workers 40`。

如需詳細資訊，請參閱定義任務屬性。

停止筆記本工作階段

若要停止筆記本工作階段，請使用魔術命令 `%stop_session`。

如果您瀏覽離開 AWS 主控台中的筆記本，則會收到警告訊息，其中可以選擇停止工作階段。

CodeWhisperer 搭配使用 AWS Glue Studio notebooks

AWS Glue Studio 允許您在基於 Jupyter 筆記本的筆記本介面中以互動方式編寫任務。使用可 CodeWhisperer 改善AWS Glue Studio筆記本中的編寫體驗。

Amazon CodeWhisperer 擴充功能支援撰寫程式碼，方法是產生程式碼建議，並建議與程式碼問題相關的改進。

什麼是 Amazon CodeWhisperer？

Amazon CodeWhisperer 是由機器學習提供支援的服務，可協助提升開發人員生產力。CodeWhisperer 通過根據開發人員在自然語言中的註釋及其在 IDE 中的代碼生成代碼建議來實現這一目標。在預覽期間 CodeWhisperer ，Amazon 可用於 Java JavaScript，Python，C# 和 TypeScript 編程語言。此服務與 Amazon SageMaker Studio JupyterLab、 Amazon SageMaker 筆記型電腦執行個體及其他整合式開發環境 (IDE) 整合。

如需詳細資訊，請參閱設定方 CodeWhisperer 式AWS Glue Studio。

主控台中的 AWS Glue 任務執行狀態

您可在 AWS Glue 擷取、轉換和載入 (ETL) 任務執行中或停止之後，檢視其狀態。您可以使用 AWS Glue 主控台檢視狀態。如需有關任務執行狀態的詳細資訊，請參閱 the section called “任務執行狀態”。

存取任務監控儀表板

若要存取任務監控儀表板，請選擇 AWS Glue 導覽窗格中的 監控連結。

任務監控儀表板的概觀

任務監控儀表板提供任務執行的整體摘要，其中狀態為執行中、已取消、成功或失敗。其他圖標可提供整體任務執行成功率、任務的預估 DPU 使用量、依任務類型、工作者類型及日期劃分的任務狀態計數細目。

圖標中的圖形是互動式。您可以選擇圖形中的任何區塊來執行篩選，該篩選僅顯示頁面底部任務執行資料表中的那些任務。

您可以變更此頁面上所顯示資訊的日期範圍，方法是使用日期範圍選取器。當您變更日期範圍時，資訊圖標會調整以顯示目前日期之前指定天數的值。如果您從日期範圍選取器選擇自訂，則也可以使用特定日期範圍。

任務執行檢視

> **Note**
>
> Job 流程和作業執行可存取 90 天的工作執行歷程記錄。

任務執行資源清單會顯示指定日期範圍和篩選的任務。

您可以根據其他準則篩選任務，例如狀態、工作者類型、任務類型和任務名稱。在資料表頂端的篩選方塊中，您可以輸入要用作篩選的文字。當您輸入文字時，資料表結果會以包含相符文字的列進行更新。

您可以從任務監控儀表板上的圖形中選擇元素，來檢視任務的子集。例如，如果您在任務執行摘要圖標中選擇執行中的任務數目，接著任務執行清單僅會顯示目前狀態為 Running 的任務。如果您選擇列在工作者類型明細長條圖中的其中一項，則只有具有相符工作者類型和狀態的任務執行會顯示在任務執行清單中。

任務執行資源清單會顯示任務執行的詳細資訊。您可以選擇資料欄標題來排序資料表中的資料列。此資料表包含以下資訊：

屬性	描述
任務名稱	任務的名稱。
Type	任務環境的類型： • Glue ETL：在由 AWS Glue 管理的 Apache Spark 環境中執行。 • Glue 串流：在 Apache Spark 環境中執行，並在資料串流上執行 ETL。 • Python shell：將 Python 指令碼作為 Shell 執行。
開始時間	此次任務執行開始的日期和時間。
結束時間	此次任務執行完成的日期和時間。
執行狀態	任務執行目前的狀態。值可以為： • STARTING • RUNNING • STOPPING • STOPPED • SUCCEEDED • FAILED

屬性	描述
	• TIMEOUT
執行時間	任務執行消耗資源所需的時間量。
容量	配置給此任務執行的 AWS Glue 資料處理單位 (DPU) 數目。如需容量規劃的詳細資訊，請參閱 AWS Glue 開發人員指南中的 DPU 容量規劃監控。

屬性	描述
工作者類型	在任務執行時配置的預先定義工作者類型。值可以為 G.1X、G.2X、G.4X 或 G.8X。 • **G.1X** – 選擇這種類型時，您也要提供 工作者數目 的值。每個工作者都會映射到 1 個 DPU (4 個 vCPU、16 GB 記憶體)，外加 84 GB 磁碟 (大約 34 GB 可用)。我們建議記憶體密集型任務採用這種工作者類型。此為 AWS Glue 2.0 版本或更新版本任務的預設工作者類型。 • **G.2X** – 選擇這種類型時，您也要提供 工作者數目 的值。每個工作者都會映射到 2 個 DPU (8 個 vCPU、32 GB 記憶體)，外加 128 GB 磁碟 (大約 77 GB 可用)。我們建議記憶體密集型任務和執行機器學習轉換的任務採用這種工作者類型。 • **G.4X** – 選擇這種類型時，您也要提供 工作者數目 的值。每個工作者都會映射到 4 個 DPU (16 個 vCPU、64 GB 記憶體)，外加 256 GB 磁碟 (大約 235 GB 可用)。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此工作者類型僅適用於以下 AWS 區域內 AWS Glue 3.0 版或更新版本的 Spark ETL 任務：美國東部 (俄亥俄)、美國東部 (維吉尼亞北部)、美國西部 (奧勒岡)、亞太區域 (新加坡)、亞太區域 (雪梨)、亞太區域 (東京)、加拿大 (中部)、歐洲 (法蘭克福)、歐洲 (愛爾蘭) 和歐洲 (斯德哥爾摩)。 • **G.8X** – 選擇這種類型時，您也要提供 工作者數目 的值。每個工作者都會映射到 8 個 DPU (32 個 vCPU、128 GB 記憶體)，外加 512 GB 磁碟 (大約 487 GB 可用)。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此背景工作類型僅適用於 3.0 AWS Glue 版或更新版本的 Spark

屬性	描述
	ETL 工作，與G.4X背景工作者類型支援的相同 AWS 區域。
DPU 時數	任務執行的預估 DPU 數目。DPU 是處理能力的相對測量。DPU 用來判斷執行任務的費用。如需詳細資訊，請參閱 AWS Glue 定價頁面。

您可以選取清單中的任何任務執行，並檢視其他資訊。選擇任務執行，然後進行下列任一項目：

- 選擇動作選單以及檢視任務選項，以在視覺化編輯器中檢視任務。
- 選擇動作選單以及停止執行選項，停止任務的目前執行。
- 選擇檢視 CloudWatch 記錄檔按鈕，檢視該工作的工作執行日誌。
- 選擇檢視詳細資料以檢視任務執行詳細資料頁面。

檢視任務執行日誌

您可以用多種方式來檢視任務日誌：

- 在 [監視] 頁面的 [Job 執行] 表格中，選擇工作執行，然後選擇 [檢視 CloudWatch 記錄]。
- 在視覺化任務編輯器中，在執行索引標籤中，選擇要檢視日誌的超連結：
  - 日誌 – 啟用任務執行的連續記錄時，所寫入 Apache Spark 任務日誌的連結。當您選擇此連結時，會將您帶到記 Amazon CloudWatch 錄群組中的記/aws-glue/jobs/logs-v2錄檔。預設情況下，日誌會排除無用的 Apache Hadoop YARN 活動訊號和 Apache Spark 驅動程式或執行器日誌訊息。如需持續記錄的詳細資訊，請參閱 AWS Glue 開發人員指南中的持續記錄 AWS Glue 任務。
  - 錯誤日誌 – 連結至此次任務執行時寫入 `stderr` 的日誌。當您選擇此連結時，它會帶您前往 `/aws-glue/jobs/error` 日誌群組中的 Amazon CloudWatch 日誌。您可以使用這些日誌來檢視任務執行期間所發生任何錯誤的詳細資訊。
  - 輸出日誌 – 連結至此次任務執行時所寫入 `stdout` 的日誌。當您選擇此連結時，它會帶您前往 `/aws-glue/jobs/output` 日誌群組中的 Amazon CloudWatch 日誌。您可以使用這些日誌來查看在 AWS Glue Data Catalog 中建立的資料表的所有詳細資訊，以及所發生的任何錯誤。

檢視任務執行的詳細資訊

您可以在監控頁面的任務執行清單選擇任務，然後選擇檢視執行詳細資訊以查看該任務執行的詳細資訊。

任務執行詳細資訊頁面上顯示的資訊包括：

屬性	描述
任務名稱	任務的名稱。
執行狀態	任務執行目前的狀態。值可以為： • STARTING • RUNNING • STOPPING • STOPPED • SUCCEEDED • FAILED • TIMEOUT
Glue 版本	任務執行所使用的 AWS Glue 版本。
最近嘗試	此任務執行的自動重試嘗試次數。
開始時間	此次任務執行開始的日期和時間。
結束時間	此次任務執行完成的日期和時間。
開始時間	準備執行任務所花的時間。
執行時間	執行任務指令碼所花的時間。
觸發條件名稱	與任務相關聯的觸發名稱。
上次修改時間	上次修改任務的日期。
安全組態	任務的安全組態，包括 Amazon S3 加密、加 CloudWatch 密和任務書籤加密設定。

屬性	描述
逾時	任務執行逾時閾值。
已配置容量	配置給此任務執行的 AWS Glue 資料處理單位 (DPU) 數目。如需容量規劃的詳細資訊，請參閱 AWS Glue 開發人員指南中的 DPU 容量規劃監控。
最大容量	任務執行的可用容量上限。
工作者數目	用於任務執行的工作者數量。
工作者類型	配置給任務執行的預先定義工作者類型。值可以為 G.1X 或 G.2X。 • **G.1X** – 選擇這種類型時，您也要提供 工作者數目 的值。每個工作者都會映射到 1 個 DPU (4 個 vCPU、16 GB 記憶體、64 GB 磁碟)，並為每個工作者提供 1 個執行器。我們建議記憶體密集型任務採用這種工作者類型。此為 AWS Glue 2.0 版本或更新版本任務的預設工作者類型。 • **G.2X** – 選擇這種類型時，您也要提供 工作者數目 的值。每個工作者都會映射到 2 個 DPU (8 個 vCPU、32 GB 記憶體、128 GB 磁碟)，並為每個工作者提供 1 個執行器。我們建議記憶體密集型任務和執行機器學習轉換的任務採用這種工作者類型。
日誌	連續記錄之任務日誌的連結 (/aws-glue/jobs/logs-v2)。
輸出日誌	任務輸出日誌檔案的連結 (/aws-glue/jobs/output)。
錯誤日誌	任務錯誤日誌檔案的連結 (/aws-glue/jobs/error)。

也可以檢視下列其他項目，可在檢視最近工作執行的資訊時使用它們。如需詳細資訊，請參閱 the section called “檢視最近任務執行的資訊”。

- 輸入引數
- 連續日誌
- 指標：您可以查看基本指標的視覺效果。如需有關包含之指標的詳細資訊，請參閱 the section called “Amazon CloudWatch 檢視 Spark 工作執行的指標”。
- Spark UI – 您可以在 Spark UI 中將工作的 Spark 日誌視覺化。如需有關使用 Spark Web UI 的詳細資訊，請參閱 the section called “使用 Spark UI 進行監控”。必須依照 the section called “為任務啟用 Spark UI” 中的程序來啟用此功能。

Amazon CloudWatch 檢視 Spark 工作執行的指標

您可以在工作執行的詳細資訊頁面的「執行詳細資訊」段落下方，檢視工作測量結果。 AWS Glue Studio會 Amazon CloudWatch 針對每個工作執行傳送工作指標至。

AWS Glue Amazon CloudWatch 每 30 秒報告一次指標。 AWS Glue 指標表示先前回報數值中的差異值。適當時，指標儀表板將會彙總 (加總) 30 秒的值，以取得最後完整一分鐘的值。不過， AWS Glue 傳遞至的 Apache Spark 量度通常 Amazon CloudWatch 是絕對值，代表報告時的目前狀態。

> **Note**
>
> 您必須設定您的帳戶才能存取 Amazon CloudWatch、。

指標提供任務執行的相關資訊，例如：

- ETL 資料移動 – 讀取或寫入 Amazon S3 的位元組數。
- 記憶體設定檔：使用的堆積 – Java 虛擬機器 (JVM) 堆積所使用的記憶體位元組數目。
- 記憶體設定檔：堆積使用率 – JVM 堆積使用的記憶體分數 (比例：0–1)，以百分比顯示。
- CPU 負載 – 使用的 CPU 系統負載分數 (比例：0–1)，以百分比顯示。

檢視 Ray 工作執行的 Amazon CloudWatch 測量結果

您可以在工作執行的詳細資訊頁面的「執行詳細資訊」段落下方，檢視工作測量結果。 AWS Glue Studio會 Amazon CloudWatch 針對每個工作執行傳送工作指標至。

AWS Glue Amazon CloudWatch 每 30 秒報告一次指標。 AWS Glue 指標表示先前回報數值中的差異值。適當時，指標儀表板將會彙總 (加總) 30 秒的值，以取得最後完整一分鐘的值。不過， AWS Glue 傳遞至的 Apache Spark 量度通常 Amazon CloudWatch 是絕對值，代表報告時的目前狀態。

> **Note**
>
> 您必須設定您的帳戶才能存取 Amazon CloudWatch，如中所述。

在 Ray 任務中，您可以檢視下列彙總指標圖表。您可以使用這些資料建立叢集和任務的設定檔，並可存取有關每個節點的詳細資訊。支援這些圖形的時間序列資料可用 CloudWatch 於進一步分析。

任務設定檔：任務狀態

顯示系統中的 Ray 任務數量。每個任務生命週期都有自己的時間序列。

任務設定檔：任務名稱

顯示系統中的 Ray 任務數量。只會顯示擱置中和作用中的任務。每種類型的任務 (依名稱) 都有自己的時間序列。

叢集設定檔：使用中的 CPU

顯示使用的 CPU 核心數量。每個節點都有自己的時間序列。節點由 IP 地址識別，這些 IP 地址是暫時的，僅用於識別。

叢集設定檔：物件存放區記憶體使用量

顯示 Ray 物件快取的記憶體使用量。每個記憶體位置 (實體記憶體、磁碟上的快取，以及 Amazon S3 中的溢出) 都有自己的時間序列。物件存放區管理叢集中所有節點的資料儲存。如需詳細資訊，請參閱 Ray 文件中的物件。

叢集設定檔：節點計數

顯示為叢集佈建的節點數量。

節點詳細資訊：CPU 使用量

以百分比顯示每個節點的 CPU 使用率。每個系列都會顯示節點上所有核心的 CPU 使用率彙總百分比。

節點詳細資訊：記憶體使用量

顯示每個節點的記憶體使用量 (GB)。每個系列都會顯示節點上所有程序之間的彙總記憶體，包括 Ray 任務和 Plasma 存放程序。這不會反映存放到磁碟或溢出到 Amazon S3 的物件。

節點詳細資訊：磁碟使用量

顯示每個節點上的磁碟使用量 (GB)。

節點詳細資訊：磁碟 I/O 速度

顯示每個節點上的磁碟 I/O (KB/s)。

節點詳細資訊：網路 I/O 輸送量

顯示每個節點上的網路 I/O (KB/s)。

節點詳細資訊：Ray 元件的 CPU 使用量

以核心分數的方式顯示 CPU 使用量。每個節點上的每個 Ray 元件都有自己的時間序列。

節點詳細資訊：Ray 元件的記憶體使用量

顯示記憶體使用量 (GiB)。每個節點上的每個 Ray 元件都有自己的時間序列。

偵測和處理敏感資料

Detect PII 轉換可識別資料來源中的個人身分識別資訊 (PII)。您可以選擇要識別的 PII 實體、掃描資料的方式，以及如何處理由 Detect PII 轉換識別的 PII 實體。

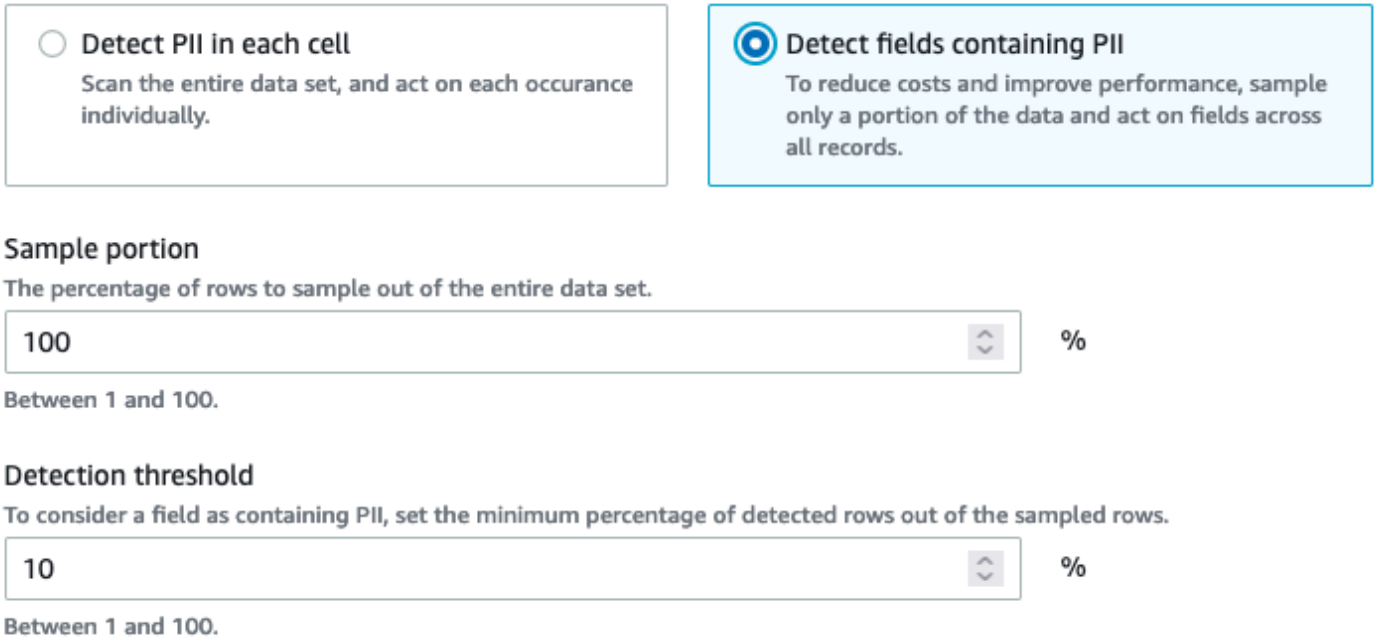
Detect PII 轉換提供偵測、遮罩或移除實體的功能，這些實體為您定義或由 AWS預先定義的實體。這樣，您就能夠提高合規性並減少責任。例如，您可能想要確保您的資料中沒有任何可讀取的個人身分資訊，並希望使用固定字串 (例如 xxx-xx-xxxx)、電話號碼或地址遮罩社會安全號碼。

若要處理 AWS Glue Studio 以外的敏感資料，請參閱 在 AWS Glue Studio 外部使用敏感數據檢測

主題

選擇掃描資料的方式

當您掃描資料集尋找敏感資料 (例如，個人身分識別資訊 (PII)) 時，可以選擇偵測每個資料列中的 PII，或偵測包含 PII 資料的資料欄。



當您選擇 Detect PII in each cell (偵測每個儲存格中的 PII) 時，就表示掃描資料來源中的所有資料列。這是一項完整的掃描，可確保識別 PII 實體。

當您選擇 Detect fields containing PII (偵測包含 PII 的欄位) 時，則表示掃描資料列範例以取得 PII 實體。這是一種降低成本和資源的方法，同時可識別找到 PII 實體的欄位。

當您選擇偵測包含 PII 的欄位時，可以透過取樣部分資料列降低成本並改善效能。選擇此選項可讓您指定其他選項：

- Sample portion (取樣部分)：這可讓您指定要取樣的列百分比。例如，如果您輸入 '50'，則表示您要為 PII 實體指定 50% 的掃描列。
- Detection threshold (偵測閾值)：這可讓您指定包含 PII 實體的列百分比，以便將整個資料行識別為具有 PII 實體。例如，如果您輸入 '10'，則指定掃描資料列中 PII 實體 (美國電話) 的數字必須占 10% 或更大，才能將欄位識別為具有 PII 實體 (美國電話)。如果包含 PII 實體的資料列百分比小於 10%，則該欄位將不會識別為具有 PII 實體 (美國電話)。

選擇要偵測的 PII 實體

若您選擇 Detect PII in each cell (偵測各儲存格中的 PII)，則您可從以下三個選項中選擇一個：

- 所有可用的 PII 模式-這包括 AWS 實體。
- 選取類別 – 在您選取類別時，PII 模式將自動包含您選取類別中的模式。

- 選取特定模式 - 僅會偵測您選取的模式。

如需受管敏感資料類型的完整清單，請參閱 Managed data types。

從所有可用的 PII 模式中選擇

如果您選擇「所有可用的 PII 樣式」，請選取預先定義的圖元。 AWS您可以選取一個、多個或所有實體。

Select entities to detect

Available entities (19) | Select all | Clear all | Create new | Manage

Find entities | All categories | 1

Entity name	Category
Person's name	Universal, HIPAA
Email (General)	Universal
Credit Card	Universal
IP Address	Networking
MAC Address	Networking
US Phone	United States, HIPAA
US Passport	United States
Social Security Number (SSN)	United States, HIPAA
US Individual Taxpayer Identification Number (ITIN)	United States, HIPAA
US/Canada bank account	United States, HIPAA
US driving license	HIPAA
Healthcare Common Procedure Coding System (HCPCS) code	HIPAA
National Drug Code (NDC)	HIPAA
National Provider Identifier (NPI)	HIPAA
Drug Enforcement Agency (DEA) Registration Number	HIPAA
Health Insurance Claim Number (HICN)	HIPAA
Medicare Beneficiary Identifier	HIPAA

選取類別

若您選擇 Select categories (選取類別) 作為要偵測的 PII 模式，則您可從下拉式選單中的選項中選取。請注意，部分實體可隸屬於多個類別。例如：Person's name (人員姓名) 是一個同時隸屬於 Universal (通用) 和 HIPAA 類別的實體。

- 通用 (範例：電子郵件、信用卡)
- HIPAA (例如：美國駕駛執照、醫療通用程序編碼系統 (HCPCS) 代碼)
- 聯網 (範例：IP 地址、MAC 地址)
- 阿根廷
- 澳洲
- 奧地利
- 比利時
- 波士尼亞
- 保加利亞
- 加拿大
- 智利
- 哥倫比亞
- 克羅埃西亞
- 賽普勒斯
- 捷克
- 丹麥
- 愛沙尼亞
- 芬蘭
- 法國
- 德國
- 希臘
- 匈牙利
- 愛爾蘭
- 韓國
- 日本
- 墨西哥

- 荷蘭
- 紐西蘭
- 挪威
- 葡萄牙
- 羅馬尼亞
- 新加坡
- 斯洛伐克
- 斯洛維尼亞
- 西班牙
- 瑞典
- 瑞士
- 土耳其
- 烏克蘭
- 美國
- 英國
- 委內瑞拉

選取特定模式

若您選擇 Select specific patterns (選取特定模式) 作為要偵測的 PII 模式，您可從已建立的模式清單中搜尋或瀏覽，或者建立新的偵測實體模式。

以下步驟說明了如何建立新的自訂模式來偵測敏感資料。您將透過輸入自訂模式的名稱來建立自訂模式、新增規則運算式，或者定義內容文字。

1. 若要建立新模式，請按一下 Create new (建立新模式) 按鈕。

2. 在 Create detection entity (建立偵測實體) 頁面中，請輸入實體名稱和常規表達式。常規表達式 (Regex) 是 AWS Glue 將用來配對實體的方式。

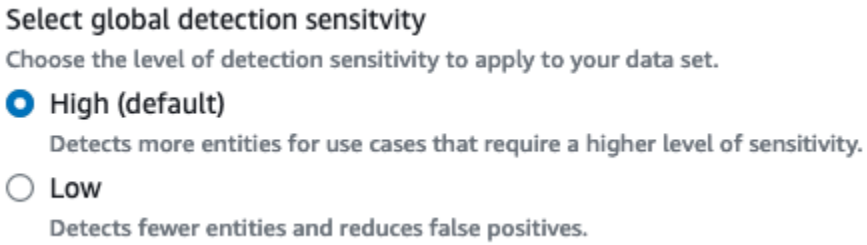
3. 按一下 Validate (驗證)。若驗證成功，您將會看到一則確認訊息，說明該字串為有效的常規表達式。若驗證未成功，您將會看到一則訊息，說明該字串不符合正確的格式和可接受的字元常值、運算子或建構。
4. 除了常規表達式之外，您可以選擇新增內容文字。內容文字可能會提高相符的可能性。若欄位名稱並非實體的描述，這些功能便十分實用。例如，社會安全號碼可被命名為 "SSN" 或 "SS"。新增這些內容文字有助於配對實體。
5. 按一下 Create (建立) 以建立偵測實體。任何已建立的實體皆在 AWS Glue Studio 主控台中可見。按一下左側導覽選單中的 Detection entities (偵測實體)。

   您可以從 Detection entities (偵測實體) 頁面中編輯、刪除或建立偵測實體。您也可以使用搜尋欄位來搜尋模式。

指定偵測敏感度等級

您可以設定使用偵測敏感資料時的敏感度等級。

- 高：(預設) 針對需要更高敏感度等級的使用案例偵測更多實體。所有在 2023 年 11 月之後建立的 AWS Glue 任務都會自動選擇加入此設定。
- 低：偵測較少的實體並減少誤報。

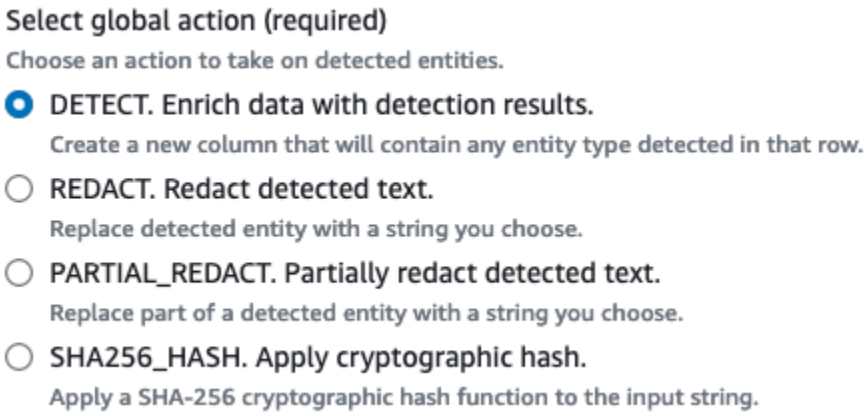


選擇如何處理已識別的 PII 資料

如果您選擇在整個資料來源中偵測 PII，則可選取要套用的全域動作：

- Enrich data with detection results (利用偵測結果豐富資料)：如果您在每個儲存格中選擇「偵測 PII」，則可以將偵測到的實體存放到新的資料行中。
- Redact detected text (將偵測到的文字設為密文)：您可以使用在選擇性的取代文字輸入欄位中指定的字串來取代偵測到的 PII 值。如果未指定任何字串，則偵測到的 PII 實體會以 '*******' 取代。

- 部分遮蔽偵測到的文字：您可以使用選擇的字串取代部分偵測到的 PII 值。其中提供兩個可能的選項：保持結尾未遮罩，或透過明確的 regex 模式進行遮罩。此功能尚無法在 AWS Glue 2.0 中使用。
- Apply cryptographic hash (套用加密雜湊)：您可以將偵測到的 PII 值傳遞給 SHA-256 密碼編譯雜湊函數，並以函數的輸出取代該值。



AWS Glue 2.0 和 3.0 以上版本之間的差異

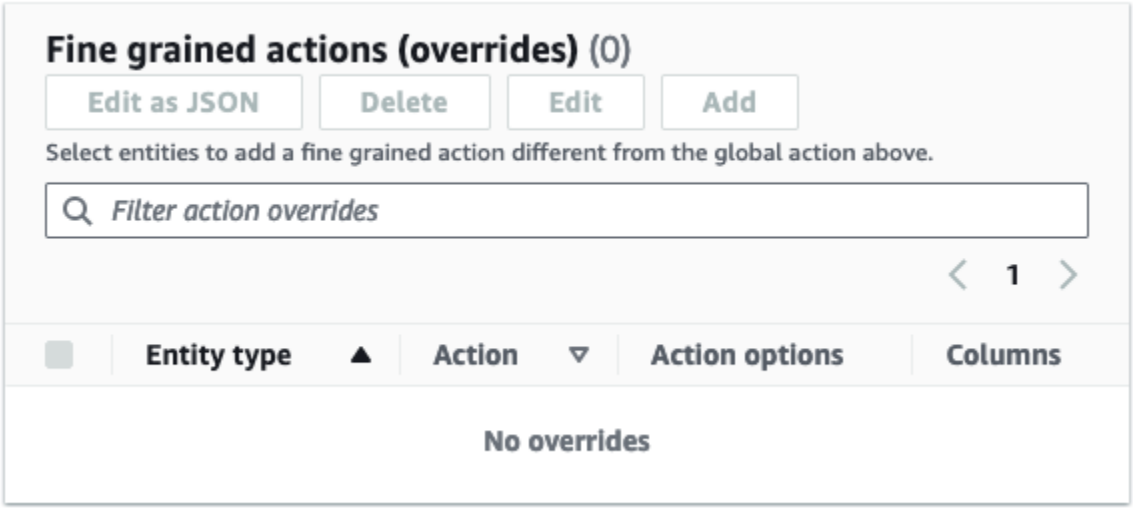
AWS Glue2.0 個工作將在補充欄中傳回一個新的工作，其中 DataFrame 包含偵測到的 PII 資訊。任何遮蔽或雜湊工作皆會顯示於視覺化索引標籤中的 AWS Glue 指令碼內。

AWS Glue3.0 和 4.0 工作將返回一個 DataFrame 具有相同補充列的新作業。"actionUsed" 的新金鑰隨即顯示，可能為 `DETECT`、`REDACT`、`PARTIAL_REDACT` 或 `SHA256_HASH` 其中一個。如果選取遮罩動作， DataFrame 將會傳回遮罩敏感資料的資料。

新增微調動作覆寫

您可以將其他偵測和動作設定新增至微調動作覆寫資料表。這可讓您：

- 包含或從偵測中排除特定資料欄：資料來源上的推論結構描述將會在資料表中填入可用的資料欄。
- 指定深入微調的特定設定，而非使用全域動作：例如，您可以為不同的實體類型指定不同的遮蔽文字設定。
- 指定全域動作以外的不同動作：如果要在不同的敏感資料類型上套用不同的動作，則可在此處進行。請注意，不能在同一列上使用兩個不同的 edit-in-place 操作（密文和哈希），但可以始終使用檢測。



使用 AWS Glue Studio 管理 ETL 任務

您可以使用 AWS Glue Studio 中的簡單圖形介面來管理您的 ETL 任務。使用導覽功能表，選擇 Jobs (任務) 來檢視 Jobs (任務) 頁面。在此頁面上，您可以查看您使用 AWS Glue Studio 或 AWS Glue 主控台建立的所有任務。您可以在此頁面上檢視、管理和執行您的任務。

您也可以在此頁面上執行下列任務：

- 開始任務執行
- 排程任務執行
- 管理任務排程
- 停止任務執行
- 檢視您的任務
- 檢視最近任務執行的資訊
- 檢視任務指令碼
- 修改任務屬性
- 儲存任務
- 複製任務
- 刪除任務

開始任務執行

在 AWS Glue Studio 中，您可以隨需執行任務。一個任務可以執行多次，每次執行該任務時，AWS Glue 都會收集有關任務活動和效能的資訊。這些資訊稱為任務執行，並由任務執行 ID 識別。

您可以透過下列方式在 AWS Glue Studio 啟動任務執行：

- 在 Jobs (任務) 頁面上，選擇您要開始的任務，然後選擇 Run job (執行任務) 按鈕。
- 如果您在視覺化編輯器中檢視任務，且任務已儲存，您可以選擇 Run (執行) 按鈕開始任務執行。

如需任務執行的詳細資訊，請參閱 AWS Glue 開發人員指南中的在 AWS Glue 主控台上使用工作。

排程任務執行

在 AWS Glue Studio 中，您可以建立排程，讓您的任務在特定時間執行。您可以指定限制條件，例如任務的執行次數、在一週中的哪一天執行，以及執行的時間。這些限制條件是根據 `cron` 並且與 `cron` 具有相同限制。例如，如果您選擇在每個月的 31 日執行您的任務，請注意有些月份不到 31 天。如需 `cron` 的詳細資訊，請參閱 AWS Glue 開發人員指南中的 Cron 表達式。

根據排程執行任務

1. 使用以下其中一個方法建立任務排程：
   - 在 Jobs (任務) 頁面上，選擇您要建立排程的任務，選擇 Actions (動作)，然後選擇 Schedule job (排程任務)。
   - 如果您在視覺化編輯器中檢視任務，且任務已儲存，請選擇 Schedules (排程) 索引標籤。然後選擇 Create Schedule (建立排程)。
2. 在 Schedule job run (排程任務執行) 頁面上，輸入下列資訊：
   - Name (名稱)：輸入任務排程的名稱。
   - Frequency (頻率)：輸入任務排程的頻率。您可以選擇下列選項：
     - Hourly (每小時)：任務將每小時執行一次，從特定分鐘開始。您可以指定任務應執行的小時 Minute (分鐘) 數。依預設，如果您選擇每小時，任務會在小時開始執行 (分鐘 0)。
     - Daily (每日)：任務將每天執行，從一個時間開始。您可以指定任務應執行的小時 Minute (分鐘) 數以及任務的 Start hour (起始小時)。小時數使用 23 小時制指定，您可以使用數字 13 到 23 表示下午的時間。分鐘和小時的預設值為 0，也就是說，如果您選取 Daily (每日)，則任務預設在午夜執行。

- Weekly (每週)：任務將在每週的一天或多天執行。除了上述與「每日」相同的設定之外，您還可以選擇在一週的哪幾天執行任務。您可以選擇一或多個天。
- Monthly (每月)：任務將在每個月的特定日期執行。除了上述與「每日」相同的設定之外，您還可以選擇在一個月的哪一天執行任務。將日指定為 1 到 31 之間的數值。如果您選取了一個月中不存在的日期，例如二月 30 日，那麼該月不會執行任務。
- Custom (自訂)：使用 `cron` 語法為您的任務排程輸入表達式。Cron 表達式允許您建立更複雜的排程，例如每月的最後一天 (而不是每月的特定日期) 或是每三個月的第 7 天和第 21 天。

  請參閱 AWS Glue 開發人員指南中的 Cron 表達式

- Description (描述)：您可以選擇性地輸入您的任務排程的說明。如果您計劃針對多個任務使用相同的排程，具有描述可讓您更容易判斷任務排程的作用。

3. 選擇 Create schedule (建立排程) 以儲存任務排程。
4. 建立排程後，成功訊息會出現在主控台頁面頂端。您可以選擇此橫幅中的 Job Details (任務詳細資訊) 以檢視任務詳細資訊。這會開啟視覺化任務編輯器頁面，其中選取了 Schedules (排程) 索引標籤。

管理任務排程

建立任務的排程後，您可以在視覺化編輯器中開啟任務，然後選擇 Schedules (排程) 索引標籤來管理排程。

在視覺化編輯器的 Schedules (排程) 索引標籤，您可以執行以下任務：

- 建立新排程。

  選擇 Create schedule (建立排程)，然後輸入排程的資訊，如the section called “排程任務執行”所述。

- 編輯現有排程。

  選擇您要編輯的排程，然後選擇 Action (動作)，接著選擇 Edit schedule (編輯排程)。當您選擇編輯現有的排程時，Frequency (頻率) 顯示為 Custom (自訂)，且排程會顯示為 `cron` 表達式。您可以修改 `cron` 表達式，或使用 Frequency (頻率) 按鈕指定新排程。當您完成變更後，請選擇 Update schedule (更新排程)。

- 暫停作用中的排程。

  選擇作用中排程，然後選擇 Actions (動作)，接著選擇 Pause schedule (暫停排程)。排程會立即停用。選擇重新整理 (重新載入) 按鈕，以查看更新的任務排程狀態。

- 繼續暫停的排程。

  選擇停用的排程，然後選擇 Actions (動作)，接著選擇 Resume schedule (繼續排程)。排程會立即啟動。選擇重新整理 (重新載入) 按鈕，以查看更新的任務排程狀態。

- 刪除排程。

  選擇您要移除的排程，然後選擇 Actions (動作)，接著選擇 Delete schedule (刪除排程)。排程會立即刪除。選擇重新整理 (重新載入) 按鈕，以查看更新的任務排程清單。排程會顯示 Deleting (正在刪除) 狀態，直到它被完全移除。

停止任務執行

您可以在任務完成任務執行之前停止任務。如果您知道任務設定不正確，或任務花太長時間無法完成，您可以選擇此選項。

在 Monitoring (監控) 頁面的 Job runs (任務執行) 清單中，選擇您要停止的任務，選擇 Actions (動作)，接著選擇 Stop run (停止執行)。

檢視您的任務

您可以在 Jobs (任務) 頁面上檢視您所有的任務。若要存取此頁面，請選擇導覽窗格中的 Jobs (任務)。

在 Jobs (任務) 頁面上，您可以看到在帳戶中建立的所有任務。Your jobs (您的任務) 清單會顯示任務名稱、其類型、該任務上次執行的狀態，以及建立任務和上次修改的日期。您可以選擇任務的名稱，以查看該任務的詳細資訊。

您也可以使用監控儀表板來檢視您的所有任務。若要存取儀表板，請選擇導覽窗格中的 Monitoring (監控)。

自訂任務顯示

您可以在 Jobs (任務) 頁面的 Your jobs (您的任務) 區段自訂任務的顯示方式。此外，您可以在搜尋文字欄位中輸入文字，以僅顯示名稱包含該文字的任務。

如果您選擇 Your jobs (您的任務) 區段中的設定圖示

，

您可以自訂 AWS Glue Studio 在資料表中顯示資訊的方式。您可以選擇在顯示中將文字行換行、變更頁面上顯示的任務數目，以及指定要顯示的欄。

檢視最近任務執行的資訊

任務可以隨著來源位置新增資料而執行多次。每次任務執行，都會為任務執行指派一個唯一的 ID，並收集有關該任務執行的資訊。您可以使用下列方法，檢視此資訊：

- 選擇視覺化編輯器的 Runs (執行) 索引標籤，以檢視目前顯示任務的任務執行資訊。

  在 Runs (執行) 索引標籤 (Recent job runs (最近任務執行) 頁面)，每個任務執行都會有一張卡片。Runs (執行) 索引標籤顯示的資訊包含：

  - 任務執行 ID
  - 嘗試執行此任務的次數
  - 任務執行的狀態
  - 任務執行的開始及結束時間
  - 任務執行的執行時間
  - 任務日誌檔的連結
  - 任務錯誤日誌檔的連結
  - 失敗任務傳回的錯誤
- 您可以選取工作執行，以檢視有關該工作的其他資訊，包括以下內容：
  - 輸入引數
  - 連續日誌
  - 指標：您可以查看基本指標的視覺效果。如需有關包含之指標的詳細資訊，請參閱 the section called “Amazon CloudWatch 檢視 Spark 工作執行的指標”。
  - Spark UI – 您可以在 Spark UI 中將工作的 Spark 日誌視覺化。如需有關使用 Spark Web UI 的詳細資訊，請參閱 the section called “使用 Spark UI 進行監控”。必須依照 the section called “為任務啟用 Spark UI” 中的程序來啟用此功能。

您可以選取檢視詳細資料，以在任務執行詳細資料頁面中檢視類似的資訊。或者，您也可以透過監控頁面，導覽至任務執行詳細資料頁面。在導覽窗格中，選擇 Monitoring (監控)。向下捲動到 Job runs (任務執行) 清單。選擇任務，然後選擇 View run details (檢視執行詳細資訊)。內容在檢視任務執行的詳細資訊中描述。

如需任務日誌的詳細資訊，請參閱檢視任務執行日誌。

檢視任務指令碼

在您提供任務中所有節點的資訊之後，AWS Glue Studio 會產生由任務用於從來源讀取資料、轉換資料，並在目標位置寫入資料的指令碼。如果您儲存任務，您可以隨時檢視此指令碼。

檢視為任務產生的指令碼

1. 在導覽窗格中，選擇 Jobs (任務)。
2. 在 Jobs (任務) 頁面的 Your jobs (您的任務) 清單中，選擇要檢閱的任務名稱。或者，您可以在清單中選擇一個任務，選擇 Actions (動作) 功能表，然後選擇 Edit job (編輯任務)。
3. 在視覺化編輯器頁面上，選擇頂端的 Script (指令碼) 索引標籤以檢視任務指令碼。

   如果想要編輯任務指令碼，請參閱AWS Glue 編程指南。

修改任務屬性

任務圖表中的節點定義了任務所執行的動作，但您也可以為任務設定幾個屬性。這些屬性會決定執行任務的環境、其使用的資源、閾值設定、安全性設定等。

自訂任務執行環境

1. 在導覽窗格中，選擇 Jobs (任務)。
2. 在 Jobs (任務) 頁面的 Your jobs (您的任務) 清單中，選擇要檢閱的任務名稱。
3. 在視覺化編輯器頁面上，選擇任務編輯窗格頂端的 Job Details (任務詳細資訊)。
4. 視需要修改任務屬性。

   如需任務屬性的詳細資訊，請參閱 AWS Glue 開發人員指南中的定義任務屬性。
5. 如果您需要指定下列額外任務屬性，請展開 Advanced properties (進階屬性) 區段：

   - Script filename (指令碼檔案名稱) – 在 Amazon S3 中儲存任務指令碼的檔案名稱。
   - Script path (指令碼路徑) – 儲存任務指令碼的 Amazon S3 位置。
   - Job metrics (任務指標) – (不適用於 Python Shell 任務) 此任務執行時開啟建立 Amazon CloudWatch 指標。
   - Continuous logging (連續記錄) – (不適用於 Python Shell 任務) 開啟連續記錄至 CloudWatch，以便在任務完成之前可以檢視記錄。
   - Spark UI 和 Spark UI logs path (Spark UI 日誌路徑) – (不適用於 Python Shell 任務) 開啟使用 Spark UI 來監控此任務，並指定 Spark UI 日誌的位置。

- Maximum concurrency (最大並行數量) – 設定此任務允許並行執行的最大數量。
- Temporary path (暫存路徑) – 提供 Amazon S3 裡的工作目錄位置，可在 AWS Glue 執行任務指令碼時寫入暫時的中繼結果。
- Delay notification threshold (minutes) (延遲通知閾值 (分鐘)) – 為任務指定延遲閾值。如果任務執行的時間超過閾值所指定的時間，AWS Glue 會將任務的延遲通知傳送至 CloudWatch。
- Security configuration (安全組態) 和 Server-side encryption (伺服器端加密) – 使用這些欄位來選擇任務的加密選項。
- Use Glue Data Catalog as the Hive metastore (使用 Glue Data Catalog 做為 Hive 中繼存放區) – 如果您想要使用 AWS Glue Data Catalog 作為 Apache Hive 中繼存放區的替代選項，請選擇此選項。
- Additional network connection (額外的網路連線) – 對於 VPC 中的資料來源，您可以指定 `Network` 連線類型，以確保您的任務透過 VPC 存取您的資料。
- Python library path (Python 程式庫路徑)、Dependent jars path (相依 jar 路徑) (不適用於 Python Shell 任務)，或 Referenced files path (參考檔案路徑) – 使用這些欄位來指定任務執行指令碼時所使用的其他檔案位置。
- Job Parameters (任務參數) – 您可新增一組索引鍵/值對，以具名參數的形式傳遞至任務指令碼。在 Python 中呼叫 AWS Glue API，最好以名稱明確傳遞參數。如需在任務指令碼中使用參數的詳細資訊，請參閱 AWS Glue開發人員指南中的在 AWS Glue 中傳遞和存取 Python 參數。
- Tags (標籤) – 您可以在任務中新增標籤，協助您整理和識別它們。

6. 修改任務屬性之後，請儲存任務。

在 Amazon S3 上儲存 Spark 隨機播放檔案

某些 ETL 任務需要讀取和合併來自多個分割區的資訊，例如，使用聯結轉換時。這項作業稱為隨機播放。在隨機播放期間，資料會寫入磁碟並透過網路傳輸。使用 AWS Glue 3.0 版，您可以將 Amazon S3 設定為這些檔案的儲存位置。AWS Glue 提供隨機播放管理器，可在 Amazon S3 之間寫入和讀取隨機檔案。與本機磁碟 (或針對 Amazon EC2 進行大量最佳化的 Amazon EBS) 相比，從 Amazon S3 寫入和讀取隨機檔案的速度較慢 (5%-20%)。不過，Amazon S3 提供無限儲存容量，因此在執行任務時無須擔心「`No space left on device`」錯誤。

將您的任務設定為使用 Amazon S3 進行隨機播放檔案

1. 在 Jobs (任務) 頁面的 Your jobs (您的任務) 清單中，選擇要修改的任務名稱。
2. 在視覺化編輯器頁面上，選擇任務編輯窗格頂端的 Job Details (任務詳細資訊)。

向下捲動到 Job Parameters (任務參數) 區段。

3. 指定下列索引鍵/值組。

   - `--write-shuffle-files-to-s3` — `true`

     這是 AWS Glue 中設定隨機播放管理器的主要參數，使用 Amazon S3 儲存貯體來寫入和讀取隨機資料。此參數的預設值為 `false`。

   - (選用) `--write-shuffle-spills-to-s3` — `true`

     此參數可讓您將溢出檔案卸載到 Amazon S3 儲存貯體，這為 AWS Glue 中的 Spark 任務提供額外的彈性。只有將大量資料溢出到磁碟的大型工作負載才需要它。此參數的預設值為 `false`。

   - (選用) `--conf spark.shuffle.glue.s3ShuffleBucket` — `S3://<shuffle-bucket>`

     此參數指定寫入隨機檔案時要使用的 Amazon S3 儲存貯體。如果您未設定此參數，則位置是為 Temporary path (暫存路徑) 指定之位置中的 `shuffle-data` 資料夾 (`--TempDir`)。

     > **Note**
     >
     > 確定隨機儲存貯體位於任務執行的同一個 AWS 區域。
     > 此外，隨機播放服務不會在任務執行完畢後清除檔案，因此您應該在隨機儲存貯體位置上設定 Amazon S3 儲存生命週期政策。如需詳細資訊，請參閱 Amazon S3 使用者指南中的管理儲存生命週期。

儲存任務

紅色的 Job has not been saved (尚未儲存任務) 標註會顯示在 Save (儲存) 按鈕左側，直到您儲存任務為止。

儲存任務

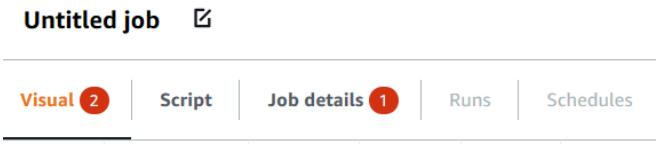
1. Visual (視覺效果) 和 Job Details (任務詳細資訊) 索引標籤中提供所有必要的資訊。
2. 選擇 Save (儲存) 按鈕。

   儲存任務後，「未儲存」標註會變更為顯示上次儲存任務的時間和日期。

如果您在儲存任務前離開 AWS Glue Studio，則下次您登入 AWS Glue Studio 時會出現通知。通知會指出有未儲存的任務，並詢問您是否要還原它。如果您選擇還原任務，您可以繼續編輯任務。

故障診斷儲存任務時發生的錯誤

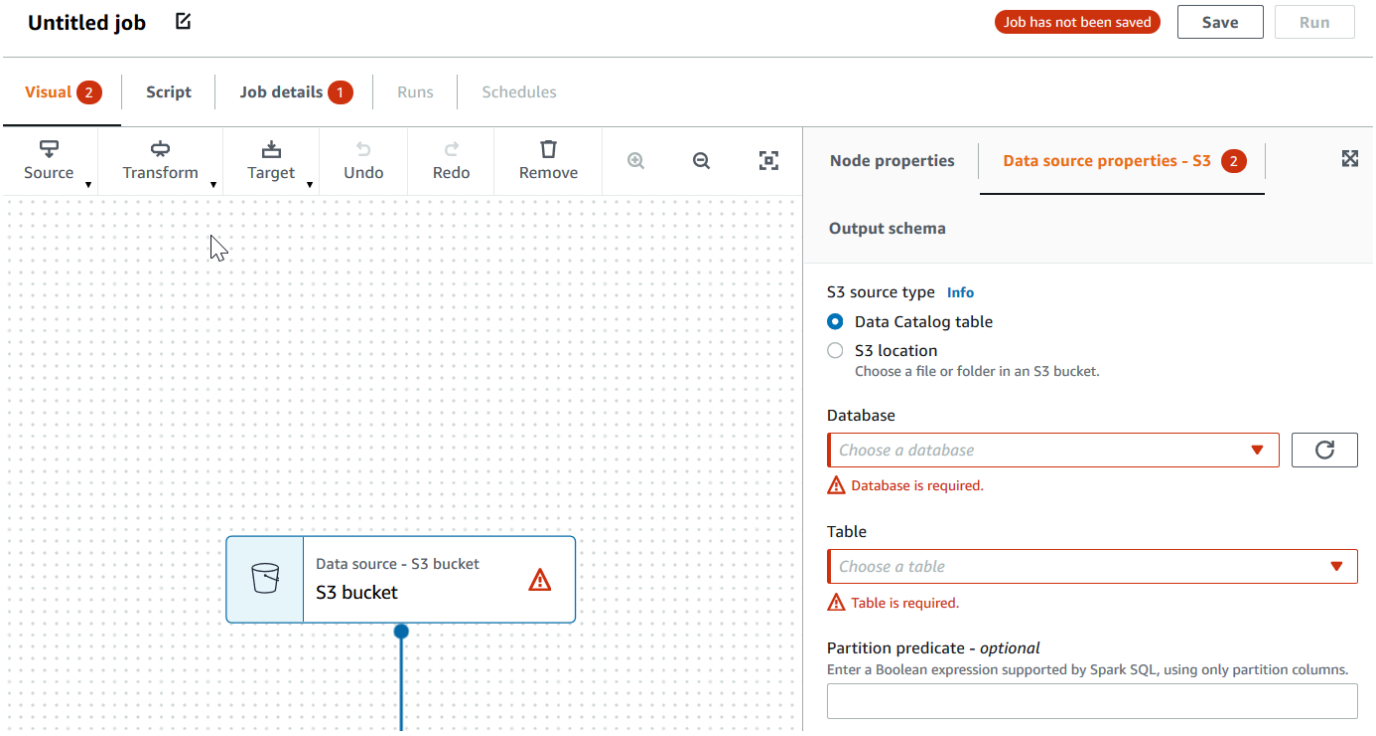
如果選擇 Save (儲存) 按鈕，但是您的任務遺失了一些必要資訊，則會在遺失資訊的索引標籤上出現紅色標註。標註中的數字表示偵測到的遺失欄位數目。



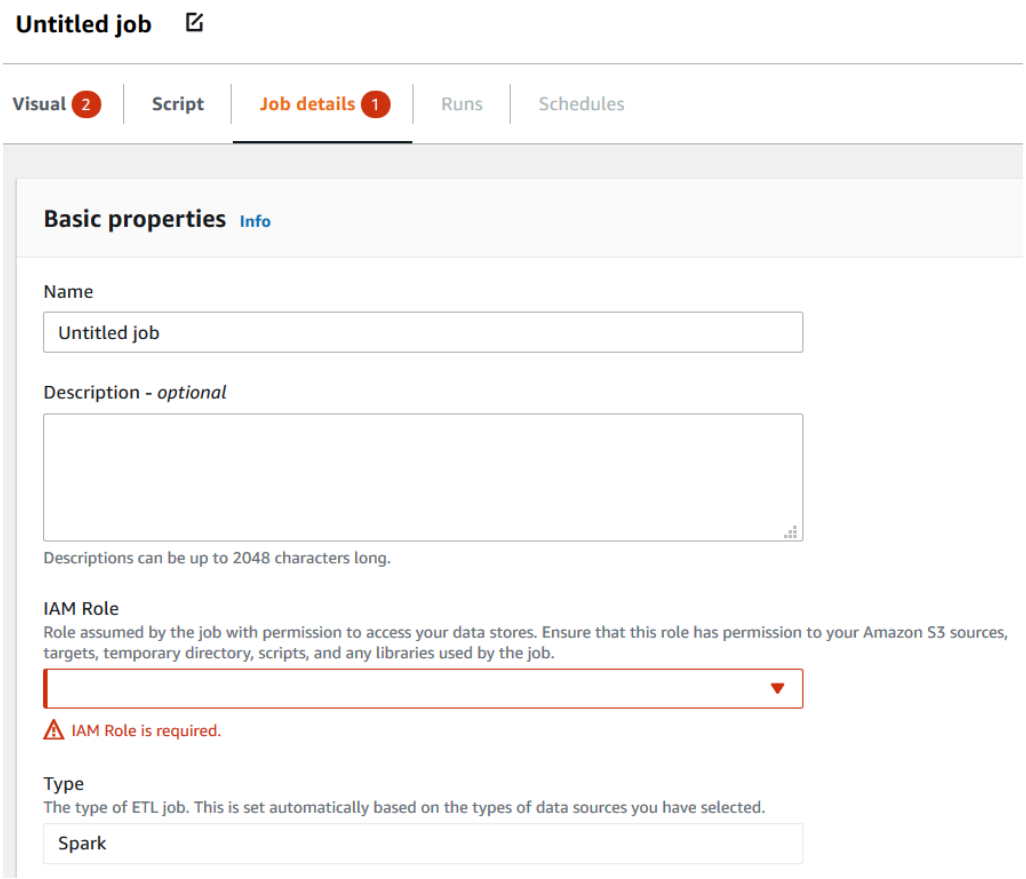
- 如果視覺化編輯器中的節點未正確設定，Visual (視覺效果) 索引標籤會顯示紅色標註，並且出現錯誤的節點顯示警告符號

  。

  1. 選擇節點。在節點詳細資訊面板中，紅色標註會出現在遺失或不正確資訊所在的索引標籤上。
  2. 在節點詳細資訊面板中選擇顯示紅色標註的索引標籤，然後找到反白顯示的問題欄位。欄位下方的錯誤訊息會提供問題的其他資訊。



- 如果任務屬性發生問題，Job Details (任務詳細資訊) 索引標籤會顯示紅色標註。選擇該標簽并确定問題欄位，這些欄位會突出顯示。欄位下方的錯誤訊息提供問題的其他資訊。



複製任務

您可以使用複製任務動作，將現有任務複製到新任務中。

透過複製現有任務建立新任務

1. 在 Jobs (任務) 頁面的 Your jobs (您的任務) 清單中，選擇要複製的任務。
2. 從 Actions (動作) 功能表，選擇 Clone job (複製任務)。
3. 輸入新任務的名稱。然後您可以儲存或編輯任務。

刪除任務

您可以移除不再需要的任務。您可以在單一作業中刪除一或多個任務。

從 AWS Glue Studio 中移除任務

1. 在 Jobs (任務) 頁面的 Your jobs (您的任務) 清單中，選擇要刪除的任務。

2. 在 Actions (動作) 選單中，選擇 Delete job (刪除任務)。
3. 確認您要刪除任務，方法是輸入**`delete`**。

您也可以在檢視視覺化編輯器中任務的 Job Details (任務詳細資訊) 索引標籤時刪除儲存的任務。

在 AWS Glue 中使用任務

以下各節提供有關 AWS Glue中 ETL 和 Ray 任務的資訊。

主題

AWS Glue 版本

您可以在新增或更新任務時設定 AWS Glue 版本參數。AWS Glue 版本決定 AWS Glue 支援的 Apache Spark 和 Python 版本。Python 版本指示針對 Spark 類型任務支援的版本。下表列出可用的 AWS Glue 版本、對應的 Spark 和 Python 版本，以及其他功能變更。

AWS Glue 版本

AWS Glue 版本	支援的執行期環境版本	支援的 Java 版本	功能變更
AWS Glue4.0	Spark 環境版本 • Spark 3.3.0 • Python 3.10	Java 8	AWS Glue 最新版本為 AWS Glue 4.0。此 AWS Glue 版本中內建了數個最佳化和升級，例如： • 許多 Spark 功能從 Spark 3.1 升級至 Spark 3.3： • 與 pandas 配對時的數個功能改進。如需詳細資

AWS Glue 版本	支援的執行期環境版本	支援的 Java 版本	功能變更
			訊，請參閱 Spark 3.3 中的新功能。 • 在 Amazon EMR 上開發的其他最佳化功能。 • 升級至 EMR 檔案系統 (EMRFS) 2.53。 • 從 Log4j 1.x 遷移至 Log4j 2 • 從 AWS Glue 3.0 開始更新了數個 Python 模組，例如 Boto 的升級版本。 • 升級數個連接器，包括預設的 Amazon Redshift 連接器。請參閱附錄 C：連接器升級。 • 升級數個 JDBC 驅動程式。請參閱附錄 B：JDBC 驅動程式升級。 • 以新的 Amazon Redshift 連接器和 JDBC 驅動程式進行更新。 • 原生支援開放式資料湖架構，包括 Apache Hudi、Delta Lake 和 Apache Iceberg。

AWS Glue 版本	支援的執行期環境版本	支援的 Java 版本	功能變更
			• 原生支援以 Amazon S3 為基礎的雲端隨機排序儲存外掛程式 (Apache Spark 外掛程式)，以使用 Amazon S3 進行隨機排序和彈性儲存容量。 限制 以下為 AWS Glue 4.0 的限制： • AWS Glue 4.0 尚未提供 AWS Glue 機器學習和個人身分識別資訊 (PII) 轉換。 如需遷移到 AWS Glue 4.0 版的詳細資訊，請參閱將 AWS Glue for Spark 任務遷移到 AWS Glue 4.0 版。

AWS Glue 版本	支援的執行期環境版本	支援的 Java 版本	功能變更
	Ray 環境版本 • Ray 2.4.0 Python 3.9	N/A	使用 Ray 建置及執行分散式 Python 應 AWS Glue 用程式。 • 透過 Python 3.9 支援 Ray-2.4.0 資料分發 (`ray[data]`)。如需此 Ray 發行版本的詳細資訊，請參閱 Ray 儲存庫中的 Ray-2.4.0。GitHub • 支援將額外的 Python 程式庫安裝至 `Ray2.4` 執行期環境。如需詳細資訊，請參閱 the section called “Ray 任務的其他 Python 模組”。 • 將 Ray 任務中的日誌和指標與 Amazon 整合 CloudWatch。如需詳細資訊，請參閱 the section called “對 Ray 錯誤進行疑難排解” 及 the section called “Ray 任務指標”。 • 在 AWS Glue Studio每個作業執行頁面上彙總和視

AWS Glue 版本	支援的執行期環境版本	支援的 Java 版本	功能變更
			覺化 Ray 工作的指標。 • 支援將檔案分發至叢集中的每個工作目錄、將物件從 Ray 物件存放區溢出至 Amazon S3，以及控制分配給 Ray 任務的工作節點數量下限。如需詳細資訊，請參閱 the section called “Ray 任務參數”。 AWS Glue 4.0 中 Ray 任務的限制 • AWS Glue Ray 的互動式工作階段會保留在此版本的預覽中。 • AWS Glue 用於與 Amazon VPC 的 Ray 集成目前不可用。如果沒有公共路由， AWS 則無法存取中的 VPC 中的資源。如需 AWS Glue 搭配 Amazon VPC 搭配使用的詳細資訊，請參閱the section called

AWS Glue 版本	支援的執行期環境版本	支援的 Java 版本	功能變更
			“VPC 端點 (AWS PrivateLink)”。 • AWS Glue in Ray 在美國東部 (維吉尼亞北部)、美國東部 (俄亥俄)、美國西部 (奧勒岡)、亞太區域 (東京) 和歐洲 (愛爾蘭) 提供。

AWS Glue 版本	支援的執行期環境版本	支援的 Java 版本	功能變更

AWS Glue3.0	• Spark 3.1.1
	• Python 3.7
		Java 8	除了 Spark 引擎升級到 3.0 之外，這個 AWS Glue 版本還內建了一些最佳化和升級，例如：
			• 根據 Spark 3.0 建置 AWS Glue ETL 程式庫，這是 Spark 的一個重要版本。
			• AWS Glue 3.0 支援串流任務。
			• 包含針對效能和可靠性的新 AWS Glue Spark 執行時間最佳化：
			  • 基於 Apache Arrow 讀取 CSV 資料的更快的記憶體直欄式處理。
			  • 使用 CSV 資料進行向量化讀取的 SIMD 基礎執行。
			  • Spark 升級還包括在 Amazon EMR 上開發的其他最佳化。
			  • 將 EMRFS 從 2.38 升級至 2.46，為 Amazon S3 存取提供新功能和錯誤修正。

AWS Glue 版本	支援的執行期環境版本	支援的 Java 版本	功能變更
			• 升級了新 Spark 版本所需的幾個相依性。請參閱附錄 A：值得注意的相依性升級。 • 針對我們原生支援的資料來源升級 JDBC 驅動程式。請參閱附錄 B：JDBC 驅動程式升級。 限制 以下為 AWS Glue 3.0 的限制： • AWS Glue 機器學習轉換尚未在 AWS Glue3.0 提供。 • 某些自訂 Spark 連接器無法用於 AWS Glue 3.0，如果它們依賴於 Spark 2.4，並且與 Spark 3.1 沒有相容性。 如需從遷移到 AWS Glue 3.0 版的詳細資訊，請參閱將 AWS Glue for Spark 任務遷移到 AWS Glue 3.0 版。

AWS Glue 版本	支援的執行期環境版本	支援的 Java 版本	功能變更
AWS Glue2.0 (已淘汰，終止支援)	• Spark 2.4.3 • Python 3.7	N/A	除了 AWS Glue 1.0 版提供的功能，AWS Glue 2.0 版也提供： • 已升級的基礎結構，可在 AWS Glue 中以縮短的啟動時間執行 Apache Spark ETL 任務。 • 預設日誌記錄現在是即時的記錄，為驅動程式和執行程序以及輸出和錯誤提供單獨的串流。 • 支援在任務層級指定其他 Python 模組或不同版本。 **Note** 由於基礎架構變化，AWS Glue 2.0 版在一些相依性和版本方面不同於 AWS Glue 1.0。請先驗證您的 AWS Glue 任務，然後再跨主要 AWS Glue 版本發行遷移。

AWS Glue 版本	支援的執行期環境版本	支援的 Java 版本	功能變更
			如需有關 AWS Glue 2.0 版功能和限制的詳細資訊，請參閱以縮短的啟動時間執行 Spark ETL 任務。

AWS Glue 版本	支援的執行期環境版本	支援的 Java 版本	功能變更
AWS Glue 1.0 (已被取代，終止支援)	• Spark 2.4.3 • Python 2.7 • Python 3.6	N/A	您可以在 AWS Glue ETL 任務 (使用 AWS Glue 1.0 版) 中維護 Parquet 和 ORC 格式的任務書籤。之前，您只能在 AWS Glue ETL 任務中將常見的 Amazon S3 來源格式加入書籤，例如 JSON、CSV、Apache Avro 和 XML。 為 ETL 輸入和輸出設定格式選項時，您可以指定使用 Apache Avro Reader/Writer 格式 1.8，以支援 Avro 邏輯類型的讀取和寫入 (使用 AWS Glue 1.0 版)。先前僅支援 1.7 版的 Avro Reader/Writer 格式。 DynamoDB 連線類型支援寫入器選項 (使用 AWS Glue 1.0 版)。 限制 以下為 AWS Glue 1.0 的限制： • AWS Glue 0.9 版和 1.0 版在亞太區

AWS Glue 版本	支援的執行期環境版本	支援的 Java 版本	功能變更
			域 (雅加達) (ap-southeast-3)、中東 (阿拉伯聯合大公國) (me-central-1) 或今後推出服務的其他新區域中不可使用。
AWS Glue 0.9 (已被取代，終止支援)	• Spark 2.2.1 • Python 2.7	N/A	建立時未指定 AWS Glue 版本的任務，預設為 AWS Glue 0.9。 限制 以下為 AWS Glue 0.9 的限制： • AWS Glue 0.9 版和 1.0 版在亞太區域 (雅加達) (ap-southeast-3)、中東 (阿拉伯聯合大公國) (me-central-1) 或今後推出服務的其他新區域中不可使用。

以縮短的啟動時間執行 Spark ETL 任務

AWS Glue 2.0 版和更高版本提供升級的基礎設施，可在 AWS Glue 中執行 Apache Spark ETL (擷取、轉換與載入) 任務，同時縮短啟動時間。藉由縮短等待時間，資料工程師可以提高生產力，並增加與 AWS Glue 的互動性。任務開始時間的差異減少，可協助您達到或超越可供分析資料的 SLA。

若要將此功能搭配 AWS Glue ETL 任務，請在建立您的任務時針對 `Glue version` 選擇 **`2.0`** 或更高版本

主題

支援的新功能

本節說明 AWS Glue 2.0 版和更高版本支援的新功能。

支援在任務層級指定其他 Python 模組

AWS Glue 2.0 版和更高版本也允許您在任務層級提供額外的 Python 模組或不同的版本。您可以使用 `--additional-python-modules` 選項與逗號分隔的 Python 模組清單來新增新模組或變更現有模組的版本。

例如，要更新或新增新的 `scikit-learn` 模組，請使用以下鍵/值：`"--additional-python-modules", "scikit-learn==0.21.3"`。

此外，在 `--additional-python-modules` 選項中，您可以指定 Python wheel 模組的 Amazon S3 路徑。例如：

```
--additional-python-modules s3://aws-glue-native-spark/tests/j4.2/ephem-3.7.7.1-cp37-cp37m-linux_x86_64.whl,s3://aws-glue-native-spark/tests/j4.2/fbprophet-0.6-py3-none-any.whl,scikit-learn==0.21.3
```

AWS Glue 使用 Python Package Installer (pip3) 來安裝其他模組。您可以傳遞由 `python-modules-installer-option` 指定的其他選項到 pip3 來安裝模組。來自 pip3 的任何不相容或限制將適用。

AWS Glue 2.0 版中已提供 Python 模組

AWS Glue 2.0 版本支援以下開箱即用的 python 模組：

- setuptools—45.2.0
- subprocess32—3.5.4
- ptvsd—4.3.2
- Pydevd—1.9.0
- PyMySQL—0.9.3

- docutils—0.15.2
- jmespath—0.9.4
- six—1.14.0
- python_dateutil—2.8.1
- urllib3—1.25.8
- botocore—1.15.4
- s3transfer—0.3.3
- boto3—1.12.4
- certifi—2019.11.28
- chardet—3.0.4
- idna—2.9
- requests—2.23.0
- pyparsing—2.4.6
- enum34—1.1.9
- pytz—2019.3
- numpy—1.18.1
- cycler—0.10.0
- kiwisolver—1.1.0
- scipy—1.4.1
- pandas—1.0.1
- pyarrow—0.16.0
- matplotlib—3.1.3
- pyhocon—0.3.54
- mpmath—1.1.0
- sympy—1.5.1
- patsy—0.5.1
- statsmodels—0.11.1
- fsspec—0.6.2
- s3fs—0.4.0

- Cython—0.29.15
- joblib—0.14.1
- pmdarima—1.5.3
- scikit-learn—0.22.1
- tbats—1.0.9

記錄行為

AWS Glue 2.0 版和更高版本支援不同的預設記錄行為。差異包括：

- 即時發生記錄。
- 驅動程式和執行程序有不同的串流。
- 對於每個驅動程式和執行程序有兩個串流：輸出串流和錯誤串流。

驅動程式和執行器串流

驅動程式串流會透過任務執行 ID 識別。執行器串流由任務 <*run id*>_<*executor task id*> 識別。例如：

- `"logStreamName": "jr_8255308b426fff1b4e09e00e0bd5612b1b4ec848d7884cebe61ed33a31789..._g-f65f617bd31d54bd94482af755b6cdf464542..."`

輸出和錯誤串流

輸出串流具有程式碼中的標準輸出 (stdout)。錯誤串流具有來自您的程式碼/程式庫的記錄訊息。

- 日誌串流：
  - 驅動程式日誌串流具有 <*jr*>，其中 <*jr*> 是任務執行 ID。
  - 執行器日誌流具有 <*jr*>_<*g*>，其中 <*g*> 是執行器的任務 ID。您可以在驅動程式錯誤日誌中查閱執行器任務 ID。

AWS Glue 2.0 版的預設日誌群組如下：

- `/aws-glue/jobs/logs/output` 用於輸出

- /aws-glue/jobs/logs/error 用於錯誤

提供安全組態時，日誌群組名稱會變更為：

- /aws-glue/jobs/<*security configuration*>-role/<*Role Name*>/output
- /aws-glue/jobs/<*security configuration*>-role/<*Role Name*>/error

在主控台上，Logs (日誌) 連結指向輸出日誌群組，而 Error (錯誤) 連結指向錯誤日誌群組。啟用連續記錄時，Logs (日誌) 連結指向連續日誌群組，而 Output (輸出) 連結指向輸出日誌群組。

記錄規則

> **Note**
>
> 連續記錄的預設日誌群組名稱為 /aws-glue/jobs/logs-v2。

在 AWS Glue 2.0 版和更高版本中，連續日誌的行為與 AWS Glue 1.0 版相同：

- 預設日誌群組：/aws-glue/jobs/logs-v2
- 驅動程式日誌串流：<*jr*>-driver
- 執行程式日誌串流：<*jr*>-<*### ID*>

  日誌群組的名稱可透過設定 --continuous-log-logGroupName 來變更

  日誌串流名稱可以透過設定 --continous-log-logStreamPrefix 來加上字首

不支援的功能

不支援以下 AWS Glue 功能：

- 開發端點
- AWS Glue 2.0 版和更高版本不執行 Apache YARN，因此不會套用 YARN 設定。
- AWS Glue 2.0 版和更高版本沒有 Hadoop 分散式檔案系統 (HDFS)
- AWS Glue 2.0 版和更高版本不使用動態配置，因此無法使用 ExecutorAllocationManager 指標
- 對於 AWS Glue 2.0 版或更高版本任務，您可以指定工作者和工作者類型的數目，但不指定 maxCapacity。

- AWS Glue 2.0 版和更高版本不支援 s3n 開箱即用。我們建議使用 s3 或 s3a。如果任務出於任何原因需要使用 s3n，您可以傳遞以下附加參數：

```
--conf spark.hadoop.fs.s3n.impl=com.amazon.ws.emr.hadoop.fs.EmrFileSystem
```

將 AWS Glue for Spark 任務遷移到 AWS Glue 3.0 版

本主題說明 AWS Glue 版本 0.9、1.0、2.0 和 3.0 之間的變更，以允許您將 Spark 應用程式和 ETL 任務遷移到 AWS Glue 3.0。

若要將此功能搭配 AWS Glue ETL 任務，請在建立您的任務時針對 `Glue version` 選擇 **`3.0`**

主題

支援的新功能

本節說明 AWS Glue 3.0 版的新功能和優點。

- 它基於 Apache Spark 3.1.1，具有來自開源 Spark 並由 AWS Glue 和 EMR 服務開發的最佳化，例如調整式查詢執行、向量化讀取器，以及最佳化的隨機和分割區合併。
- 升級 JDBC 驅動程式的所有 Glue 原生來源，包括 MySQL、Microsoft SQL Server、Oracle、PostgreSQL、MongoDB 和由 Spark 3.1.1 帶來的升級 Spark 程式庫和相依性。
- 透過升級的 EMRFS 最佳化 Amazon S3 存取，並預設啟用 Amazon S3 最佳化的輸出提交器。
- 最佳化 Data Catalog 存取與分割區索引、下推述詞、分割區清單，以及升級的 Hive 中繼存放區用戶端。
- 與 Lake Formation 整合，以控管的目錄資料表具有儲存格層級篩選和資料湖交易。

- 使用新的 Spark 執行器記憶體指標和 Spark 結構化串流指標，改善 Spark 3.1.1 的 Spark UI 體驗。
- 減少啟動延遲，可改善整體任務完成時間和互動性，類似於 AWS Glue 2.0。
- Spark 任務以 1 秒的增量計費，最短計費期間縮短 10 倍，從最短 10 分鐘到最短 1 分鐘，類似於 AWS Glue 2.0。

遷移至 AWS Glue 3.0 的動作

對於現有的任務，請將舊版 `Glue version` 變更為任務組態中的 `Glue 3.0`。

- 在主控台中，在 `Glue version` 中選擇 `Spark 3.1, Python 3 (Glue Version 3.0) or Spark 3.1, Scala 2 (Glue Version 3.0)`。
- 在 AWS Glue Studio 中，選擇 `Glue version` 中的 `Glue 3.0 - Supports spark 3.1, Scala 2, Python 3`。
- 在 API 中，選擇 `UpdateJob` API `GlueVersion` 參數中的 **`3.0`**。

對於新任務，當您建立任務時請選擇 `Glue 3.0`。

- 在主控台中，在 `Glue version` 中選擇 `Spark 3.1, Python 3 (Glue Version 3.0) or Spark 3.1, Scala 2 (Glue Version 3.0)`。
- 在 AWS Glue Studio 中，選擇 `Glue version` 中的 `Glue 3.0 - Supports spark 3.1, Scala 2, Python 3`。
- 在 API 中，選擇 `CreateJob` API `GlueVersion` 參數中的 **`3.0`**。

若要查看 AWS Glue3.0 的 Spark 事件日誌，請使用 CloudFormation 或 Docker 啟動 Glue 3.0 的升級 Spark 歷史伺服器。

遷移檢查清單

檢閱此檢查清單以進行遷移。

- 您的任務是否依賴於 HDFS ？ 如果是，請嘗試用 S3 替換 HDFS。
  - 在任務指令碼程式碼中搜尋以 `hdfs://` 或 `/` 開頭的檔案系統路徑，作為 DFS 路徑。
  - 檢查您的預設檔案系統沒有配置 HDFS。如果它是明確配置的，則需要移除 `fs.defaultFS` 組態。
  - 檢查您的任務是否包含任何 `dfs.*` 參數。如果有包含，則需要驗證停用參數是否可行。

- 您的任務是否依賴於 YARN？ 如果是，請檢查任務是否包含下列參數，以確認影響。如果有包含，則需要驗證停用參數是否可行。
  - `spark.yarn.*`

    例如：

    ```
    spark.yarn.executor.memoryOverhead
    spark.yarn.driver.memoryOverhead
    spark.yarn.scheduler.reporterThread.maxFailures
    ```

  - `yarn.*`

    例如：

    ```
    yarn.scheduler.maximum-allocation-mb
    yarn.nodemanager.resource.memory-mb
    ```

- 您的任務是否依賴於 Spark 2.2.1 或 Spark 2.4.3？ 如果是，請檢查您的任務是否使用 Spark 3.1.1 中變更的功能來驗證影響。
  - `https://spark.apache.org/docs/latest/sql-migration-guide.html#upgrading-from-spark-sql-22-to-23`

    例如 `percentile_approx` 函數，或存在現有 `SparkContext` 時 `SparkSession` 搭配 `SparkSession.builder.getOrCreate()`。
  - `https://spark.apache.org/docs/latest/sql-migration-guide.html#upgrading-from-spark-sql-23-to-24`

    例如 `array_contains` 函數，或 `CURRENT_DATE`、`CURRENT_TIMESTAMP` 函數搭配 `spark.sql.caseSensitive=true`。
- 您的任務額外的 jars 在 Glue 3.0 中衝突嗎？
  - 從 AWS Glue 0.9/1.0：現有 AWS Glue 0.9/1.0 任務中提供的任何額外 jars 可能會導致 classpath 衝突，因為 Glue 3.0 中的升級或新的相依性。您可以避免 AWS Glue 3.0 與 `--user-jars-first` AWS Glue 任務參數的 classpath 衝突或透過陰影相依性。
  - 從 AWS Glue 2.0：您仍然可以避免 AWS Glue 3.0 與 `--user-jars-first` AWS Glue 任務參數的 classpath 衝突或透過陰影相依性。
- 您的任務是否依賴於 Scala 2.11？
  - AWS Glue 3.0 使用 Scala 2.12，所以如果您的程式庫依賴於 Scala 2.11，您需要用 Scala 2.12 重建您的程式庫。

- 您任務的外部 Python 程式庫是否依賴於 Python 2.7/3.6？
  - 使用 `--additional-python-modules` 參數，而不是在 Python 程式庫路徑中設定 egg/wheel/zip 檔案。
  - 將相依程式庫從 Python 2.7/3.6 更新為 Python 3.7，因為 Spark 3.1.1 刪除了 Python 2.7 的支援。

從 AWS Glue 0.9 遷移到 AWS Glue 3.0

遷移時請注意下列變更：

- AWS Glue 0.9 使用開源 Spark 2.2.1 而 AWS Glue 3.0 使用 EMR 最佳化 Spark 3.1.1。
  - 單獨的數個 Spark 變更可能需要修改指令碼，以確保沒有被引用刪除的功能。
  - 例如，Spark 3.1.1 不啟用 Scala 無類型的 UDF，但 Spark 2.2 確實允許它們。
- AWS Glue 3.0 中的所有任務都將以顯著改善的啟動時間執行。Spark 任務將以 1 秒的增量計費，最小計費持續時間縮短 10 倍，因為啟動延遲將從最大 10 分鐘到最大 1 分鐘。
- 自 AWS Glue 2.0 以來，日誌記錄行為發生了變更。
- 數個相依性更新，在 附錄 A：值得注意的相依性升級 中反白顯示。
- Scala 也從 2.11 更新到 2.12，而 Scala 2.12 不回溯相容 Scala 2.11。
- Python 3.7 也是用於 Python 指令碼的預設版本，因為 AWS Glue 0.9 只利用 Python 2。
  - Python 2.7 不支援 Spark 3.1.1。
  - 提供安裝其他 Python 模組的新機制。
- AWS Glue 3.0 不執行 Apache YARN，因此不會套用 YARN 設定。
- AWS Glue 3.0 沒有 Hadoop 分散式檔案系統 (HDFS)。
- 在現有 AWS Glue 0.9 任務中提供的任何額外 jars 可能會帶來衝突的相依性，因為 3.0 中的幾個相依性已從 0.9 升級。您可以避免 AWS Glue 3.0 與 `--user-jars-first` AWS Glue 任務參數的 classpath 衝突。
- AWS Glue 3.0 尚未支援動態配置，因此無法使用 ExecutorAllocationManager 指標。
- 在 AWS Glue 3.0 版任務，您可以指定工作者和工作者類型的數目，但不指定 `maxCapacity`。
- AWS Glue 3.0 尚未支援機器學習轉換。
- AWS Glue 3.0 尚未支援開發端點。

請參閱 Spark 遷移文件：

- 請參閱從 Spark SQL 2.2 升級到 2.3
- 請參閱從 Spark SQL 2.3 升級到 2.4
- 請參閱從 Spark SQL 2.4 升級到 3.0
- 請參閱從 Spark SQL 3.0 升級到 3.1
- 請參閱自 Spark 3.0 以來預期日期時間行為的變化。

從 AWS Glue 1.0 遷移到 AWS Glue 3.0

遷移時請注意下列變更：

- AWS Glue 1.0 使用開源 Spark 2.4 而 AWS Glue 3.0 使用 EMR 最佳化 Spark 3.1.1。
  - 單獨的數個 Spark 變更可能需要修改指令碼，以確保沒有被引用刪除的功能。
  - 例如，Spark 3.1.1 不啟用 Scala 無類型的 UDF，但 Spark 2.4 確實允許它們。
- AWS Glue 3.0 中的所有任務都將以顯著改善的啟動時間執行。Spark 任務將以 1 秒的增量計費，最小計費持續時間縮短 10 倍，因為啟動延遲將從最大 10 分鐘到最大 1 分鐘。
- 自 AWS Glue 2.0 以來，日誌記錄行為發生了變更。
- 數個相依性更新，凸顯於
- Scala 也從 2.11 更新到 2.12，而 Scala 2.12 不回溯相容 Scala 2.11。
- Python 3.7 也是用於 Python 指令碼的預設版本，因為 AWS Glue 0.9 只利用 Python 2。
  - Python 2.7 不支援 Spark 3.1.1。
  - 提供安裝其他 Python 模組的新機制。
- AWS Glue 3.0 不執行 Apache YARN，因此不會套用 YARN 設定。
- AWS Glue 3.0 沒有 Hadoop 分散式檔案系統 (HDFS)。
- 在現有 AWS Glue 1.0 任務中提供的任何額外 jars 可能會帶來衝突的相依性，因為 3.0 中的幾個相依性已從 1.0 升級。您可以避免 AWS Glue 3.0 與 `--user-jars-first` AWS Glue 任務參數的 classpath 衝突。
- AWS Glue 3.0 尚未支援動態配置，因此無法使用 ExecutorAllocationManager 指標。
- 在 AWS Glue 3.0 版任務，您可以指定工作者和工作者類型的數目，但不指定 `maxCapacity`。
- AWS Glue 3.0 尚未支援機器學習轉換。
- AWS Glue 3.0 尚未支援開發端點。

請參閱 Spark 遷移文件：

- 請參閱從 Spark SQL 2.4 升級到 3.0
- 請參閱自 Spark 3.0 以來預期日期時間行為的變化。

從 AWS Glue 2.0 遷移到 AWS Glue 3.0

遷移時請注意下列變更：

- AWS Glue 2.0 中所有現有任務參數和主要功能都將存在於 AWS Glue 3.0。
  - 預設情況下，在 AWS Glue 3.0 中啟用將 Parquet 資料寫入 Amazon S3 的 EMRFS S3 最佳化遞交者。不過，您仍然可以透過將 `--enable-s3-parquet-optimized-committer` 設定為 `false` 來停用它。
- AWS Glue 2.0 使用開源 Spark 2.4 而 AWS Glue 3.0 使用 EMR 最佳化 Spark 3.1.1。
  - 單獨的數個 Spark 變更可能需要修改指令碼，以確保沒有被引用刪除的功能。
  - 例如，Spark 3.1.1 不啟用 Scala 無類型的 UDF，但 Spark 2.4 確實允許它們。
- AWS Glue 3.0 還提供 EMRFS 的更新、更新的 JDBC 驅動程式，以及 AWS Glue 提供的對 Spark 本身的其他最佳化。
- AWS Glue 3.0 中的所有任務都將以顯著改善的啟動時間執行。Spark 任務將以 1 秒的增量計費，最小計費持續時間縮短 10 倍，因為啟動延遲將從最大 10 分鐘到最大 1 分鐘。
- Python 2.7 不支援 Spark 3.1.1。
- 數個相依性更新，在 附錄 A：值得注意的相依性升級 中反白顯示。
- Scala 也從 2.11 更新到 2.12，而 Scala 2.12 不回溯相容 Scala 2.11。
- 在現有 AWS Glue 2.0 任務中提供的任何額外 jars 可能會帶來衝突的相依性，因為 3.0 中的幾個相依性已從 2.0 升級。您可以避免 AWS Glue 3.0 與 `--user-jars-first` AWS Glue 任務參數的 classpath 衝突。
- 與 AWS Glue 2.0 相比，AWS Glue 3.0 對於驅動程式/執行程式組態採用不同的 Spark 任務平行處理，同時提高效能並更妥善地利用可用的資源。`spark.driver.cores` 和 `spark.executor.cores` 已在 AWS Glue 3.0 中設定為核心數 (標準為 4 和 `G.1X` 工作者，以及在 `G.2X` 工作者時為 8)。這些組態不會變更 AWS Glue 任務的工作者類型或硬體。您可以使用這些組態來計算分割區或分割的數量，以匹配 Spark 應用程式中的 Spark 任務平行處理。

  一般而言，與 AWS Glue 2.0 相較，任務的效能會相似或有所提升。如果任務執行速度較慢，您可以傳遞下列任務引數來增加任務平行處理：

  - 索引鍵：`--executor-cores` 值：<############>

- 該值不應超過工作者類型上 vCPU 數量的 2 倍，即分別為 G.1X 上 8 個、G.2X 上 16 個、G.4X 上 32 個和 G.8X 上 64 個。更新此組態時應小心，這可能會影響任務效能，因為增加任務平行處理會導致記憶體和磁碟壓力，並且可能會對來源和目標系統限流。

- AWS Glue 3.0 使用 Spark 3.1，後者將行為變更為從 parquet 檔案載入時間戳記以及將時間戳記儲存至此類檔案。有關更多詳細資訊，請參閱從 Spark SQL 3.0 升級到 3.1。

  建議在讀取/寫入包含時間戳記資料行的 parquet 資料時設定以下參數。針對 AWS Glue 動態框架和 Spark 資料框架，設定這些參數可以解決從 Spark 2 升級到 Spark 3 期間發生的行事曆不相容問題。使用「更正」選項原樣讀取 datetime 值；使用「舊式」選項可以在讀取過程中針對行事曆差異重新設定 datetime 值的基準。

  ```
  - Key: --conf
  - Value: spark.sql.legacy.parquet.int96RebaseModeInRead=[CORRECTED|LEGACY] --
  conf spark.sql.legacy.parquet.int96RebaseModeInWrite=[CORRECTED|LEGACY] --conf
   spark.sql.legacy.parquet.datetimeRebaseModeInRead=[CORRECTED|LEGACY]
  ```

請參閱 Spark 遷移文件：

- 請參閱從 Spark SQL 2.4 升級到 3.0
- 請參閱自 Spark 3.0 以來預期日期時間行為的變化。

附錄 A：值得注意的相依性升級

以下是相依性升級：

相依性	AWS Glue 0.9 中的版本	AWS Glue 1.0 中的版本	AWS Glue 2.0 中的版本	AWS Glue 3.0 中的版本
Spark	2.2.1	2.4.3	2.4.3	3.1.1-amzn-0
Hadoop	2.7.3-amzn-6	2.8.5-amzn-1	2.8.5-amzn-5	3.2.1-amzn-3
Scala	2.11	2.11	2.11	2.12
Jackson	2.7.x	2.7.x	2.7.x	2.10.x
Hive	1.2	1.2	1.2	2.3.7-amzn-4

相依性	AWS Glue 0.9 中的版本	AWS Glue 1.0 中的版本	AWS Glue 2.0 中的版本	AWS Glue 3.0 中的版本
EMRFS	2.20.0	2.30.0	2.38.0	2.46.0
Json4s	3.2.x	3.5.x	3.5.x	3.6.6
Arrow	N/A	0.10.0	0.10.0	2.0.0
AWS Glue 目錄用戶端	N/A	N/A	1.10.0	3.0.0

附錄 B：JDBC 驅動程式升級

以下是 JDBC 驅動程式升級：

驅動程式	過去 AWS Glue 版本的 JDBC 驅動程式版本	AWS Glue 3.0 的 JDBC 驅動程式版本
MySQL	5.1	8.0.23
Microsoft SQL Server	6.1.0	7.0.0
Oracle 資料庫	11.2	21.1
PostgreSQL	42.1.0	42.2.18
MongoDB	2.0.0	4.0.0

將 AWS Glue for Spark 任務遷移到 AWS Glue 4.0 版

本主題說明 AWS Glue 版本 0.9、1.0、2.0 和 3.0 之間的變更，以允許您將 Spark 應用程式和 ETL 任務遷移到 AWS Glue 4.0。其中也介紹 AWS Glue 4.0 中的功能及其使用優點。

若要將此功能搭配 AWS Glue ETL 任務，請在建立您的任務時針對 `Glue version` 選擇 **`4.0`**

主題

- 支援的新功能

支援的新功能

本節說明 AWS Glue 4.0 版的新功能和優點。

- 其基於 Apache Spark 3.3.0，但包括 AWS Glue 和 Amazon EMR 中的最佳化，例如調整式查詢執行、向量化讀取器，以及最佳化的隨機和分割區合併。
- 升級 JDBC 驅動程式的所有 AWS Glue 原生來源，包括 MySQL、Microsoft SQL Server、Oracle、PostgreSQL、MongoDB 和由 Spark 3.3.0 帶來的升級 Spark 程式庫和相依性。
- 以新的 Amazon Redshift 連接器和 JDBC 驅動程式進行更新。
- 透過升級的 EMR File System (EMRFS) 最佳化 Amazon S3 存取，並預設啟用 Amazon S3 最佳化的輸出提交器。
- 最佳化 Data Catalog 存取與分割區索引、下推述詞、分割區清單，以及升級的 Hive 中繼存放區用戶端。
- 與 Lake Formation 整合，以控管的目錄資料表具有儲存格層級篩選和資料湖交易。
- 減少啟動延遲，可改善整體任務完成時間和互動性。
- Spark 任務以 1 秒的增量計費，最短計費持續時間變為原來的十分之一，即從最短 10 分鐘變為最短 1 分鐘。
- 原生支援開放式資料湖架構，包括 Apache Hudi、Delta Lake 和 Apache Iceberg。
- 原生支援以 Amazon S3 為基礎的雲端隨機排序儲存外掛程式 (Apache Spark 外掛程式)，以使用 Amazon S3 進行隨機排序和彈性儲存容量。

從 Spark 3.1.1 到 Spark 3.3.0 的主要增強功能

請注意下列增強功能：

- 資料列層級執行時間篩選 (SPARK-32268)。
- ANSI 增強功能 (SPARK-38860)。
- 錯誤訊息改善項目 (SPARK-38781)。
- 支援 Parquet 向量化讀取器的複雜類型 (SPARK-34863)。
- 隱藏 Spark SQL 的檔案中繼資料支援 (SPARK-37273)。
- 提供 Python/pandas UDF 的分析工具 (SPARK-37443)。
- 導入 Trigger.AvailableNow，以執行分成多批次執行的 Trigger.Once 等串流查詢 (SPARK-36533)。
- 更全面的 Datasource V2 下推功能 (SPARK-38788)。
- 從 log4j 1 遷移到 log4j 2 (SPARK-37814)。

其他顯著的變更

請注意下列變更：

- 重大變更
  - 在文件和 Python/文件中加入 Python 3.6 支援的參考 (SPARK-36977)。
  - 將內建 pickle 更換成 cloudpickle，以移除命名的 tuple hack (SPARK-32079)。
  - 將 pandas 從最小版本增加到 1.0.5 (SPARK-37465)。

遷移到 AWS Glue 4.0 的動作

對於現有的任務，請將舊版 `Glue version` 變更為任務組態中的 `Glue 4.0`。

- 在 AWS Glue Studio 中，選擇 `Glue version` 中的 `Glue 4.0 - Supports Spark 3.3, Scala 2, Python 3`。
- 在 API 中，選擇 `UpdateJob` API 操作 `GlueVersion` 參數中的 **`4.0`**。

對於新任務，當您建立任務時請選擇 `Glue 4.0`。

- 在主控台中，在 `Glue version` 中選擇 `Spark 3.3, Python 3 (Glue Version 4.0) or Spark 3.3, Scala 2 (Glue Version 3.0)`。

- 在 AWS Glue Studio 中，選擇 `Glue version` 中的 `Glue 4.0 - Supports Spark 3.3, Scala 2, Python 3`。
- 在 API 中，選擇 `CreateJob` API 操作 `GlueVersion` 參數中的 **`4.0`**。

若要查看來自 AWS Glue 2.0 或更早版本的 AWS Glue 4.0 Spark 事件日誌，請使用 AWS CloudFormation 或 Docker 啟動 AWS Glue 4.0 的升級 Spark 歷史伺服器。

遷移檢查清單

檢閱此檢查清單以進行遷移：

> **Note**
>
> 如需與 AWS Glue 3.0 相關的檢查清單項目，請參閱遷移檢查清單。

- 您任務的外部 Python 程式庫是否依賴於 Python 2.7/3.6？
  - 將相依程式庫從 Python 2.7/3.6 更新為 Python 3.10，因為 Spark 3.3.0 完全刪除了 Python 2.7 和 3.6 的支援。

從 AWS Glue 3.0 遷移到 AWS Glue 4.0

遷移時請注意下列變更：

- AWS Glue 3.0 中所有現有任務參數和主要功能都將存在於 AWS Glue 4.0。
- AWS Glue 3.0 使用 Amazon EMR 最佳化 Spark 3.1.1，而 AWS Glue 4.0 使用 Amazon EMR 最佳化 Spark 3.3.0。

  單獨的數個 Spark 變更可能需要修改指令碼，以確保沒有引用被刪除的功能。
- AWS Glue 4.0 也有 EMRFS 和 Hadoop 的更新。如需特定版本，請參閱附錄 A：值得注意的相依性升級。
- 在 ETL 任務中提供的 AWS SDK 現在已從 1.11 升級到 1.12。
- 所有 Python 任務都將使用 3.10 版。Python 3.7 之前用於 AWS Glue 3.0。

  因此，升級了 AWS Glue 提供的部分立即可用的 pymodules。
- Log4j 已升級到 Log4j2。

  - 如需 Log4j2 遷移路徑的資訊，請參閱 Log4j 文件。
  - 您必須以適當的 log4j2 屬性將任何自訂 log4j 屬性檔案重新命名為 log4j2.properties 檔案。
- 如需遷移某些連接器，請參閱AWS Glue 4.0 的連接器和 JDBC 驅動程式遷移。
- AWS Encryption SDK 已從 1.x 升級到 2.x。這會影響使用 AWS Glue 安全組態的 AWS Glue 任務，以及相依在執行階段中所提供 AWS Encryption SDK 相依性的任務。請參閱 AWS Glue 任務遷移的說明。

  由於 AWS Glue 2.0/3.0 已包含 AWS Encryption SDK 橋接器版本，因此您可安全地將 AWS Glue 2.0/3.0 任務升級到 AWS Glue 4.0 任務。

請參閱 Spark 遷移文件：

- 從 Spark SQL 3.1 升級到 3.2
- 從 Spark SQL 3.2 升級到 3.3

從 AWS Glue 2.0 遷移到 AWS Glue 4.0

遷移時請注意下列變更：

> **Note**
>
> 如需與 AWS Glue 3.0 相關的步驟，請參閱從 AWS Glue 3.0 遷移到 AWS Glue 4.0。

- AWS Glue 2.0 中所有現有任務參數和主要功能都將存在於 AWS Glue 4.0。
- 自 AWS Glue 3.0 起，預設啟用將 Parquet 資料寫入 Amazon S3 的 EMRFS S3 最佳化遞交者。不過，您仍然可以透過將 `--enable-s3-parquet-optimized-committer` 設定為 `false` 來停用它。
- AWS Glue 2.0 使用開源 Spark 2.4 而 AWS Glue 4.0 使用 Amazon EMR 最佳化 Spark 3.3.0。
  - 單獨的數個 Spark 變更可能需要修改指令碼，以確保沒有引用被刪除的功能。
  - 例如，Spark 3.3.0 不啟用 Scala 無類型的 UDF，但 Spark 2.4 確實允許它們。
- 在 ETL 任務中提供的 AWS SDK 現在已從 1.11 升級到 1.12。
- AWS Glue 4.0 還提供 EMRFS 的更新、更新的 JDBC 驅動程式，以及 AWS Glue 提供的對 Spark 本身的其他最佳化。

- Scala 從 2.11 更新到 2.12，而 Scala 2.12 不回溯相容 Scala 2.11。
- Python 3.10 也是用於 Python 指令碼的預設版本，因為 AWS Glue 2.0 只利用 Python 3.7 和 2.7。
  - Python 2.7 不支援 Spark 3.3.0。在任務組態中請求 Python 2 的任何任務都將失敗，並出現 IllegalArgumentException。
  - 自 AWS Glue 2.0 起，提供安裝其他 Python 模組的新機制。
- 數個相依性更新，在 附錄 A：值得注意的相依性升級 中反白顯示。
- 在現有 AWS Glue 2.0 任務中提供的任何額外 JAR 檔案可能會帶來衝突的相依性，因為 4.0 中的幾個相依性已從 2.0 升級。您可以避免 AWS Glue 4.0 與 `--user-jars-first` AWS Glue 任務參數的 classpath 衝突。
- AWS Glue 4.0 使用 Spark 3.3。從 Spark 3.1 開始，從 Parquet 檔案載入時間戳記以及將時間戳記儲存至此類檔案的行為發生了變化。有關更多詳細資訊，請參閱從 Spark SQL 3.0 升級到 3.1。

  建議在讀取/寫入包含時間戳記資料行的 parquet 資料時設定以下參數。針對 AWS Glue 動態框架和 Spark 資料框架，設定這些參數可以解決從 Spark 2 升級到 Spark 3 期間發生的行事曆不相容問題。使用「更正」選項原樣讀取 datetime 值；使用「舊式」選項可以在讀取過程中針對行事曆差異重新設定 datetime 值的基準。

  ```
  - Key: --conf
  - Value: spark.sql.legacy.parquet.int96RebaseModeInRead=[CORRECTED|LEGACY] --
  conf spark.sql.legacy.parquet.int96RebaseModeInWrite=[CORRECTED|LEGACY] --conf
   spark.sql.legacy.parquet.datetimeRebaseModeInRead=[CORRECTED|LEGACY]
  ```
- 如需遷移某些連接器，請參閱AWS Glue 4.0 的連接器和 JDBC 驅動程式遷移。
- AWS Encryption SDK 已從 1.x 升級到 2.x。這會影響使用 AWS Glue 安全組態的 AWS Glue 任務，以及相依在執行階段中所提供 AWS Encryption SDK 相依性的任務。請參閱 AWS Glue 任務遷移的說明：
  - 由於 AWS Glue 2.0 已包含 AWS Encryption SDK 橋接器版本，因此您可安全地將 AWS Glue 2.0 任務升級到 AWS Glue 4.0 任務。

請參閱 Spark 遷移文件：

- 從 Spark SQL 2.4 升級到 3.0
- 從 Spark SQL 3.1 升級到 3.2
- 從 Spark SQL 3.2 升級到 3.3
- 自 Spark 3.0 以來預期日期時間行為的變化。

從 AWS Glue 1.0 遷移到 AWS Glue 4.0

遷移時請注意下列變更：

- AWS Glue 1.0 使用開源 Spark 2.4 而 AWS Glue 4.0 使用 Amazon EMR 最佳化 Spark 3.3.0。
  - 單獨的數個 Spark 變更可能需要修改指令碼，以確保沒有引用被刪除的功能。
  - 例如，Spark 3.3.0 不啟用 Scala 無類型的 UDF，但 Spark 2.4 確實允許它們。
- AWS Glue 4.0 中的所有任務都將以顯著改善的啟動時間執行。Spark 任務將以 1 秒的增量計費，最短計費持續時間變為原來的十分之一，因為啟動延遲將從最大 10 分鐘到最大 1 分鐘。
- 記錄行為在 AWS Glue 4.0 中有顯著變更，而 Spark 3.3.0 有最低的 Log4j2 要求。
- 數個相依性更新，在附錄中反白顯示。
- Scala 也從 2.11 更新到 2.12，而 Scala 2.12 不回溯相容 Scala 2.11。
- Python 3.10 也是用於 Python 指令碼的預設版本，因為 AWS Glue 0.9 只利用 Python 2。

  Python 2.7 不支援 Spark 3.3.0。在任務組態中請求 Python 2 的任何任務都將失敗，並出現 IllegalArgumentException。
- 自 AWS Glue 2.0 起，提供透過 pip 安裝其他 Python 模組的新機制。如需詳細資訊，請參閱使用 pip 在 AWS Glue 2.0+ 中安裝其他 Python 模組。
- AWS Glue 4.0 不在 Apache YARN 上執行，因此不會套用 YARN 設定。
- AWS Glue 4.0 沒有 Hadoop 分散式檔案系統 (HDFS)。
- 在現有 AWS Glue 1.0 任務中提供的任何額外 JAR 檔案可能會帶來衝突的相依性，因為 4.0 中的幾個相依性已從 1.0 升級。我們已預設啟用包含 `--user-jars-first` AWS Glue 任務參數的 AWS Glue 4.0，以避免此問題。
- AWS Glue 4.0 支援 Auto Scaling。因此，啟用 Auto Scaling 後，將提供 ExecutorAllocationManager 指標。
- 在 AWS Glue 4.0 版任務，您可以指定工作者和工作者類型的數目，但不指定 `maxCapacity`。
- AWS Glue 4.0 尚未支援機器學習轉換。
- 如需遷移某些連接器，請參閱AWS Glue 4.0 的連接器和 JDBC 驅動程式遷移。
- AWS Encryption SDK 已從 1.x 升級到 2.x。這會影響使用 AWS Glue 安全組態的 AWS Glue 任務，以及相依在執行階段中所提供 AWS Encryption SDK 相依性的任務。請參閱 AWS Glue 任務遷移的這些說明。
  - 您無法直接將 AWS Glue 0.9/1.0 任務遷移到 AWS Glue 4.0 任務。這是因為當直接升級到 2.x 版或更高版本並立即啟用所有新功能時，AWS Encryption SDK 將無法解密透過 AWS Encryption SDK 之前版本加密的密文。

- 若要安全地升級，建議您先遷移到包含 AWS Encryption SDK 橋接器版本的 AWS Glue 2.0/3.0 任務。執行任務一次，即可使用 AWS Encryption SDK 橋接器版本。
- 完成後，您可以安全地將 AWS Glue 2.0/3.0 任務遷移到 AWS Glue 4.0。

請參閱 Spark 遷移文件：

- 從 Spark SQL 2.4 升級到 3.0
- 從 Spark SQL 3.0 升級到 3.1
- 從 Spark SQL 3.1 升級到 3.2
- 從 Spark SQL 3.2 升級到 3.3
- 自 Spark 3.0 以來預期日期時間行為的變化。

從 AWS Glue 0.9 遷移到 AWS Glue 4.0

遷移時請注意下列變更：

- AWS Glue 0.9 使用開源 Spark 2.2.1 而 AWS Glue 4.0 使用 Amazon EMR 最佳化 Spark 3.3.0。
  - 單獨的數個 Spark 變更可能需要修改指令碼，以確保沒有引用被刪除的功能。
  - 例如，Spark 3.3.0 不啟用 Scala 無類型的 UDF，但 Spark 2.2 確實允許它們。
- AWS Glue 4.0 中的所有任務都將以顯著改善的啟動時間執行。Spark 任務將以 1 秒的增量計費，最短計費持續時間變為原來的十分之一，因為啟動延遲將從最大 10 分鐘變為最大 1 分鐘。
- 自 AWS Glue 4.0 起，記錄行為已有顯著的變更，Spark 3.3.0 具有 Log4j2 的最低要求，如下所述 (https://spark.apache.org/docs/latest/core-migration-guide.html#upgrading-from-core-32-to-33)。
- 數個相依性更新，在附錄中反白顯示。
- Scala 也從 2.11 更新到 2.12，而 Scala 2.12 不回溯相容 Scala 2.11。
- Python 3.10 也是用於 Python 指令碼的預設版本，因為 AWS Glue 0.9 只利用 Python 2。
  - Python 2.7 不支援 Spark 3.3.0。在任務組態中請求 Python 2 的任何任務都將失敗，並出現 IllegalArgumentException。
  - 提供透過 pip 安裝其他 Python 模組的新機制。
- AWS Glue 4.0 不在 Apache YARN 上執行，因此不會套用 YARN 設定。
- AWS Glue 4.0 沒有 Hadoop 分散式檔案系統 (HDFS)。

- 在現有 AWS Glue 0.9 任務中提供的任何額外 JAR 檔案可能會帶來衝突的相依性，因為 3.0 中的幾個相依性已從 0.9 升級。您可以避免 AWS Glue 3.0 與 `--user-jars-first` AWS Glue 任務參數的 classpath 衝突。
- AWS Glue 4.0 支援 Auto Scaling。因此，啟用 Auto Scaling 後，將提供 ExecutorAllocationManager 指標。
- 在 AWS Glue 4.0 版任務，您可以指定工作者和工作者類型的數目，但不指定 `maxCapacity`。
- AWS Glue 4.0 尚未支援機器學習轉換。
- 如需遷移某些連接器，請參閱AWS Glue 4.0 的連接器和 JDBC 驅動程式遷移。
- AWS Encryption SDK 已從 1.x 升級到 2.x。這會影響使用 AWS Glue 安全組態的 AWS Glue 任務，以及相依在執行階段中所提供 AWS Encryption SDK 相依性的任務。請參閱 AWS Glue 任務遷移的這些說明。
  - 您無法直接將 AWS Glue 0.9/1.0 任務遷移到 AWS Glue 4.0 任務。這是因為當直接升級到 2.x 版或更高版本並立即啟用所有新功能時，AWS Encryption SDK 將無法解密透過 AWS Encryption SDK 之前版本加密的密文。
  - 若要安全地升級，建議您先遷移到包含 AWS Encryption SDK 橋接器版本的 AWS Glue 2.0/3.0 任務。執行任務一次，即可使用 AWS Encryption SDK 橋接器版本。
  - 完成後，您可以安全地將 AWS Glue 2.0/3.0 任務遷移到 AWS Glue 4.0。

請參閱 Spark 遷移文件：

- 從 Spark SQL 2.2 升級到 2.3
- 從 Spark SQL 2.3 升級到 2.4
- 從 Spark SQL 2.4 升級到 3.0
- 從 Spark SQL 3.0 升級到 3.1
- 從 Spark SQL 3.1 升級到 3.2
- 從 Spark SQL 3.2 升級到 3.3
- 自 Spark 3.0 以來預期日期時間行為的變化。

AWS Glue 4.0 的連接器和 JDBC 驅動程式遷移

如需已升級的 JDBC 和資料湖連接器版本，請參閱：

- 附錄 B：JDBC 驅動程式升級
- 附錄 C：連接器升級

Hudi

- Spark SQL 支援改善項目：
  - 透過 `Call Procedure` 命令新增升級、降級、引導、清理和維修的支援。`Create/Drop/Show/Refresh Index` 語法可在 Spark SQL 中使用。
  - 透過 Spark DataSource 的使用消除對照於 Spark SQL 的效能落差。過去的 Datasource 寫入曾比 SQL 快。
  - 所有內建金鑰產生器都會實作更多效能 Spark 特定 API 操作。
  - 用 RDD 轉換取代大量 `insert` 操作中的 UDF 轉換，以削減使用 SerDe 的成本。
  - 使用 Hudi 的 Spark SQL 需要由 `tblproperites` 或 SQL 陳述式中的選項指定 `primaryKey`。更新和刪除操作也需要 `preCombineField`。
- 自 0.10.0 版起，在 0.10.0 版之前建立且無 `primaryKey` 的任何 Hudi 資料表都需要使用 `primaryKey` 欄位重新建立。

PostgreSQL

- 已處理數個安全漏洞 (CVE)。
- 原生支援 Java 8。
- 如果任務正使用陣列的陣列，此情境可以作為多維陣列處理，但不包括位元組陣列。

MongoDB

- 目前 MongoDB 連接器支援 Spark 3.1 版或更高版本和 MongoDB 4.0 版或更高版本。
- 由於連接器升級，部分屬性名稱也已變更。例如，URI 屬性名稱已變更為 `connection.uri`。如需目前選項的詳細資訊，請參閱 MongoDB Spark Connector 部落格。
- 使用由 Amazon DocumentDB 託管的 MongoDB 4.0 時有些功能差異。如需詳細資訊，請參閱以下主題：
  - 功能差異：Amazon DocumentDB 和 MongoDB
  - 支援的 MongoDB API、操作和資料類型。
- "Partitioner" 選項僅限於 `ShardedPartitioner`、`PaginateIntoPartitionsPartitioner` 和 `SinglePartitionPartitioner`。該選項不能將預設 `SamplePartitioner` 和 `PaginateBySizePartitioner` 用於 Amazon DocumentDB，因為階段運算子不支援 MongoDB API。如需詳細資訊，請參閱 Supported MongoDB APIs, Operations, and Data Types (支援的 MongoDB API、操作和資料類型)。

Delta Lake

- Delta Lake 現在支援 SQL 中的時間移動操作，可輕鬆查詢舊資料。透過此更新，現在可在 Spark SQL 中以及透過 DataFrame API 提供時間移動操作。已在 SQL 中新增對目前 TIMESTAMP 版本的支援。
- Spark 3.3 導入 Trigger.AvailableNow，以執行串流查詢 (等同於用於批次查詢的 `Trigger.Once`)。使用 Delta 資料表作為串流來源時，也提供此支援。
- 支援傳回資料表中資料欄清單的 SHOW COLUMNS。
- 支援 Scala 和 Python DeltaTable API 中的 DESCRIBE DETAIL。其會使用 DeltaTable API 或 Spark SQL 擷取 Delta 資料表的相關詳細資訊。
- 支援從 SQL Delete、Merge 和 Update 命令傳回操作指標。這些 SQL 命令之前會傳回空的 DataFrame，現在則會傳回包含已執行操作之實用指標的 DataFrame。
- 最佳化效能改善項目：
  - 設定組態選項 `spark.databricks.delta.optimize.repartition.enabled=true` 以在 Optimize 命令中使用 `repartition(1)` 而非 `coalesce(1)`，以便在精簡小型檔案時實現更高的效能。
  - 使用佇列式方法平行化精簡任務，從而改善效能。
- 其他顯著的變更：
  - 支援將變數用於 VACUUM 和 OPTIMIZE SQL 命令。
  - 目錄資料表 CONVERT TO DELTA 的改善功能，包括：
    - 未提供時，自動填入分割區結構描述。
    - 使用目錄的分割區資訊來尋找要遞交的資料檔案，而不進行完整的目錄掃描。只會遞交使用中分割區目錄下的資料檔案，而不遞交資料表目錄中的所有資料檔案。
  - 未使用 DROP COLUMN 與 RENAME COLUMN 時，在已啟用資料欄對應的資料表上支援變更資料饋送 (CDF) 批次讀取。如需詳細資訊，請參閱 Delta Lake 文件。
  - 在第一次傳遞中啟用結構描述剔除，以改善 Update 命令的效能。

Apache Iceberg

- 已新增掃描規劃和 Spark 查詢的數項效能改善項目。
- 已新增常見的 REST 目錄用戶端，該用戶端使用以變更為基礎的遞交來解決服務端的遞交衝突。
- 已支援 SQL 時間移動查詢的 `AS OF` 語法。
- 已新增 MERGE 和 UPDATE 查詢的「讀取時合併」支援。

- 已新增使用 Z-order 重新寫入分割區的支援。
- 已新增 Puffin 的規格和實作，此為大型統計資料和索引 blob 的格式，就像是 Theta 示意圖或 bloom 篩選條件。
- 已新增遞增取用資料的新介面 (附加和變更日誌掃描)。
- 已新增對於大量操作和遠端讀取 FileIO 介面的支援。
- 已新增更多中繼資料表，以在中繼資料樹狀結構中顯示刪除檔案。
- 已變更放置資料表行為。在 Iceberg 0.13.1 中，執行 `DROP TABLE` 會從目錄移除資料表，並刪除資料表內容。在 Iceberg 1.0.0 中，`DROP TABLE` 僅從目錄刪除資料表。若要刪除資料表內容，請使用 `DROP TABLE PURGE`。
- 已在 Iceberg 1.0.0 中啟用 Parquet 向量化讀取。如要停用向量化讀取，請將 `read.parquet.vectorization.enabled` 設定為 `false`。

Oracle

僅為次要變更。

MySQL

僅為次要變更。

Amazon Redshift

AWS Glue 4.0 採用配備新 JDBC 驅動程式的全新 Amazon Redshift 連接器。如需有關增強功能以及如何從之前 AWS Glue 版本遷移的資訊，請參閱the section called “Redshift 連線”。

附錄 A：值得注意的相依性升級

以下是相依性升級：

相依性	AWS Glue 4.0 中的版本	AWS Glue 3.0 中的版本	AWS Glue 2.0 中的版本	AWS Glue 1.0 中的版本
Spark	3.3.0-amzn-1	3.1.1-amzn-0	2.4.3	2.4.3
Hadoop	3.3.3-amzn-0	3.2.1-amzn-3	2.8.5-amzn-5	2.8.5-amzn-1
Scala	2.12	2.12	2.11	2.11

相依性	AWS Glue 4.0 中的版本	AWS Glue 3.0 中的版本	AWS Glue 2.0 中的版本	AWS Glue 1.0 中的版本
Jackson	2.13.3	2.10.x	2.7.x	2.7.x
Hive	2.3.9-amzn-2	2.3.7-amzn-4	1.2	1.2
EMRFS	2.54.0	2.46.0	2.38.0	2.30.0
Json4s	3.7.0-M11	3.6.6	3.5.x	3.5.x
Arrow	7.0.0	2.0.0	0.10.0	0.10.0
AWS Glue Data Catalog 用戶端	3.7.0	3.0.0	1.10.0	N/A
Python	3.10	3.7	2.7 和 3.6	2.7 和 3.6
Boto	1.26	1.18	1.12	N/A

附錄 B：JDBC 驅動程式升級

以下是 JDBC 驅動程式升級：

驅動程式	過去 AWS Glue 版本的 JDBC 驅動程式版本	AWS Glue 3.0 的 JDBC 驅動程式版本	AWS Glue 4.0 中的 JDBC 驅動程式版本
MySQL	5.1	8.0.23	8.0.23
Microsoft SQL Server	6.1.0	7.0.0	9.4.0
Oracle 資料庫	11.2	21.1	21.7
PostgreSQL	42.1.0	42.2.18	42.3.6
MongoDB	2.0.0	4.0.0	4.7.2
Amazon Redshift	redshift-jdbc41-1.2.12.1017	redshift-jdbc41-1.2.12.1017	redshift-jdbc42-2.1.0.16

附錄 C：連接器升級

以下是連接器升級：

驅動程式	AWS Glue 3.0 中的連接器版本	AWS Glue 4.0 中的連接器版本
MongoDB	3.0.0	10.0.4
Hudi	0.10.1	0.12.1
Delta Lake	1.0.0	2.1.0
Iceberg	0.13.1	1.0.0
DynamoDB	1.11	1.12

將 AWS Glue for Ray 從預覽版遷移至 Ray2.4 執行期環境

> **Warning**
>
> 當您將 AWS Glue for Ray (預覽版) 任務儲存在 AWS Glue Studio 中時，其將會自動升級至 `Ray2.4` 執行期。如果您的指令碼遇到相容性問題，請聯絡支援人員。

您應將在 AWS Glue for Ray (預覽版) 期間建立的 AWS Glue 任務遷移至 AWS Glue for Ray。這將涉及對您的任務組態進行一些並行變更。

- 在 `Runtime` 欄位中，提供 `Ray2.4` 執行期值。這會將基礎的 Ray 版本從 2.0.0 升級至 2.4.0。
- 預覽版中預設包含的某些 Python 程式庫已不再提供。如果任務利用適用於 pandas (awswrangler)、dask、modin 或 pymars 的 AWS SDK，您將需要將其作為附加程式庫包含在內。如需有關包含其他 Python 程式庫的詳細資訊，請參閱 the section called “Ray 任務的其他 Python 模組”。
- 如果您使用 `--additional-python-modules` 參數，則用於支援此工作流程的參數已分為 `--pip-install` 和 `--s3-py-modules`。如需這些參數的詳細資訊，請參閱 the section called “Ray 任務的其他 Python 模組”。
- 如果您使用的是 `--auto-scaling-ray-min-workers` 參數，則該參數已重新命名為 `--min-workers`。

AWS Glue 版本支援政策

AWS Glue 是無伺服器資料整合服務，可讓您輕鬆探索、準備及合併資料，以進行分析、機器學習和應用程式開發。AWS Glue 任務包含在 AWS Glue 中執行資料整合工作的商業邏輯。AWS Glue 中有兩種任務類型：Spark (批次和串流)、Ray 和 Python shell。在定義任務時，您可以指定 AWS Glue 版本，該版本會在基礎 Spark、Ray 或 Python 執行期環境中設定版本。例如：AWS Glue 2.0 版 Spark 任務支援 Spark 2.4.3 和 Python 3.7。

支援政策

AWS Glue 偶爾會停止對早期 AWS Glue 版本的支援。然而，在已過時版本上所執行的任務無法再獲得技術支援。AWS Glue 不再將安全性修補程式或其他更新套用至已過時版本。在已過時版本上執行任務時，AWS Glue 也不會再遵守 SLA。

當 AWS Glue 2.0 版或更新版本的支援結束時，您將無法建立任務，只能編輯或執行任務。

下列 AWS Glue 版本已到達或排定終止支援。終止支援的時間從指定日期的午夜（太平洋時區）開始。

類型	Glue 版本	終止支援
Spark	Spark 2.2，Scala 2（Glue 0.9 版）	6/1/2022
Spark	Spark 2.2，Python 2（Glue 0.9 版）	6/1/2022
Spark	Spark 2.4，Python 2（Glue 1.0 版）	6/1/2022
Spark	Spark 2.4，Python 3（Glue 1.0 版）	2022 年 9 月 30 日
Spark	Spark 2.4，Scala 2（Glue 1.0 版）	2022 年 9 月 30 日
Spark	Glue 2.0 版	1/31/2024

類型	Python 版本	終止支援
Python shell	Python 2（Glue 1.0 版）	6/1/2022
類型	筆記本版本	終止支援
開發端點	Zeppelin 筆記本	2022 年 9 月 30 日

AWS 強烈建議您將任務遷移至支援的版本中。

如需有關將 Spark 任務遷移至最新 AWS Glue 版本的資訊，請參閱將 AWS Glue 任務遷移至 AWS Glue 4.0 版。

將您的 Python shell 任務遷移到最新的 AWS Glue 版本：

- 在主控台中，選擇 `Python 3 (Glue Version 4.0)`。
- 在 CreateJob/UpdateJobAPI 中，將`GlueVersion`參數設定為`2.0`，並在`PythonVersionCommand`參數`3`下設定為。該`GlueVersion`配置不會影響 Python 外殼作業的行為，因此增加`GlueVersion`沒有任何優勢。
- 您需要讓自己的任務指令碼與 Python 3 相容。

> **Note**
>
> 在 2022 年 8 月推出印尼雅加達 (ap-Southest-3) 區域之前推出的所有 AWS 區域，皆有允許執行 AWS Glue 0.9/1.0 版任務執行的客戶允許清單。在這些較舊的區域中，您可以使用 Null 值建立任務，並根據「區域」預設為 0.9/1.0 版。針對所有後續推出的 AWS 區域，您必須在 API 中明確設定 AWS Glue 版本。AWS Glue 不再接受 Null 參數。若您在參數中傳遞 0.9 或 1.0，則會遇到以下錯誤：「不支援 Glue 0.9 (或) 1.0 版」。

使用星火工作於 AWS Glue

提供有關星火 ETL 工作 AWS Glue 的資訊。

主題

- AWS Glue 任務參數
- AWS Glue 火花和 PySpark 工作

- 在 AWS Glue 中串流 ETL 任務
- 使用 AWS Lake Formation FindMatches 比對記錄
- 將 Apache Spark 程式遷移到 AWS Glue

AWS Glue 任務參數

建立 AWS Glue 工作時，您可以設定一些標準欄位，例如Role和WorkerType。您可以透過 Argument 欄位 (主控台中的任務參數) 提供其他組態資訊。在這些欄位中，您可以為 AWS Glue 工作提供本主題所列的引數 (參數)。如需有關 AWS Glue 合 Job API 的詳細資訊，請參閱the section called “任務”。

設定任務參數

您可以透過主控台在 Job Parameters (任務參數) 標題下的 Job details (任務詳細資訊) 索引標籤中設定任務。您也可以透 AWS CLI 過設定DefaultArguments或工作，或NonOverridableArguments在工作執行中設定Arguments來設定工作。每次執行任務時，都會傳入任務上設定的引數，而在任務執行中設定的引數只會針對該個別執行傳入。

例如，下列是執行使用 --arguments 來設定任務參數之任務的語法。

```
$ aws glue start-job-run --job-name "CSV to CSV" --arguments='--scriptLocation="s3://
my_glue/libraries/test_lib.py"'
```

存取任務參數

撰寫 AWS Glue 指令碼時，您可能會想要存取工作參數值，以改變您自己程式碼的行為。我們提供協助程式方法來在我們的程式庫中執行此操作。這些方法會解析覆寫任務參數值的任務執行參數值。解析在多個位置設定的參數時，任務 NonOverridableArguments 會覆寫任務執行 Arguments，這將覆寫任務 DefaultArguments。

在 Python 中：

在 Python 任務中，我們提供了一個名為 getResolvedParameters 的函數。如需詳細資訊，請參閱 the section called “getResolvedOptions”。任務參數可在 sys.argv 變數中使用。

在 Scala 中：

在 Scala 任務中，我們提供了一個名為 `GlueArgParser` 的物件。如需詳細資訊，請參閱 the section called “GlueArgParser”。任務參數可在 `sysArgs` 變數中使用。

任務參數參考

AWS Glue 會辨識以下引數名稱，您可以用來為您的任務和任務執行設定指令碼環境：

`--additional-python-modules`

以逗號分隔的清單，代表要安裝的一組 Python 套件。您可以從 PyPI 安裝套件或提供自訂發行版本。PyPI 套件項目的格式為 *`package`*==*`version`*，其中包含 PyPI 名稱和目標套件的版本。自訂發行版本項目是發行版本的 S3 路徑。

項目使用 Python 版本比對來比對套件和版本。這意味著您將需要使用兩個等號，例如 ==。還有其他版本的比對運算子，如需更多資訊，請參閱 PEP 440。

若要將模組安裝選項傳遞給 `pip3`，請使用 --python-modules-installer-option 參數。

`--auto-scale-within-microbatch`

預設值為 false。此參數只能用於 AWS Glue 串流工作，這些工作會以一系列微型批次處理串流資料，且必須啟用 auto 調整規模。將此值設定為 false 時，會運算已完成微型批次之批次處理持續時間的指數移動平均值，並將此值與間格大小進行比較，以判斷是縱向擴展執行器數目還是縮減其規模。只有在微型批次完成時才會進行擴展。將此值設定為 true 時，在微型批次期間，當 Spark 任務數量保持不變 30 秒，或目前的批次處理大於間格大小時，它會縱向擴展。如果執行器閒置超過 60 秒，或批次處理持續時間的指數移動平均值較低，則執行器數量將下降。

`--class`

可作為您的 Scala 指令碼進入點的 Scala 類別。這只在您的 `--job-language` 設為 `scala` 才適用。

`--continuous-log-conversionPattern`

為啟用連續記錄的任務指定自訂轉換日誌模式。轉換模式僅適用於驅動程式日誌和執行程式日誌。它不會影響 AWS Glue 進度列。

`--continuous-log-logGroup`

為啟用連續記錄的任務指定自訂 Amazon CloudWatch 日誌群組名稱。

`--continuous-log-logStreamPrefix`

為啟用連續記錄的工作指定自訂 CloudWatch 記錄資料流前置詞。

--customer-driver-env-vars 和 --customer-executor-env-vars

這些參數會分別為每個 Worker (驅動程式或執行程式) 在作業系統上設定環境變數。在 AWS Glue 之上建置平台和自訂架構時，您可以使用這些參數，讓使用者在其上撰寫工作。啟用這兩個標誌將允許您分別在驅動程序和執行程序上設置不同的環境變量，而不必在作業腳本本身中注入相同的邏輯。

範例使用方式

以下是使用這些參數的範例：

```
"–customer-driver-env-vars", "CUSTOMER_KEY1=VAL1,CUSTOMER_KEY2=\"val2,val2 val2\"",
"–customer-executor-env-vars", "CUSTOMER_KEY3=VAL3,KEY4=VAL4"
```

在工作執行引數中設定這些值等同於執行下列命令：

在驅動程式中：

- 導出客戶密鑰 1 = 值 1
- 導出客戶密鑰 2 =「值 2，值 2」

在遺囑執行人中：

- 導出客戶密鑰 3 = 值 3

然後，在工作命令檔本身中，您可以使用os.environ.get("CUSTOMER_KEY1")或擷取環境變數System.getenv("CUSTOMER_KEY1")。

強制語法

定義環境變數時，請注意下列標準：

- 每個金鑰都必須有CUSTOMER_ prefix.

  例如：for"CUSTOMER_KEY3=VAL3,KEY4=VAL4"，KEY4=VAL4將被忽略而不設置。
- 每個鍵和值對必須用一個逗號劃定。

  例如："CUSTOMER_KEY3=VAL3,CUSTOMER_KEY4=VAL4"
- 如果「值」有空格或逗號，則必須在引號中定義它。

  例如：CUSTOMER_KEY2=\"val2,val2 val2\"

此語法密切模型設置 bash 環境變量的標準。

--datalake-formats

在 AWS Glue 3.0 及更新版本中支援。

指定要使用的資料湖架構。 AWS Glue 會為您指定的架構新增必要的 JAR 檔案`classpath`。如需詳細資訊，請參閱 將資料湖架構與 AWS Glue ETL 任務搭配使用。

您可以指定以下一或多個值，以逗號分隔：

- `hudi`
- `delta`
- `iceberg`

例如，傳遞下列引數來指定所有三個架構。

```
'--datalake-formats': 'hudi,delta,iceberg'
```

--disable-proxy-v2

停用服務代理以允許透過 VPC 從指令碼傳AWS Glue送至 Amazon S3 的 AWS 服務呼叫。CloudWatch如需詳細資訊，請參閱設定 AWS 呼叫通過您的 VPC。若要停用服務代理，請將此參數的值設定為 `true`。

--enable-auto-scaling

將此值設定為 `true` 時，開啟自動擴展和按每名工作者計費。

--enable-continuous-cloudwatch-log

啟用即時、持續的 AWS Glue 任務記錄。您可以檢視即時 Apache 星火工作記錄中 CloudWatch。

--enable-continuous-log-filter

當您建立或編輯已啟用持續記錄的任務時，指定標準篩選條件 (`true`) 或無篩選條件 (`false`)。選擇標準篩選條件可剔除無用的 Apache Spark 驅動程式/執行器，以及 Apache Hadoop YARN 活動訊號日誌訊息。選擇無篩選條件可讓您獲得所有日誌訊息。

--enable-glue-datacatalog

可讓您使用 AWS Glue 資料目錄做為 Apache 星火蜂巢中繼存放區。若要啟用此功能，請將此值設定為 `true`。

--enable-job-insights

使用 AWS Glue 工作執行深入解析，啟用額外的錯誤分析 如需詳細資訊，請參閱 the section called "使用 AWS Glue 任務執行見解進行監控"。依預設，此值設定為 `true` 並啟用任務執行見解。

此選項適用於 AWS Glue 版本 2.0 和 3.0。

--enable-metrics

針對此任務執行啟用任務分析的指標集合。這些指標可在AWS Glue主控台和 Amazon 主控 CloudWatch 台上取得。此參數的值不相關。若要啟用此功能，您可為此參數提供任何值，但為了確保清晰明瞭，建議您使用 `true`。若要停用此功能，請從任務組態中移除此參數。

--enable-observability-metrics

在AWS Glue主控台和主控台下的「Job 執行監視」頁面上，啟用一組「觀察性」指標，以產生每個工作執行中的情況的 Amazon CloudWatch 見解。若要啟用此功能，請將此參數值設定為 true。若要停用此功能，請將其設為 `false` 或從作業組態中移除此參數。

--enable-rename-algorithm-v2

將 EMRFS 重新命名演算法版本設定為第 2 版。當 Spark 任務使用動態分割區覆寫模式時，有可能會建立重複的分割區。例如，您最終可能會得到一個重複的分割區，例如 `s3://bucket/table/location/p1=1/p1=1`。在這裡，P1 是被覆寫的分割區。重新命名演算法版本 2 會修正此問題。

此選項僅適用於 AWS Glue 1.0 版。

--enable-s3-parquet-optimized-committer

啟用 EMRFS S3 最佳化遞交者，以將 Parquet 資料寫入 Amazon S3。您可以在建立或更新 AWS Glue 任務時，透過 AWS Glue 主控台提供參數/值對。將值設為 **`true`** 以啟用遞交者。在 AWS Glue 3.0 中，旗標預設為開啟，並在 G AWS lue 2.0 中關閉。

如需詳細資訊，請參閱使用 EMRFS S3 最佳化遞交者。

--enable-spark-ui

當設定為 `true` 時，開啟該功能以使用 Spark UI 來監控和偵錯 AWS Glue ETL 任務。

--executor-cores

可以平行執行的 Spark 任務數量。 AWS Glue 3.0 以上支援此選項。該值不應超過工作者類型上 vCPU 數量的 2 倍，即分別為 `G.1X` 上 8 個、`G.2X` 上 16 個、`G.4X` 上 32 個和 `G.8X` 上 64 個。更新此組態時應小心，這可能會影響任務效能，因為增加任務平行處理會導致記憶體和磁碟壓力，並且可能會對來源和目標系統限流 (例如：它會在 Amazon RDS 上造成更多並行連線)。

--extra-files

組態檔案等其他檔案的 Amazon S3 路徑；在系統執行您的指令碼前，AWS Glue 即會將其複製至指令碼的工作目錄。多個值必須是以英文逗號 (,) 分隔的完整路徑。僅支援個別檔案路徑，而非目錄路徑。Python Shell 任務類型不支援此選項。

--extra-jars

其他 Java `.jar` 檔案的 Amazon S3 路徑；在系統執行您的指令碼前，AWS Glue 即會將其新增至 Java classpath。多個值必須是以英文逗號 (,) 分隔的完整路徑。

--extra-py-files

其他 Python 模組的 Amazon S3 路徑；在系統執行您的指令碼之前，AWS Glue 即會將其新增至 Python 路徑。多個值必須是以英文逗號 (,) 分隔的完整路徑。僅支援個別檔案路徑，而非目錄路徑。

--job-bookmark-option

控制任務書籤的行為。可設定以下選項值：

--job-bookmark-option 值	描述
`job-bookmark-enable`	追蹤先前已處理的資料。當任務執行時，從最後一個檢查點開始處理新資料。
`job-bookmark-disable`	一律處理整個資料集。您需負責管理之前任務執行的輸出。
`job-bookmark-pause`	處理上次成功執行後的遞增資料，或由下列子選項識別範圍內的資料，而不更新最後一個書籤的狀態。您需負責管理之前任務執行的輸出。兩個子選項如下： • **job-bookmark-from** `<from-value>` 是執行 ID，它表示所有被處理，直到最後一次成功執行之前，包括指定的執行 ID 的輸入。對應的輸入會被忽略。 • **job-bookmark-to** `<to-value>` 是執行 ID，它表示所有被處理，直到最後一次成功執行之前，包括指定的執行 ID 的輸入。由 `<from-value>` 識別的輸入以外的對應的輸入由該任務處理。此輸入後的任何輸入也會排除而不進行處理。

--job-bookmark-option 值	描述
	指定此選項組時，不會更新任務書籤狀態。 子選項是選擇性的。但是，使用時必須提供兩個子選項。

例如，若要啟用任務書籤，請傳遞以下參數：

```
'--job-bookmark-option': 'job-bookmark-enable'
```

--job-language

指令碼程式設計語言。此值必須為 scala 或 python。如果此參數不存在，則預設值為 python。

--python-modules-installer-option

使用 --additional-python-modules 安裝模組時，針對要傳送到 pip3 的選項給予定義的純文字字串。以您在命令列中執行的相同方式提供選項，以空格分隔各項，並在前方加上英文破折號。如需使用方面的詳細資訊，請參閱 the section called “使用 pip 在 AWS Glue 2.0+ 中安裝其他 Python 模組”。

> **Note**
>
> 當您使用 Python 3.9 時， AWS Glue 合工作不支援這個選項。

--scriptLocation

ETL 指令碼所在的 Amazon Simple Storage Service (Amazon S3) 位置 (格式是 s3://path/to/my/script.py)。此參數會覆寫在 JobCommand 物件中設定的指令碼位置。

--spark-event-logs-path

指定 Amazon S3 路徑。使用 Spark UI 監控功能時，AWS Glue 會每隔 30 秒將此 Amazon S3 路徑的 Spark 事件日誌排清到儲存貯體，該儲存貯體可用作存放 Spark UI 事件的暫時目錄。

--TempDir

指定儲存貯體的 Amazon S3 路徑做為任務的暫時目錄。

例如，若要設定臨時目錄，請傳遞以下引數：

```
'--TempDir': 's3-path-to-directory'
```

> **Note**
>
> 如果區域中尚未存在儲存貯體，則 AWS Glue Glue 會為任務建立暫時儲存貯體。此儲存貯體可能允許公開存取。您可以修改 Amazon S3 中的儲存貯體以設定的公有存取封鎖，或稍後在該區域的所有任務完成後刪除儲存貯體。

`--use-postgres-driver`

將此值設定為 `true`，即會在類別路徑中優先考慮 Postgres JDBC 驅動程式，以避免與 Amazon Redshift JDBC 驅動程式發生衝突。此選項僅能在 AWS Glue 2.0 版中使用。

`--user-jars-first`

將此值設定為 `true`，即會在 classpath 中優先考慮客戶的額外 JAR 檔案。此選項僅能在 AWS Glue 2.0 版或更新版中使用。

`--conf`

控制 Spark 組態參數。適用於進階使用案例。

`--encryption-type`

舊參數。應使用安全組態設定對應的行為。如需有關安全組態的詳細資訊，請參閱 the section called “對 AWS Glue 寫入的資料加密”。

AWS Glue 在內部使用以下引數，您不應該使用它們：

- `--debug` — AWS Glue 內部引數名稱。請勿設定。
- `--mode` — AWS Glue 內部引數名稱。請勿設定。
- `--JOB_NAME` — AWS Glue 內部引數名稱。請勿設定。
- `--endpoint` — AWS Glue 內部引數名稱。請勿設定。

AWS Glue 支援透過 Python `site` 模組使用 `sitecustomize` 引導環境，以執行網站特定的自訂。建議僅將自己的初始化函數引導用於進階使用案例，並且 AWS Glue 4.0 上會竭盡全力提供相應支援。

環境變數字首 GLUE_CUSTOMER 會保留供客戶使用。

AWS Glue 火花和 PySpark 工作

以下各節提供有關 AWS Glue Spark 和 PySpark 工作的資訊。

主題

- 在 AWS Glue 中新增 Spark 和 PySpark 任務
- 使用任務書籤追蹤處理的資料
- AWS Glue Spark 隨機排序外掛程式與 Amazon S3
- 監控 AWS Glue Spark 任務

在 AWS Glue 中新增 Spark 和 PySpark 任務

以下章節將提供在 AWS Glue 中新增 Spark 和 PySpark 任務的資訊。

主題

- 設定 Spark 工作的工作屬性 AWS Glue
- 編輯 AWS Glue 主控台中的 Spark 指令碼
- 任務 (舊版)

設定 Spark 工作的工作屬性 AWS Glue

AWS Glue 任務封裝了一個指令碼，會連線至您的來源資料、處理資料，然後將它寫出至您的資料目標。一般而言，任務會執行擷取、轉換和載入 (ETL) 指令碼。任務也可以執行一般用途的 Python 指令碼 (Python Shell 任務)。AWS Glue 觸發程序可以根據排程或事件，或隨需啟動任務。您可以監控任務執行以了解執行時間指標，例如完成狀態、持續時間和開始時間。

您可以使用 AWS Glue 產生的指令碼，也可以提供自己的指令碼。透過原始碼結構描述和目標位置或結構描述，程式AWS Glue碼產生器可以自動建立 Apache Spark API (PySpark) 指令碼。您可以將此指令碼做為起點，編輯其內容以符合您的目標。

AWS Glue 可以用多種資料格式撰寫輸出檔案，包括 JSON、CSV、ORC (最佳化資料列單欄式)、Apache Parquet 和 Apache Avro。某些資料格式也可寫入常見的壓縮格式。

AWS Glue 中有三種任務類型：Spark、串流 ETL 和 Python shell。

- 星火作業是在由管理的 Apache 星火環境中運行 AWS Glue。它會分批次處理資料。
- 串流 ETL 任務類似 Spark 任務，不同之處在於它會在資料串流上執行 ETL。它使用 Apache Spark 結構化串流框架。某些 Spark 任務功能不適用於串流 ETL 任務。
- 將 Python 指令碼作為 shell 執行，並根據您使用的 AWS Glue 版本支援 Python 版本的 Python shell 任務執行。您可以使用這些任務來排程及執行不需要 Apache Spark 環境的任務。

定義 Spark 任務的任務屬性

若是在 AWS Glue 主控台定義任務，請提供屬性值以控制 AWS Glue 執行階段環境。

以下清單說明 Spark 任務的屬性。如需了解 Python shell 任務的屬性，請參閱 為 Python shell 任務定義任務屬性。如需了解串流 ETL 任務的屬性，請參閱 the section called “定義串流 ETL 任務的任務屬性”。

這些內容會依它們在 AWS Glue 主控台 Add job (新增任務) 精靈中出現的順序列出。

名稱

提供長度上限為 255 個字元的 UTF-8 字串。

描述

提供最多 2048 個字元的選擇性說明。

IAM 角色

指定 IAM 角色以授權使用執行工作和存取資料存放區所需的資源。如需在 AWS Glue 執行任務之許可的詳細資訊，請參閱 AWS Glue 的身分識別與存取管理。

Type

ETL 工作的類型。這會根據您選取的資料來源類型自動設定。

- 星火運行與作業命令`glueetl`阿帕奇星火 ETL 腳本。
- 星火流運行與作業命令`gluestreaming`的 Apache 星火流 ETL 腳本。如需詳細資訊，請參閱 the section called “串流 ETL 任務”。
- Python 外殼使用作業命令運行一個 Python 腳本`pythonshell`。如需詳細資訊，請參閱 設定 Python 殼層工作的工作屬性 AWS Glue。

AWS Glue 版本

AWS Glue 版本會決定任務可用的 Apache Spark 和 Python 版本，如下表中指定。

AWS Glue 版本	支援的 Spark 和 Python 版本
4.0	• Spark 3.3.0 • Python 3.10
3.0	• Spark 3.1.1 • Python 3.7
2.0	• Spark 2.4.3 • Python 3.7
1.0	• Spark 2.4.3 • Python 2.7 • Python 3.6
0.9	• Spark 2.2.1 • Python 2.7

工作者類型

可使用以下工作者類型：

AWS Glue 工作者上可用的資源是以 DPU 來衡量。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。

- **G.1X** – 選擇這種類型時，您也要提供 工作者數目 的值。每個工作者都會映射到 1 個 DPU (4 個 vCPU、16 GB 記憶體)，外加 84 GB 磁碟 (大約 34 GB 可用)。我們建議將此工作者類型用於資料轉換、聯結和查詢等工作負載，以提供可擴展且符合成本效益的方式來執行大部分任務。
- **G.2X** – 選擇這種類型時，您也要提供 工作者數目 的值。每個工作者都會映射到 2 個 DPU (8 個 vCPU、32 GB 記憶體)，外加 128 GB 磁碟 (大約 77 GB 可用)。我們建議將此工作者類型用於資料轉換、聯結和查詢等工作負載，以提供可擴展且符合成本效益的方式來執行大部分任務。
- **G.4X** – 選擇這種類型時，您也要提供 工作者數目 的值。每個工作者都會映射到 4 個 DPU (16 個 vCPU、64 GB 記憶體)，外加 256 GB 磁碟 (大約 235 GB 可用)。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此 Worker 類型僅適用於以下 AWS 區域 3.0 或更新AWS Glue版本的 Spark ETL 工作：美國東部 (俄亥俄)、美國東部 (維吉尼亞北部)、美國西部 (奧勒岡)、亞太區域 (新加坡)、亞太區域 (雪梨)、亞太區域 (東京)、加拿大 (中部)、歐洲 (法蘭克福)、歐洲 (愛爾蘭) 和歐洲 (斯德哥爾摩)。

- **G.8X** – 選擇這種類型時，您也要提供 工作者數目 的值。每個工作者都會映射到 8 個 DPU (32 個 vCPU、128 GB 記憶體)，外加 512 GB 磁碟 (大約 487 GB 可用)。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此背景工作類型僅適用於 3.0 AWS Glue 版或更新版本的 Spark ETL 工作，與G.4X背景工作者類型支援的相同 AWS 區域。
- **G.025X** – 選擇這種類型時，您也要提供 工作者數目 的值。每個工作者都會映射到 0.25 個 DPU (2 個 vCPU、4 GB 記憶體)，外加 84 GB 磁碟 (大約 34 GB 可用)。我們建議低容量串流任務採用這種工作者類型。此工作者類型僅適用於 AWS Glue 3.0 版串流任務。

我們會根據您執行 ETL 任務所使用的 DPU 數量，以小時費率計費。如需詳細資訊，請參閱 AWS Glue 定價頁面。

對於 AWS Glue 1.0 版或更早版本的任務，當您使用主控台設定任務並指定 Worker type (工作者類型) 為 Standard (標準)，會設定 Maximum capacity (容量上限)，並且 Number of workers (工作者數目) 會變成 Maximum capacity (容量上限) - 1。如果您使用 AWS Command Line Interface (AWS CLI) 或 AWS SDK，您可以指定最大容量參數，或同時指定 Worker 類型和 Worker 數目。

對於 AWS Glue 2.0 版或更新版本的任務，您無法指定容量上限。反之，您必須指定 Worker type (工作者類型) 與 Number of workers (工作者數目)。

語言

ETL 指令碼中的程式碼定義了任務的邏輯。指令碼可在 Python 或 Scala 中編寫。您可以選擇任務執行的指令碼是由 AWS Glue 產生或由您提供。您可以在 Amazon Simple Storage Service (Amazon S3) 中提供指令碼名稱和位置。請確認路徑中沒有跟指令碼目錄名稱相同的檔案。若要進一步了解如何編寫指令碼，請參閱 AWS Glue 編程指南。

要求的工人數

對於大多數工作者類型，您必須指定任務時執行所分配的工作者數目。

任務書籤

指定 AWS Glue 在任務執行時如何處理狀態資訊。您可以讓任務書籤記住之前處理過的資料、更新狀態資訊，或是忽略狀態資訊。如需詳細資訊，請參閱 the section called “使用任務書籤追蹤處理的資料”。

柔性執行

當您使用 AWS Studio 或 API 設定工作時，您可以指定標準或彈性的工作執行類別。您的任務可能具有不同程度的優先順序和時間敏感度。標準執行類別非常適合需要快速的任務啟動和專用資源的時間敏感型工作負載。

彈性執行類別適用於非緊急任務，例如生產前任務、測試和一次性資料載入。使用 AWS Glue 3.0 或更高版本和 G.1X 或 G.2X 工作者類型的任務支援彈性任務執行。

彈性任務執行基於在任何時間點執行的工作者數目計費。針對執行中的彈性任務執行，可新增或移除工作者數目。計費方式不是簡單的 Max Capacity * Execution Time，每個工作者都會計入任務執行期間的時間。計費為 (Number of DPUs per worker * time each worker ran) 的總和。

如需詳細資訊，請參閱 AWS Studio 中的說明面板，或任務和任務執行。

重試次數

指定 AWS Glue 若失敗時應自動重新啟動任務的次數 (從 0 到 10)。達到逾時限制的任務不會重新啟動。

任務逾時

設定執行時間上限 (單位為分鐘)。批次任務的預設值為 2880 分鐘 (48 小時)。當任務執行時間超過此上限時，任務執行狀態會變更為 TIMEOUT。

串流工作的逾時值必須小於 7 天或 10080 分鐘。當值保留空白時，如果您尚未設定維護時段，工作將在 7 天後重新啟動。如果您已設定維護時段，則維護時段將在 7 天後重新啟動。

> **工作逾時的最佳做法**
>
> 工作會根據執行時間計費。為避免未預期的費用，請針對工作的預期執行時間設定適當的逾時值。

進階屬性

腳本文件名

工作的唯一指令碼名稱。無法命名為 「無標題」工作。

腳本路徑

指令碼的 Amazon S3 位置。該路徑的格式必須是 s3://bucket/prefix/path/。它必須以斜線 (/) 結尾，且不包含任何檔案。

任務指標

執行此任務時，開啟或關閉 Amazon CloudWatch 指標的建立。若要檢視分析資料，您必須啟用此選項。如需如何開啟和視覺化呈現指標的詳細資訊，請參閱任務監控與偵錯。

Job 觀察度量

執行此工作時，開啟其他可觀察 CloudWatch 度量的建立。如需詳細資訊，請參閱 the section called “使用 AWS Glue 可觀測性指標進行監控”。

連續記錄

開啟連續日誌記錄到 Amazon CloudWatch。如果未啟用此選項，則只有在任務完成後才能使用日誌。如需詳細資訊，請參閱 the section called “持續記錄 AWS Glue 任務”。

Spark UI

啟用 Spark UI 來監控此任務。如需詳細資訊，請參閱 為 AWS Glue 任務啟用 Apache Spark web UI。

星火 UI 日誌路徑

啟用 Spark UI 時寫入日誌的路徑。

星火 UI 日誌記錄和監控配置

請選擇下列其中一個選項：

- 標準：使用 AWS Glue 作業執行 ID 作為檔案名稱來寫入記錄。開啟主控台中的星火使用者介面監 AWS Glue 控。
- 舊版：使用「火花應用程序-{時間戳}」作為文件名寫入日誌。不要打開星火 UI 監控。
- 標準和舊版：將記錄檔寫入標準和舊版位置。開啟主控台中的星火使用者介面監 AWS Glue 控。

最大並行數量

設定此任務允許並行執行的最大數量。預設為 1。達到此閾值時，會傳回錯誤。可指定的最大值由服務限制來控制。例如，新的執行個體啟動時，如果有先前的任務仍在執行，您可能會想要傳回錯誤，以防相同任務的兩個執行個體同時執行。

暫時路徑

提供 Amazon S3 裡的任務目錄位置，可在 AWS Glue 執行指令碼時寫入暫時的中繼結果。請確認路徑中沒有跟暫時目錄名稱相同的檔案。當 AWS Glue 讀取和寫入至 Amazon Redshift 並使用特定的 AWS Glue 轉換時，會使用到此目錄。

> **Note**
>
> 如果區域中尚未存在儲存貯體，則 AWS Glue Glue 會為任務建立暫時儲存貯體。此儲存貯體可能允許公開存取。您可以修改 Amazon S3 中的儲存貯體以設定的公有存取封鎖，或稍後在該區域的所有任務完成後刪除儲存貯體。

延遲通知閾值 (分鐘)

設定傳送延遲通知之前所要經過的時間閾值 (以分鐘為單位)。您可以設定這個閾值，以在 RUNNING、STARTING 或 STOPPING 任務執行所花的時間超過預期分鐘數時傳送通知。

安全組態

從清單中選擇安全組態。安全設定指定在 Amazon S3 目標上如何加密資料：不加密、使用 AWS KMS受管金鑰 (SSE-KMS) 或 Amazon S3 管理的加密金鑰 (SSE-S3) 進行伺服器端加密。

伺服器端加密

如果您選擇此選項，則 ETL 任務寫入 Amazon S3 時，資料會在靜態時加密 (使用 SSE-S3 加密)。您的 Amazon S3 資料目標和寫入 Amazon S3 暫時目錄的任何資料都會加密。系統會以任務參數的形式傳遞此選項。如需詳細資訊，請參閱 Amazon Simple Storage Service 使用者指南中的透過 Amazon S3 受管加密金鑰 (SSE-S3) 使用伺服器端加密來保護資料。

> **Important**
>
> 如果指定安全組態，則會忽略此選項。

使用 Glue 資料目錄做為 Hive 中繼存放區

選取以使用 AWS Glue Data Catalog 做為 Hive 中繼存放區。此任務所用的 IAM 角色必須具有 `glue:CreateDatabase` 許可。稱為「預設」的資料庫隨即在 Data Catalog 中建立 (如當時該資料庫尚未存在)。

連線

選擇虛擬私有雲端組態以存取位於虛擬私有雲 (VPC) 中的 Amazon S3 資料來源。您可以在中建立和管理網路連線 AWS Glue。如需詳細資訊，請參閱 連線至資料。

Libraries (程式庫)

Python 庫路徑，從屬 JAR 路徑和引用的文件路徑

如果您的指令碼需要這些選項，請進行指定。定義此任務時，您可以為這些選項定義以逗號分隔的 Amazon S3 路徑。執行任務時，您可以覆寫這些路徑。如需詳細資訊，請參閱 提供您的自訂指令碼。

任務參數

一組金鑰/值對，會以具名參數的形式傳遞至指令碼。這些是指令碼執行時使用的預設值，但您可以在觸發時或在執行任務階段覆寫。您必須在索引鍵名稱前加上 `--`；例如：`--myKey`。使用時，您可以將工作參數作為地圖傳遞 AWS Command Line Interface。

如需範例，請參閱 在 AWS Glue 中傳遞和存取 Python 參數 的 Python 參數。

標籤

使用 Tag key (標籤金鑰) 和選用的 Tag value (標籤值)，標記您的任務。標籤鍵在建立後，將是唯讀狀態。應用標籤至某些資源，有助於您對其進行整理和識別。如需詳細資訊，請參閱 AWS 中的標籤 AWS Glue。

存取 Lake Formation 受管資料表任務的限制

建立讀取或寫入由所管理之資料表的工作時，請記住下列注意事項和限制 AWS Lake Formation：

- 使用儲存格層級篩選條件存取資料表的任務不支援下列功能：
  - 任務書籤和限制執行
  - Push-down 述詞
  - 伺服器端類別目錄分割述詞
  - enableUpdateCatalog

編輯 AWS Glue 主控台中的 Spark 指令碼

腳本包含從源中提取數據，對其進行轉換並將其加載到目標的代碼。 AWS Glue 啟動工作時執行指令碼。

AWS Glue ETL 指令碼可以在 Python 或 Scala 中進行程式碼編寫。Python 腳本使用一種語言，該語言是 PySpark Python 方言的擴展，用於提取，轉換和加載（ETL）作業。此指令碼包含了延伸的架構，來處理 ETL 的轉換作業。當您針對任務自動產生原始程式碼邏輯時，指令碼也會隨之產生。您可以編輯這個指令碼，也可以提供自己的指令碼來處理您的 ETL 任務。

如需有關在 AWS Glue 中定義和編輯指令碼的資訊，請參閱 AWS Glue 編程指南。

其他程式庫或檔案

如果指令碼需要其他程式庫或檔案，您可加以指定，如下所示：

Python library path (Python 程式庫路徑)

以逗號分隔的 Amazon Simple Storage Service (Amazon S3) 路徑，連接至您指令碼所需的 Python 程式庫。

> **Note**
>
> 只有純 Python 程式庫才可使用。目前尚未支援使用 C 延伸模組的程式庫 (例如 pandas Python 資料分析程式庫)。

Dependent jars path (相依的 jar 路徑)

以逗號分隔的 Amazon S3 路徑，連接至指令碼所需的 JAR 檔案。

> **Note**
>
> 目前只有純 Java 或 Scala (2.11) 程式庫才可使用。

參考檔案路徑

以逗號分隔的 Amazon S3 路徑，連接至指令碼所需的其他檔案 (例如組態檔案)。

任務 (舊版)

指令碼包含執行擷取、轉換和載入 (ETL) 任務的程式碼。您可以提供自己的指令碼，或者 AWS Glue 可依照您的指引產生指令碼。如需編寫自己專用指令碼的詳細資訊，請參閱提供您的自訂指令碼。

您可以在 AWS Glue 主控台編輯指令碼。編輯指令碼時，您可以新增來源、目標和轉換。

編輯指令碼

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。接著選擇 Jobs (任務) 索引標籤。

2. 在清單中選擇任務，然後選擇 Action, Edit script (動作，編輯指令碼) 以開啟指令碼編輯器。

   您也可以從任務詳細資訊頁面存取指令碼編輯器。選擇 Script (指令碼) 索引標籤，接著選擇 Edit script (編輯指令碼)。

指令碼編輯器

AWS Glue 指令碼編輯器可讓您插入、修改、刪除指令碼中的來源、目標和轉換。指令碼編輯器會顯示指令碼以及圖表，協助您將資料流程視覺化。

要建立指令碼的圖表，請選擇 Generate diagram (產生圖表)。AWS Glue 在指令碼中使用以 ## 開頭的註釋行來呈現圖表。要在圖表中正確展示指令碼，您必須讓註釋中的參數和 Apache Spark 程式碼中的參數保持同步。

無論您的游標置於指令碼中何處，指令碼編輯器都能讓您新增程式碼範本。在編輯器最上方，從下列選項中選擇：

- 若要新增來源資料表到指令碼，請選擇 Source (來源)。
- 若要新增目標到指令碼，請選擇 Target (目標)。
- 若要新增目標位置到指令碼，請選擇 Target location (目標位置)。
- 若要新增轉換到指令碼，請選擇 Transform (轉換)。如需在指令碼中呼叫函數的詳細資訊，請參閱程式 AWS Glue ETL 指令碼 PySpark。
- 若要新增 Spigot 轉換到指令碼，請選擇 Spigot。

在插入的程式碼中，修改註釋和 Apache Spark 程式碼中的 `parameters`。例如您新增了 Spigot 轉換，請確認 `path` 在 `@args` 註釋和 `output` 程式碼中都已被取代。

Logs (日誌) 索引標籤會顯示任務執行時與其關聯的日誌。顯示範圍為最近的 1,000 行。

Schema (結構描述) 索引標籤會顯示在 Data Catalog 中所選來源和目標的結構描述。

使用任務書籤追蹤處理的資料

AWS Glue 會持續保留前次執行 ETL 任務的狀態資訊，以追蹤在該任務執行期間所處理的資料。此保存狀態資訊稱為任務書籤。任務書籤可協助 AWS Glue 維護狀態資訊，以及防止重新處理舊資料。透過任務書籤，您可以在依排定間隔重新執行時處理新資料。任務書籤包括各種任務元素的狀態，例如來源、轉換和目標。例如，ETL 任務可能會讀取 Amazon S3 檔案中的新分割區。AWS Glue 會追蹤已由任務成功處理的分割區，避免重複處理以及任務目標資料存放區中出現重複的資料。

任務書籤實作於 JDBC 資料來源、Relationalize 轉換和部分 Amazon Simple Storage Service (Amazon S3) 來源。下表列出 AWS Glue 針對任務書籤支援的 Amazon S3 來源格式。

AWS Glue 版本	Amazon S3 來源格式
0.9 版	JSON、CSV、Apache Avro、XML
1.0 版和更新版本	JSON、CSV、Apache Avro、XML、Parquet、ORC

如需 AWS Glue 版本的詳細資訊，請參閱定義 Spark 任務的任務屬性。

任務書籤功能在透過 AWS Glue 指令碼存取時具有其他功能。瀏覽產生的指令碼時，您可能會看到與此功能相關的轉換內容。如需詳細資訊，請參閱 the section called “使用任務書籤”。

主題

- 在 AWS Glue 中使用任務書籤
- 任務書籤功能的操作詳細資訊

在 AWS Glue 中使用任務書籤

當任務開始之後，會將任務書籤選項當成參數傳遞。下表說明在 AWS Glue 主控台中設定任務書籤的選項。

任務書籤	描述
Enable	讓任務在執行之後更新狀態，以追蹤先前處理的資料。如果您任務的來源具有任務書籤支援，則會追蹤已處理的資料，而且在任務執行時，會處理最後一個檢查點之後的新資料。
停用	不會使用任務書籤，而且任務一律會處理整個資料集。您需負責管理之前任務執行的輸出。此為預設值。
暫停	處理上次成功執行後的遞增資料，或由下列子選項識別範圍內的資料，而不更新最後一個書籤的狀態。您需負責管理之前任務執行的輸出。這兩個子選項是：

任務書籤	描述
	• job-bookmark-from <from-value> 是執行 ID，它表示所有被處理，直到最後一次成功執行之前，包括指定的執行 ID 的所有輸入。對應的輸入會被忽略。 • job-bookmark-to <to-value> 是執行 ID，它表示所有被處理，直到最後一次成功執行之前，包括指定的執行 ID 的輸入。由 <from-value> 識別的輸入以外的對應的輸入由該任務處理。此輸入後的任何輸入也會排除而不進行處理。 指定此選項組時，不會更新任務書籤狀態。 子選項是可選的，但是當使用時，必須提供這兩個子選項。

有關在命令列傳遞至任務的參數 (特別是任務標籤) 的詳細資訊，請參閱 AWS Glue 任務參數。

若是 Amazon S3 輸入來源，AWS Glue 任務書籤會檢查物件的上次修改時間，以確認需要重新處理哪些物件。如果自上次任務執行之後已修改您的輸入來源資料，則在您重新執行任務時重新處理檔案。

如為 JDBC 來源，則適用的規定如下：

- AWS Glue 會針對每個資料表使用一個或多個欄做為書籤索引鍵，以決定新資料和已處理的資料。書籤索引鍵結合在一起，形成單一複合索引鍵。
- 依預設，AWS Glue 會使用主索引鍵作為書籤索引鍵，但條件是其依序遞增或遞減 (沒有間隙)。
- 您可以在 AWS Glue 指令碼中指定要作為書籤索引鍵使用的資料欄。如需有關在 AWS Glue 指令碼中使用任務書籤的詳細資訊，請參閱 the section called “使用任務書籤”。
- AWS Glue 不支援使用名稱區分大小寫的資料欄作為任務書籤索引鍵。

您可以將 AWS Glue Spark ETL 任務的任務書籤倒轉至任何之前的任務執行。現在，您可以透過將任務書籤倒轉至任何之前的任務執行，更好地支援資料回填方案，從而在後續的任務執行中，僅重新處理已加入書籤的任務執行中的資料。

如果您想要使用相同的任務來重新處理所有資料，則請重設任務書籤。若要重設任務書籤狀態，請使用 AWS Glue 主控台、ResetJobBookmark 行動（Python：重置工作書籤）API 操作或 AWS CLI。例如，使用 AWS CLI 輸入下列命令：

```
    aws glue reset-job-bookmark --job-name my-job-name
```

當您倒轉或重設書籤時，AWS Glue 不會清除目標檔案，因為可能有多個目標，且未使用任務書籤追蹤目標。只有來源檔案會使用任務書籤來追蹤。您可以在倒轉和重新處理來源檔案時建立不同的輸出目標，以避免輸出中的資料重複。

AWS Glue 會依任務來追蹤任務書籤。如果您刪除任務，任務書籤將不會保留。

在某些情況下，您可能已啟用 AWS Glue 任務書籤，但 ETL 任務正在重新處理已在稍早執行中處理的資料。如需解決此錯誤常見原因的資訊，請參閱疑難排解 Spark 中 AWS Glue 的錯誤。

任務書籤功能的操作詳細資訊

本節說明如何使用任務書籤的更多操作詳細資訊。

任務書籤會存放任務的狀態。都會依任務名稱和版本號碼輸入狀態的每個執行個體。指令碼呼叫 `job.init` 時，會擷取其狀態，而且一律會取得最新版本。在狀態內，有多個狀態元素，即指令碼中每個來源、轉換和目的地執行個體所特有的。這些狀態元素是依連接到指令碼中對應元素 (來源、轉換或目的地) 的轉換內容所識別。從使用者指令碼呼叫 `job.commit` 時，會自動儲存狀態元素。指令碼會從引數取得任務名稱以及任務書籤的控制選項。

任務書籤中的狀態元素是來源、轉換或目的地特定的資料。例如，假設您想要從上游任務或程序持續寫入的 Amazon S3 位置讀取遞增資料。在此情況下，指令碼必須判斷到目前為止已處理哪些項目。Amazon S3 來源的任務書籤實作會儲存資訊，因此，在重新執行任務時，只能使用儲存的資訊來篩選新物件，以及重新計算下次執行任務的狀態。時間戳記用來篩選新的檔案。

除了狀態元素之外，任務書籤還會有執行次數、嘗試次數和版本號碼。執行次數會追蹤任務的執行，而嘗試次數會記錄任務執行的嘗試。任務執行次數是每次成功執行所遞增的依序增加數目。嘗試次數會追蹤每次執行的嘗試，而且只有在失敗嘗試之後執行時才會遞增。版本號碼會依序遞增，並追蹤任務書籤的更新。

在 AWS Glue 服務資料庫中，所有轉換的書籤狀態將作為鍵值對一起存放：

```
{
  "job_name" : ...,
  "run_id": ...,
  "run_number": ..,
  "attempt_number": ...
  "states": {
    "transformation_ctx1" : {
      bookmark_state1
```

```
    },
    "transformation_ctx2" : {
      bookmark_state2
    }
  }
}
```

最佳實務

下列是使用任務書籤的最佳實務。

- 不要在已啟用書籤的情況下更改資料來源屬性。例如，有一個資料來源 0 指向 Amazon S3 輸入路徑 A，且該任務啟用了書籤並從已執行了好幾輪的來源進行讀取。如果您將資料來源 0 的輸入路徑更改為 Amazon S3 路徑 B，而不更改 `transformation_ctx`，則 AWS Glue 任務將使用儲存的舊書籤狀態。這將導致輸入路徑 B 中的檔案遺失或跳過，因為 AWS Glue 會假定這些檔案已在先前的執行中處理過。
- 使用帶有書籤的目錄表，以便更好地管理分割區。書籤既適用於 Data Catalog 中的資料來源，也適用於來自選項的資料來源。但是，使用來自選項的方法會難以刪除/新增分割區。將目錄表與爬蟲程式結合使用可以提供更好的自動化來追蹤新增的分割區，並可讓您靈活地使用 pushdown 述詞選擇特定的分割區。
- 將 AWS Glue Amazon S3 檔案清單建立工具用於大型資料集。書籤將列出每個輸入分割區下的所有檔案並執行篩選，因此如果單個分割區下的檔案太多，書籤可能會遇到驅動程式 OOM 的問題。使用 AWS Glue Amazon S3 檔案清單建立工具，避免一次性列出記憶體中的所有檔案。

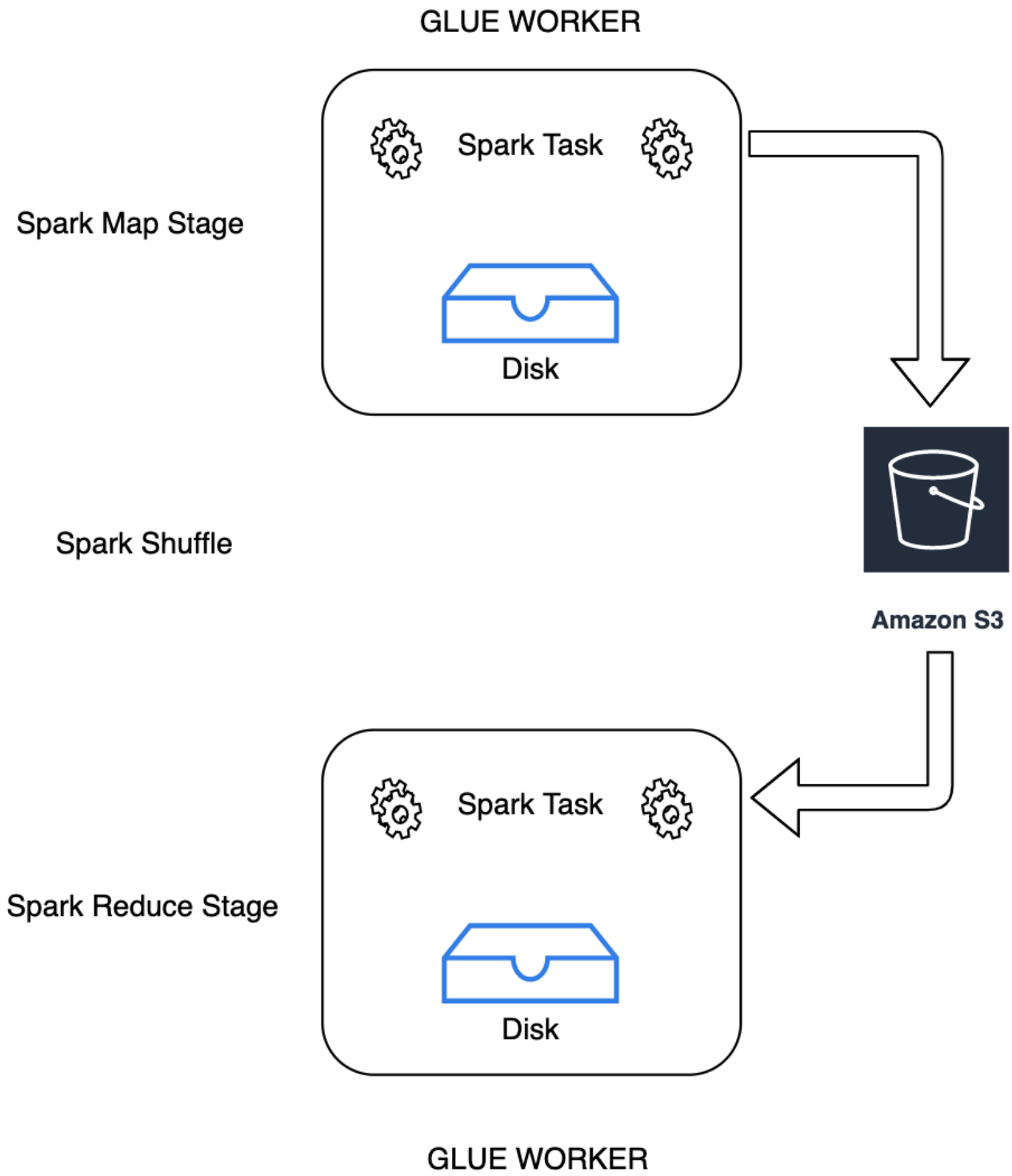
AWS Glue Spark 隨機排序外掛程式與 Amazon S3

隨機排序是每當資料分割區之間重新排列 Spark 任務的重要步驟。這是必需的，因為諸如 `join`、`groupByKey`、`reduceByKey` 以及 `repartition` 等廣泛轉換需要來自其他分割區的資訊才能完成處理。Spark 會從每個分割區收集所需的資料，並將其結合成一個新的分割區。在隨機排序期間，資料會寫入磁碟並透過網路傳輸。因此，隨機排序操作與本機磁碟容量密切相關。當執行器沒有足夠的磁碟空間並且沒有復原時，Spark 擲出 `No space left on device` 或 `MetadataFetchFailedException` 錯誤。

> **Note**
>
> 搭配 Amazon S3 的 AWS Glue Spark 隨機排序外掛程式僅對於 AWS Glue ETL 任務受支援。

解決方案

透過 AWS Glue，您現在可以使用 Amazon S3 來存放 Spark 隨機排序資料。Amazon S3 是一項物件儲存服務，提供領先業界的可擴展性、資料可用性、安全性和效能。此解決方案可為您的 Spark 任務分解運算和儲存，並提供完整的彈性和低成本的隨機排序儲存，讓您可靠地執行最密集的隨機排序工作負載。



我們正推出新的 Apache Spark 雲端隨機排序儲存外掛程式，以便您有效地使用 Amazon S3。如果已知任務受到大型隨機操作的本機磁碟容量限制，您可以開啟 Amazon S3 隨機排序來可靠地執行您的 AWS Glue 任務，而不會失敗。在某些情況下，如果您有大量的小分割區或隨機排序檔案已寫入 Amazon S3，則隨機排序到 Amazon S3 會比本機磁碟 (或 EBS) 稍慢一些。

使用雲端隨機排序儲存外掛程式的先決條件

若要將雲端隨機排序儲存外掛程式與 AWS Glue ETL 任務搭配使用，您需要下列項目：

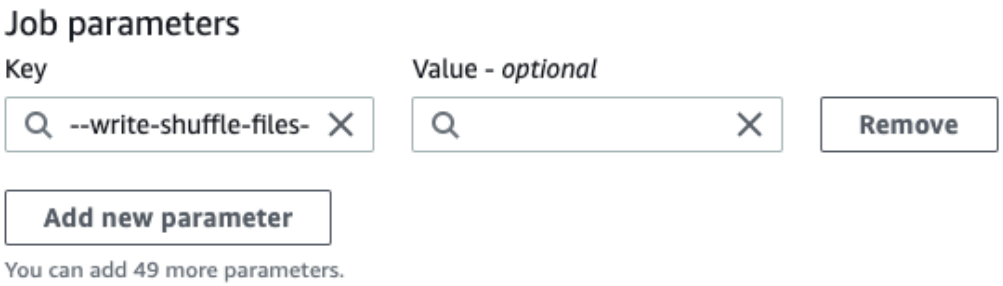
- 與任務執行位於相同區域的 Amazon S3 儲存貯體，用於存放中繼隨機排序和溢出的資料。您可以使用 `--conf spark.shuffle.glue.s3ShuffleBucket=s3://shuffle-bucket/prefix/` 指定隨機排序儲存的 Amazon S3 字首，如下列範例所示：

```
--conf spark.shuffle.glue.s3ShuffleBucket=s3://glue-shuffle-123456789-us-east-1/glue-shuffle-data/
```

- 在字首上設定 Amazon S3 儲存生命週期政策 (如 `glue-shuffle-data`)，因為隨機排序管理器在任務完成後不會清除檔案。任務完成後，應刪除中繼隨機排序和溢出的資料。使用者可以在字首上設定簡短的生命週期政策。如需設定 Amazon S3 生命週期政策的說明，請參閱《Amazon Simple Storage Service 使用者指南》中的設定儲存貯體的生命週期組態。

使用 AWS 主控台的 AWS Glue Spark 隨機排序管理器

若要在設定任務時，使用 AWS Glue 主控台或 AWS Glue Studio 設定 AWS Glue Spark 隨機排序管理器：選擇 --write-shuffle-files-to-s3 任務參數以開啟該任務的 Amazon S3 隨機排序。



使用 AWS Glue Spark 隨機排序外掛程式

下列任務參數會開啟並微調 AWS Glue 隨機排序管理器。這些參數是旗標，因此不會考慮提供的任何值。

- `--write-shuffle-files-to-s3`：主要旗標，啟用 AWS Glue Spark 隨機排序管理器以使用 Amazon S3 儲存貯體來寫入和讀取隨機排序資料。當未指定旗標時不使用隨機排序管理器。
- `--write-shuffle-spills-to-s3` – (僅對 AWS Glue 2.0 版本提供支援)。此為選用旗標，可讓您將溢出檔案卸載到 Amazon S3 儲存貯體，這可為 Spark 任務提供額外的彈性。只有將大量資料溢出到磁碟的大型工作負載才需要它。如果未指定旗標，則不會寫入中繼溢出檔案。

- `--conf spark.shuffle.glue.s3ShuffleBucket=s3://<shuffle-bucket>` — 另一個可選標記，它指定您在其中寫入隨機排序檔案的 Amazon S3 儲存貯體。依預設，`--TempDir/隨機排序資料`。AWS Glue 3.0+ 支援透過使用逗號分隔符號指定儲存貯體，將隨機排序檔案寫入多個儲存貯體，如 `--conf spark.shuffle.glue.s3ShuffleBucket=s3://shuffle-bucket-1/prefix,s3://shuffle-bucket-2/prefix/` 中所示。使用多個儲存貯體可改善效能。

您需要提供安全組態設定，才能為隨機資料啟用靜態加密。如需安全組態的詳細資訊，請參閱 the section called “設定加密”。AWS Glue 支援 Spark 提供的所有其他隨機顯示相關組態。

雲端隨機排序儲存外掛程式的軟體二進位檔

您也可以依據 Apache 2.0 許可證下載 Apache Spark 雲端隨機排序儲存外掛程式的軟體二進位檔，並在任何 Spark 環境中執行該二進位檔。這個新的外掛程式隨附對 Amazon S3 的立即支援，也可以輕鬆設定為使用其他形式的雲端儲存，例如 Google Cloud Storage 和 Microsoft Azure Blob 儲存體。如需詳細資訊，請參閱 Apache Spark 雲端隨機排序儲存外掛程式。

備註與限制

以下是 AWS Glue 隨機排序管理器的備註或限制：

- 任務完成後，AWS Glue 隨機排序管理員不會自動刪除存放在 Amazon S3 儲存貯體中的 (暫時) 隨機排序資料檔案。若要確保資料保護，請在啟用雲端隨機排序儲存外掛程式之前遵循 使用雲端隨機排序儲存外掛程式的先決條件 中的說明進行操作。
- 如果資料偏斜，您可以使用此功能。

Apache Spark 雲端隨機排序儲存外掛程式

雲端隨機排序儲存外掛程式是與 `ShuffleDataIO` API 相容的 Apache Spark 外掛程式，允許在雲端儲存系統 (如 Amazon S3) 上儲存隨機排序資料。此外掛程式可以幫助您補充或更換本機磁碟儲存容量，以進行大型隨機排序操作，此類操作通常由 Spark 應用程式中的轉換 (例如 `join`、`reduceByKey`、`groupByKey` 和 `repartition`) 觸發，從而減少無伺服器資料分析任務和管道的常見故障或對價格/效能的負面影響。

AWS Glue

AWS Glue 3.0 和 4.0 版本隨附預先安裝的外掛程式，並可啟用對 Amazon S3 的隨機排序，而無需任何額外的步驟。如需詳細資訊，請參閱 AWS Glue Spark 隨機排序外掛程式與 Amazon S3，以啟用 Spark 應用程式的功能。

其他 Spark 環境

該外掛程式需要在其他 Spark 環境中設定以下 Spark 組態：

- `--conf spark.shuffle.sort.io.plugin.class=com.amazonaws.spark.shuffle.io.cloud.Choppe`
  這會通知 Spark 使用此外掛程式對 IO 進行隨機排序。
- `--conf spark.shuffle.storage.path=s3://`*`bucket-name`*`/`*`shuffle-file-dir`*：隨機排序檔案的存放路徑。

> **Note**
>
> 該外掛程式會覆寫一個 Spark 核心類別。因此，需要在 Spark jar 之前載入外掛程式 jar。如果在 AWS Glue 外部使用此外掛程式，您可以在內部部署 YARN 環境中使用 `userClassPathFirst` 來達成此目的。

將外掛程式與 Spark 應用程式綁定在一起

您可以在本機開發 Spark 應用程式時，透過在 Maven `pom.xml` 中新增外掛程式相依性，將外掛程式與 Spark 應用程式和 Spark 發行版本 (3.1 及以上版本) 綁定在一起。如需有關此外掛程式和 Spark 版本的詳細資訊，請參閱外掛程式版本。

```
<repositories>
   ...
    <repository>
        <id>aws-glue-etl-artifacts</id>
        <url>https://aws-glue-etl-artifacts.s3.amazonaws.com/release/ </url>
    </repository>
</repositories>
...
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>chopper-plugin</artifactId>
    <version>3.1-amzn-LATEST</version>
</dependency>
```

您也可以按下文所示，改為直接從 AWS Glue Maven 成品下載二進位檔案，並將其包含在 Spark 應用程式中。

```
#!/bin/bash
sudo wget -v https://aws-glue-etl-artifacts.s3.amazonaws.com/release/com/amazonaws/
chopper-plugin/3.1-amzn-LATEST/chopper-plugin-3.1-amzn-LATEST.jar -P /usr/lib/spark/
jars/
```

spark-submit 範例

```
spark-submit --deploy-mode cluster \
--conf spark.shuffle.storage.s3.path=s3://<ShuffleBucket>/<shuffle-dir> \
--conf spark.driver.extraClassPath=<Path to plugin jar> \
--conf spark.executor.extraClassPath=<Path to plugin jar> \
--class <your test class name> s3://<ShuffleBucket>/<Your application jar> \
```

選用組態

這些是控制 Amazon S3 隨機排序行為的選用組態值。

- `spark.shuffle.storage.s3.enableServerSideEncryption`：啟用/停用將檔案隨機排序和溢位的 S3 SSE。預設值為 `true`。
- `spark.shuffle.storage.s3.serverSideEncryption.algorithm`：要使用的 SSE 演算法。預設值為 `AES256`。
- `spark.shuffle.storage.s3.serverSideEncryption.kms.key`：啟用 SSE aws:kms 時的 KMS 金鑰 ARN。

除了這些組態之外，您可能需要設定組態，例如 `spark.hadoop.fs.s3.enableServerSideEncryption` 和其他特定於環境的組態，以確保針對您的使用案例套用適當加密。

外掛程式版本

這個外掛程式支援與每個 AWS Glue 版本相關的 Spark 版本。下表顯示外掛程式軟體二進位檔的 Amazon S3 位置的 AWS Glue 版本、Spark 版本和相關外掛程式版本。

AWS Glue 版本	Spark 版本	外掛程式版本	Amazon S3 位置
3.0	3.1	3.1-amzn-LATEST	s3://aws-glue-etl-artifacts/release/com/amazonaws/chopper-

AWS Glue 版本	Spark 版本	外掛程式版本	Amazon S3 位置
			plugin/3.1-amzn-0/chopper-plugin-3.1-amzn-LATEST.jar
4.0	3.3	3.3-amzn-LATEST	s3://aws-glue-etl-artifacts/release/com/amazonaws/chopper-plugin/3.3-amzn-0/chopper-plugin-3.3-amzn-LATEST.jar

授權

此外掛程式的軟體二進位檔依據 Apache 2.0 許可證獲得授權。

監控 AWS Glue Spark 任務

主題

- 可用的星火指標 AWS Glue Studio
- 使用 Apache Spark web UI 監控任務
- 使用 AWS Glue 任務執行見解進行監控
- 使用 Amazon CloudWatch 監控
- 任務監控與偵錯

可用的星火指標 AWS Glue Studio

Metrics (指標) 索引標籤會顯示在任務執行時和分析功能開啟時，所收集到的指標。Spark 任務中會顯示下列圖表：

- ETL 資料移動
- 記憶體使用狀況：驅動程式和執行器

選擇 View additional metrics (檢視其他指標)，來顯示下列的圖表：

- ETL 資料移動

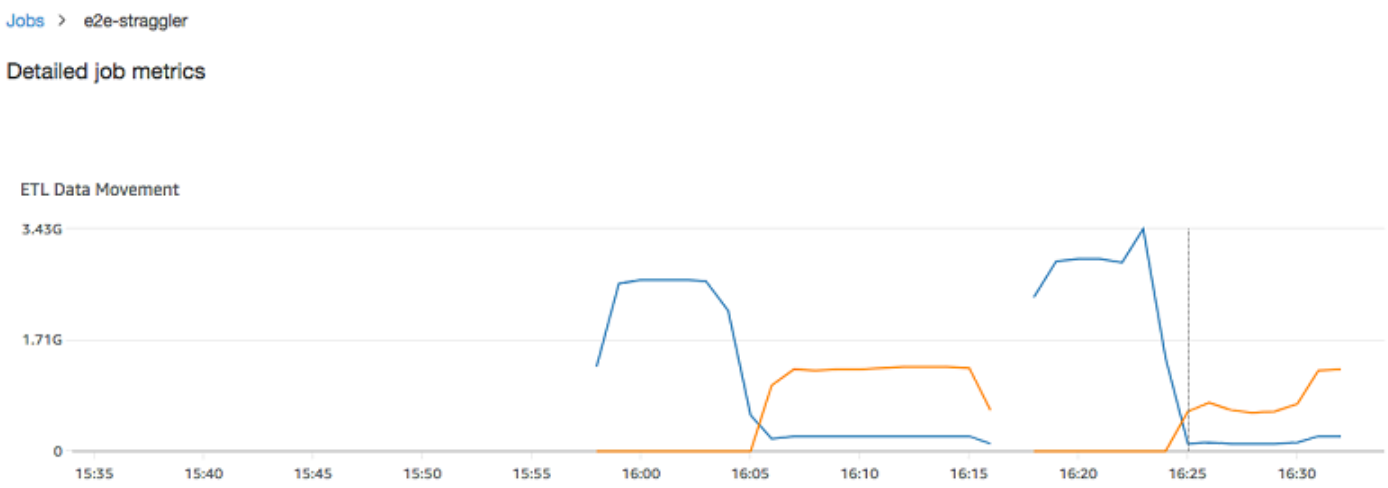
- 記憶體使用狀況：驅動程式和執行器
- 在執行器之間的資料隨機移動
- CPU 負載：驅動程式和執行器
- 任務執行：運作中的執行器、已完成的階段和所需執行器的數量上限

如果工作設定為收集 CloudWatch 測量結果，則會將這些圖形的資料推送至測量結果。如需如何開啟指標和解讀圖表的詳細資訊，請參閱 任務監控與偵錯。

Example ETL 資料移動圖表

ETL 資料移動圖表顯示下列指標：

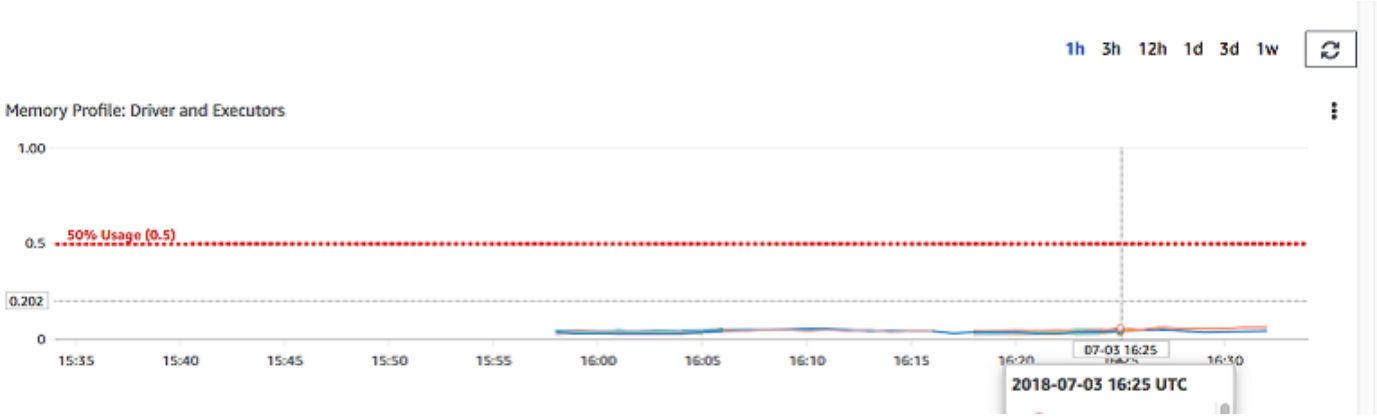
- 所有執行器從 Amazon S3 讀取的位元組數量—glue.ALL.s3.filesystem.read_bytes
- 所有執行器寫入至 Amazon S3 位元組數量—glue.ALL.s3.filesystem.write_bytes



Example 記憶體使用狀況圖表

記憶體使用狀況圖表顯示下列指標：

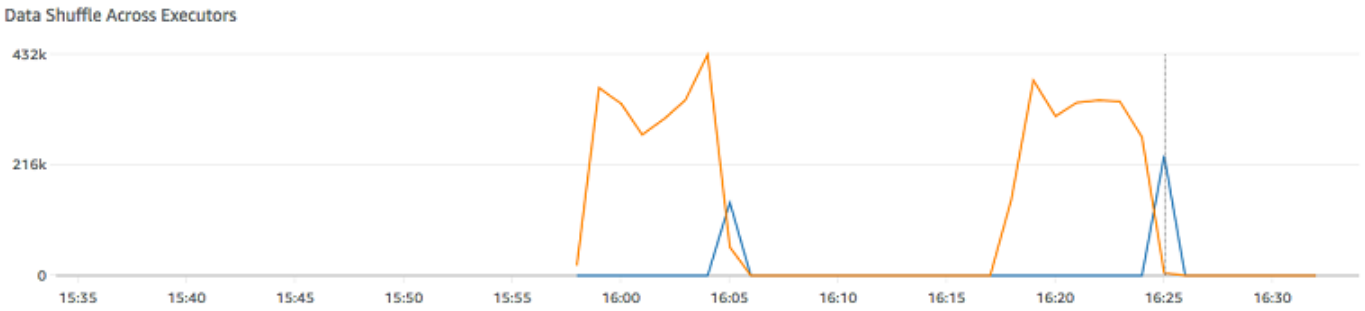
- 此驅動程式的 JVM 堆疊所使用的記憶體佔比 (比例：0-1)，依驅動程式、由 executorId 所辨識的執行器，或所有執行器劃分 —
  - glue.driver.jvm.heap.usage
  - glue.*executorId*.jvm.heap.usage
  - glue.ALL.jvm.heap.usage



Example 執行器之間的資料隨機移動圖表

執行器之間的資料隨機移動圖表顯示下列指標：

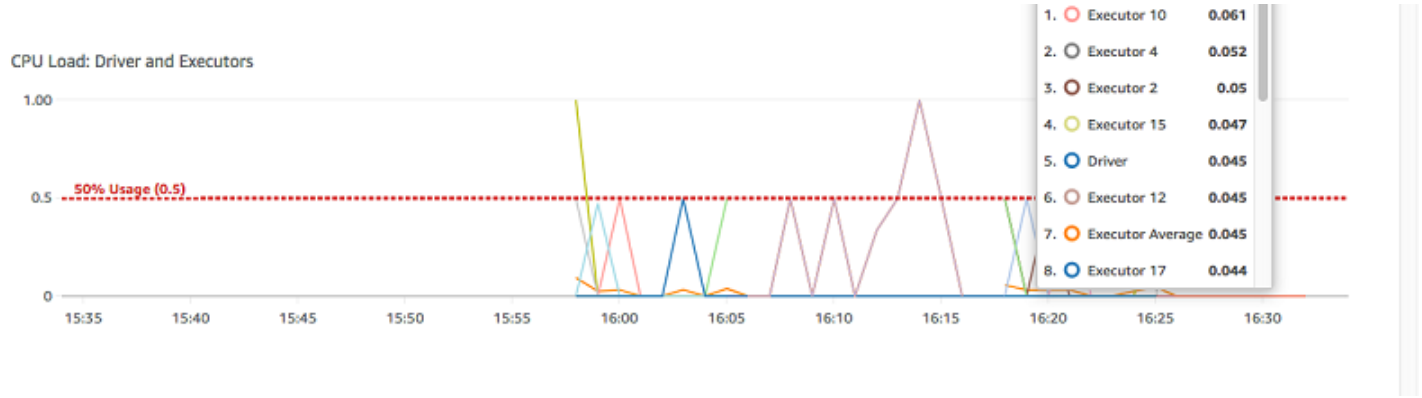
- 所有執行器讀取的位元組數量 (以在這些執行器之間隨機移動資料) — glue.driver.aggregate.shuffleLocalBytesRead
- 所有執行器寫入的位元組數量 (以在這些執行器之間隨機移動資料) — glue.driver.aggregate.shuffleBytesWritten



Example CPU 負載圖表

CPU 負載圖表顯示下列指標：

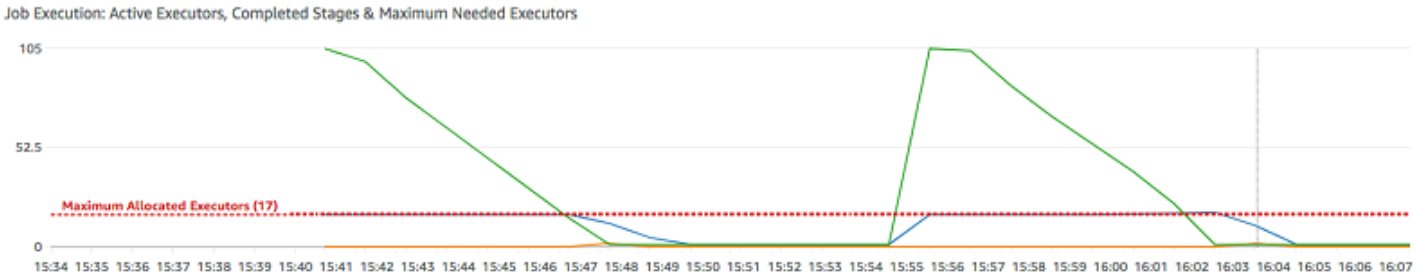
- 驅動程式、由 executorId 所辨識的執行器，或所有執行器使用的 CPU 系統負載佔比 (比例：0-1) —
  - glue.driver.system.cpuSystemLoad
  - glue.*executorId*.system.cpuSystemLoad
  - glue.ALL.system.cpuSystemLoad



Example 任務執行圖表

任務執行圖表顯示下列指標：

- 目前正在運作中執行器的數量 — glue.driver.ExecutorAllocationManager.executors.numberAllExecutors
- 已完成階段的數量 — glue.aggregate.numCompletedStages
- 所需執行器數量的上限 — glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors



使用 Apache Spark web UI 監控任務

您可以使用 Apache Spark Web UI 來監控和偵錯在 AWS Glue 任務系統上執行的 AWS Glue ETL 任務，以及在 AWS Glue 開發端點上執行的 Spark 應用程式。Spark UI 可讓您為每個任務檢查下列項目：

- 每個 Spark 階段的事件時間軸
- 任務的有向無環圖 (DAG)
- SparkSQL 查詢的實體和邏輯計畫

- 每個任務的基礎 Spark 環境變數

如需有關使用 Spark Web UI 的詳細資訊，請參閱《Spark 文件》中的 Web UI。如需如何解譯 Spark UI 結果以改善工作效能的指引，請參閱《 AWS 規範指引》中有關 Apache Spark 工作效能調整 AWS Glue 的最佳做法。

您可以在 AWS Glue 控制台中看到星火 UI。如果 AWS Glue 工作在 AWS Glue 3.0 或更新版本上執行，且記錄檔以標準 (而非舊版) 格式產生，這是較新工作的預設值。如果您的記錄檔大於 0.5 GB，您可以為 AWS Glue 4.0 或更新版本的工作執行啟用滾動記錄支援，以簡化記錄封存、分析和疑難排解。

您可以使用AWS Glue控制台或 AWS Command Line Interface （ AWS CLI ）啟用星火 UI。當您啟用 Spark UI 時，AWS Glue 開發端點上的 AWS Glue ETL 任務和 Spark 應用程式可以將 Spark 事件日誌備份到您在 Amazon Simple Storage Service (Amazon S3) 中指定的位置。可搭配使用 Amazon S3 中已備份的事件日誌與 Spark UI，即可在工作運作時即時使用，也可在工作完成後使用。當日誌保留在 Amazon S3 中時， AWS Glue 控制台中的 Spark UI 可以查看它們。

許可

若要在 AWS Glue 主控台中使用 Spark UI，您可以使用`UseGlueStudio`或新增所有個別服務 API。需要所有 API 才能完全使用 Spark UI，但是使用者可以透過在其 IAM 權限中新增其服務 API 來存取 SparKui 功能，以進行更精細的存取。

`RequestLogParsing`是最關鍵的，因為它執行日誌的解析。其餘的 API 用於讀取相應的解析數據。例如，`GetStages`可讓您存取 Spark 工作所有階段的相關資料。

對應至的 Spark UI 服務 API 清單`UseGlueStudio`如下範例政策。以下政策提供僅使用 Spark UI 功能的存取權。若要新增更多許可，例如 Amazon S3 和 IAM，請參閱建立的自訂 IAM 政策 AWS Glue Studio。

對應至的 Spark UI 服務 API 清`UseGlueStudio`單如下範例政策。使用星火 UI 服務 API 時，請使用以下命名空間：`glue:<ServiceAPI>`。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AllowGlueStudioSparkUI",
   "Effect": "Allow",
            "Action": [
                "glue:RequestLogParsing",
```

```
                "glue:GetLogParsingStatus",
                "glue:GetEnvironment",
                "glue:GetJobs",
                "glue:GetJob",
                "glue:GetStage",
                "glue:GetStages",
                "glue:GetStageFiles",
                "glue:BatchGetStageFiles",
                "glue:GetStageAttempt",
                "glue:GetStageAttemptTaskList",
                "glue:GetStageAttemptTaskSummary",
                "glue:GetExecutors",
                "glue:GetExecutorsThreads",
                "glue:GetStorage",
                "glue:GetStorageUnit",
                "glue:GetQueries",
                "glue:GetQuery"
            ],
    "Resource": [
     "*"
    ]
   }
  ]
}
```

限制

- AWS Glue 主控台中的 Spark UI 不適用於 2023 年 11 月 20 日之前發生的工作執行，因為它們是舊版記錄檔格式。
- AWS Glue 控制台中的 Spark UI 支持 AWS Glue 4.0 的滾動日誌，例如在流式作業中默認生成的日誌。所有產生的捲動記錄事件檔總和上限為 2 GB。對於沒有復原記錄支援的 AWS Glue 工作，SparKui 支援的記錄事件檔案大小上限為 0.5 GB。
- 無伺服器 Spark 使用者介面不適用於只能由您的 VPC 存取的 Amazon S3 儲存貯體中存取的 Spark 事件日誌。

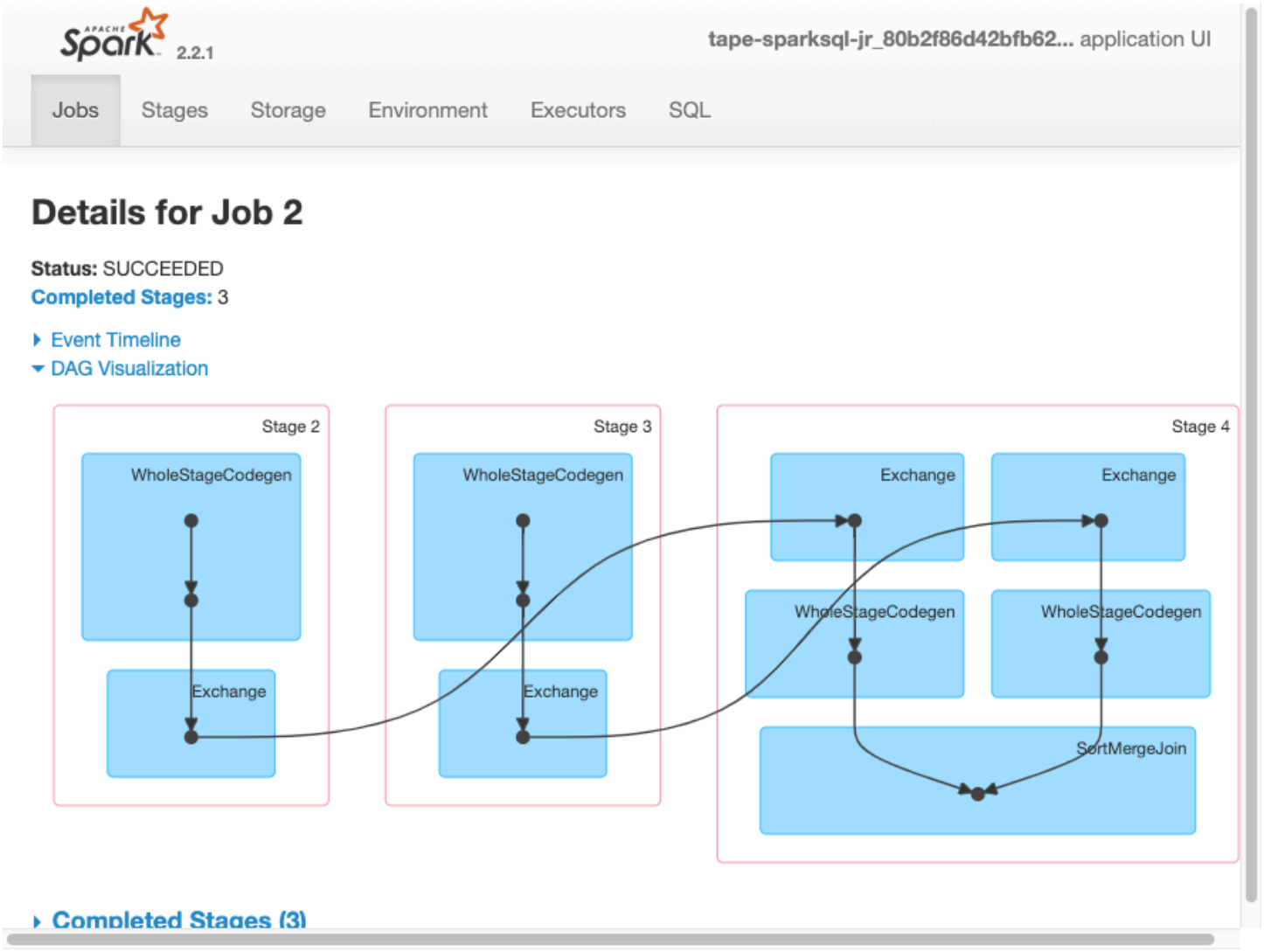
範例：Apache Spark Web UI

此範例顯示如何使用 Spark UI 了解工作效能。螢幕擷取畫面顯示自我管理的 Spark 歷史記錄伺服器提供的 Spark Web UI。在 AWS Glue 控制台星火 UI 提供了類似的意見。如需有關使用 Spark Web UI 的詳細資訊，請參閱《Spark 文件》中的 Web UI。

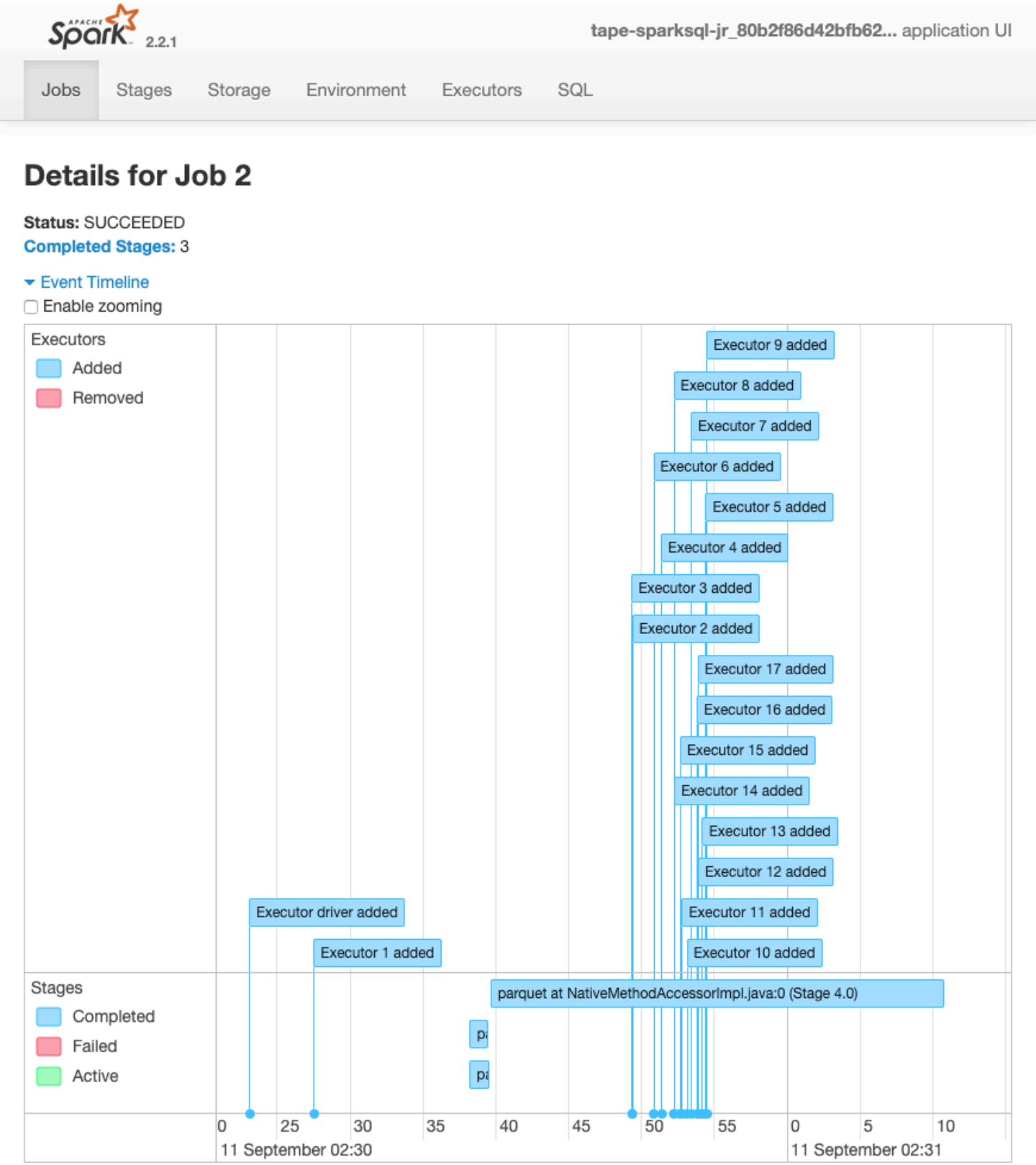
以下是 Spark 應用程式的範例，它會讀取兩個資料來源、執行聯結轉換，然後以 Parquet 格式將其寫入至 Amazon S3。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql.functions import count, when, expr, col, sum, isnull
from pyspark.sql.functions import countDistinct
from awsglue.dynamicframe import DynamicFrame

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

job = Job(glueContext)
job.init(args['JOB_NAME'])

df_persons = spark.read.json("s3://awsglue-datasets/examples/us-legislators/all/
persons.json")
df_memberships = spark.read.json("s3://awsglue-datasets/examples/us-legislators/all/
memberships.json")

df_joined = df_persons.join(df_memberships, df_persons.id == df_memberships.person_id,
 'fullouter')
df_joined.write.parquet("s3://aws-glue-demo-sparkui/output/")

job.commit()
```

下列 DAG 視覺化顯示此 Spark 任務中的不同階段。

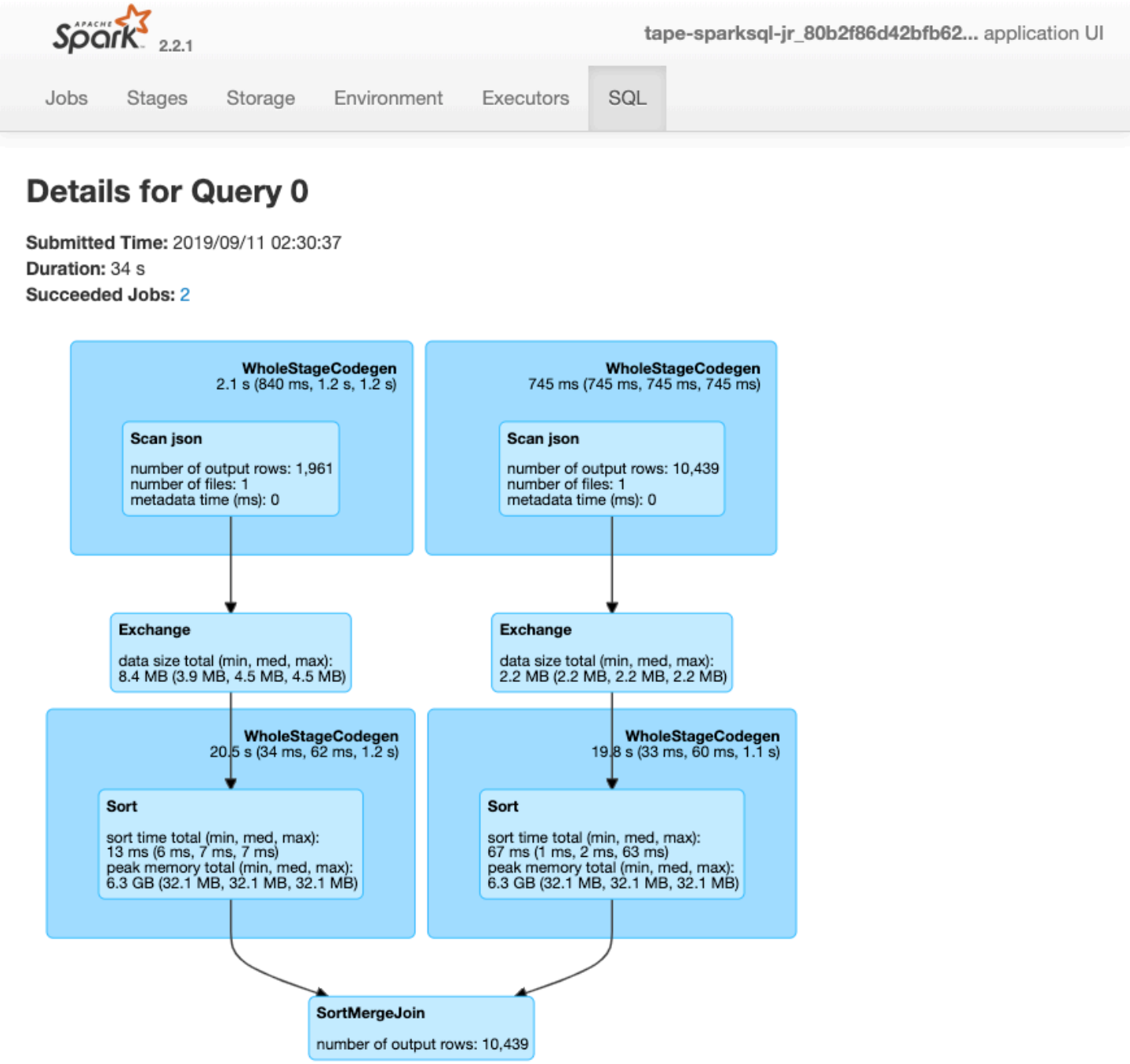


下列任務的事件時間軸顯示不同 Spark 執行器的啟動、執行和終止。


Spark 2.2.1
tape-sparksql-jr_80b2f86d42bfb62... application UI
Jobs
Stages
Storage
Environment
Executors
SQL
Details for Job 2
Status: SUCCEEDED
Completed Stages: 3
Event Timeline
Enable zooming
Executors
Added
Removed
Executor 9 added
Executor 8 added
Executor 7 added
Executor 6 added
Executor 5 added
Executor 4 added
Executor 3 added
Executor 2 added
Executor 17 added
Executor 16 added
Executor 15 added
Executor 14 added
Executor 13 added
Executor 12 added
Executor 11 added
Executor 10 added
Executor driver added
Executor 1 added
Stages
Completed
Failed
Active
parquet at NativeMethodAccessorImpl.java:0 (Stage 4.0)
0
25
30
35
40
45
50
55
0
5
10
11 September 02:30
11 September 02:31
DAG Visualization
Completed Stages (3)

下列畫面顯示 SparkSQL 查詢計畫的詳細資訊：

- 已剖析的邏輯計畫
- 已分析的邏輯計畫
- 已最佳化的邏輯計畫
- 要執行的實體計畫


Spark 2.2.1
tape-sparksql-jr_80b2f86d42bfb62... application UI
Jobs
Stages
Storage
Environment
Executors
SQL
Details for Query 0
Submitted Time: 2019/09/11 02:30:37
Duration: 34 s
Succeeded Jobs: 2
WholeStageCodegen
2.1 s (840 ms, 1.2 s, 1.2 s)
Scan json
number of output rows: 1,961
number of files: 1
metadata time (ms): 0
WholeStageCodegen
745 ms (745 ms, 745 ms, 745 ms)
Scan json
number of output rows: 10,439
number of files: 1
metadata time (ms): 0
Exchange
data size total (min, med, max):
8.4 MB (3.9 MB, 4.5 MB, 4.5 MB)
Exchange
data size total (min, med, max):
2.2 MB (2.2 MB, 2.2 MB, 2.2 MB)
WholeStageCodegen
20.5 s (34 ms, 62 ms, 1.2 s)
Sort
sort time total (min, med, max):
13 ms (6 ms, 7 ms, 7 ms)
peak memory total (min, med, max):
6.3 GB (32.1 MB, 32.1 MB, 32.1 MB)
WholeStageCodegen
19.8 s (33 ms, 60 ms, 1.1 s)
Sort
sort time total (min, med, max):
67 ms (1 ms, 2 ms, 63 ms)
peak memory total (min, med, max):
6.3 GB (32.1 MB, 32.1 MB, 32.1 MB)
SortMergeJoin
number of output rows: 10,439
Details
== Parsed Logical Plan ==
Join FullOuter, (id#14 = person_id#50)
:-
Relation[birth_date#8,contact_details#9,death_date#10,family_name#11,gender#12,given_name#13,id#14,identifiers#15
,image#16,images#17,links#18,name#19,other_names#20,sort_name#21] json
+-
Relation[area_id#45,end_date#46,legislative_period_id#47,on_behalf_of_id#48,organization_id#49,person_id#50,role#
51,start_date#52] json
== Analyzed Logical Plan ==
birth_date: string, contact_details: array<struct<type:string,value:string>>, death_date: string, family_name:
string, gender: string, given_name: string, id: string, identifiers:
array<struct<identifier:string,scheme:string>>, image: string, images: array<struct<url:string>>, links:
array<struct<note:string,url:string>>, name: string, other_names:
array<struct<lang:string,name:string,note:string>>, sort_name: string, area_id: string, end_date: string,
legislative_period_id: string, on_behalf_of_id: string, organization_id: string, person_id: string, role: string,
start_date: string
Join FullOuter, (id#14 = person_id#50)

主題

為 AWS Glue 任務啟用 Apache Spark web UI

您可以使用 Apache Spark Web UI 來監控和偵錯在 AWS Glue 任務系統上執行的 AWS Glue ETL 任務。您可以使用 AWS Glue 主控台或 AWS Command Line Interface (AWS CLI) 來設定 Spark UI。

每 30 秒，AWS Glue 會將 Spark 事件日誌備份至您指定的 Amazon S3 路徑。

主題

設定 Spark UI (主控台)

使用 AWS Management Console，依照以下步驟來設定 Spark UI。建立 AWS Glue 工作時，Spark UI 預設為啟用。

建立或編輯工作時開啟 Spark UI

1. 請登入 AWS Management Console 並開啟AWS Glue主控台，網址為 https://console.aws.amazon.com/glue/。
2. 在導覽窗格中，選擇 Jobs (任務)。
3. 選擇新增任務，或選取現有任務。
4. 在任務詳細資料中，開啟進階屬性。
5. 在 Spark UI 索引標籤下方，選擇將 Spark UI 日誌寫入 Amazon S3。
6. 指定 Amazon S3 路徑以用於存放任務的 Spark 事件日誌。請注意，如果在工作中使用安全組態，加密也會套用至 Spark UI 日誌檔案。如需詳細資訊，請參閱 對 AWS Glue 寫入的資料加密。
7. 在 Spark UI 記錄和監控組態下方：
    - 如果您要產生要在 AWS Glue 主控台中檢視的記錄，請選取 [標準]。

- 如果要產生日誌以在 Spark 歷史記錄伺服器中檢視，請選取舊版。
- 您也可以選擇產生兩者。

設定 Spark UI (AWS CLI)

若要產生使用 Spark UI 檢視的記錄檔，請在 AWS Glue 主控台中使用 AWS CLI 將下列工作參數傳遞至AWS Glue工作。如需詳細資訊，請參閱 the section called “任務參數”。

```
'--enable-spark-ui': 'true',
'--spark-event-logs-path': 's3://s3-event-log-path'
```

若要將日誌分發至其舊版位置，請將 `--enable-spark-ui-legacy-path` 參數設定為 `"true"`。如果您不想要產生這兩種格式的日誌，請移除 `--enable-spark-ui` 參數。

使用筆記本為工作階段設定 Spark UI

> **Warning**
>
> AWS Glue 互動式工作階段目前不支援主控台中的 Spark UI。設定 Spark 歷史記錄伺服器。

如果您使用 AWS Glue 筆記本，請在開始工作階段之前設定 SparKui 設定。為此，請使用 `%%configure` 儲存格魔術命令：

```
%%configure { “--enable-spark-ui”: “true”, “--spark-event-logs-path”: “s3://path” }
```

啟用滾動記錄檔

為 AWS Glue 工作啟用 SparKui 和滾動記錄事件檔可提供數個好處：

- 滾動記錄事件檔案 — 啟用滾動記錄事件檔案後， AWS Glue 會為工作執行的每個步驟產生個別的記錄檔，以便更輕鬆地識別和疑難排解特定階段或轉換的特定問題。
- 更好的日誌管理 — 滾動日誌事件文件有助於更有效地管理日誌文件。記錄會根據工作執行階段，將記錄分割成較小、更容易管理的檔案，而不是具有單一、可能較大的記錄檔。這樣可以簡化記錄封存、分析和疑難排解作業。
- 改善容錯 — 如果 AWS Glue 工作失敗或中斷，滾動記錄事件檔案可以提供有關上一個成功階段的寶貴資訊，讓您可以更輕鬆地從該點繼續工作，而不是從頭開始。

- 成本最佳化 — 藉由啟用滾動記錄事件檔案，您可以節省與記錄檔相關聯的儲存成本。您可以儲存較小、更易於管理的記錄檔，而不是儲存單一、可能較大的記錄檔，這樣可能更具成本效益，特別是對於長時間執行或複雜的工作而言。

在新環境中，使用者可透過下列方式明確啟用滾動記錄：

```
'—conf': 'spark.eventLog.rolling.enabled=true'
```

或

```
'—conf': 'spark.eventLog.rolling.enabled=true —conf
spark.eventLog.rolling.maxFileSize=128m'
```

啟動滾動記錄檔時，請`spark.eventLog.rolling.maxFileSize`指定事件記錄檔在復原之前的大小上限。如果未指定此選用參數的預設值為 128 MB。最小值為 10 MB。

所有產生的捲動記錄事件檔總和上限為 2 GB。對於沒有滾動記錄支援的 AWS Glue 工作，SparKui 支援的記錄事件檔案大小上限為 0.5 GB。

可以透過傳遞其他組態來關閉串流工作的滾動日誌。請注意，非常大的日誌檔案可能需要昂貴的維護成本。

若要關閉滾動日誌，請提供下列組態：

```
'--spark-ui-event-logs-path': 'true',
'--conf': 'spark.eventLog.rolling.enabled=false'
```

啟動 Spark 歷史記錄伺服器

可使用 Spark 歷史記錄伺服器，在您自己的基礎設施中視覺化 Spark 日誌。對於 AWS Glue 4.0 或更新版本中的 AWS Glue 任務執行 (其中包含以標準 (而非舊版) 格式產生的日誌)，您可以在 AWS Glue 主控台中看到相同的視覺效果。如需更多詳細資訊，請參閱 the section called “使用 Spark UI 進行監控”。

您可以使用在 EC2 執行個體上託管伺服器的 AWS CloudFormation 範本啟動 Spark 歷史記錄伺服器，或使用 Docker 在本機啟動。

主題

- 啟動 Spark 歷史記錄伺服器並使用 AWS CloudFormation 檢視 Spark UI

- 啟動 Spark 歷史記錄伺服器並使用 Docker 檢視 Spark UI

啟動 Spark 歷史記錄伺服器並使用 AWS CloudFormation 檢視 Spark UI

您可以使用 AWS CloudFormation 範本來啟動 Apache Spark 歷史記錄伺服器並檢視 Spark Web UI。這些範本是範例，您應加以修改以符合您的需求。

啟動 Spark 歷史記錄伺服器並使用 AWS CloudFormation 檢視 Spark UI

1. 選擇下表中的其中一個 Launch Stack (啟動堆疊) 按鈕。這會在 AWS CloudFormation 主控台上啟動堆疊。

區域	啟動
美國東部 (俄亥俄)	Launch Stack
美國東部 (維吉尼亞北部)	Launch Stack
美國西部 (加利佛尼亞北部)	Launch Stack
美國西部 (奧勒岡)	Launch Stack
非洲 (開普敦)	Launch Stack
亞太區域 (香港)	Launch Stack
亞太區域 (孟買)	Launch Stack
亞太區域 (大阪)	Launch Stack
亞太區域 (首爾)	Launch Stack
亞太區域 (新加坡)	Launch Stack

區域	啟動
亞太區域 (雪梨)	Launch Stack
亞太區域 (東京)	Launch Stack
加拿大 (中部)	Launch Stack
歐洲 (法蘭克福)	Launch Stack
歐洲 (愛爾蘭)	Launch Stack
歐洲 (倫敦)	Launch Stack
Europe (Milan)	Launch Stack
歐洲 (巴黎)	Launch Stack
歐洲 (斯德哥爾摩)	Launch Stack
中東 (巴林)	Launch Stack
南美洲 (聖保羅)	Launch Stack

2. 在 Specify template (指定範本) 頁面上，選擇 Next (下一步)。
3. 在 Specify stack details (指定堆疊詳細資訊) 頁面上，輸入 Stack name (堆疊名稱)。在 Parameters (參數) 下輸入其他資訊。

   a. Spark UI 組態

      請提供下列資訊：

      - IP address range (IP 地址範圍) - 可用於檢視 Spark UI 的 IP 地址範圍。如果您想要限制來自特定 IP 地址範圍的存取，您應該使用自訂值。

- History server port (歷史記錄伺服器連接埠) - Spark UI 的連接埠。您可以使用預設值。
- Event log directory (事件日誌目錄) - 選擇從 AWS Glue 任務或開發端點存放 Spark 事件日誌的位置。您必須使用 **`s3a://`** 做為事件日誌路徑配置。
- Spark package location (Spark 套件位置) - 您可以使用預設值。
- Keystore path (金鑰存放區路徑) - HTTPS 的 SSL/TLS 金鑰存放區路徑。如果您要使用自訂金鑰存放區檔案，可以在此指定 S3 路徑 `s3://path_to_your_keystore_file`。如果您將此參數保留空白，將會產生並使用以自我簽署憑證為基礎的金鑰存放區。
- Keystore password (金鑰存放區密碼) – 輸入 HTTPS 的 SSL/TLS 金鑰存放區密碼。

b. EC2 執行個體組態

請提供下列資訊：

- Instance type (執行個體類型) - 託管 Spark 歷史記錄伺服器的 Amazon EC2 執行個體類型。由於此範本啟動您帳戶中的 Amazon EC2 執行個體，因此會另外向您的帳戶收取 Amazon EC2 費用。
- Latest AMI ID (最新的 AMI ID) - Spark 歷史記錄伺服器執行個體之 Amazon Linux 2 的 AMI ID。您可以使用預設值。
- VPC ID - Spark 歷史記錄伺服器執行個體的 Virtual Private Cloud (VPC) ID。您可以使用您帳戶中可用的任何 VPC，不建議使用預設 VPC 搭配預設網路 ACL。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的預設 VPC 和預設子網路以及建立 VPC。
- Subnet ID (子網路 ID) - Spark 歷史記錄伺服器執行個體的 ID。您可以使用 VPC 中的任何子網路。您必須能夠從用戶端連線到網路以連接至子網路。如果您要透過網際網路存取，必須使用在路由表中具有網際網路閘道的公有子網路。

c. 選擇 Next (下一步)。

4. 在 Configure stack options (設定堆疊選項) 頁面上，若要使用目前使用者憑證來決定 CloudFormation 如何在堆疊中建立、修改或刪除資源，請選擇 Next (下一步)。您也可以在許可區段指定一個角色，而非使用目前使用者的許可，然後選擇下一步。
5. 在 Review (檢閱) 頁面上檢閱範本。

   選擇 I acknowledge that AWS CloudFormation might create IAM resources (我知道 AWS CloudFormation 可能會建立 IAM 資源)，然後選擇 Create stack (建立堆疊)。
6. 等待堆疊建立。
7. 開啟 Outputs (輸出) 標籤。

   a. 如果您使用公有子網路，請複製 SparkUiPublicUrl 的 URL。

b. 如果您使用私有子網路，請複製 SparkUiPrivateUrl 的 URL。

8. 開啟 Web 瀏覽器，然後貼入該 URL。這可讓您使用 HTTPS 在指定連接埠上存取伺服器。您的瀏覽器可能會無法辨識伺服器的憑證，在此情況下，您必須覆寫其保護並繼續。

啟動 Spark 歷史記錄伺服器並使用 Docker 檢視 Spark UI

如果您偏好本機存取 (不使用 EC2 執行個體做為 Apache Spark 歷史記錄伺服器)，您也可以使用 Docker 來啟動 Apache Spark 歷史記錄伺服器，並在本機檢視 Spark UI。此 Dockerfile 為範例，您應該修改以符合您的需求。

先決條件

如需有關如何在筆記型電腦上安裝 Docker 的資訊，請參閱 Docker 引擎社群。

啟動 Spark 歷史記錄伺服器並使用 Docker 在本機檢視 Spark UI

1. 從 GitHub 下載檔案。

   從 AWS Glue 程式碼範例下載 Dockerfile 和 `pom.xml`。

2. 決定是否要使用您的使用者憑證或聯合身分使用者憑證來存取 AWS。

   - 若要使用目前的使用者憑證來存取 AWS，取得用於 `docker run` 命令中 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 的值。如需詳細資訊，請參閱《IAM 使用者指南》中的管理 IAM 使用者的存取金鑰。
   - 若要使用 SAML 2.0 聯合身分使用者來存取 AWS，請取得 `AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY` 和 `AWS_SESSION_TOKEN` 的值。如需詳細資訊，請參閱請求暫時安全憑證。

3. 判斷您的事件日誌目錄的位置，以在 `docker run` 命令中使用。

4. 使用本機目錄中的檔案建置 Docker 映像檔，對其使用名稱 `glue/sparkui` 以及標籤 `latest`。

   ```
   $ docker build -t glue/sparkui:latest .
   ```

5. 建立並啟動 Docker 容器。

   在下列命令中，使用先前在步驟 2 與 3 中取得的值。

   a. 若要使用您的使用者憑證建立 Docker 容器，請使用類似如下的命令。

```
docker run -itd -e SPARK_HISTORY_OPTS="$SPARK_HISTORY_OPTS -
Dspark.history.fs.logDirectory=s3a://path_to_eventlog
 -Dspark.hadoop.fs.s3a.access.key=AWS_ACCESS_KEY_ID -
Dspark.hadoop.fs.s3a.secret.key=AWS_SECRET_ACCESS_KEY"
 -p 18080:18080 glue/sparkui:latest "/opt/spark/bin/spark-class
 org.apache.spark.deploy.history.HistoryServer"
```

b. 要使用暫時憑證建立 Docker 容器，請使用 `org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider` 做為提供者，並提供在步驟 2 中取得的憑證值。如需詳細資訊，請參閱 Hadoop：與 Amazon Web Services 整合文件中的將工作階段憑證與 TemporaryAWSCredentialsProvider 搭配使用。

```
docker run -itd -e SPARK_HISTORY_OPTS="$SPARK_HISTORY_OPTS -
Dspark.history.fs.logDirectory=s3a://path_to_eventlog
 -Dspark.hadoop.fs.s3a.access.key=AWS_ACCESS_KEY_ID -
Dspark.hadoop.fs.s3a.secret.key=AWS_SECRET_ACCESS_KEY
 -Dspark.hadoop.fs.s3a.session.token=AWS_SESSION_TOKEN
 -
Dspark.hadoop.fs.s3a.aws.credentials.provider=org.apache.hadoop.fs.s3a.TemporaryAWSCred
 -p 18080:18080 glue/sparkui:latest "/opt/spark/bin/spark-class
 org.apache.spark.deploy.history.HistoryServer"
```

> **Note**
>
> 這些組態參數來自 Hadoop-AWS 模組。您可能需要根據您的使用案例來新增具體的組態。例如：隔離區域中的使用者需要設定 `spark.hadoop.fs.s3a.endpoint`。

6. 在瀏覽器中開啟 `http://localhost:18080` 以在本機檢視 Spark UI。

使用 AWS Glue 任務執行見解進行監控

AWS Glue工作執行深入解析AWS Glue是簡化工作偵錯和最佳化 AWS Glue 工作的功能。 AWS Glue 提供 Spark UI，以及用於監視 AWS Glue 工作的CloudWatch 日誌和指標。使用此功能，您可以獲得有關 AWS Glue 工作執行的以下信息：

- AWS Glue 任務指令碼中出現故障的行號。
- 在任務失敗之前，Spark 查詢計劃中最近執行的 Spark 動作。
- 與時間排序日誌串流中顯示的故障相關的 Spark 異常事件。

- 根本原因分析和修正問題的建議動作 (例如調校指令碼)。
- 常見的 Spark 事件 (與 Spark 動作相關的日誌訊息)，其中包含解決根本原因的建議動作。

您可以在任AWS Glue務的記錄中使用兩個新的 CloudWatch 記錄資料流來取得所有這些見解。

要求

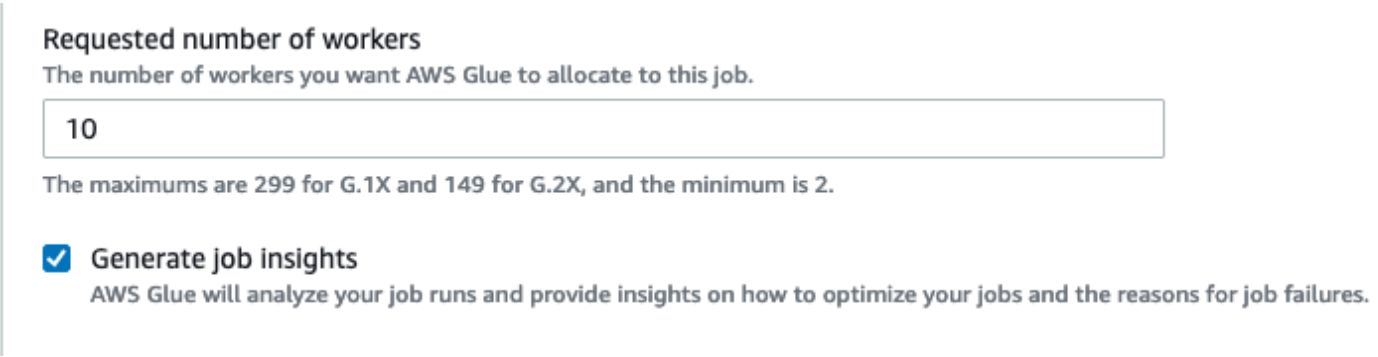
AWS Glue作業執行見解功能適用於 2.0、3.0 和 4.0 AWS Glue 版。您可以按照遷移指南，將現有的任務從舊的 AWS Glue 版本遷移。

啟用 AWS Glue ETL 工作的工作執行深入解析

您可以透過 AWS Glue Studio 或 CLI 啟用任務執行見解。

AWS Glue Studio

透過 AWS Glue Studio 建立任務時，可以在 Job Details (任務詳細資訊) 索引標籤下啟用或停用任務執行見解。檢查是否已選取 [產生工作見解] 方塊。



命令列

如果透過 CLI 建立任務，則可以使用單個新的任務參數啟動任務：`--enable-job-insights = true`。

預設情況下，在 AWS Glue 連續記錄使用的相同預設日誌群組下建立任務執行見解日誌串流，即 `/aws-glue/jobs/logs-v2/`。您可以使用相同的連續記錄引數集來設定自訂日誌群組名稱、日誌篩選條件和日誌群組組態。如需詳細資訊，請參閱啟用 AWS Glue 任務的持續記錄。

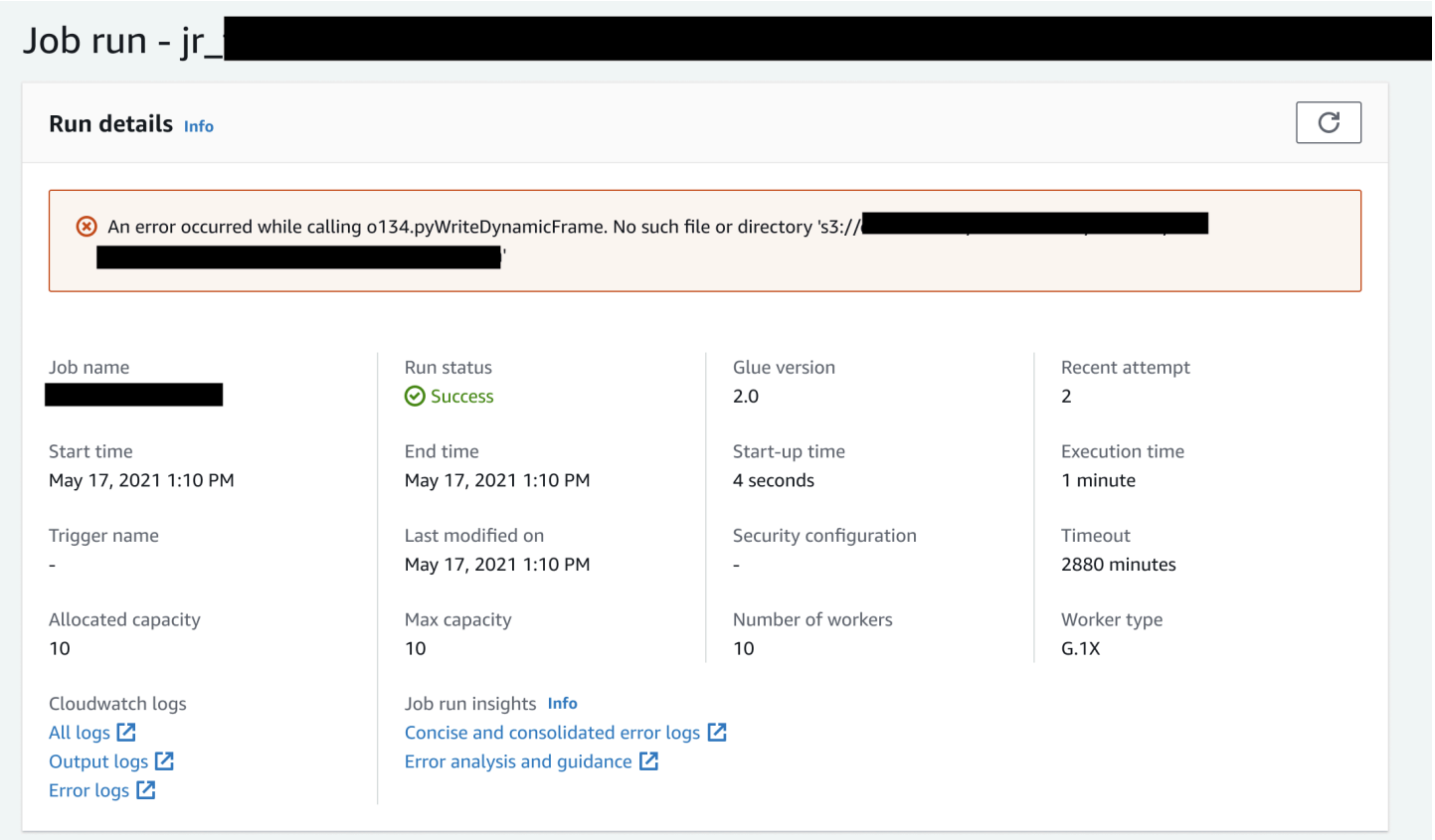
存取工作執行見解記錄資料流 CloudWatch

啟用任務執行見解功能後，當任務執行失敗時，可能會建立兩個日誌串流。任務成功完成後，兩個串流均不會產生。

1. 異常分析日誌串流：<job-run-id>-job-insights-rca-driver。此串流提供下列資訊：
   - AWS Glue 任務指令碼中導致失敗的行號。
   - 在 Spark 查詢計劃 (DAG) 中最近執行的 Spark 動作。
   - Spark 驅動程式和執行器中與異常相關的簡明時間排序事件。您可以從中找到詳細資訊，例如完整的錯誤訊息、失敗的 Spark 任務及其執行器 ID，這些資訊可協助您專注於特定執行器的日誌串流，以便在需要時進行更深入的調查。
2. 基於規則的見解串流：
   - 根本原因分析和關於如何修正錯誤的建議 (例如使用特定的任務參數以最佳化效能)。
   - 作為根本原因分析基礎的相關 Spark 事件和建議的動作。

> **Note**
>
> 僅當任何異常 Spark 事件可用於失敗的任務執行時，第一個串流才會存在；只有在見解可用於失敗的任務執行時，第二個串流才存在。例如，如果您的任務成功完成，則不會產生任何串流；如果任務失敗，但沒有可以與故障場景相符的服務定義規則，則僅產生第一個串流。

如果從 AWS Glue Studio 執行任務，則在任務執行詳細資訊索引標籤 (任務執行見解) 下也會提供指向上述串流的連結，作為「簡明和合併的錯誤日誌」和「錯誤分析和指導」。



AWS Glue 任務執行見解的範例

在本部分中，我們展示任務執行見解功能如何協助您解決失敗任務中問題的範例。在此範例中，用户忘記將所需的模組 (tensorflow) 匯入 AWS Glue 任務來分析和建置其資料的機器學習模型。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql.types import *
from pyspark.sql.functions import udf,col

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
```

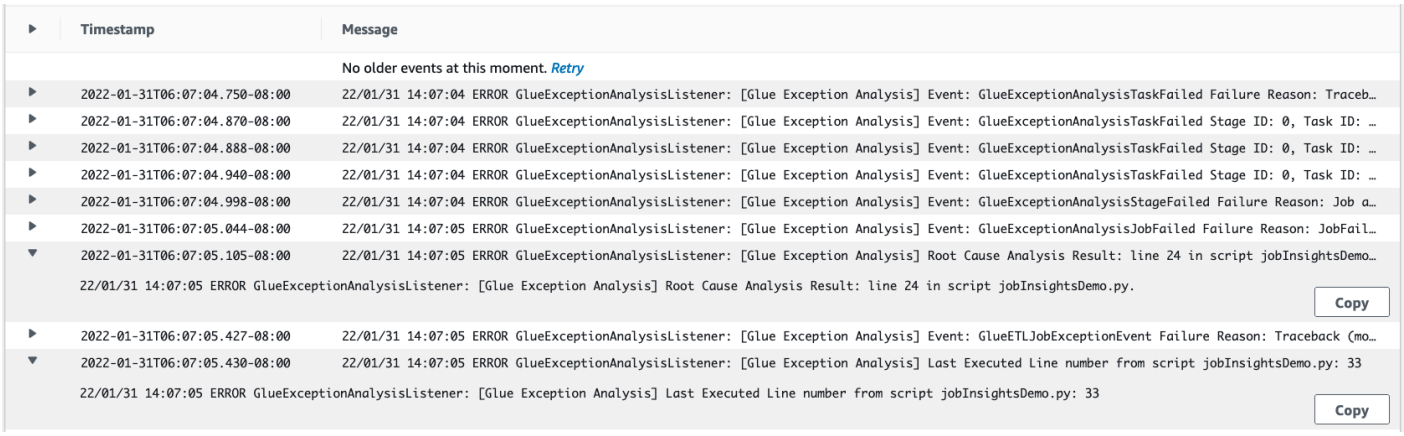
```
data_set_1 = [1, 2, 3, 4]
data_set_2 = [5, 6, 7, 8]

scoresDf = spark.createDataFrame(data_set_1, IntegerType())

def data_multiplier_func(factor, data_vector):
    import tensorflow as tf
    with tf.compat.v1.Session() as sess:
        x1 = tf.constant(factor)
        x2 = tf.constant(data_vector)
        result = tf.multiply(x1, x2)
        return sess.run(result).tolist()

data_multiplier_udf = udf(lambda x:data_multiplier_func(x, data_set_2),
 ArrayType(IntegerType(),False))
factoredDf = scoresDf.withColumn("final_value", data_multiplier_udf(col("value")))
print(factoredDf.collect())
```

如果沒有任務執行見解功能，當任務失敗時，您只會看到 Spark 擲回的訊息：

```
An error occurred while calling o111.collectToPython. Traceback (most
recent call last):
```

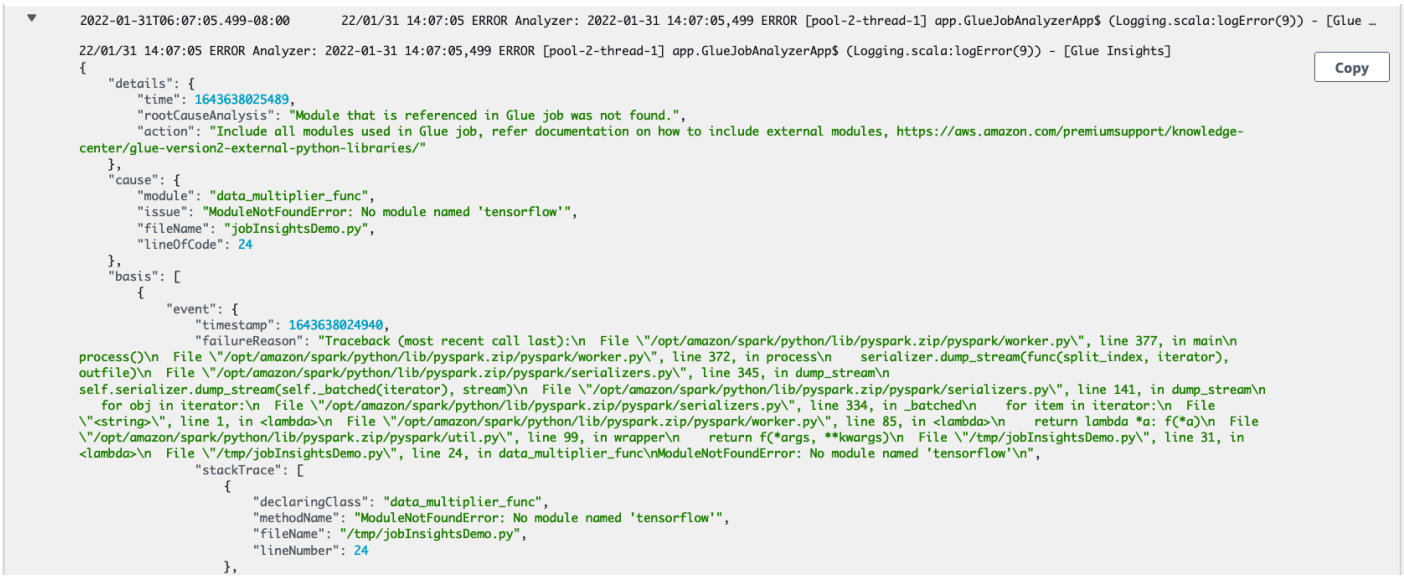
該訊息含糊不清，並限制了您的偵錯體驗。在此情況下，此功能會在兩個 CloudWatch 記錄串流中為您提供其他見解：

1. `job-insights-rca-driver` 日誌串流：
   - 異常事件：此日誌串流提供與從 Spark 驅動程式和不同分佈式工作者收集的故障相關的 Spark 異常事件。這些事件可協助您瞭解異常的時間排序傳播，因為故障程式碼跨 Spark 任務、執行器和分佈在 AWS Glue 工作者中的階段執行。
   - 行號：此日誌串流標識第 21 行，該行叫用以匯入導致失敗的 Python 模組；它還標識第 24 行，其中叫用 Spark Action `collect()`，作為指令碼中最後執行的一行。



2. `job-insights-rule-driver` 日誌串流：

   - 根本原因和建議：除了指令碼中故障的行號和最後執行的行號外，此日誌串流還顯示了根本原因分析和建議，供您遵循 AWS Glue 文件並設定必要的任務參數，以便在 AWS Glue 任務中使用附加的 Python 模組。
   - 基礎事件：此日誌串流還顯示了使用服務定義規則評估的 Spark 異常事件，以推斷根本原因並提供建議。



使用 Amazon CloudWatch 監控

您可以使用 Amazon CloudWatch 來監控 AWS Glue；該服務會收集並處理來自 AWS Glue 的原始資料，進而將這些資料轉換為便於讀取且幾近即時的指標。這些統計資料會記錄兩週的時間，讓您可以存取歷史資訊，以更清楚 Web 應用程式或服務的執行效能。根據預設，系統會自動將 AWS Glue 指標資料傳送至 CloudWatch。如需更多詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的什麼是 Amazon CloudWatch？，以及 AWS Glue 指標。

連續記錄

AWS Glue 也支援即時持續記錄 AWS Glue 任務。當任務的持續記錄啟用時，您就可以在 AWS Glue 主控台或 CloudWatch 主控台儀表板檢視即時日誌。如需更多詳細資訊，請參閱 持續記錄 AWS Glue 任務。

可觀測性指標

當工作可觀測性指標啟用後，其他 Amazon CloudWatch 指標會在工作執行時產生。使用 AWS Glue 可觀測性指標產生 AWS Glue 內部所發生之事件的深入解析，以改善問題的分類和分析。

主題

- 使用 Amazon CloudWatch 指標監控 AWS Glue
- 在 AWS Glue 任務設定檔上設定 Amazon CloudWatch 警示
- 持續記錄 AWS Glue 任務
- 使用 AWS Glue 可觀測性指標進行監控

使用 Amazon CloudWatch 指標監控 AWS Glue

您可以使用 AWS Glue 任務分析器來分析及監控 AWS Glue 操作。它會收集來自 AWS Glue 任務的原始資料，將這些資料處理成可讀取且幾近即時的指標並存放於 Amazon CloudWatch。系統會將這些統計資料保留於 CloudWatch 中並加以彙整，以便您存取歷史資訊，讓您更深入掌握應用程式的執行情況。

> **Note**
>
> 啟用任務指標並建立 CloudWatch 自訂指標時，您可能會產生額外費用。如需詳細資訊，請參閱 Amazon CloudWatch 定價。

AWS Glue 指標概觀

當您與 AWS Glue 互動時，它會傳送指標至 CloudWatch。您可使用 AWS Glue 主控台 (偏好的方法)、CloudWatch 主控台儀表板或 AWS Command Line Interface (AWS CLI) 中檢視這些指標。

使用 AWS Glue 主控台儀表板檢視指標

您能檢視任務指標的摘要或詳細圖表，或是任務執行的詳細圖表。

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格中，選擇任務執行監控。
3. 在任務執行中，選擇動作以停止目前正在執行的任務、檢視任務或倒轉任務書籤。
4. 選取任務，然後選擇檢視執行詳細資訊，以檢視有關任務執行的其他資訊。

使用 CloudWatch 主控台儀表板檢視指標

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 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在導覽窗格中，選擇 Metrics (指標)。
3. 選擇 Glue (Glue) 命名空間。

若要使用 AWS CLI 來檢視指標

- 在命令提示中，使用下列命令。

```
aws cloudwatch list-metrics --namespace Glue
```

AWS Glue 每 30 秒就會向 CloudWatch 回報指標，且 CloudWatch 指標儀表板也設定成每分鐘顯示一次指標。AWS Glue 指標表示先前回報數值中的差異值。適當時，指標儀表板將會彙總 (加總) 30 秒的值，以取得最後完整一分鐘的值。

Spark 任務的 AWS Glue 指標行為

當指令碼中的 `GlueContext` 進行初始化時，系統便已啟用 AWS Glue 指標。通常，唯有在 Apache Spark 任務快結束時，系統才會更新指標。這些指標代表目前所有完成的 Spark 任務彙總值。

但是，AWS Glue 傳遞至 CloudWatch 的 Spark 指標通常是絕對值，可代表這些指標在回報時的目前狀態。AWS Glue 每 30 秒就會向 CloudWatch 回報這些指標，且指標儀表板通常會顯示過去 1 分鐘內收到的資料點平均值。

AWS Glue 指標名稱皆有以下其中一個類型的前綴：

- `glue.driver.` – 若是以此字首為名稱開頭，即代表該指標屬於 Spark 驅動程式中所有執行器的彙總 AWS Glue 指標，或是對應至 Spark 驅動程式的 Spark 指標。

- glue.executorId. – executorId 是特定 Spark 執行器的數量。它對應至日誌中的執行器。
- glue.ALL. – 名稱開頭為此字首的指標即是所有 Spark 執行器的彙總值。

AWS Glue 指標

AWS Glue 會設定並將下列指標每隔 30 秒傳送至 CloudWatch，而 AWS Glue 指標儀表板每分鐘報告一次：

指標	描述
glue.driver.aggregate.bytesRead	由所有執行器中執行的所有已完成 Spark 任務，從所有資料來源讀取的位元組數。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計數)。 有效的統計數字：總和。此指標是最後一個報告值的差異值，因此在 AWS Glue 指標儀表板，SUM 統計數字用於彙總。 單位：位元組 可用於監控： • 讀取位元組數。 • 任務進度。 • JDBC 資料來源。 • 任務書籤問題。 • 任務執行間的變異。 此指標的使用方式與 glue.ALL.s3.filesystem.read_bytes 指標相同，差別在於此指標會在 Spark 任務結束時更新並擷取非 S3 資料來源。
glue.driver.aggregate.elapsedTime	ETL 經過時間 (以毫秒為單位) (不包括任務啟動程序時間)。

指標	描述
	有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計數)。 有效的統計數字：總和。此指標是最後一個報告值的差異值，因此在 AWS Glue 指標儀表板，SUM 統計數字用於彙總。 單位：毫秒 可用於判斷任務執行平均所需的時間。 使用資料的一些方法： • 設定落後者的警示。 • 測量任務執行間的變異。

指標	描述
glue.driver.aggregate.numCompletedStages	任務中已完成階段的數量。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計數)。 有效的統計數字：總和。此指標是最後一個報告值的差異值，因此在 AWS Glue 指標儀表板，SUM 統計數字用於彙總。 單位：計數 可用於監控： • 任務進度。 • 任務執行的每個階段時間表 (與其他指標相關)。 使用資料的一些方法： • 識別任務執行過程中要求嚴苛的階段。 • 針對任務執行中的相關尖峰 (需求階段) 設定警示。

指標	描述
glue.driver.aggregate.numCompletedTasks	任務中已完成任務的數目。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計數)。 有效的統計數字：總和。此指標是最後一個報告值的差異值，因此在 AWS Glue 指標儀表板，SUM 統計數字用於彙總。 單位：計數 可用於監控： • 任務進度。 • 階段內的平行性。
glue.driver.aggregate.numFailedTasks	失敗的任務數量。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計數)。 有效的統計數字：總和。此指標是最後一個報告值的差異值，因此在 AWS Glue 指標儀表板，SUM 統計數字用於彙總。 單位：計數 可用於監控： • 導致任務作業失敗的資料異常。 • 造成任務作業失敗的叢集異常。 • 導致任務作業失敗的指令碼異常。 這些資料可用來設定警示，以防止增加失敗，這些失敗可能會提出資料、叢集或指令碼的異常情況。

指標	描述
glue.driver.aggregate.numKilledTasks	終止的任務數量。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計數)。 有效的統計數字：總和。此指標是最後一個報告值的差異值，因此在 AWS Glue 指標儀表板，SUM 統計數字用於彙總。 單位：計數 可用於監控： • 導致終止任務的異常資料偏斜例外狀況 (OOM)。 • 導致終止任務的例外狀況 (OOM) 的指令碼異常情況。 使用資料的一些方法： • 對於指出資料異常的失敗增加設定警示。 • 對於指出叢集異常的失敗增加設定警示。 • 對於指出指令碼異常的失敗增加設定警示。

指標	描述
glue.driver.aggregate.recordsRead	由所有執行器中執行的所有已完成 Spark 任務，從所有資料來源讀取的記錄數。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計數)。 有效的統計數字：總和。此指標是最後一個報告值的差異值，因此在 AWS Glue 指標儀表板，SUM 統計數字用於彙總。 單位：計數 可用於監控： • 記錄讀取。 • 任務進度。 • JDBC 資料來源。 • 任務書籤問題。 • 任務執行中隨著天數的偏斜。 此指標的使用方式類似於 glue.ALL.s3.filesystem.read_bytes 指標，差別在於此指標會在 Spark 任務結束時更新。

指標	描述
glue.driver.aggregate.shuffleBytesWritten	自上次報告以來，所有執行器寫入的位元組數量 (以在這些執行器之間隨機移動資料) (由 AWS Glue 指標儀表板作為前一分鐘內為此目的而寫入的位元組數目)。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計數)。 有效的統計數字：總和。此指標是最後一個報告值的差異值，因此在 AWS Glue 指標儀表板，SUM 統計數字用於彙總。 單位：位元組 可用於監視：任務中的資料隨機播放 (大型聯結、GroupBy、重新分割、聯合)。 使用資料的一些方法： • 在進一步處理之前，重新分割或解壓縮大型輸入檔案。 • 更均勻地重新分割資料，以避免熱鍵。 • 在聯結或 GroupBy 操作之前預先篩選資料。

指標	描述
glue.driver.aggregate.shuffleLocalBytesRead	自上次報告以來，所有執行器讀取的位元組數量 (以在這些執行器之間隨機移動資料) (由 AWS Glue 指標儀表板作為前一分鐘內為此目的而讀取的位元組數目)。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計數)。 有效的統計數字：總和。此指標是最後一個報告值的差異值，因此在 AWS Glue 指標儀表板，SUM 統計數字用於彙總。 單位：位元組 可用於監視：任務中的資料隨機播放 (大型聯結、GroupBy、重新分割、聯合)。 使用資料的一些方法： • 在進一步處理之前，重新分割或解壓縮大型輸入檔案。 • 使用熱鍵更均勻地重新分割資料。 • 在聯結或 GroupBy 操作之前預先篩選資料。

指標	描述
glue.driver.BlockManager.disk.diskSpaceUsed_MB	在所有執行器中使用的 MB 磁碟空間數目。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計量)。 有效的統計數字：平均。這是 Spark 指標，報告為絕對值。 單位：MB 可用於監控： • 用於代表快取 RDD 分割區的區塊的磁碟空間。 • 用於表示中間隨機輸出的區塊的磁碟空間。 • 用於代表廣播的區塊的磁碟空間。 使用資料的一些方法： • 識別由於磁碟使用量增加而造成的任務失敗。 • 識別導致溢出或隨機播放的大型分割區。 • 增加佈建的 DPU 容量以修正這些問題。

指標	描述
glue.driver.ExecutorAllocationManager.executors.numberAllExecutors	目前正在執行中任務執行器的數量。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計量)。 有效的統計數字：平均。這是 Spark 指標，報告為絕對值。 單位：計數 可用於監控： • 任務活動。 • 跨越執行器 (只有幾個執行器執行中) • 目前執行器層級的平行處理原則。 使用資料的一些方法： • 如果叢集未充分利用，事先重新分割或解壓縮大型輸入檔案。 • 識別由於落後案例造成的階段或任務執行延遲。 • • 與 numberMaxNeededExecutors 進行比較，了解積壓項目以佈建更多 DPU。

指標	描述
glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors	滿足目前負載所需的最大值 (主動執行中和擱置中) 任務執行器數目。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計量)。 有效的統計數字：上限。這是 Spark 指標，報告為絕對值。 單位：計數 可用於監控： • 任務活動。 • 由於 DPU 容量或終止/失敗的執行器而無法使用的執行器，目前執行器層級並行性和待處理任務的積存尚未排程。 使用資料的一些方法： • 識別排程佇列的擱置/積存。 • 識別由於落後案例造成的階段或任務執行延遲。 • 與 numberAllExecutors 進行比較，了解積壓項目以佈建更多 DPU。 • 增加佈建的 DPU 容量，以修正擱置的執行器待處理項目。

指標	描述
glue.driver.jvm.heap.usage glue.executorId.jvm.heap.usage glue.ALL.jvm.heap.usage	此驅動程式之 JVM 堆疊所使用的記憶體佔比 (比例：0-1)、由 executorId 所辨識的執行器，或所有執行器。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計量)。 有效的統計數字：平均。這是 Spark 指標，報告為絕對值。 單位：百分比 可用於監控： • 使用 glue.driver.jvm.heap.usage 的驅動程式記憶體不足情況 (OOM)。 • 使用 glue.ALL.jvm.heap.usage 的執行器記憶體不足情況 (OOM)。 使用資料的一些方法： • 識別消耗記憶體的執行器 ID 和階段。 • 識別落後的執行器 ID 和階段。 • 識別驅動程式記憶體不足情況 (OOM)。 • 識別執行器記憶體不足條件 (OOM) 並獲取相應的執行器 ID，以便能夠從執行器日誌中取得堆疊追蹤。 • 識別可能導致資料偏差或記憶體不足狀況 (OOM) 的檔案或分割區。

指標	描述
glue.driver.jvm.heap.used glue.executorId.jvm.heap.used glue.ALL.jvm.heap.used	JVM 堆積針對驅動程式、由 executorId 識別的執行器或所有執行器，所使用的記憶體位元組數目。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計量)。 有效的統計數字：平均。這是 Spark 指標，報告為絕對值。 單位：位元組 可用於監控： • 驅動程式記憶體不足情況 (OOM)。 • 執行器記憶體不足情況 (OOM)。 使用資料的一些方法： • 識別消耗記憶體的執行器 ID 和階段。 • 識別落後的執行器 ID 和階段。 • 識別驅動程式記憶體不足情況 (OOM)。 • 識別執行器記憶體不足條件 (OOM) 並獲取相應的執行器 ID，以便能夠從執行器日誌中取得堆疊追蹤。 • 識別可能導致資料偏差或記憶體不足狀況 (OOM) 的檔案或分割區。

指標	描述
glue.driver.s3.filesystem.read_bytes glue.executorId.s3.filesystem.read_bytes glue.ALL.s3.filesystem.read_bytes	自上一次報告以來，驅動程式、由 executorId 識別的執行器或所有執行器從 Amazon S3 讀取的位元組數目 (由 AWS Glue 指標儀表板彙總為前一分鐘內讀取的位元組總數)。 有效維度：JobName、JobRunId 以及 Type (計量)。 有效的統計數字：總和。此指標是最後一個報告值的差異值，因此在 AWS Glue 指標儀表板，SUM 統計數字用於彙總。AWS Glue 指標儀表板上曲線下的區域可用來以視覺化方式比較兩個不同任務執行讀取的位元組。 單位：位元組。 可用於監控： • ETL 資料移動。 • 任務進度。 • 任務書籤問題 (已處理、重新處理和略過的資料)。 • 將讀取率和外部資料來源的擷取率相比較。 • 任務執行間的變異。 產生的資料可用於： • DPU 容量規劃。 • 針對任務執行和任務階段的資料讀取中的大量尖峰或下降設定警示。

指標	描述
glue.driver.s3.filesystem.write_bytes glue.executorId.s3.filesystem.write_bytes glue.ALL.s3.filesystem.write_bytes	自上一次報告以來，驅動程式、由 executorId 識別的執行器或所有執行器從 Amazon S3 寫入的位元組數目 (由 AWS Glue 指標儀表板彙總為前一分鐘內寫入的位元組總數)。 有效維度：JobName、JobRunId 以及 Type (計量)。 有效的統計數字：總和。此指標是最後一個報告值的差異值，因此在 AWS Glue 指標儀表板，SUM 統計數字用於彙總。AWS Glue 指標儀表板上曲線下的區域可用來以視覺化方式比較兩個不同任務執行寫入的位元組。 單位：位元組 可用於監控： • ETL 資料移動。 • 任務進度。 • 任務書籤問題 (已處理、重新處理和略過的資料)。 • 將讀取率和外部資料來源的擷取率相比較。 • 任務執行間的變異。 使用資料的一些方法： • DPU 容量規劃。 • 針對任務執行和任務階段的資料讀取中的大量尖峰或下降設定警示。

指標	描述
glue.driver.streaming.numRecords	在微批次中接收的記錄數目。此指標僅適用於搭配 AWS Glue2.0 及更新版本的 AWS Glue 串流任務。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計數)。 有效的統計數字：總和、上限、下限、平均、百分位數 單位：計數 可用於監控： • 記錄讀取。 • 任務進度。
glue.driver.streaming.batchProcessingTimeInMs	處理批次所需的時間 (以毫秒為單位)。此指標僅適用於搭配 AWS Glue2.0 及更新版本的 AWS Glue 串流任務。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計數)。 有效的統計數字：總和、上限、下限、平均、百分位數 單位：計數 可用於監控： • 任務進度。 • 指令碼效能。

指標	描述
glue.driver.system.cpuSystemLoad glue.executorId.system.cpuSystemLoad glue.ALL.system.cpuSystemLoad	驅動程式使用的 CPU 系統負載佔比 (比例：0-1)、由 executorId 所辨識的執行器，或所有執行器。 有效維度：JobName (AWS Glue 任務的名稱)、JobRunId (JobRun ID 或 ALL)，以及Type (計量)。 有效的統計數字：平均。此指標會報告為絕對值。 單位：百分比 可用於監控： • 驅動程式 CPU 負載。 • 執行器 CPU 負載。 • 偵測任務中的 CPU 密集型或 IO 密集型執行器或階段。 使用資料的一些方法： • DPU 容量規劃以及 IO 指標 (位元組讀取/隨機位元組、工作平行程度) 以及所需執行器指標上限的數目。 • 識別 CPU /IO 密集型比率。如此可針對具有較低 CPU 使用率的分割表資料集，對長時間執行的任務進行重新分割並增加佈建容量。

AWS Glue 指標的維度

AWS Glue 指標使用 AWS Glue 命名空間，並提供下列維度的指標：

維度	描述
JobName	此維度會篩選特定 AWS Glue 任務的所有任務執行的指標。

維度	描述
JobRunId	此維度會依照 JobRun ID 或 ALL 篩選特定 AWS Glue 任務執行的指標。
Type	此維度會依 count (彙總數字) 或 gauge (某個時間點的值) 篩選指標。

如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》。

在 AWS Glue 任務設定檔上設定 Amazon CloudWatch 警示

AWS Glue 指標也可於 Amazon CloudWatch 中使用。您可以針對已排程任務的任何 AWS Glue 指標，來設定警示。

下列是設定警示的幾個常見案例：

- 執行任務的記憶體用盡 (OOM)：設定警示，當 AWS Glue 任務的驅動程式或執行器之記憶體使用量超過正常平均值時，發出警示。
- 落後的執行器：設定警示，當 AWS Glue 任務中執行器的數量長時間持續低於某個閾值時，發出警示。
- 資料後端記錄或重新處理：使用 CloudWatch 的數學表達式，比較任務流程中個別任務的指標。然後，您就可以根據產生的運算式數值 (例如任務所寫入位元組的比率，以及後續任務所讀取的位元組數)，來觸發警示。

如需設定警示的詳細指示，請參閱 Amazon CloudWatch Events 使用者指南中的建立或編輯 CloudWatch 警示。

如需使用 CloudWatch 的監控與除錯案例，請參閱任務監控與偵錯。

持續記錄 AWS Glue 任務

AWS Glue 提供即時、持續的 AWS Glue 任務記錄。您可以在 Amazon 中查看實時 Apache Spark 任務日誌 CloudWatch，包括驅動程序日誌，執行程序日誌和 Apache Spark 任務進度條。檢視即時日誌可讓您更了解執行中的任務。

當您啟動AWS Glue作業時，它會在 Spark 應用程式開始執行之後，將即時記錄資訊傳送至 CloudWatch (每個執行程式終止之前每個執行程式終止一次)。您可以在AWS Glue控制台或控制 CloudWatch 台儀表板上查看日誌。

持續記錄功能包含下列功能：

- 連續記錄
- 以自訂指令碼記錄器記錄應用程式特定訊息
- 主控台進度列可追蹤目前 AWS Glue 任務的執行狀態

如需 AWS Glue 2.0 版如何支援連續記錄的資訊，請參閱以縮短的啟動時間執行 Spark ETL 任務。

您可以限制 IAM 角色讀取記 CloudWatch 錄群組或串流的存取權限。如需有關限制存取權限的詳細資訊，請參閱文件中的針對 CloudWatch 記錄使用身分型政策 (IAM 政策)。 CloudWatch

> **Note**
>
> 當您啟用連續記錄並建立其他記 CloudWatch 錄事件時，可能會產生額外費用。如需詳細資訊，請參閱 Amazon CloudWatch 定價。

主題

- 啟用持續記錄 AWS Glue 任務
- 檢視 AWS Glue 任務的持續記錄

啟用持續記錄 AWS Glue 任務

您可以使用AWS Glue主控台或透過 AWS Command Line Interface (AWS CLI) 啟用連續記錄。

您可以在建立新工作、編輯現有工作或透過啟用時啟用連續記錄 AWS CLI。

您也可以指定自訂組態選項，例如 Amazon CloudWatch 記錄群組名稱、AWS Glue工作執行 ID 驅動程式/執行程式 ID 之前的 CloudWatch 記錄資料流前置詞，以及記錄訊息的記錄檔轉換模式。這些設定可協助您在具有不同到期原則的自訂記 CloudWatch 錄群組中設定彙總記錄，並使用自訂記錄串流前置詞和轉換模式進一步分析這些記錄。

主題

- 使用 AWS Management Console
- 使用自訂指令碼記錄器記錄應用程式特定訊息
- 啟用進度列來顯示任務進度

- 具有連續記錄的安全組態

使用 AWS Management Console

依照以下步驟使用主控台，在建立或編輯 AWS Glue 任務時啟用持續記錄。

建立持續記錄的新 AWS Glue 任務

1. 請登入 AWS Management Console 並開啟 AWS Glue 主控台，網址為 https://console.aws.amazon.com/glue/。
2. 在瀏覽窗格中，選擇 ETL 工作。
3. 選擇「視覺 ETL」。
4. 在 [Job 詳細資訊] 索引標籤中，展開 [進階屬性] 區段
5. 在 [連續記錄] 下選取 [啟用登入] CloudWatch。

啟用持續記錄現有 AWS Glue 任務

1. 請在以下位置開啟 AWS Glue 主控台。 https://console.aws.amazon.com/glue/
2. 在導覽窗格中，選擇 Jobs (任務)。
3. 從 Jobs (任務) 清單中選擇現有的任務。
4. 選擇 Action (動作)、Edit job (編輯任務)。
5. 在 [Job 詳細資訊] 索引標籤中，展開 [進階屬性] 區段
6. 在 [連續記錄] 下選取 [啟用登入] CloudWatch。

使用 AWS CLI

若要啟用持續記錄，您必須將任務參數傳遞至 AWS Glue 任務。傳遞下列與其他作業參數類似的特殊 AWS Glue工作參數。如需詳細資訊，請參閱 AWS Glue 任務參數。

```
'--enable-continuous-cloudwatch-log': 'true'
```

您可以指定自訂的 Amazon CloudWatch 日誌群組名稱。如果未指定，則預設日誌群組名稱為 `/aws-glue/jobs/logs-v2/`。

```
'--continuous-log-logGroup': 'custom_log_group_name'
```

您可以指定自訂的 Amazon CloudWatch 日誌串流前置詞。如果未指定，則預設日誌資料流字首為任務執行 ID。

```
'--continuous-log-logStreamPrefix': 'custom_log_stream_prefix'
```

您可以指定自訂持續記錄轉換模式。如果未指定，則預設轉換模式為 %d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n。請注意，轉換模式僅適用於驅動程式日誌和執行程式日誌。它不會影響 AWS Glue 進度列。

```
'--continuous-log-conversionPattern': 'custom_log_conversion_pattern'
```

使用自訂指令碼記錄器記錄應用程式特定訊息

您可以使用 AWS Glue 記錄器，記錄即時傳送至驅動程式日誌串流之指令碼中的任何應用程式特定訊息。

以下範例顯示 Python 指令碼。

```
from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)
logger = glueContext.get_logger()
logger.info("info message")
logger.warn("warn message")
logger.error("error message")
```

以下範例顯示 Scala 指令碼。

```
import com.amazonaws.services.glue.log.GlueLogger

object GlueApp {
  def main(sysArgs: Array[String]) {
    val logger = new GlueLogger
    logger.info("info message")
    logger.warn("warn message")
    logger.error("error message")
  }
}
```

啟用進度列來顯示任務進度

AWS Glue 在 JOB_RUN_ID-progress-bar 日誌串流下方提供即時的進度列，以檢查 AWS Glue 任務執行狀態。目前它僅支援初始化 glueContext 的任務。如果您執行無需初始化 glueContext 的純 Spark 任務，AWS Glue 進度列將不會出現。

進度列每 5 秒顯示一次以下的最新進度。

```
Stage Number (Stage Name): > (numCompletedTasks + numActiveTasks) /
 totalNumOfTasksInThisStage]
```

具有連續記錄的安全組態

如果已啟用 CloudWatch 記錄檔的安全性設定，AWS Glue將會針對連續記錄建立名稱如下的記錄群組：

```
<Log-Group-Name>-<Security-Configuration-Name>
```

預設和自訂日誌群組將如下所示：

- 預設的連續日誌群組為 /aws-glue/jobs/logs-v2-*<Security-Configuration-Name>*
- 自訂的連續日誌群組為 *<custom-log-group-name>*-*<Security-Configuration-Name>*

如果您使用 CloudWatch 記錄啟用安全設定，則需要將 IAM 角色許可新增logs:AssociateKmsKey至您的 IAM 角色許可。如果未包含該許可，則會停用連續記錄。此外，若要設定日誌的加密，請按照 Amazon CloudWatch 日誌使用者指南中的使用加密 CloudWatch 日誌中的日 CloudWatch 誌資料 AWS Key Management Service中的說明進行操作。

如需建立安全組態的詳細資訊，請參閱在 AWS Glue 主控台上使用安全組態。

檢視 AWS Glue 任務的持續記錄

您可以使用 AWS Glue 主控台或 Amazon CloudWatch 主控台檢視即時日誌。

使用 AWS Glue 主控台儀表板檢視即時日誌

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格中，選擇 Jobs (任務)。
3. 新增或開始現有的任務。選擇 Action (動作)、Run job (執行任務)。

當您開始執行任務時，請導覽到頁面，其中包含有關執行中任務的資訊：

- Logs (日誌) 標籤會顯示較舊的彙總應用程式日誌 。
- Continuous logging (連續記錄) 索引標籤會在任務搭配 `glueContext` 初始化執行時顯示即時進度列。
- Continuous logging (持續記錄) 標籤也包含 Driver logs (驅動程式日誌)，它會擷取即時 Apache Spark 驅動程式日誌，以及任務執行時使用 AWS Glue 應用程式記錄器記錄之指令碼中的應用程式日誌。

4. 對於較舊的任務，您也可以在 Job History (任務歷史記錄) 檢視中透過選擇 Logs (日誌) 來檢視即時日誌。此動作會移至 CloudWatch 主控台，並顯示所有 Spark 驅動程式、執行器，以及該任務執行的進度列日誌串流。

使用 CloudWatch 主控台儀表板檢視即時日誌

1. 在 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在導覽窗格中，選擇 Log (日誌)。
3. 選擇 /aws-glue/jobs/logs-v2/ 日誌群組。
4. 在 Filter (篩選條件) 方塊中，貼上任務執行 ID。

   您可以檢視驅動程式日誌、執行器日誌，以及進度列 (如果使用 Standard filter (標準篩選條件))。

使用 AWS Glue 可觀測性指標進行監控

> **Note**
>
> AWS Glue 可觀測性指標可在 AWS Glue 4.0 及更新版本上使用。

使用 AWS Glue 可觀測性指標針對 Apache Spark 任務產生 AWS Glue 內部所發生之事件的深入解析，以改善問題的分類和分析。可觀測性指標能透過 Amazon CloudWatch 儀表板以視覺化呈現，並可用來協助執行錯誤的根本原因分析，以及診斷效能瓶頸。您可以減少大規模偵錯問題所花費的時間，使您能夠專注於更快、更有效地解決問題。

AWS Glue可觀測性提供以下四個群組分類的 Amazon CloudWatch 量度：

- 可靠性 (即錯誤類別)：可輕鬆識別特定時間範圍內您可能想要解決之常見失敗原因。

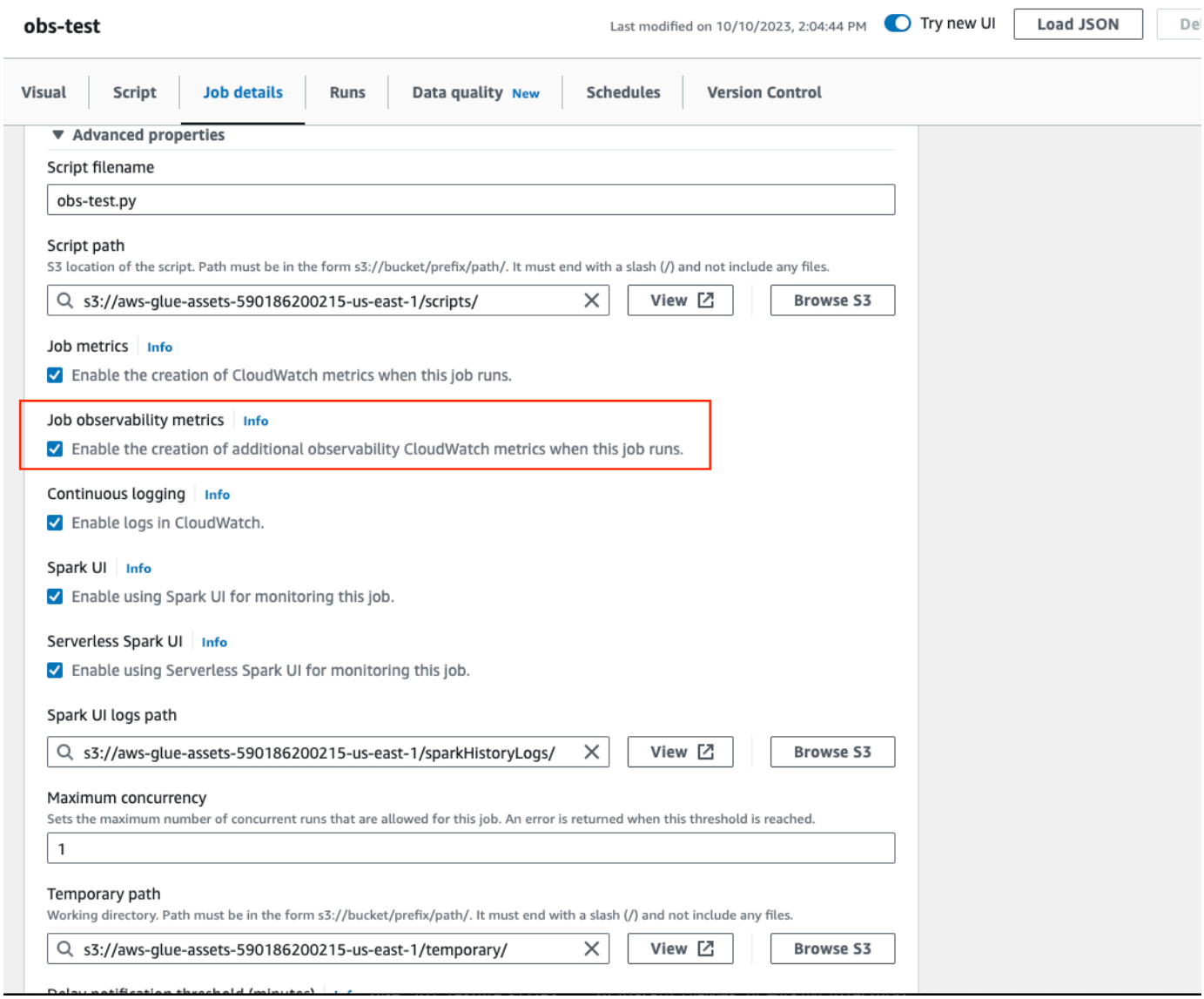
- 效能 (即偏態)：識別效能瓶頸並套用調整技術。例如，當您因任務偏態而遇到效能降低時，您可能會想要啟用 Spark 調適性查詢執行，並微調偏斜聯結閾值。
- 輸送量 (即每個來源/接收器輸送量)：監控資料讀取和寫入的趨勢。您也可以設定異常 Amazon CloudWatch 警示。
- 資源使用率 (即工作者、記憶體和磁碟使用率)：有效率地找出容量使用率低的任務。您可能會想要為這些任務啟用 AWS Glue 自動擴展。

開始使用 AWS Glue 可觀測性指標

> **Note**
>
> 設會在 AWS Glue Studio 主控台中啟用新的指標。

要在 AWS Glue Studio 中設定可觀測性指標：

1. 登入 AWS Glue 主控台，然後從主控台功能表選擇 ETL 任務。
2. 按一下您的任務區段中的任務名稱，即可選擇任務。
3. 選擇 Job details (任務詳細資訊) 索引標籤。
4. 捲動至底部，然後選擇進階屬性，接著選擇任務可觀測性指標。



若要使用以下方式啟用 AWS CLI「AWS Glue觀察性」

- 將輸入 JSON 檔案中的下列鍵值新增至 `--default-arguments` 對應：

```
--enable-observability-metrics, true
```

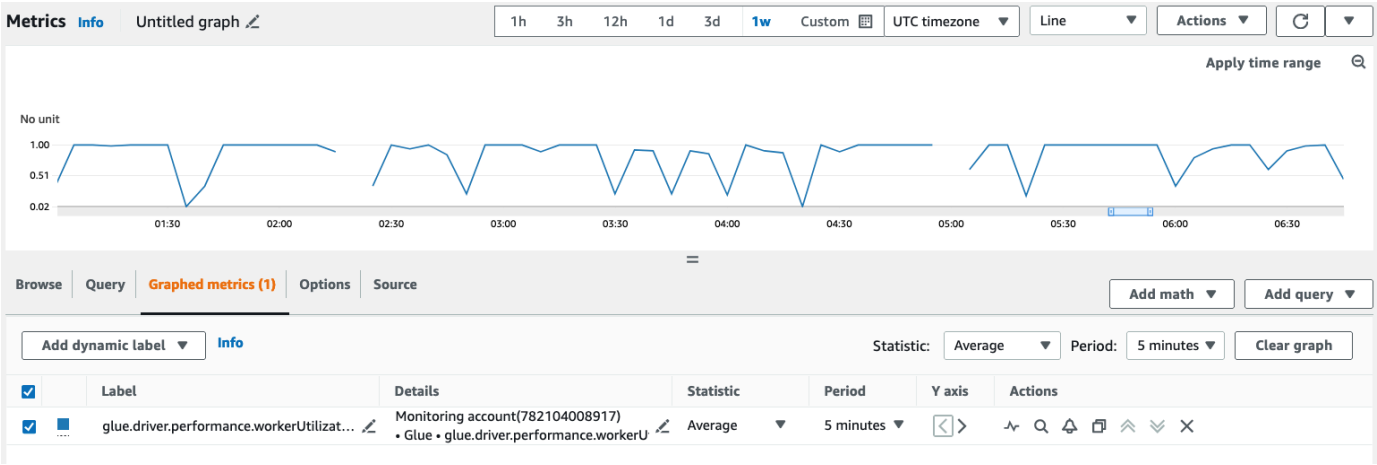
使用 AWS Glue 可觀測性

由於AWS Glue觀察度量是透過提供的 Amazon CloudWatch，因此您可以使用 Amazon CloudWatch 主控台 AWS CLI、SDK 或 API 來查詢可觀察性指標資料點。請參閱使用 Glue 可觀測性監控資源使用率以降低成本，了解何時使用 AWS Glue 可觀測性指標的範例使用案例。

在控制AWS GlueAmazon CloudWatch 台中使用可觀察性

若要在 Amazon CloudWatch 主控台中查詢和視覺化指標：

1. 開啟 Amazon CloudWatch 主控台，然後選擇 [所有指標]。
2. 在自訂命名空間下方，選取 AWS Glue。
3. 選擇任務可觀測性指標、每個來源的可觀測性指標，或每個接收器的可觀測性指標。
4. 搜尋特定的指標名稱、任務名稱、任務執行 ID，然後加以選取。
5. 在圖形化指標索引標籤下方，設定您偏好的統計資料、期間和其他選項。



若要使 AWS CLI用下列方式查詢觀測量結果：

1. 建立指標定義 JSON 檔案並使用自己的值取代 `your-Glue-job-name` 和 `your-Glue-job-run-id`。

```
$ cat multiplequeries.json
[
    {
        "Id": "avgWorkerUtil_0",
        "MetricStat": {
            "Metric": {
                "Namespace": "Glue",
```

```
                "MetricName": "glue.driver.workerUtilization",
                "Dimensions": [
                    {
                        "Name": "JobName",
                        "Value": "<your-Glue-job-name-A>"
                    },
                    {
                        "Name": "JobRunId",
                        "Value": "<your-Glue-job-run-id-A>"
                    },
                    {
                        "Name": "Type",
                        "Value": "gauge"
                    },
                    {
                        "Name": "ObservabilityGroup",
                        "Value": "resource_utilization"
                    }
                ]
            },
            "Period": 1800,
            "Stat": "Minimum",
            "Unit": "None"
        }
    },
    {
        "Id": "avgWorkerUtil_1",
        "MetricStat": {
            "Metric": {
                "Namespace": "Glue",
                "MetricName": "glue.driver.workerUtilization",
                "Dimensions": [
                    {
                        "Name": "JobName",
                        "Value": "<your-Glue-job-name-B>"
                    },
                    {
                        "Name": "JobRunId",
                        "Value": "<your-Glue-job-run-id-B>"
                    },
                    {
                        "Name": "Type",
                        "Value": "gauge"
                    },
```

```
                    {
                        "Name": "ObservabilityGroup",
                        "Value": "resource_utilization"
                    }
                ]
            },
            "Period": 1800,
            "Stat": "Minimum",
            "Unit": "None"
        }
    }
]
```

2. 執行 get-metric-data 命令：

```
$ aws cloudwatch get-metric-data --metric-data-queries file: //multiplequeries.json
 \
     --start-time '2023-10-28T18: 20' \
     --end-time '2023-10-28T19: 10'  \
     --region us-east-1
{
    "MetricDataResults": [
        {
            "Id": "avgWorkerUtil_0",
            "Label": "<your-label-for-A>",
            "Timestamps": [
                "2023-10-28T18:20:00+00:00"
            ],
            "Values": [
                0.06718750000000001
            ],
            "StatusCode": "Complete"
        },
        {
            "Id": "avgWorkerUtil_1",
            "Label": "<your-label-for-B>",
            "Timestamps": [
                "2023-10-28T18:50:00+00:00"
            ],
            "Values": [
                0.5959183673469387
            ],
```

```
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

可觀測性指標

AWS Glue「觀察性」設定檔並將下列測量結果傳送至 Amazon CloudWatch 每 30 秒一次，而其中一些測量結果則會顯示在「AWS Glue StudioJob 執行監督」頁面中。

指標	描述	類別
glue.driver.skewness.stage	指標類別：job_performance Spark 階段執行偏態：此指標擷取執行偏態，此偏態可能是由輸入資料偏態或轉換 (例如，偏斜聯結) 引起。此指標的值落在 [0, infinity[的範圍內，其中 0 表示該階段所有任務中，任務執行時間的最大值與中位數的比率小於特定階段偏態因子。預設階段偏態因子是「5」，可透過 spark conf 覆寫此值：spark.metrics.conf.driver.source.glue.jobPerformance.skewnessFactor 階段偏態值為 1 表示比率是階段偏態因子的兩倍。 階段偏態的值會每 30 秒更新一次，可反映目前的偏態。階段結束時的值會反映最後階段偏態。	job_performance

指標	描述	類別
	有效維度： JobName (J AWS Glue ob 的名稱)、 JobRunId (JobRun ID 或全部)、類型 (量測規) 和 ObservabilityGroup (job_效能) 有效的統計資料：平均值、最大值、最小值、百分位數 單位：計數	
glue.driver.skewness.job	指標類別：job_performance 作業偏態是作業階段偏態的加權平均值。加權平均值會為需要更長執行時間的階段提供更多權重。這是為了避免以下極端情況，即相對於其他階段，嚴重偏斜的階段實際執行時間很短 (因此其偏態對整體作業效能影響不大，不值得費力嘗試解決其偏態)。 此指標會在每個階段完成時更新，因此最後一個值會反映實際的整體作業偏態。 有效維度： JobName (J AWS Glue ob 的名稱)、 JobRunId (JobRun ID 或全部)、類型 (量測規) 和 ObservabilityGroup (job_效能) 有效的統計資料：平均值、最大值、最小值、百分位數 單位：計數	job_performance

指標	描述	類別
glue.succeed.ALL	指標類別：錯誤 成功的作業執行總數，可完成失敗類別的圖片 有效維度： JobName (AWS GlueJob 名稱)、 JobRunId (JobRun ID 或 ALL)、類型 (計數) 和 ObservabilityGroup (錯誤) 有效的統計資料：總和 單位：計數	error
glue.error.ALL	指標類別：錯誤 作業執行錯誤總數，可完成失敗類別的圖片 有效維度： JobName (AWS GlueJob 名稱)、 JobRunId (JobRun ID 或 ALL)、類型 (計數) 和 ObservabilityGroup (錯誤) 有效的統計資料：總和 單位：計數	error

指標	描述	類別
glue.error.[error category]	指標類別：錯誤 這實際上是一組指標，只有在作業執行失敗時才會更新。錯誤分類有助於分類和偵錯。當作業執行失敗時，會將造成失敗的錯誤分類，並將對應的錯誤類別指標設為 1。這有助於針對所有作業錯誤分析執行隨著時間變化的失敗分析，以找出最常見的失敗類別並開始解決這些問題。AWS Glue 有 28 個錯誤類別，包括 OUT_OF_MEMORY (驅動程式和執行程式)、PERMISSION、SYNTAX 和 THROTTLING 錯誤類別。錯誤類別還包括 COMPILATION、LAUNCH 和 TIMEOUT 錯誤類別。 有效維度： JobName (AWS GlueJob 名稱)、 JobRunId (JobRun ID 或 ALL)、類型 (計數) 和 ObservabilityGroup (錯誤) 有效的統計資料：總和 單位：計數	error

指標	描述	類別
glue.driver.workerUtilization	指標類別：resource_utilization 實際使用的已配置工作者百分比。如果不理想，自動擴展可以有所助益。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值、最大值、最小值、百分位數 單位：百分比	resource_utilization
glue.driver.memory.heap.[available | used]	指標類別：resource_utilization 在作業執行期間，驅動程式可用/已用的堆積記憶體。這有助於了解記憶體使用量趨勢，特別是隨著時間的推移，除了對記憶體相關的失敗進行偵錯，還有助於避免潛在的失敗。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：位元組	resource_utilization

指標	描述	類別
glue.driver.memory.heap.used.percentage	指標類別：resource_utilization 在作業執行期間，驅動程式已用的 (%) 堆積記憶體。這有助於了解記憶體使用量趨勢，特別是隨著時間的推移，除了對記憶體相關的失敗進行偵錯，還有助於避免潛在的失敗。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：百分比	resource_utilization

指標	描述	類別
glue.driver.memory.non-heap.[available | used]	指標類別：resource_utilization 在作業執行期間，驅動程式可用/已用的非堆積記憶體。這有助於了解記憶體使用量趨勢，特別是隨著時間的推移，除了對記憶體相關的失敗進行偵錯，還有助於避免潛在的失敗。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：位元組	resource_utilization

指標	描述	類別
glue.driver.memory.non-heap.used.percentage	指標類別：resource_utilization 在作業執行期間，驅動程式已用的 (%) 非堆積記憶體。這有助於了解記憶體使用量趨勢，特別是隨著時間的推移，除了對記憶體相關的失敗進行偵錯，還有助於避免潛在的失敗。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：百分比	resource_utilization

指標	描述	類別
glue.driver.memory.total.[available | used]	指標類別：resource_utilization 在作業執行期間，驅動程式可用/已用的記憶體總量。這有助於了解記憶體使用量趨勢，特別是隨著時間的推移，除了對記憶體相關的失敗進行偵錯，還有助於避免潛在的失敗。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：位元組	resource_utilization

指標	描述	類別
glue.driver.memory.total.used.percentage	指標類別：resource_utilization 在作業執行期間，驅動程式已用的 (%) 記憶體總量。這有助於了解記憶體使用量趨勢，特別是隨著時間的推移，除了對記憶體相關的失敗進行偵錯，還有助於避免潛在的失敗。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：百分比	resource_utilization
glue.ALL.memory.heap.[available | used]	指標類別：resource_utilization 執行程式可用/已用的堆積記憶體。ALL 表示所有執行程式。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：位元組	resource_utilization

指標	描述	類別
glue.ALL.memory.heap.used.percentage	指標類別：resource_utilization 執行程式已用的 (%) 堆積記憶體。ALL 表示所有執行程式。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：百分比	resource_utilization
glue.ALL.memory.non-heap.[available | used]	指標類別：resource_utilization 執行程式可用/已用的非堆積記憶體。ALL 表示所有執行程式。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：位元組	resource_utilization

指標	描述	類別
glue.ALL.memory.non-heap.used.percentage	指標類別：resource_utilization 執行程式已用的 (%) 非堆積記憶體。ALL 表示所有執行程式。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：百分比	resource_utilization
glue.ALL.memory.total.[available | used]	指標類別：resource_utilization 執行程式可用/已用的記憶體總量。ALL 表示所有執行程式。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：位元組	resource_utilization

指標	描述	類別
glue.ALL.memory.total.used.percentage	指標類別：resource_utilization 執行程式已用的 (%) 記憶體總量。ALL 表示所有執行程式。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：百分比	resource_utilization
glue.driver.disk.[available_GB | used_GB]	指標類別：resource_utilization 在作業執行期間，驅動程式可用/已用的磁碟空間。這有助於了解磁碟使用量趨勢，特別是隨著時間的推移，除了對磁碟空間不足相關的失敗進行偵錯，還有助於避免潛在的失敗。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：千兆位元組 (GB)	resource_utilization

指標	描述	類別
glue.driver.disk.used.percentage]	指標類別：resource_utilization 在作業執行期間，驅動程式可用/已用的磁碟空間。這有助於了解磁碟使用量趨勢，特別是隨著時間的推移，除了對磁碟空間不足相關的失敗進行偵錯，還有助於避免潛在的失敗。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：百分比	resource_utilization
glue.ALL.disk.[available_GB | used_GB]	指標類別：resource_utilization 執行程式可用/已用的磁碟空間。ALL 表示所有執行程式。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：千兆位元組 (GB)	resource_utilization

指標	描述	類別
glue.ALL.disk.used.percentage	指標類別：resource_utilization 執行程式可用/已用/已用(%) 的磁碟空間。ALL 表示所有執行程式。 有效維度： JobName (「AWS GlueJob」的名稱)、JobRunId (JobRun ID 或「全部」)、「類型」(量測計) 及 ObservabilityGroup (資源使用率) 有效的統計資料：平均值 單位：百分比	resource_utilization
glue.driver.bytesRead	指標類別：輸送量 此作業執行中每個輸入來源以及針對所有來源讀取的位元組數目。這有助於了解資料磁碟區及其隨時間的變化，進而幫助解決資料偏態等問題。 有效維度： JobName (「AWS GlueJob」的名稱)、 JobRunId (JobRun ID 或「全部」)、「類型」(量測計)、 ObservabilityGroup (Resource_Utilise) 和來源 (來源資料位置) 有效的統計資料：平均值 單位：位元組	輸送量

指標	描述	類別
glue.driver.[recordsRead | filesRead]	指標類別：輸送量 此作業執行中每個輸入來源以及針對所有來源讀取的記錄/檔案數目。這有助於了解資料磁碟區及其隨時間的變化，進而幫助解決資料偏態等問題。 有效維度： JobName (「AWS GlueJob」的名稱)、 JobRunId (JobRun ID 或「全部」)、「類型」(量測計)、 ObservabilityGroup (Resource_Utilise) 和來源 (來源資料位置) 有效的統計資料：平均值 單位：計數	輸送量
glue.driver.partitionsRead	指標類別：輸送量 此作業執行中每個 Amazon S3 輸入來源以及針對所有來源讀取的分割區數目。 有效維度： JobName (「AWS GlueJob」的名稱)、 JobRunId (JobRun ID 或「全部」)、「類型」(量測計)、 ObservabilityGroup (Resource_Utilise) 和來源 (來源資料位置) 有效的統計資料：平均值 單位：計數	輸送量

指標	描述	類別
glue.driver.bytesWrittten	指標類別：輸送量 此作業執行中每個輸出接收器以及針對所有接收器寫入的位元組數目。這有助於了解資料磁碟區及其隨時間的演進，進而幫助解決處理偏態等問題。 有效維度： JobName (AWS GlueJob 的名稱)、 JobRunId (JobRun ID 或全部)、類型 (量測計)、(資源 _ 使用率) 和接收器 ObservabilityGroup (接收器資料位置) 有效的統計資料：平均值 單位：位元組	輸送量

指標	描述	類別
glue.driver.[recordsWritten | filesWritten]	指標類別：輸送量 此作業執行中每個輸出接收器以及針對所有接收器寫入的記錄/檔案數目。這有助於了解資料磁碟區及其隨時間的演進，進而幫助解決處理偏態等問題。 有效維度： JobName (AWS GlueJob 的名稱)、 JobRunId (JobRun ID 或全部)、類型 (量測計)、(資源 _ 使用率) 和接收器 ObservabilityGroup (接收器資料位置) 有效的統計資料：平均值 單位：計數	輸送量

錯誤類別

錯誤類別	描述
COMPILATION_ERROR	編譯 Scala 程式碼期間出現錯誤。
CONNECTION_ERROR	連線到服務/遠端主機/資料庫服務等項目時出現錯誤。
DISK_NO_SPACE_ERROR	驅動程式/執行程式的磁碟中沒有剩餘空間時出現錯誤。
OUT_OF_MEMORY_ERROR	驅動程式/執行程式的記憶體中沒有剩餘空間時出現錯誤。
IMPORT_ERROR	匯入相依性時出現錯誤。

錯誤類別	描述
INVALID_ARGUMENT_ERROR	當輸入參數無效/非法時出現錯誤。
PERMISSION_ERROR	缺少服務、資料等項目的許可時出現錯誤。
RESOURCE_NOT_FOUND_ERROR	資料、位置等項目不存在時出現錯誤。
QUERY_ERROR	因 Spark SQL 查詢執行而出現錯誤。
SYNTAX_ERROR	指令碼中存在語法錯誤時出現錯誤。
THROTTLING_ERROR	達到服務並行限制或超出服務配額限制時出現錯誤。
DATA_LAKE_FRAMEWORK_ERROR	因 AWS Glue 原生支援的資料湖架構 (例如 Hudi、Iceberg 等) 而出現錯誤。
UNSUPPORTED_OPERATION_ERROR	進行不支援的操作時出現錯誤。
RESOURCES_ALREADY_EXISTS_ERROR	要建立或新增的資源已存在時出現錯誤。
GLUE_INTERNAL_SERVICE_ERROR	發生 AWS Glue 內部服務問題時出現錯誤。
GLUE_OPERATION_TIMEOUT_ERROR	AWS Glue 操作逾時時出現錯誤。
GLUE_VALIDATION_ERROR	無法驗證 AWS Glue 作業所需的值時出現錯誤。
GLUE_JOB_BOOKMARK_VERSION_MISMATCH_ERROR	在相同的來源儲存貯體上執行同一個作業並同時寫入相同/不同的目的地 (並行數 > 1) 時出現錯誤
LAUNCH_ERROR	在 AWS Glue 作業啟動階段出現錯誤。
DYNAMODB_ERROR	一般錯誤來自 Amazon DynamoDB 服務。
GLUE_ERROR	因 AWS Glue 服務而出現一般錯誤。
LAKEFORMATION_ERROR	一般錯誤來自 AWS Lake Formation 服務。
REDSHIFT_ERROR	一般錯誤來自 Amazon Redshift 服務。

錯誤類別	描述
S3_ERROR	因 Amazon S3 服務而出現一般錯誤。
SYSTEM_EXIT_ERROR	一般系統結束錯誤。
TIMEOUT_ERROR	作業因操作逾時而失敗時出現一般錯誤。
UNCLASSIFIED_SPARK_ERROR	因 Spark 而出現一般錯誤。
UNCLASSIFIED_ERROR	預設錯誤類別。

限制

> **Note**
>
> 必須將 `glueContext` 初始化才能發布指標。

在來源維度中，值可以是 Amazon S3 路徑或資料表名稱，具體取決於來源類型。此外，如果來源為 JDBC 且使用的是查詢選項，則會在來源維度中設定查詢字串。如果該值超過 500 個字元，則會修剪至 500 個字元以內。值的限制如下：

- 非 ASCII 字元會被移除。
- 如果來源名稱不包含任何 ASCII 字元，則會將該名稱轉換為 <非 ASCII 輸入>。

輸送量指標的限制和考量

- DataFrame 支持 DataFrame基於 DynamicFrame （例如 JDBC，從 Amazon S3 上的實木地板讀取），但是，不支持基於 RDS 的 DynamicFrame （例如，在 Amazon S3 上讀取 csv，json 等）。從技術上來講，支援 Spark UI 上可見的所有讀取和寫入。
- 如果資料來源是目錄資料表且格式為 JSON、CSV、文字或 Iceberg，則會發出 `recordsRead` 指標。
- JDBC 和 Iceberg 資料表中不提供 `glue.driver.throughput.recordsWritten`、`glue.driver.throughput.bytesWritten` 和 `glue.driver.throughput.filesWritten` 指標。

- 指標可能會延遲出現。如果工作在大約一分鐘內完成，「測量結果」中可能沒有輸送量測量 Amazon CloudWatch 結果。

任務監控與偵錯

您可以收集 AWS Glue 任務的指標，並且在 AWS Glue 和 Amazon CloudWatch 主控台上，用視覺化的方式呈現這些指標，來找出和修正問題。分析您的 AWS Glue 任務需要執行下列步驟：

1. 啟用指標：

   a. 在任務定義中啟用 Job metrics (任務指標) 選項。您可以在 AWS Glue 主控台中啟用分析功能，或是將分析功能做為任務的參數。如需詳細資訊，請參閱定義 Spark 任務的任務屬性或AWS Glue 任務參數。

   b. 在任務定義中啟用 AWS Glue 可觀測性指標選項。您可以在 AWS Glue 主控台中啟用可觀測性指標，或是將其作為作業的參數。如需詳細資訊，請參閱使用 AWS Glue 可觀測性指標進行監控。

2. 確認任務指令碼會將 `GlueContext` 初始化。例如，下列的指令碼片段會將 `GlueContext` 初始化，並顯示分析程式碼在指令碼中的所在位置。這個一般格式會在後續的除錯案例中使用。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
import time

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

...
...
code-to-profile
...
```

```
...


job.commit()
```

3. 執行任務。
4. 以視覺化的方式呈現指標：
   a. 在 AWS Glue 主控台中以視覺化的方式呈現指標，並找出驅動程式或執行程式的異常指標。
   b. 在作業執行監控頁面、作業執行詳細資訊頁面或 Amazon CloudWatch 上查看可觀測性指標。如需更多詳細資訊，請參閱 使用 AWS Glue 可觀測性指標進行監控。
5. 利用辨識出的指標來縮小問題根源的範圍。
6. 或者，使用已辨識驅動程式或任務執行器的日誌串流，來確認問題的根源。

AWS Glue 可觀測性指標的使用案例

- 進行 OOM 例外與任務異常的除錯
- 針對高要求的階段和落後任務進行除錯
- 監控多個任務的進度
- DPU 容量規劃監控
- 使用 AWS Glue 可觀測性監控資源使用率以降低成本

進行 OOM 例外與任務異常的除錯

您可以針對 AWS Glue 中的記憶體不足 (OOM) 例外和任務異常進行除錯。下列的段落所說明的除錯案例，是針對 Apache Spark 驅動程式或 Spark 執行器的記憶體不足 (OOM) 例外進行除錯。

- 進行驅動程式 OOM 例外的除錯
- 進行執行器 OOM 例外的除錯

進行驅動程式 OOM 例外的除錯

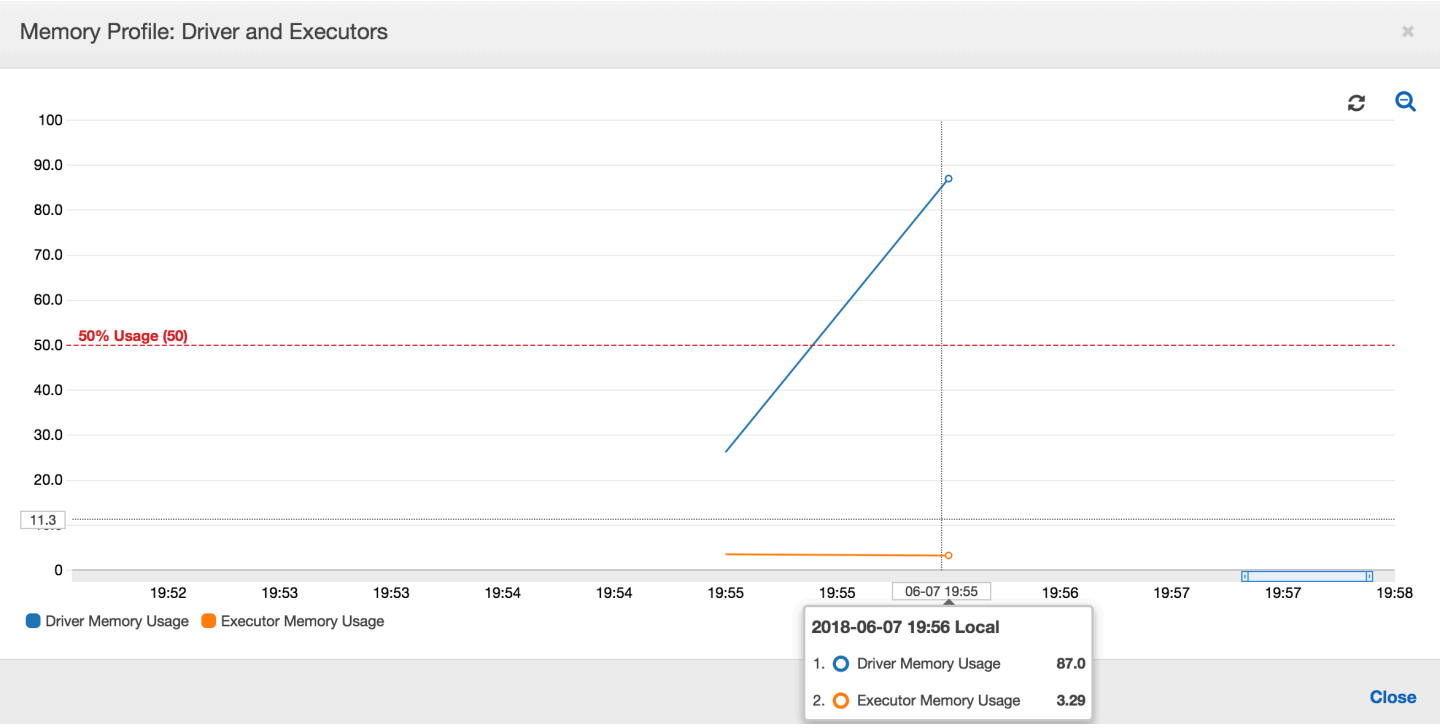
在這個案例中，Spark 任務會從 Amazon Simple Storage Service (Amazon S3) 讀取大量的小型檔案。接著將檔案轉換成 Apache Parquet 格式，然後寫入到 Amazon S3。Spark 驅動程式的記憶體即將用盡。輸入的 Amazon S3 資料在不同的 Amazon S3 分割區中，具有超過 100 萬個檔案。

分析程式碼如下：

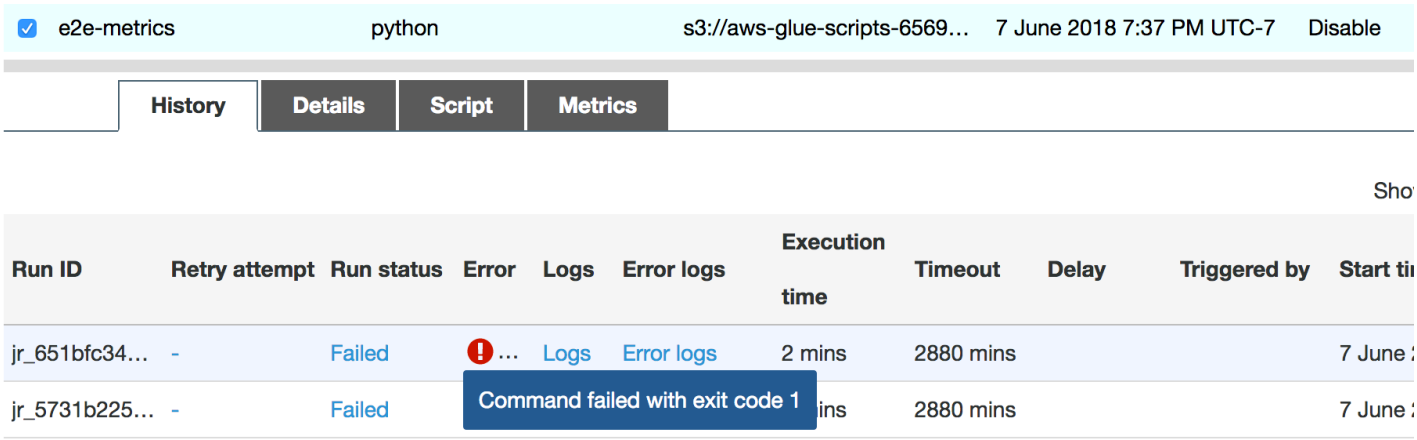
```
data = spark.read.format("json").option("inferSchema", False).load("s3://input_path")
data.write.format("parquet").save(output_path)
```

在 AWS Glue 主控台上視覺化已分析的指標

下列的圖表顯示驅動程式和執行器的記憶體使用量百分比。此用量曲線是由一個資料點繪製而成，而此資料點是過去一分鐘內所呈報值的平均值。您可以看到在任務的記憶體使用狀況圖中，驅動程式記憶體很快地就超過 50% 的安全使用量門檻。另一方面，所有執行器的整體平均記憶體使用量仍然低於 4%。這明顯地顯示出這項 Spark 任務中驅動程式執行的異常狀況。



工作的執行很快地失敗，在 主控台的 History (歷程記錄)AWS Glue 索引標籤中，顯示了下列的錯誤訊息：Command Failed with Exit Code 1 (指令執行失敗，結束代碼 1)。這個錯誤字串代表任務因系統錯誤而失敗，在此案例中，是因為驅動程式的記憶體用盡。



在主控台的 History (歷程記錄) 索引標籤中，選擇 Error logs (錯誤日誌) 連結，來確認 CloudWatch Logs 所提供關於驅動程式 OOM 的資訊。在任務的錯誤日誌中搜尋「**`Error`**」，確認這的確是讓任務失敗的 OOM 例外：

```
# java.lang.OutOfMemoryError: Java heap space
# -XX:OnOutOfMemoryError="kill -9 %p"
# Executing /bin/sh -c "kill -9 12039"...
```

在任務的 History (歷程記錄) 索引標籤中，選擇 Logs (日誌)。您可於任務開始時在 CloudWatch Logs 找到驅動程式執行的追蹤。Spark 驅動程式會試著列出所有目錄中的所有檔案、建立 `InMemoryFileIndex`，並針對每個檔案啟動一個任務。這會導致 Spark 驅動程式必須在記憶體中維持大量的狀態顯示，以追蹤所有的任務。該驅動程式會針對大量檔案的完整清單建立快取，以維持記憶體內的索引，進而造成驅動程式的 OOM。

利用分組功能來修正處理多個檔案的問題

您可以利用 中的分組AWS Glue功能，來修正處理多個檔案的問題。使用動態框架時，以及輸入資料集包含大量檔案 (超過 50,000 個) 時，分組功能就會自動啟用。分組功能可讓您將多個檔案合併為一個群組，讓任務處理整個群組而非單一檔案。因此，Spark 驅動程式儲存於記憶體中的狀態顯示會大幅減少，追蹤的任務也減少。關於手動啟用資料集的分組功能，詳細資訊請參閱 讀取在大型群組中的輸入檔案。

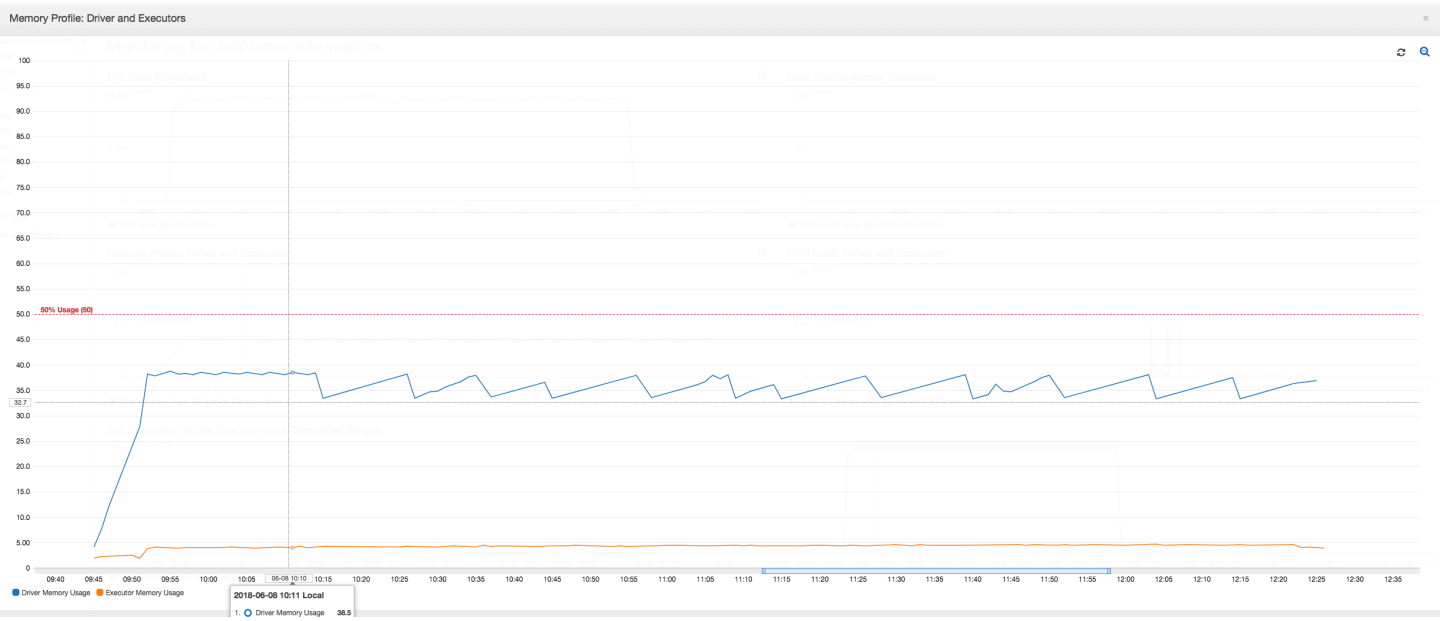
若要查看 AWS Glue 任務的記憶體使用狀況，請在啟用分組功能的狀況下，分析下列程式碼：

```
df = glueContext.create_dynamic_frame_from_options("s3", {'paths': ["s3://input_path"],
 "recurse":True, 'groupFiles': 'inPartition'}, format="json")
```

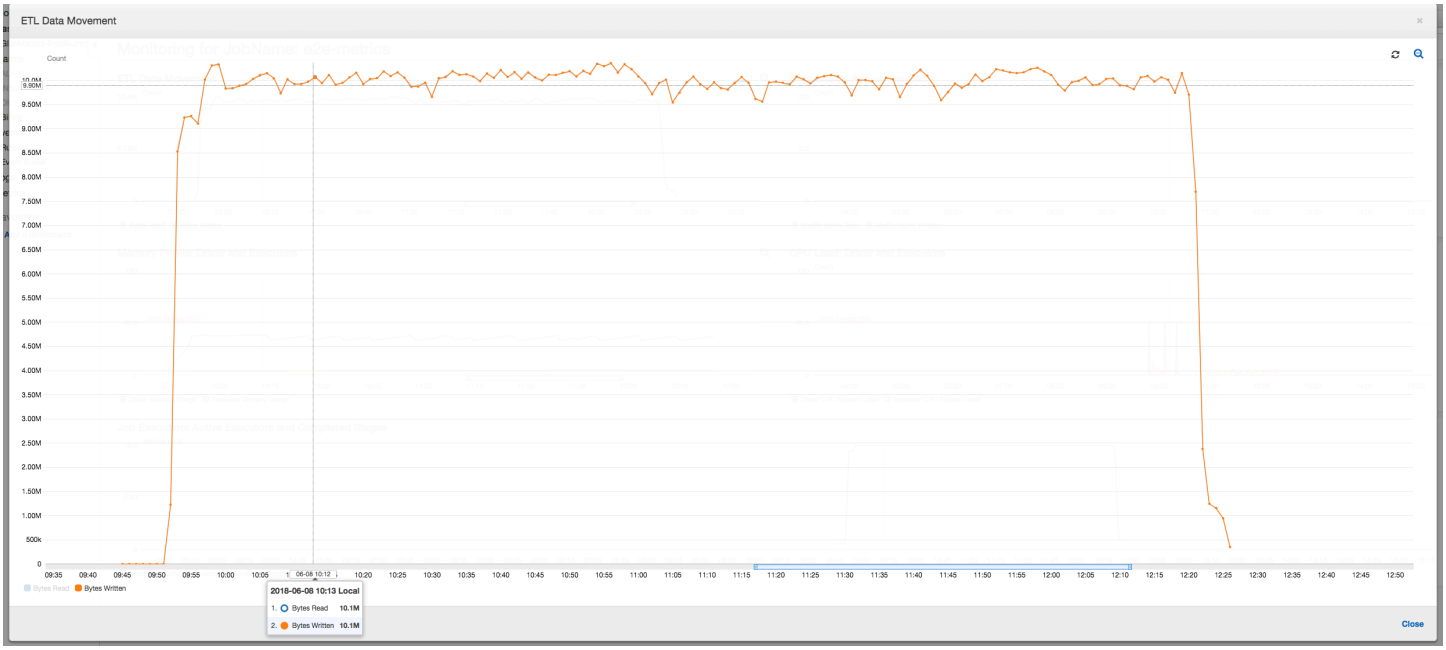
```
datasink = glueContext.write_dynamic_frame.from_options(frame = df, connection_type
 = "s3", connection_options = {"path": output_path}, format = "parquet",
 transformation_ctx = "datasink")
```

您可以在 AWS Glue 任務狀況分析中，監控記憶體的使用狀況和 ETL 資料的移動。

在 AWS Glue 任務的整個任務期間內，驅動程式都會以低於 50% 的記憶體使用量閥值來執行。執行器會串流來自 Amazon S3 的資料、進行處理，然後將資料輸出寫入到 Amazon S3。因此，執行器在任何時點所使用的記憶體皆少於 5%。



下列的資料移動狀況圖，顯示了所有執行器隨著任務的進行，過去一分鐘內讀取和寫入的 Amazon S3 位元組總數。這兩種動作都會遵循相同的模式，讓資料分散透過所有執行器串流。這項任務會在不到三小時內處理完所有的一百萬個檔案。



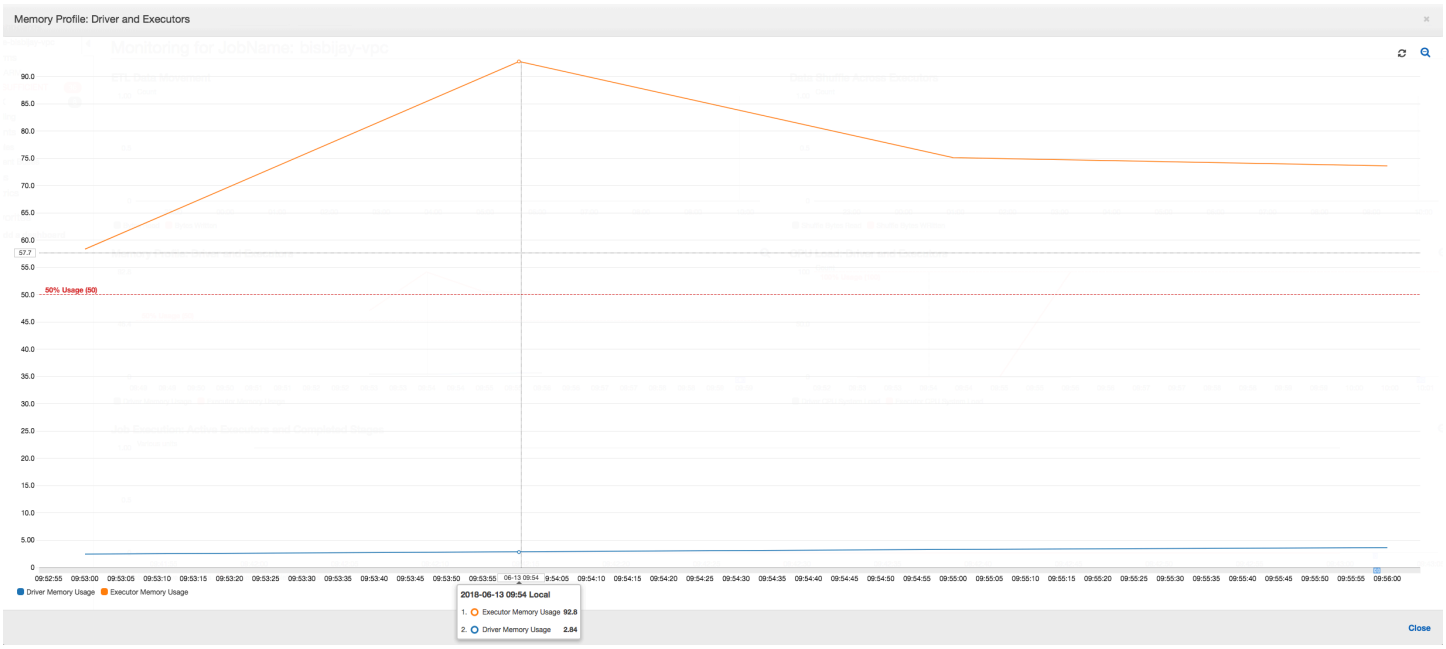
進行執行器 OOM 例外的除錯

在此案例中，您可以了解如何針對 Apache Spark 執行器可能會發生的 OOM 例外，來進行除錯。下列的程式碼使用了 Spark MySQL 讀取器，來將包含約 3,400 萬行的大型資料表，讀取到 Spark DataFrame 中。接著再以 Parquet 格式，將資料輸出寫入至 Amazon S3。您可以提供連線屬性，並使用預設的 Spark 組態來讀取資料表。

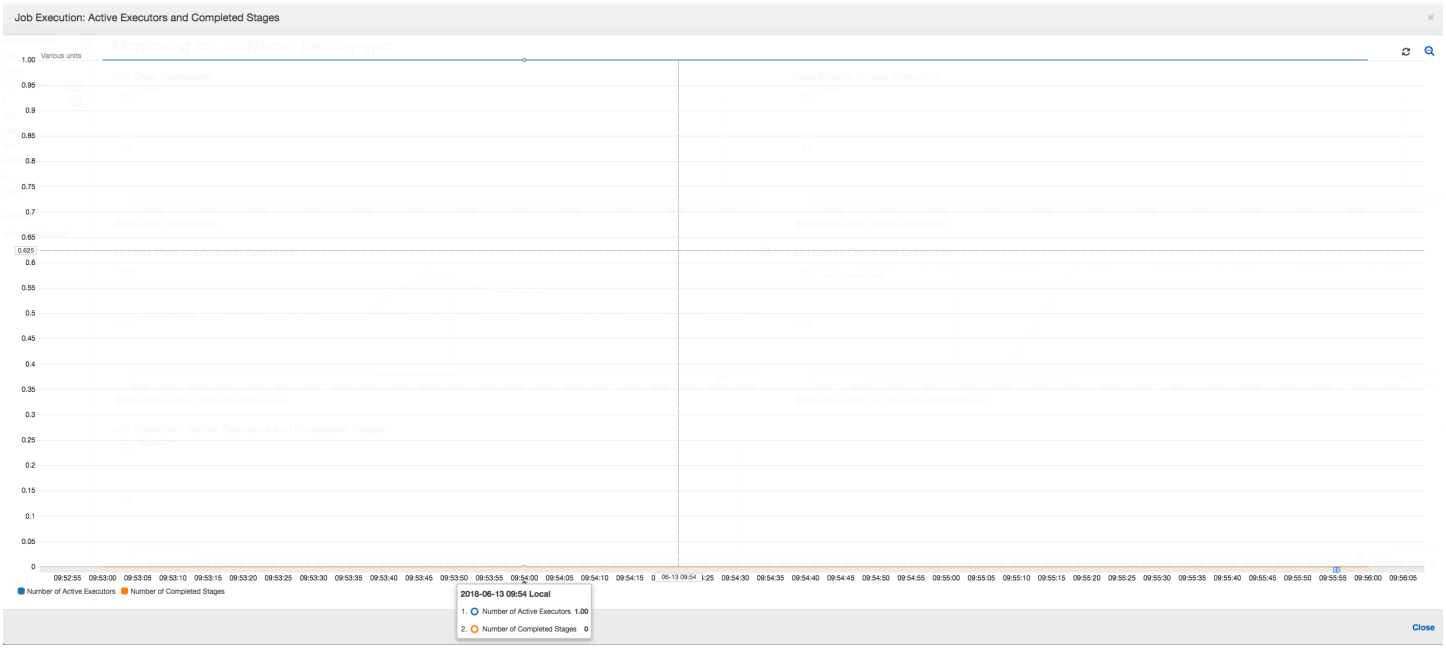
```
val connectionProperties = new Properties()
connectionProperties.put("user", user)
connectionProperties.put("password", password)
connectionProperties.put("Driver", "com.mysql.jdbc.Driver")
val sparkSession = glueContext.sparkSession
val dfSpark = sparkSession.read.jdbc(url, tableName, connectionProperties)
dfSpark.write.format("parquet").save(output_path)
```

在 AWS Glue 主控台上視覺化已分析的指標

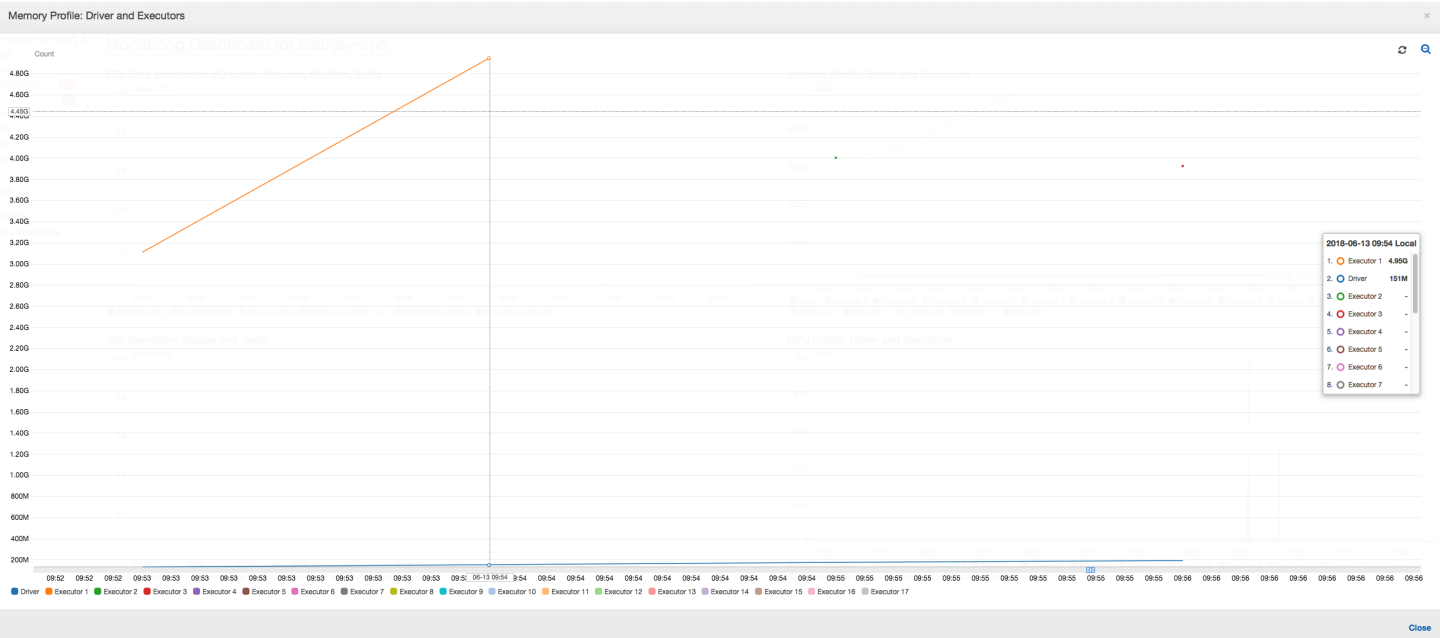
如果記憶體使用量圖表的斜率為正值且跨越 50%，則如果在發出下一個指標之前任務失敗，則記憶體耗盡很可能是造成失敗的原因。下列圖表顯示執行開始的一分鐘內，所有執行器的平均記憶體使用量快速暴增超過 50%。使用量達到 92%，而執行此執行器的容器被 Apache Hadoop YARN 停止。



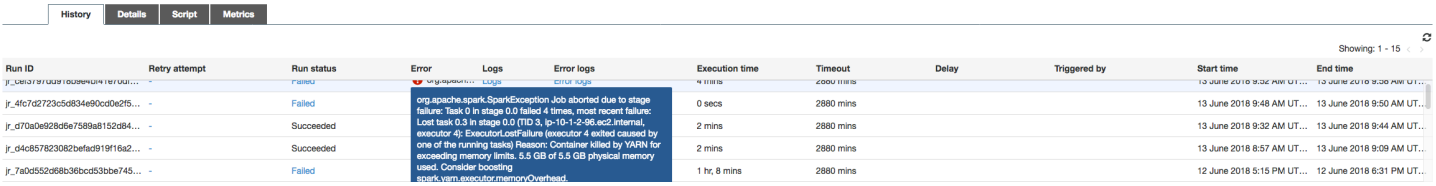
如下列圖表所示，到任務失敗之前，一直都有單一執行器在執行。這是由於啟動了新的執行器來取代已停止的執行器。JDBC 資料來源的讀取，預設並非平行進行，因為這會需要從欄分割資料表，並開啟多個連線。因此，只有一個執行器會以序列方式讀取完整的資料表。



如下列圖表所示，在任務失敗之前，Spark 嘗試啟動新的任務四次。您可以查看三個執行器的記憶體使用狀況。每個執行器很快地就用完自己所有的記憶體。當第四個執行器用完記憶體後，任務就失敗了。因此，並未立即呈報其指標。



您可以從 AWS Glue 主控台上的錯誤字串，確認任務是因為 OOM 例外而失敗，如下列圖像所示。



任務輸出日誌：為了進一步確認執行器 OOM 例外的情況，請檢視 CloudWatch Logs。搜尋 **`Error`** 時，您會發現四個執行器大約是在相同的期間內遭到停止，如指標儀表板上所示。因為超過了記憶體使用量的限制，這些執行器全都遭到 YARN 終止。

執行器 1

```
18/06/13 16:54:29 WARN YarnAllocator: Container killed by YARN for exceeding
 memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:54:29 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed
 by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider
 boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:54:29 ERROR YarnClusterScheduler: Lost executor 1 on
 ip-10-1-2-175.ec2.internal: Container killed by YARN for exceeding
 memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:54:29 WARN TaskSetManager: Lost task 0.0 in stage 0.0 (TID 0,
 ip-10-1-2-175.ec2.internal, executor 1): ExecutorLostFailure (executor 1
 exited caused by one of the running tasks) Reason: Container killed by YARN for
```

```
 exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
```

執行器 2

```
18/06/13 16:55:35 WARN YarnAllocator: Container killed by YARN for exceeding
 memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:55:35 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed
 by YARN for exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider
 boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:55:35 ERROR YarnClusterScheduler: Lost executor 2 on
 ip-10-1-2-16.ec2.internal: Container killed by YARN for exceeding
 memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:55:35 WARN TaskSetManager: Lost task 0.1 in stage 0.0 (TID 1,
 ip-10-1-2-16.ec2.internal, executor 2): ExecutorLostFailure (executor 2 exited
 caused by one of the running tasks) Reason: Container killed by YARN for
 exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
```

執行器 3

```
18/06/13 16:56:37 WARN YarnAllocator: Container killed by YARN for exceeding
 memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:56:37 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed
 by YARN for exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider
 boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:56:37 ERROR YarnClusterScheduler: Lost executor 3 on
 ip-10-1-2-189.ec2.internal: Container killed by YARN for exceeding
 memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:56:37 WARN TaskSetManager: Lost task 0.2 in stage 0.0 (TID 2,
 ip-10-1-2-189.ec2.internal, executor 3): ExecutorLostFailure (executor 3
 exited caused by one of the running tasks) Reason: Container killed by YARN for
 exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
```

執行器 4

```
18/06/13 16:57:18 WARN YarnAllocator: Container killed by YARN for exceeding
 memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:57:18 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed
 by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider
 boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:57:18 ERROR YarnClusterScheduler: Lost executor 4 on
 ip-10-1-2-96.ec2.internal: Container killed by YARN for exceeding
 memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:57:18 WARN TaskSetManager: Lost task 0.3 in stage 0.0 (TID 3,
 ip-10-1-2-96.ec2.internal, executor 4): ExecutorLostFailure (executor 4 exited
 caused by one of the running tasks) Reason: Container killed by YARN for
 exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
```

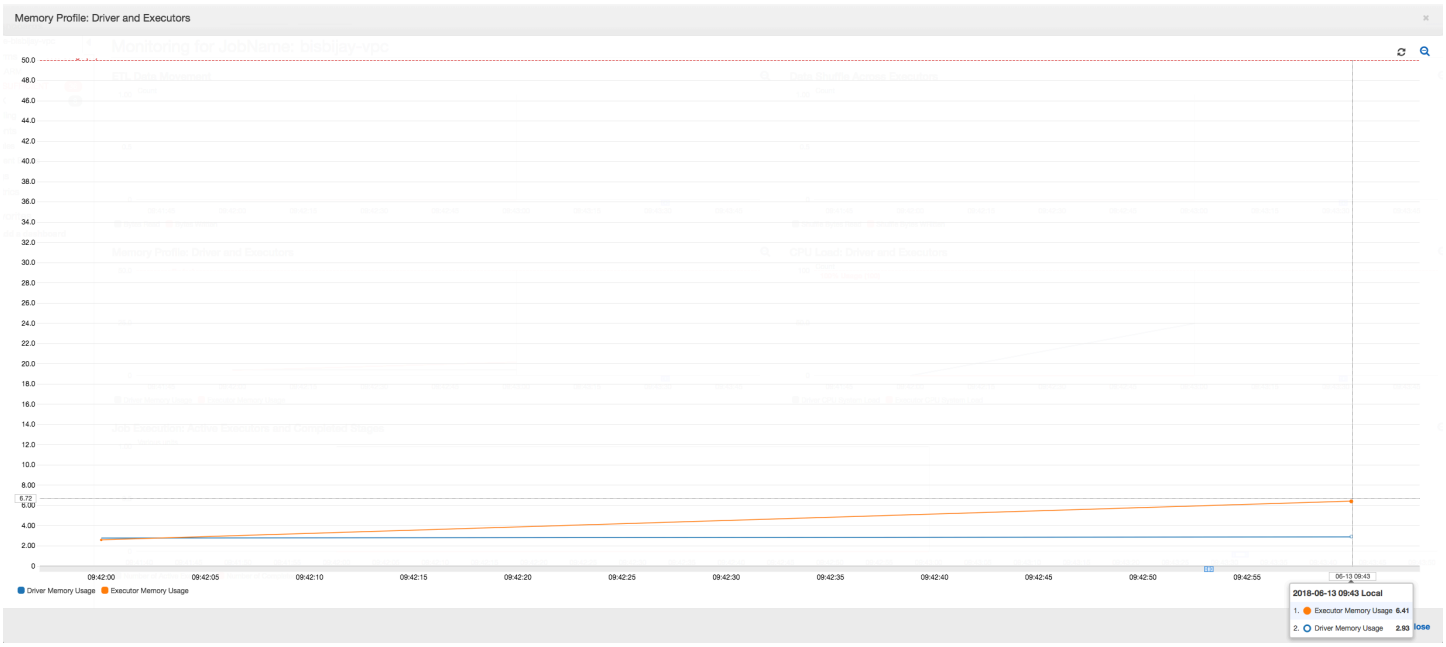
利用 AWS Glue 動態框架來修正擷取大小

由於預設的 Spark JDBC 擷取大小為 0，因此執行器在讀取 JDBC 資料表時用盡了記憶體。這代表 Spark 執行器上的 JDBC 驅動程式，試著一次從資料庫擷取 3,400 萬行的資料，並建立這些資料的快取 (即使 Spark 是一次一行的進行串流作業)。使用 Spark 時，您可以將擷取大小的參數，設定為非 0 的預設值，來避免這種狀況發生。

您也可以改用 AWS Glue 動態框架，來修正這個問題。在預設情況下，動態框架會使用 1,000 個資料列的擷取大小，通常這個設定值已經夠用。因此，執行器不會使用超過總記憶體 7% 的記憶體量。AWS Glue 任務只會使用單一執行器，在不到兩分鐘內完成。雖然我們建議使用 AWS Glue 動態框架，但是您也可以使用 Apache Spark `fetchsize` 屬性來設定擷取大小。請參閱 Spark SQL、DataFrames 和資料集指南。

```
val (url, database, tableName) = {
 ("jdbc_url", "db_name", "table_name")
 }
val source = glueContext.getSource(format, sourceJson)
val df = source.getDynamicFrame
glueContext.write_dynamic_frame.from_options(frame = df, connection_type = "s3",
 connection_options = {"path": output_path}, format = "parquet", transformation_ctx =
 "datasink")
```

正常分析指標：使用 動態框架的執行器記憶體AWS Glue絕對不會超過安全門檻，如下列圖像所示：此等記憶體會從資料庫串流資料行，而且任何時點只會針對 JDBC 驅動程式中的 1,000 行建立快取。不會發生記憶體不足的例外狀況。



針對高要求的階段和落後任務進行除錯

您可以利用 AWS Glue 任務分析功能，在擷取、轉換和載入 (ETL) 任務中，找出高要求階段和落後任務。在 AWS Glue 任務的階段中，落後任務所花費的作業時間，遠多於其他的任務。因此，此等任務需要花較長的時間完成作業，也延遲了任務的總執行時間。

將小型的輸入檔案合併為較大的輸出檔案

當不同任務之間的作業量分配不均，或是資料傾斜造成某個任務需要處理更多的資料時，就可能會出現落後的任務。

您可以分析下列程式碼 (在 Apache Spark 中的通用模式)，來將大量的小型檔案合併為較大的輸出檔案。在這個範例中，輸入資料集是 32 GB 的 JSON Gzip 壓縮檔案。輸出資料集約包含 190 GB 的未壓縮 JSON 檔案。

分析程式碼如下：

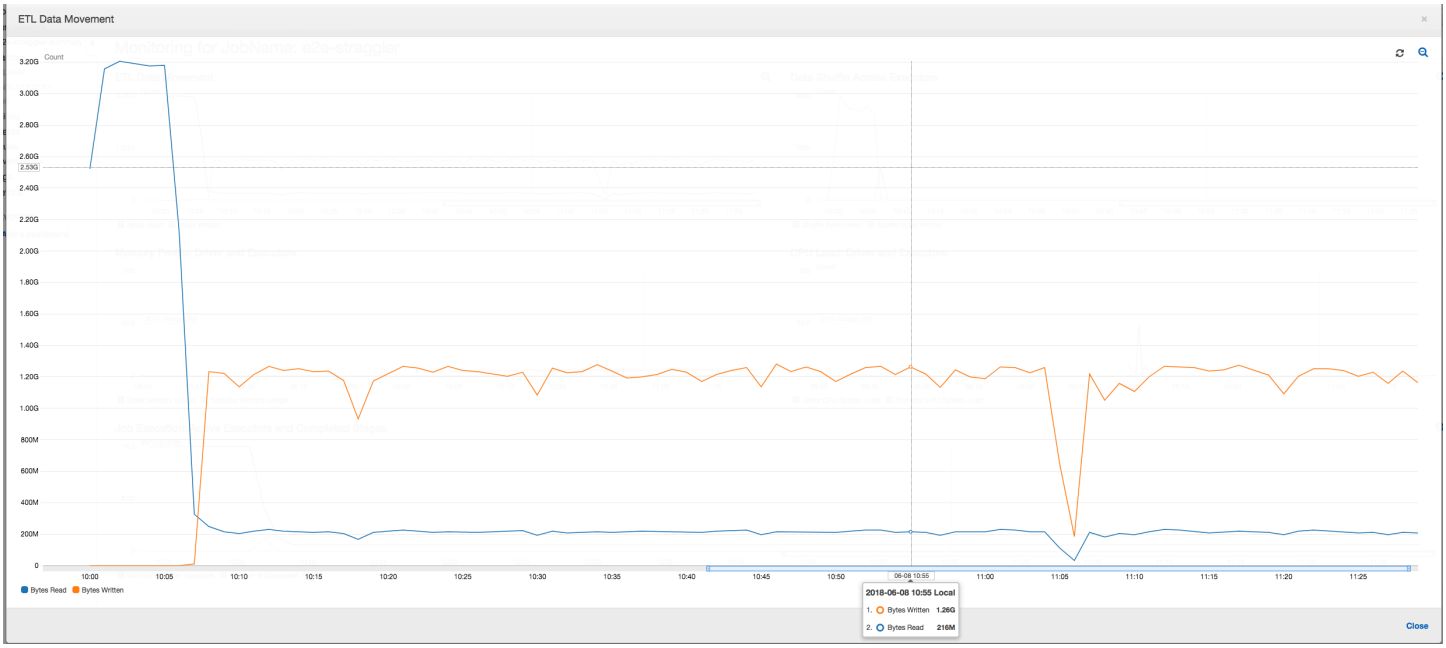
```
datasource0 = spark.read.format("json").load("s3://input_path")
df = datasource0.coalesce(1)
df.write.format("json").save(output_path)
```

在 AWS Glue 主控台上視覺化已分析的指標

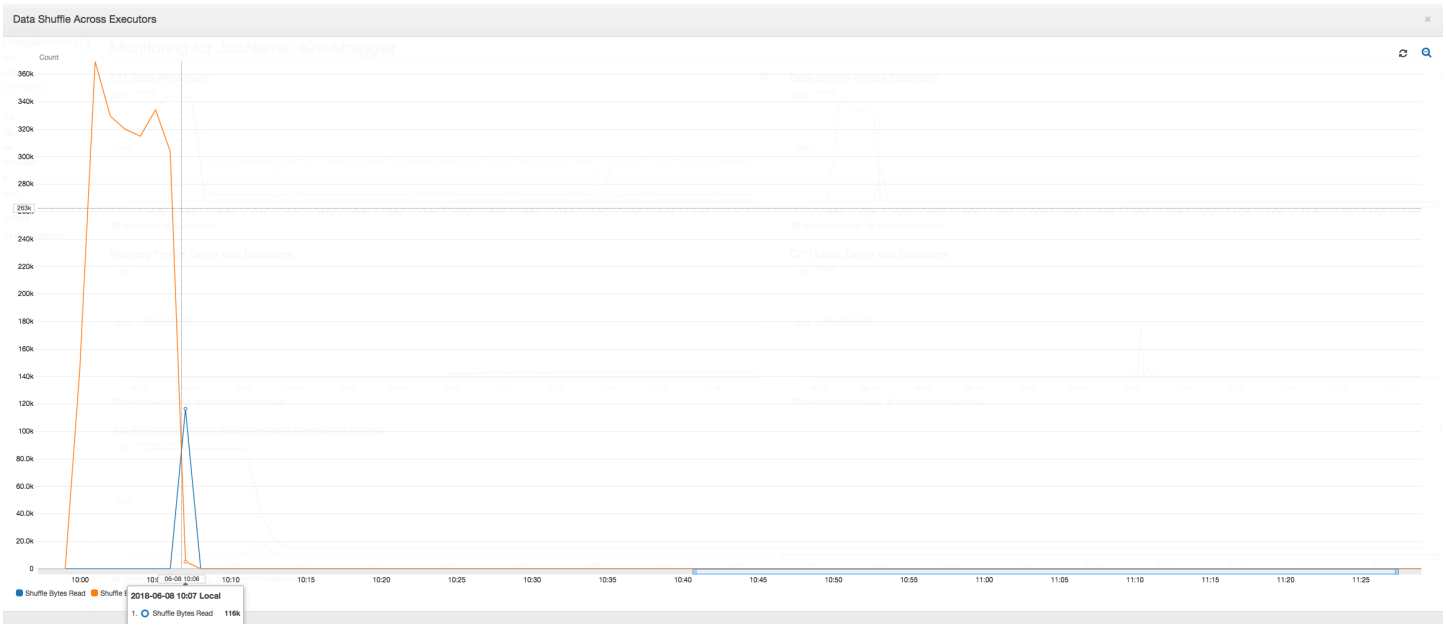
您可以分析任務，檢驗四組不同的指標：

- ETL 資料移動
- 在執行器之間的資料隨機移動
- 任務執行
- 記憶體使用狀況

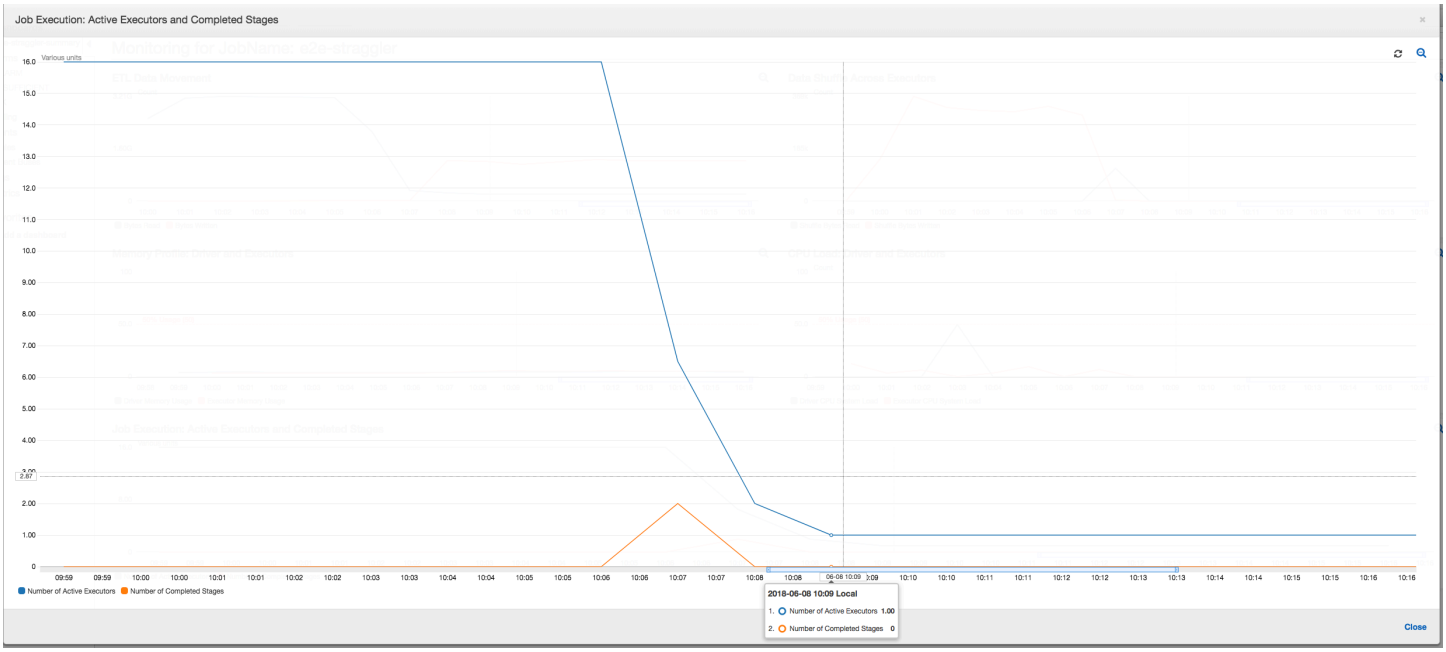
ETL 資料移動：在 ETL Data Movement (ETL 資料移動) 分析圖表中，所有執行器在前 6 分鐘內就完成的第一個階段中，相當快速地讀取資料 (單位：位元組)。不過，任務執行的總時間約為一個小時，大部分為資料寫入作業。



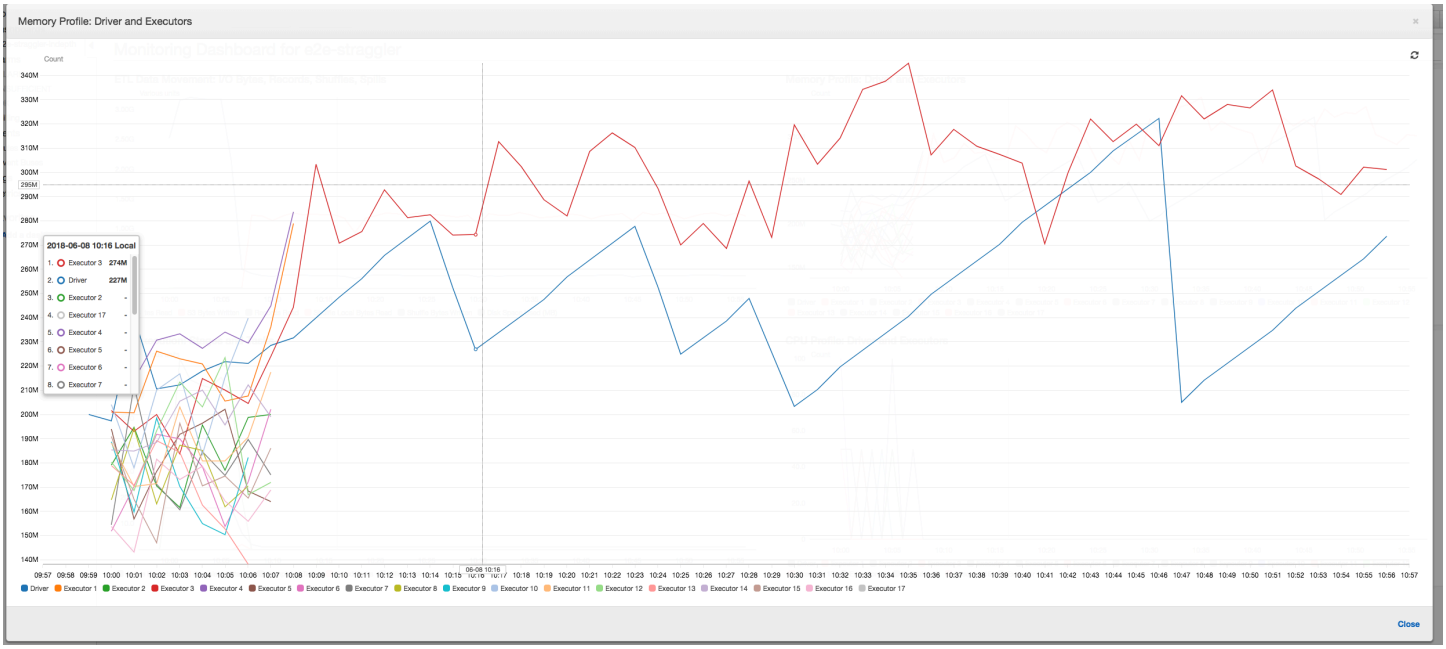
執行器之間的資料隨機移動：在第二階段結束之前，隨機移動資料期間讀取和寫入的位元組數量出現突增的峰值，如 Job Execution (任務執行) 和 Data Shuffle (資料隨機移動) 指標所示。在所有執行器隨機移動資料之後，就只從第 3 號執行器進行讀取和寫入。



任務執行：如下列圖表所示，其他所有的執行器皆處於閒置狀態，最後在 10:09 之前終止。在這個時間點，執行器的總數減少到只剩下一個。這清楚地顯示出，第 3 號執行器包含落後任務，此任務所花費的執行時間最長，而且佔了大部分的任務執行時間。



記憶體使用狀況：經過前兩階段後，只有第 3 號執行器正在使用記憶體來運作處理資料。剩下的執行器只是處於閒置狀態，或是在前兩個階段完成之後，就很快地遭到終止。



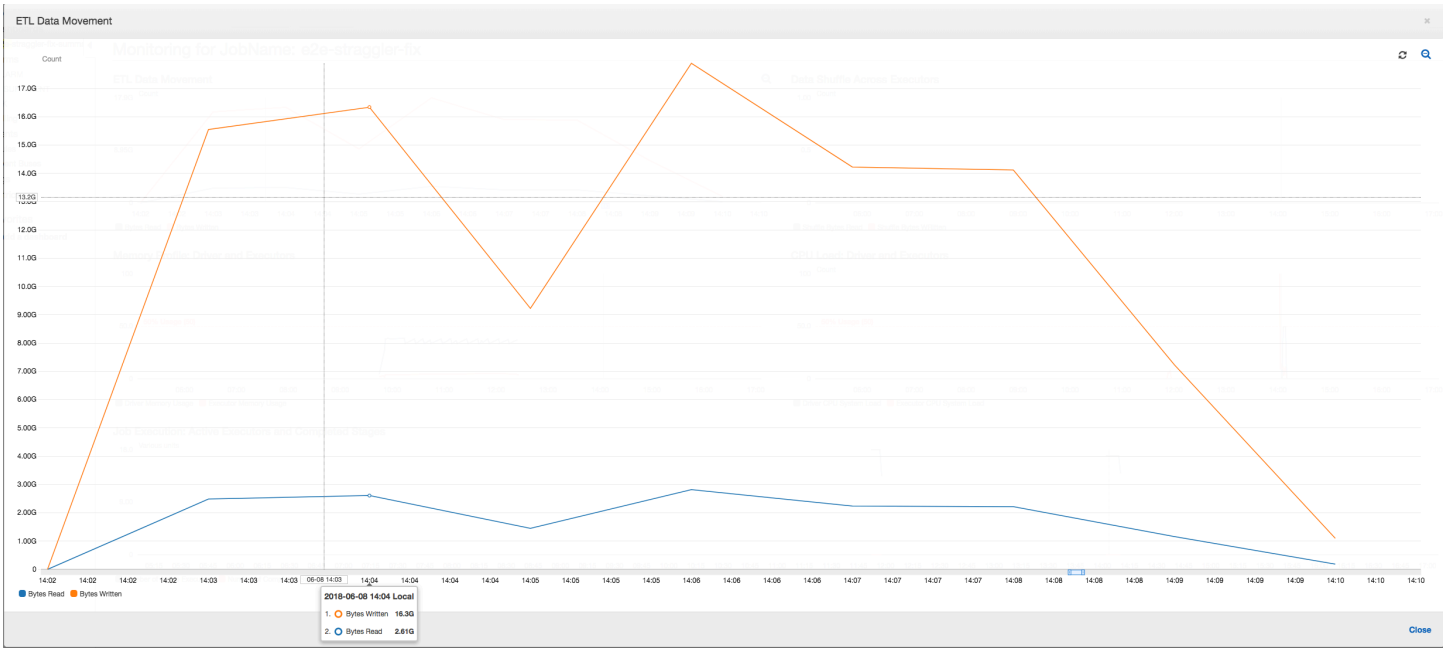
利用分組來解決落後執行器的問題

您可以利用 中的分組AWS Glue功能，來避免出現落後的執行器。利用分組功能，將資料平均地分配給所有執行器，並使用叢集上所有可用的執行器，將檔案合併為更大型的檔案。如需詳細資訊，請參閱讀取在大型群組中的輸入檔案。

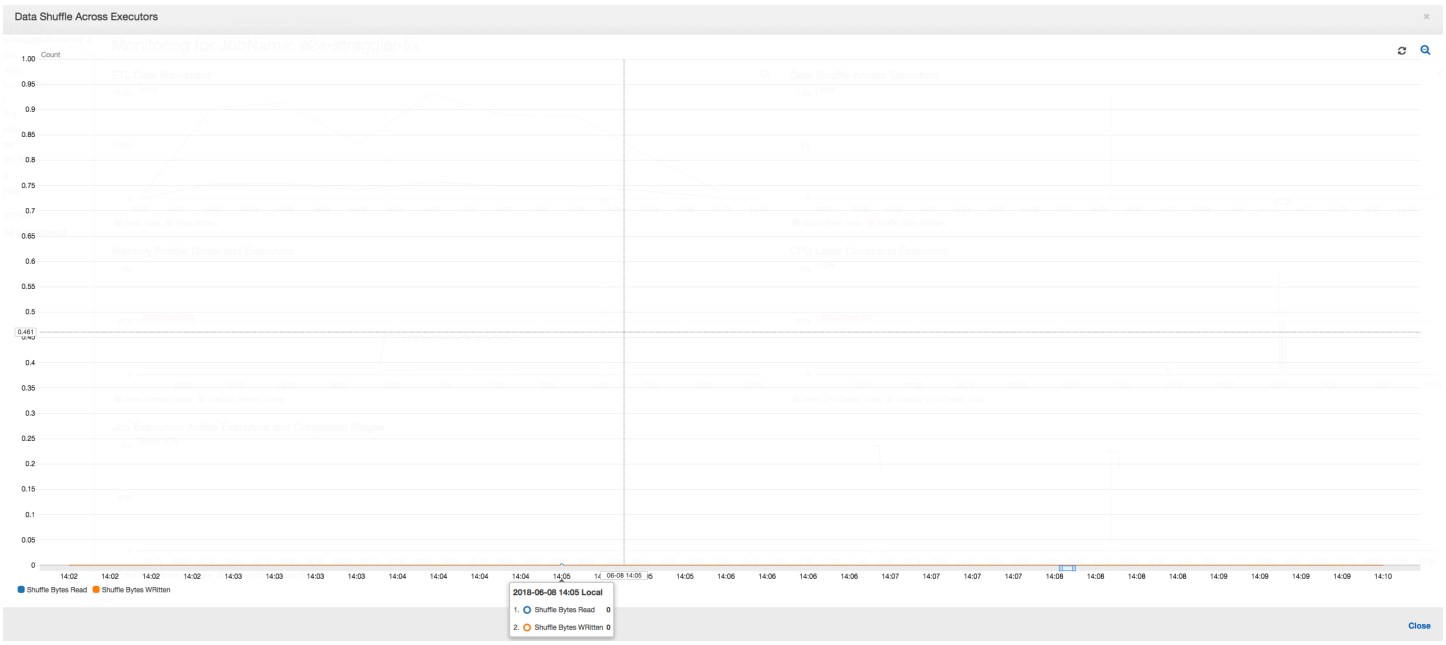
若要查看 AWS Glue 任務中的 ETL 資料移動狀況，請在啟用分組功能的情況下，分析下列程式碼：

```
df = glueContext.create_dynamic_frame_from_options("s3", {'paths': ["s3://input_path"],
 "recurse":True, 'groupFiles': 'inPartition'}, format="json")
datasink = glueContext.write_dynamic_frame.from_options(frame = df, connection_type =
 "s3", connection_options = {"path": output_path}, format = "json", transformation_ctx
 = "datasink4")
```

ETL 資料移動：在整個任務執行期間，資料寫入作業現在會與資料讀取作業並行串流。因此，任務會在 8 分鐘之內完成，速度比之前快上許多。

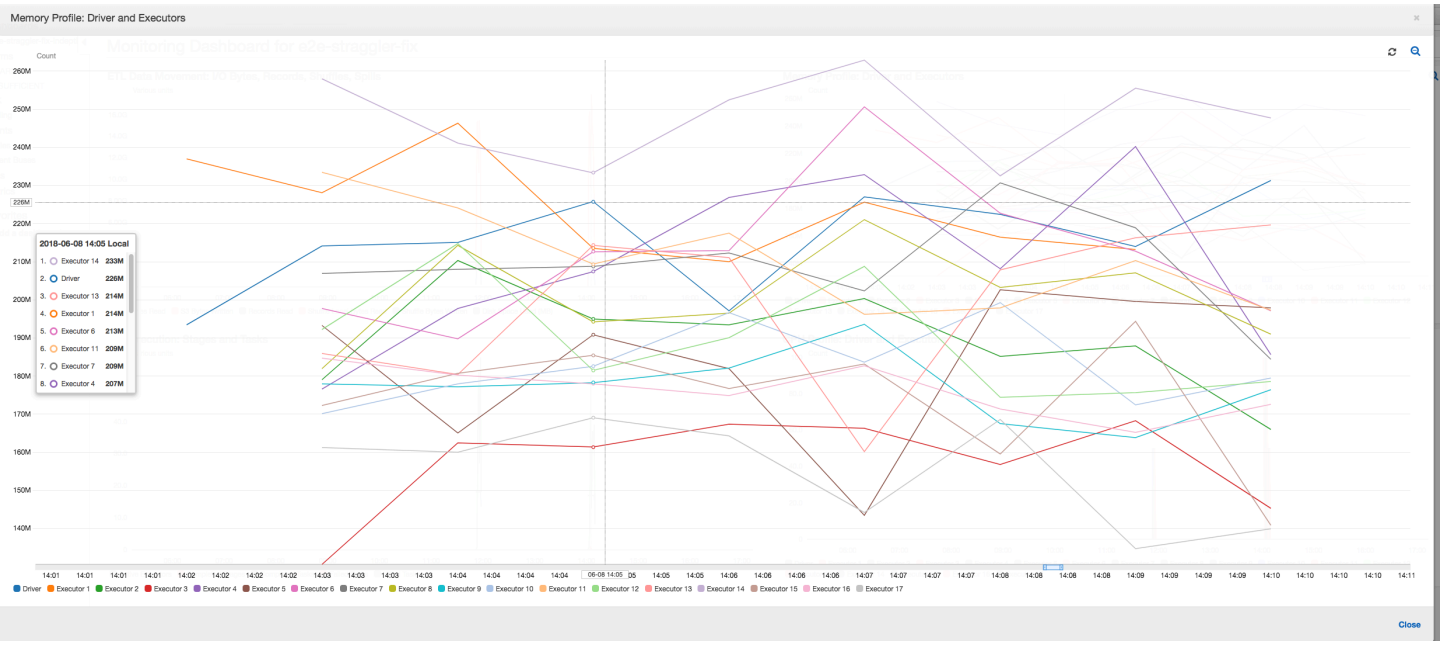


執行器之間的資料隨機移動：在進行讀取時，使用分組功能將輸入檔案合併之後，就不用在資料讀取之後，進行成本昂貴的資料隨機移動作業。



任務執行：任務執行指標會顯示運作中執行器的總數，以及處理的資料維持地相當穩定。在任務中沒有任何落後作業。所有的執行器皆處於運作狀態，在任務尚未完成之前，不會遭到終止。由於不需在執行器之間進行中間的資料隨即移動作業，因此任務中只有一個階段。

記憶體使用狀況：此指標顯示所有執行器的現用記憶體使用量—重新確認所有的執行器都在活動中。隨著資料同時串流傳入和輸出寫入，所有執行器使用的總記憶體大致平均，而且遠低於所有執行器的安全門檻值。



監控多個任務的進度

您可以分析多個 AWS Glue 任務，並監控這些任務之間的資料流量。這是一個常見的任務流程模式，且需要監控個別任務進度、資料處理後端紀錄、資料重新處理和任務書籤。

主題

- 已分析的程式碼
- 在 AWS Glue 主控台上視覺化已分析的指標
- 修正檔案處理

已分析的程式碼

在此任務流程中，您有兩個任務：輸入任務和輸出任務。輸入任務透過定期觸發排定每隔 30 分鐘執行。在每個輸入任務成功執行之後，即排定要執行的輸出任務。這些排定的任務皆由任務觸發器控制。

輸入任務：此任務從 Amazon Simple Storage Service (Amazon S3) 位置讀取的資料，會使用 `ApplyMapping` 轉換該任務，並將其寫入至暫存 Amazon S3 位置。以下程式碼是已分析的輸入任務程式碼：

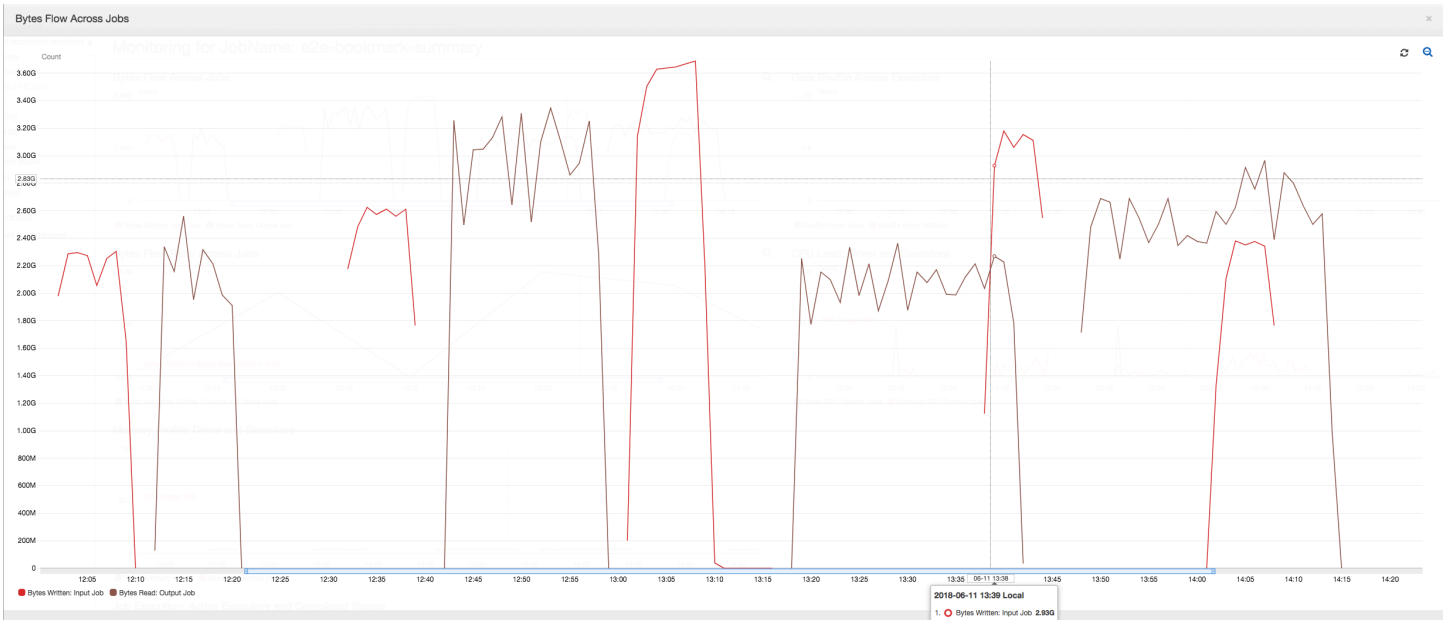
```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
 connection_options = {"paths": ["s3://input_path"],
 "useS3ListImplementation":True,"recurse":True}, format="json")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [map_spec])
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
 connection_type = "s3", connection_options = {"path": staging_path, "compression":
 "gzip"}, format = "json")
```

輸出任務：此任務從 Amazon S3 的暫存位置讀取輸入任務中的輸出，將其再次轉換並寫入至目的地：

```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
 connection_options = {"paths": [staging_path],
 "useS3ListImplementation":True,"recurse":True}, format="json")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [map_spec])
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
 connection_type = "s3", connection_options = {"path": output_path}, format = "json")
```

在 AWS Glue 主控台上視覺化已分析的指標

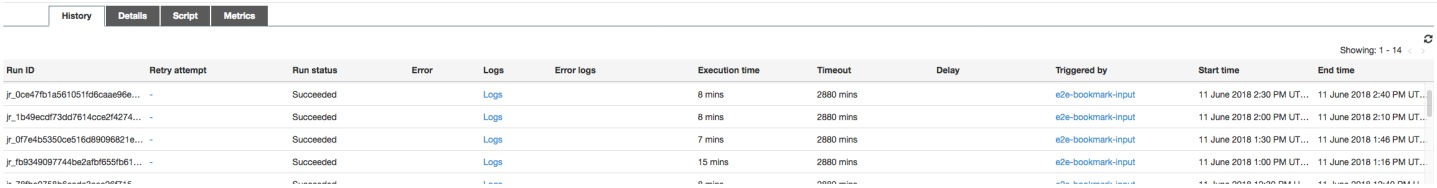
在相同的輸出任務時間軸上，以下儀表板會將輸入任務的 Amazon S3 位元組寫入指標，加乘至 Amazon S3 位元組讀取指標。該時間軸顯示了輸入與輸出任務的不同任務執行。輸入任務 (紅色) 每 30 分鐘啟動一次。輸出任務 (褐色) 會於輸入任務完成時啟動，最大並行數量為 1。



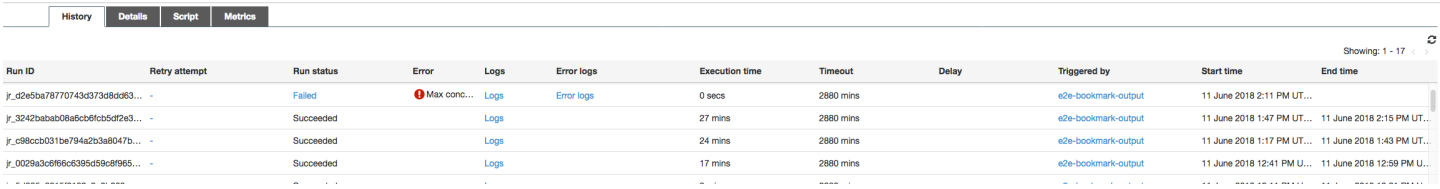
在此範例中並未啟用任務書籤。在該指令碼程式碼中沒有使用轉換細節來啟用任務書籤。

任務歷史記錄：從下午 12 點開始，輸入和輸出任務會有多項執行，如 History (歷史記錄) 標籤中所示。

AWS Glue 主控台的輸入任務如下所示：

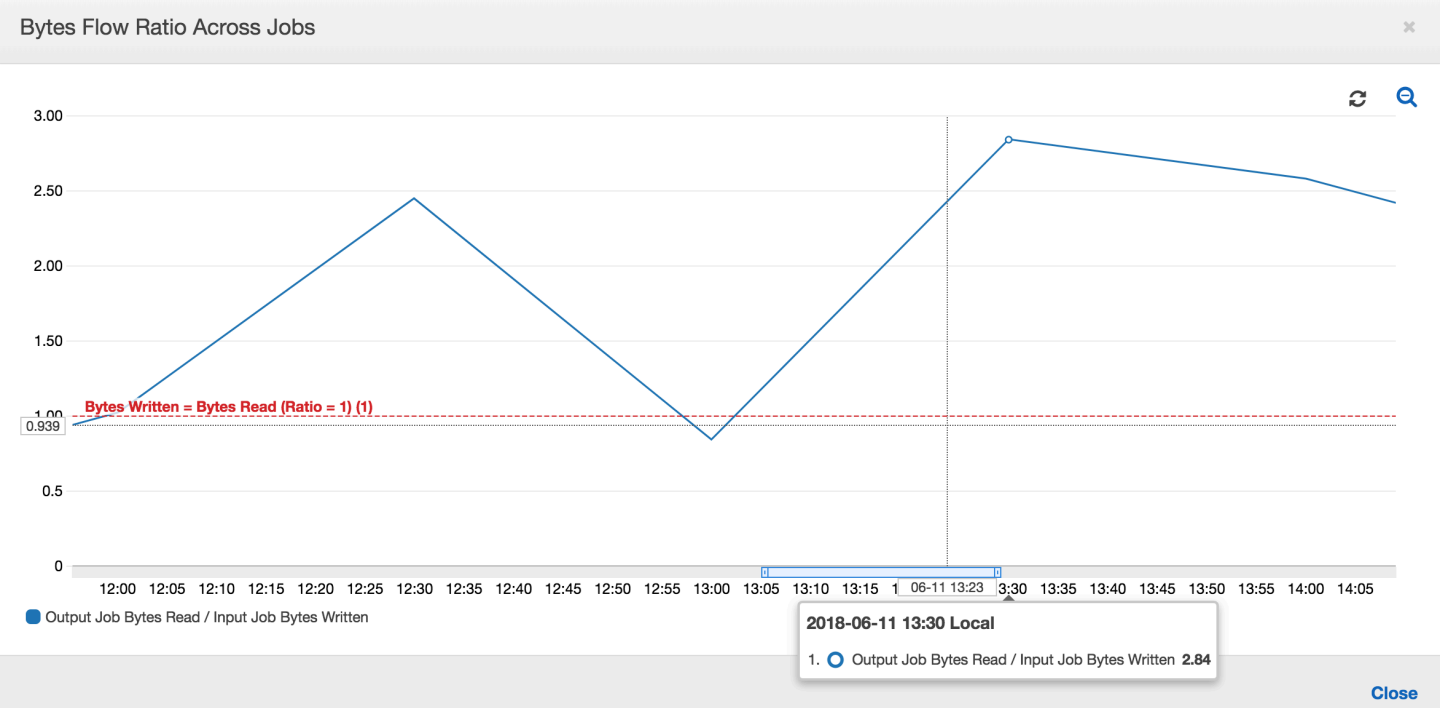


下列映像顯示了輸出任務：



第一個任務執行：如下方的 Data Bytes Read and Written (資料位元組讀取和寫入圖形) 所示，在 12:00 和 12:30 之間的第一個輸入與輸出任務的任務執行，顯示了曲線下大致的相同區域。那些領域代表了輸入任務寫入的 Amazon S3 位元組，以及輸出任務讀取的 Amazon S3 位元組。此資料也由 Amazon S3 位元組寫入的比例所確認 (超過 30 分鐘的加總 - 輸入任務的任務觸發器頻率)。在下午 12 點開始的輸入任務執行比例資料點也是 1。

以下圖表顯示所有任務執行的資料流程比例：



第二個任務執行：在第二個任務執行中，輸出任務讀取的位元數和輸入任務寫入的位元數之間有清楚的差異。(在輸出任務的兩個任務執行中比較曲線下的區域，或在輸入與輸出任務中的第二個執行中比較區域。) 讀取和寫入的位元組比例，顯示第二個範圍 (從 12:30 至 13:00) 的 30 分鐘內，輸出任務資料讀取約是輸入任務資料寫入的 2.5 倍。這是因為任務書籤並未啟用，故輸出任務重新處理了輸入任務的第一個任務執行的輸出。超過 1 的比率，說明有由輸出任務處理的額外資料後端紀錄。

第三個任務執行：輸入任務與寫入的位元組數相當的一致 (請參閱紅色曲線以下的區域)。不過，輸入任務第三個任務執行的執行時間超出預期 (請參閱紅色曲線的尾部)。因此，輸出任務的第三個任務執行會延遲開始。第三個任務執行僅在 13:00 到 13:30 之間剩餘的 30 分鐘內處理暫存位置中累積的一小部分資料。位元組流程的比例，顯示它僅會處理由輸入任務的第三個任務執行寫入資料的 0.83 (參見 13:00 時的比例)。

輸入和輸出任務的重疊：輸出任務的第四個任務執行依各排程在 13:30 開始，即輸出任務的第三個任務執行完成之前。這兩個任務執行間有部分重疊。不過，輸出任務的第三個任務執行約在 13:17 開始，它僅擷取 Amazon S3 暫存位置中列出的檔案。這包含了輸入任務的第一個任務執行的所有資料輸出。13:30 的實際比例為 2.75。從 13:30 至 14:00 間，輸出任務的第三個任務執行處理的寫入資料，是輸入任務的第四個任務執行的 2.75 倍。

如這些映像所示，輸出任務正在重新處理資料，這些資料位於輸入任務的所有優先任務暫存位置。因此，輸出任務的第四個任務執行時間最長，並與輸入任務的整個第五個任務執行重疊。

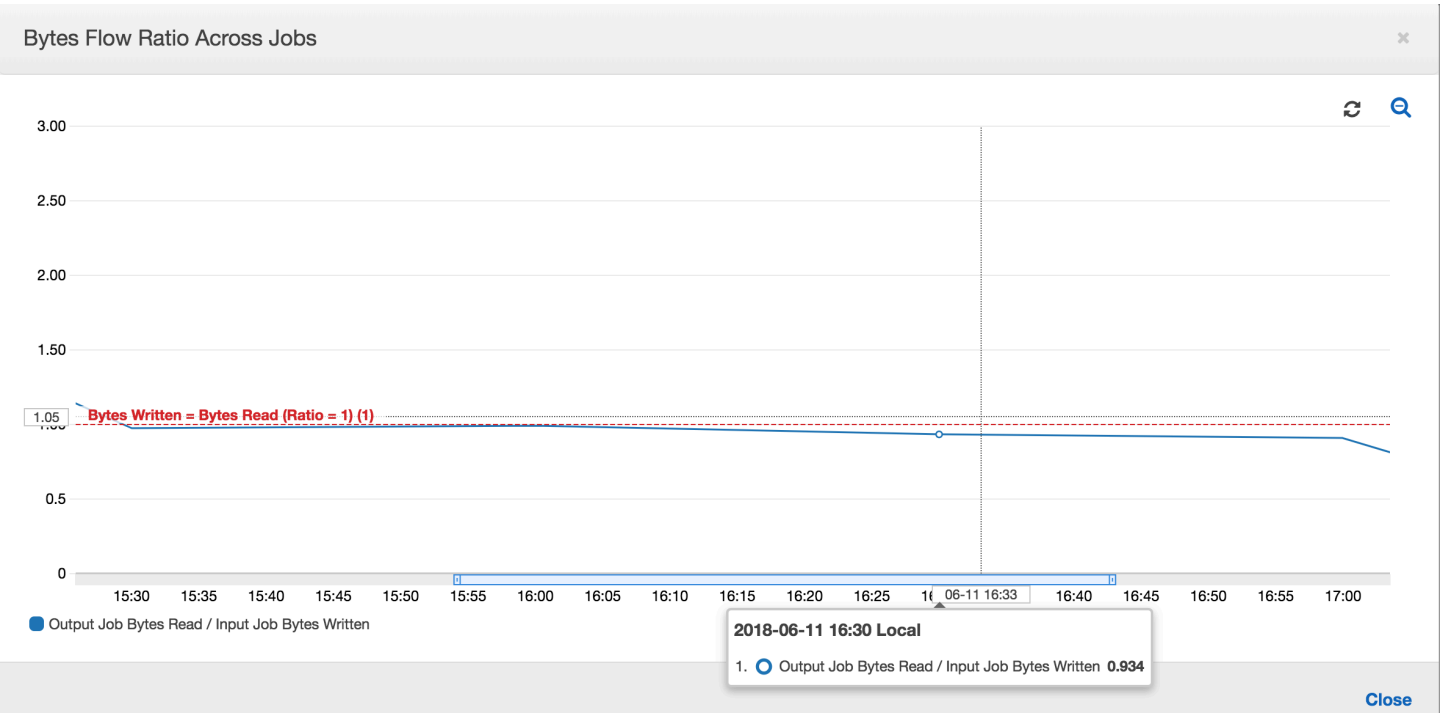
修正檔案處理

您應該確保輸出任務僅會處理尚未由輸出任務的前一任務執行處理過的檔案。若要執行此操作，請啟用任務書籤並在輸出任務中設定轉換內容，如下所示：

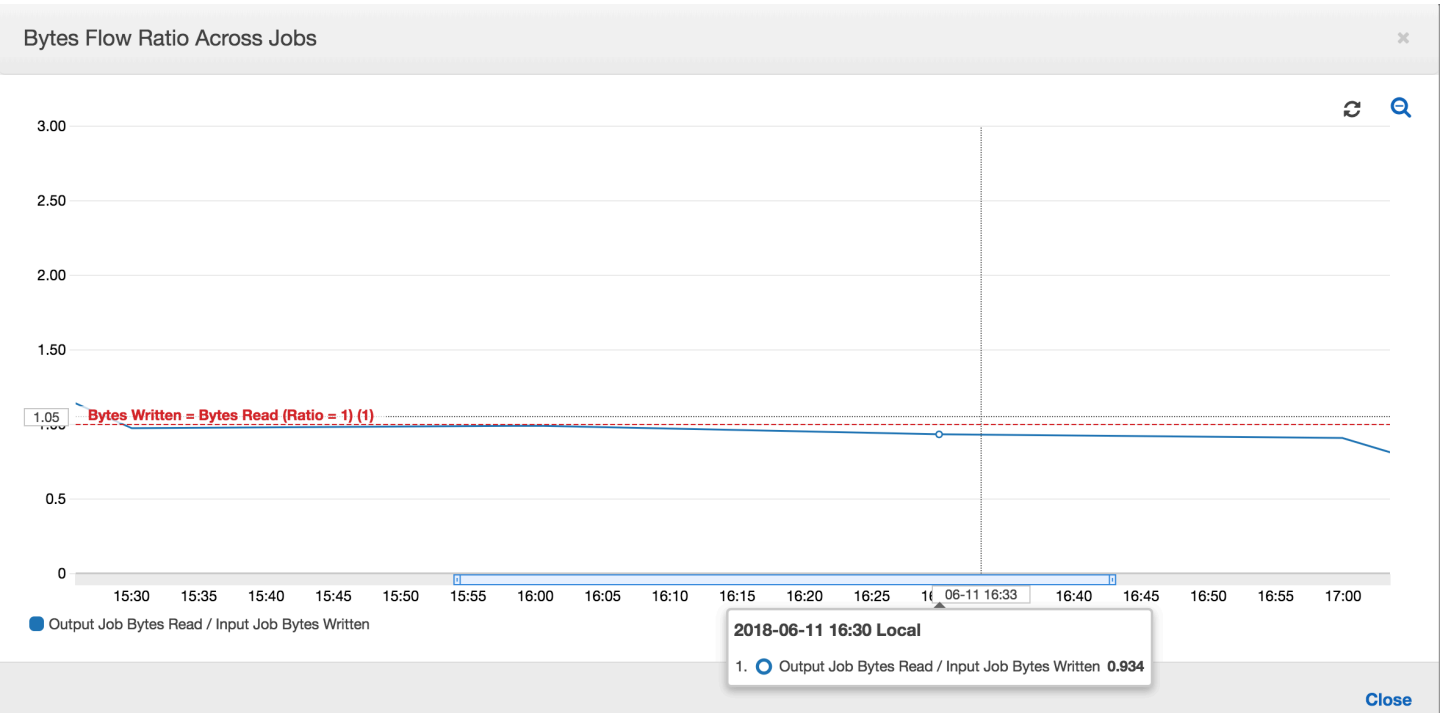
```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
 connection_options = {"paths": [staging_path],
 "useS3ListImplementation":True,"recurse":True}, format="json", transformation_ctx =
 "bookmark_ctx")
```

任務書籤啟用後，輸出任務將不會自輸入任務的所有先前任務執行中重新處理暫存位置裡的資料。下列映像顯示了資料的讀取和寫入，棕色曲線下的面積與紅色曲線相當一致且相似。

由於沒有額外的資料處理，位元組流程比例也會大致保持接近為 1。



在下一個輸入任務執行開始將更多資料放入暫存位置之前，輸出任務的任務執行將開始並擷取暫存位置中的檔案。只要此操作持續執行，它僅會處理之前的輸入任務執行所擷取的檔案，且比例保持接近 1。



假設輸入任務花費的時間超過預期，於是，輸出任務將從兩個輸入任務執行中擷取暫存位置中的檔案。然後，該輸出任務執行的比例將高於 1。不過，以下輸出任務的任務執行，不會處理任何輸出任務之前的任務執行已處裡的檔案。

DPU 容量規劃監控

您可在 AWS Glue 使用任務指標來估計可在 AWS Glue 任務上用來進行擴展的資料處理單元 (DPU) 數量。

> **Note**
>
> 此頁面僅適用於 AWS Glue 0.9 和 1.0 版。更高版本的 AWS Glue 包含節約成本的功能，這些功能在執行容量規劃時會引入額外的考慮因素。

主題

- 已分析的程式碼
- 在 AWS Glue 主控台上視覺化已分析的指標
- 判斷最佳 DPU 容量

已分析的程式碼

以下指令碼會讀取包含 428 個 gzip JSON 檔案的 Amazon Simple Storage Service (Amazon S3) 分割區。該指令碼套用對應來變更欄位名稱，並以 Apache Parquet 格式將它們轉換並寫入 Amazon S3。您將根據預設佈建 10 個 DPU 並執行此任務。

```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
 connection_options = {"paths": [input_path],
 "useS3ListImplementation":True,"recurse":True}, format="json")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [(map_spec])
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
 connection_type = "s3", connection_options = {"path": output_path}, format =
 "parquet")
```

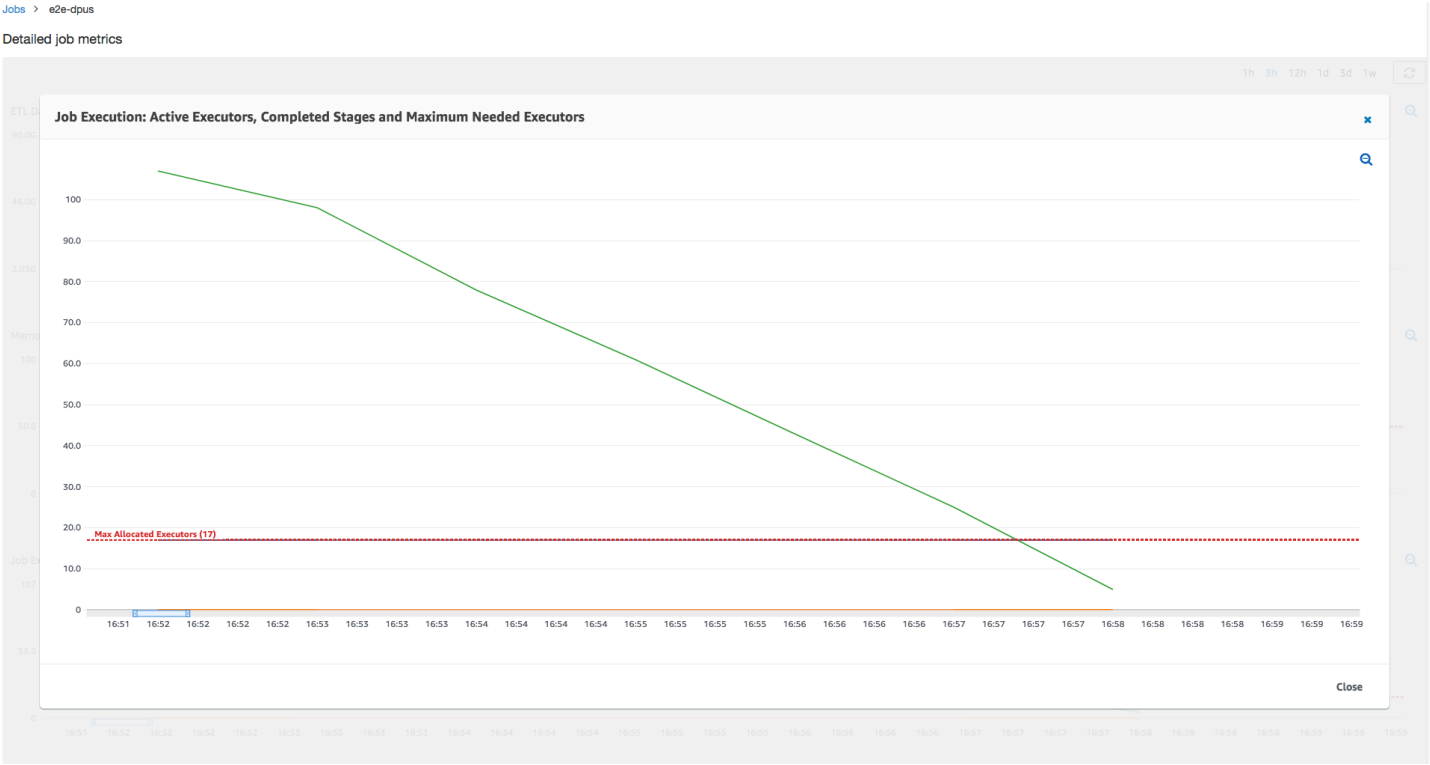
在 AWS Glue 主控台上視覺化已分析的指標

任務執行 1：在此任務執行中，我們將展示如何尋找叢集中是否存在佈建不足的 DPU。在 AWS Glue 中的任務執行功能顯示所有主動執行的執行器數量、完成階段的數量，以及所需執行器的最大數量。

所需執行器最大數量的計算，是透過新增執行中任務和待處理任務總數除以每個執行器的任務。該結果是滿足目前負載所需的執行器總數的測量。

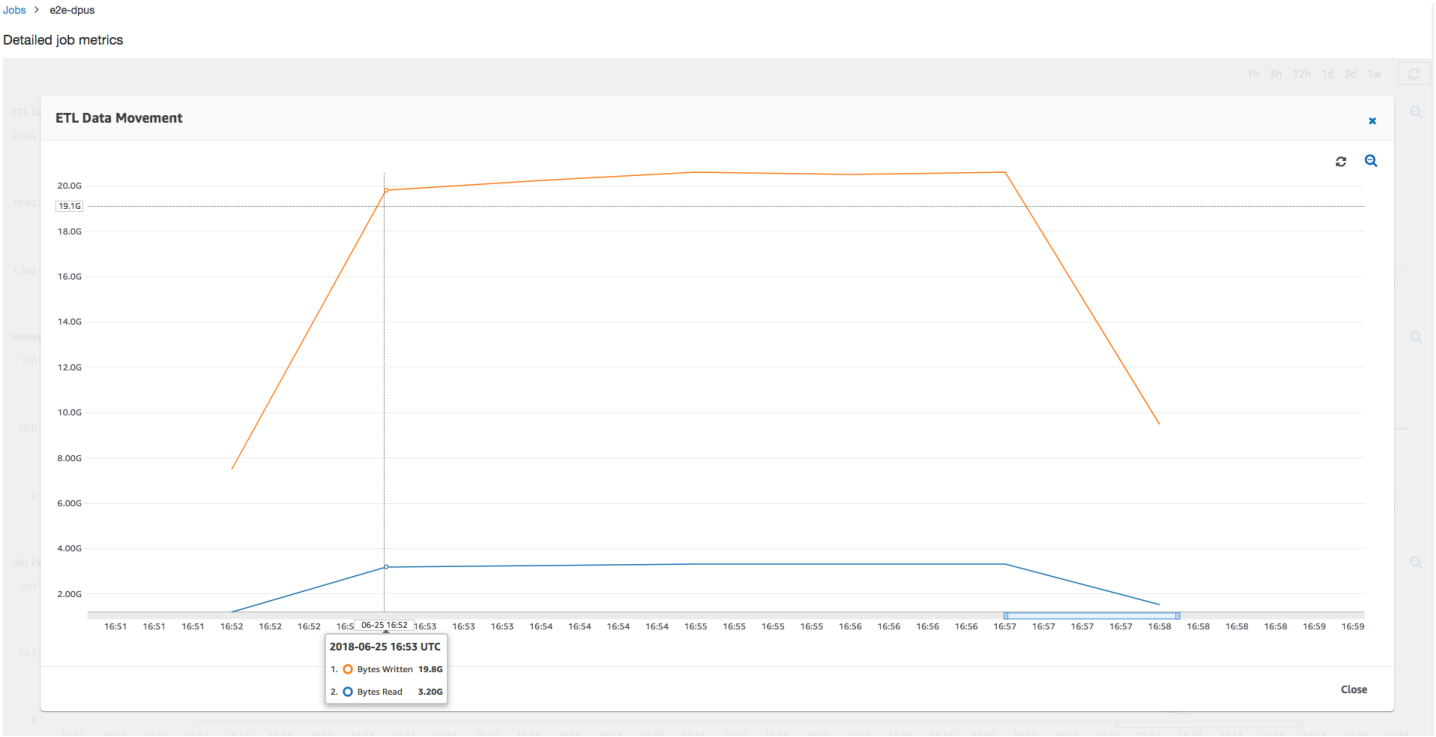
相反的，主動執行的執行器數量，會衡量有多少執行器正在執行主動 Apache Spark 任務。隨著任務的進行，所需的最大執行器可以更改，並且通常會在等待中的任務佇列降低時向下移動到任務的末端。

下圖中的水平紅線顯示了分配執行器的最大數量，這取決於您為任務分配的 DPU 數。在本例中，您需要為該任務執行分配 10 個 DPU。一個 DPU 保留給管理使用。另外九個 DPU 則會分別執行兩個執行器，且會保留其中一個執行器給 Spark 驅動程式。Spark 驅動程式會在主要應用程式內執行。因此，分配的執行器最大數量是 2*9 - 1 = 17 個執行器。



如圖所示，所需執行器的最大數量在任務開始時從 107 開始，而中動執行器數量保持為 17。這與具有 10 個 DPU 的分配執行器最大數量相同。所需執行器最大數量和所分配執行器最大數量間的比率 (對於 Spark 驅動程式，兩者都新增 1 個) 會提供佈建不足因數：108/18 = 6 倍。您可以佈建 6 (佈建不足比率) *9 (目前 DPU 容量 - 1) + 1 個 DPU = 55 個 DPU 來擴展該任務，即可以最高平行處理速度執行並加速完成。

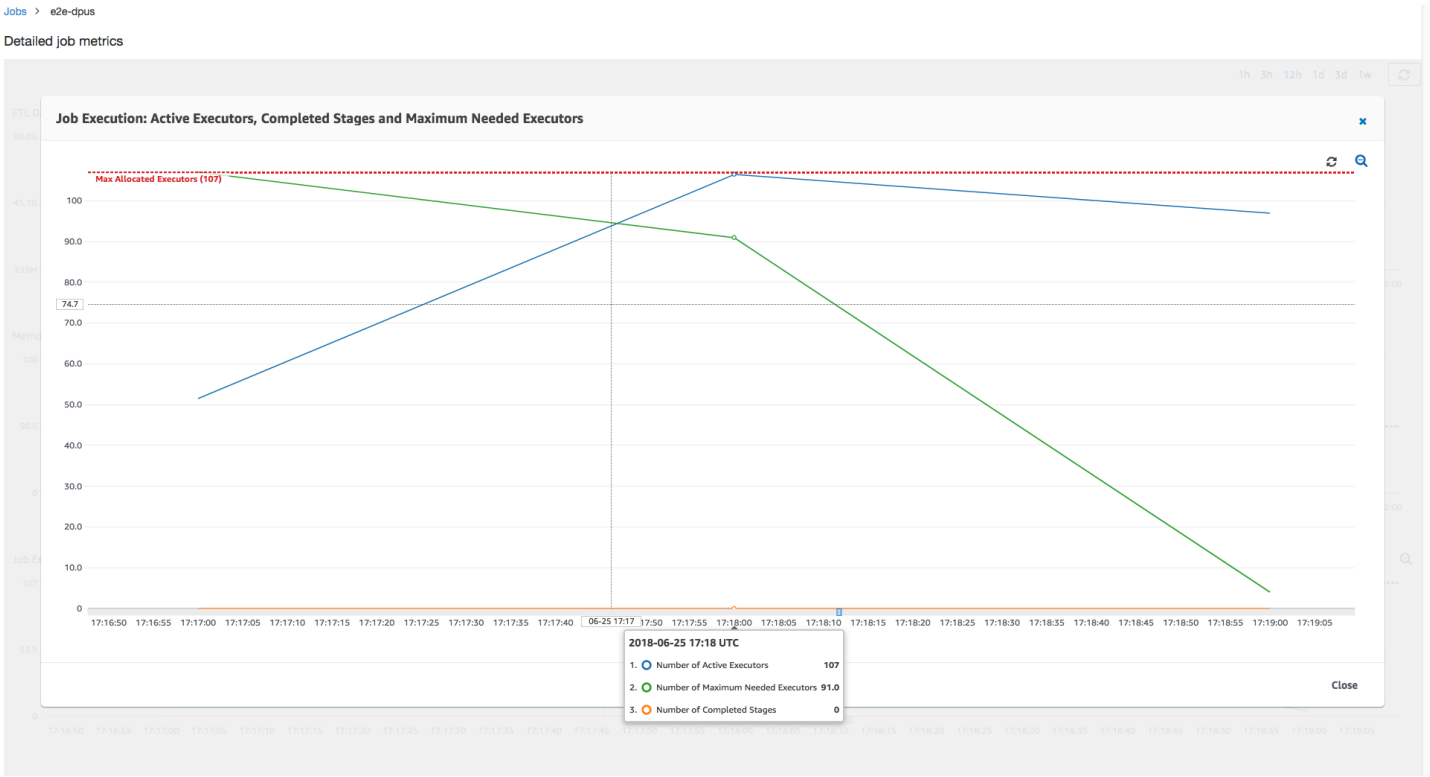
AWS Glue 主控台會以靜態線 (表示分配執行器原始數量上限) 顯示詳細的任務指標。該主控台會透過指標的任務定義來計算分配執行器的上限。相反地，如需詳細任務執行指標，主控台會透過任務執行組態來計算分配執行器的上限，特別是針對任務執行分配的 DPU 數。若要檢視個別任務執行的指標，請選取任務執行並選擇 View run metrics (檢視執行指標)。



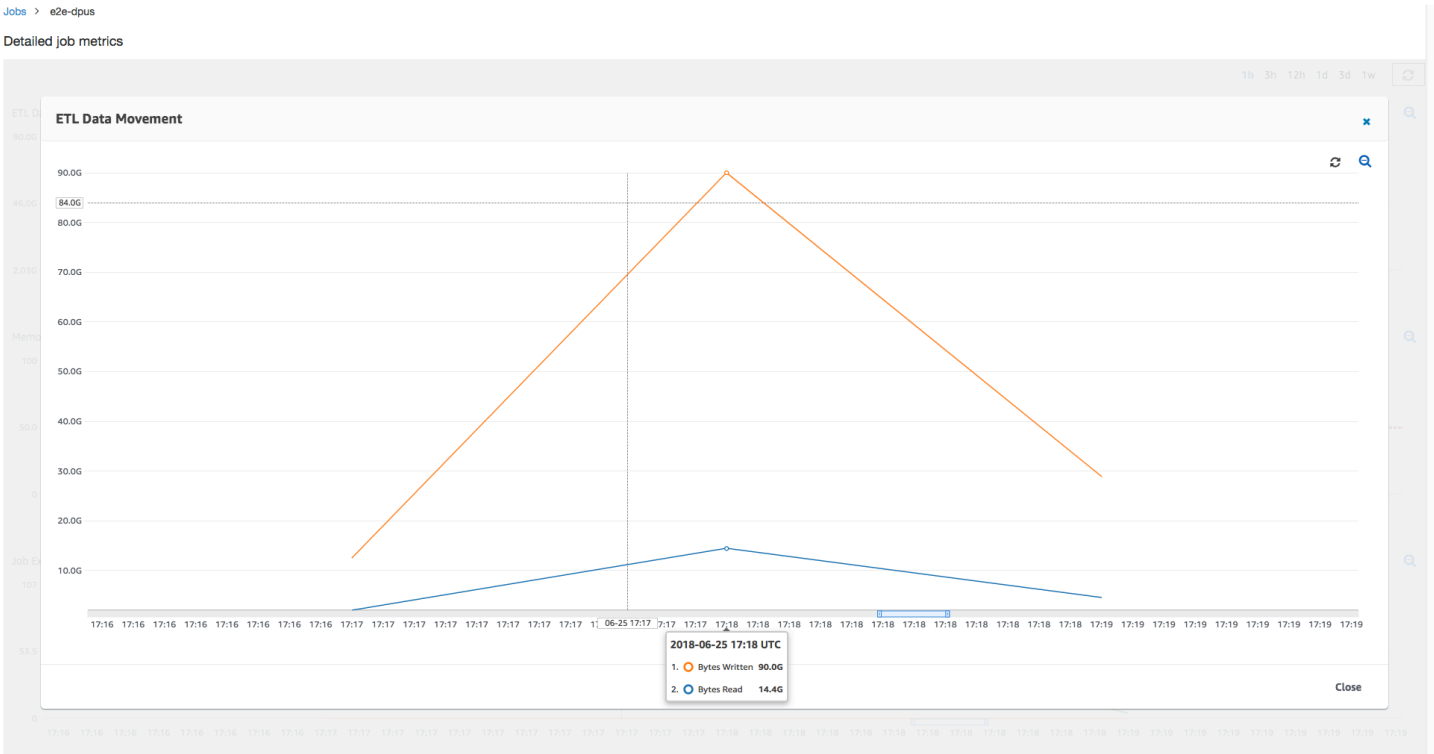
查看 Amazon S3 位元組讀取和寫入，請注意，該任務會在 Amazon S3 資料中花費所有的六分鐘串流，並將其平行寫出。分配的 DPU 上的所有核心都正在讀取和寫入 Amazon S3。所需執行器的最大數量為 107，也符合輸入 Amazon S3 路徑中的檔案數 – 428。每個執行器可以啟動四個 Spark 任務以處理四個輸入檔案 (JSON gzip)。

判斷最佳 DPU 容量

根據先前任務執行的結果，您可以將分配的 DPU 總數增加到 55，並查看任務的執行情況。該任務可在三個分鐘內完成 – 是先前所需時間的一半。此案例中的任務擴展並非線性的，因為它是一種短期執行任務。具有長期執行任務或大量任務的任務 (大量的最大所需執行器) 受益於接近線性的 DPU 擴展效能加速。



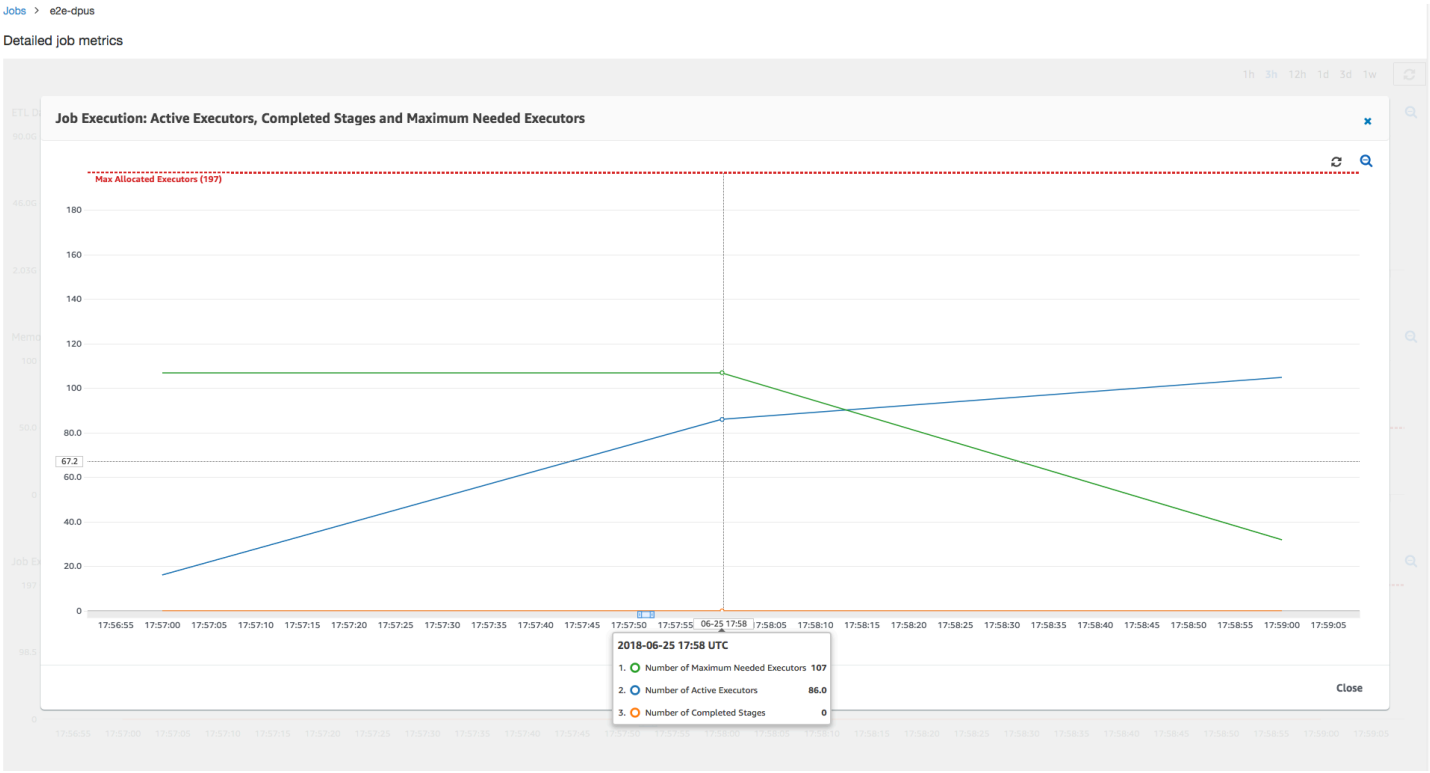
如上圖所示，主動執行器的總數達到最大分配數 – 107 個執行器。同樣地，所需執行器的最大數量永遠不會超過分配的執行器最大數量。所需的最大執行器數量是根據主動執行和等待中任務計數計算的，因此可能小於活動執行器的數量。這是因為可能有執行器在短時間內部分或完全閒置，且尚未淘汰。



此任務執行使用 6 倍以上的執行器從 Amazon S3 平行讀取和寫入。因此，此任務執行會為讀取和寫入使用更多的 Amazon S3 頻寬，並且更快完成。

識別過度佈建的 DPU

接下來，您可以判斷使用 100 DPU (99 * 2 = 198 執行器) 擴展任務是否有助於進一步擴展。如下圖顯示，該任務仍然需要三分鐘來完成。同樣，任務不會超出 107 個執行器 (55 個 DPU 組態)，其餘 91 個執行器被過度佈建而完全不使用。這表示增加的 DPU 數量有時可能無法提高效能，這從所需的最大執行器數量中可以看出。



比較時間差異

下表中顯示的三個任務執行總結了 10 個 DPU、55 個 DPU 和 100 個 DPU 的任務執行時間。您可以使用監控第一個任務執行建立的評估來尋找 DPU 容量以改善任務執行時間。

任務 ID	DPU 數量	執行時間
jr_c894524c8ef5048a4d9...	10	6 分鐘。
jr_1a466cf2575e7ffe6856...	55	3 分鐘。
jr_34fa1ed4c6aa9ff0a814...	100	3 分鐘。

在 AWS Glue 中串流 ETL 任務

您可以建立串流擷取、轉換和載入 (ETL) 任務，讓它連續執行並從 Amazon Kinesis Data Streams、Apache Kafka 和 Amazon Managed Streaming for Apache Kafka (Amazon MSK) 的串流來源使用資料。這些任務會清理並轉換資料，然後將結果載入 Amazon S3 資料湖或 JDBC 資料存放區。

此外，您可以為 Amazon Kinesis Data Streams 產生資料。只有在撰寫 AWS Glue 指令碼時，才能使用此功能。如需詳細資訊，請參閱 the section called “Kinesis 連線”。

依預設，AWS Glue 以 100 秒的間隔處理和寫出資料。這樣可以有效處理資料，並且可在資料到達時間比預期晚時執行彙總。您可以修改此時段大小，以提高適時性或彙總正確性。AWS Glue 串流任務使用檢查點而不是任務書籤來追蹤已讀取的資料。

> **Note**
>
> AWS Glue 在串流 ETL 任務執行時按小時計費。

本影片討論串流 ETL 成本挑戰，以及中的節省成本功能。 AWS Glue

建立串流 ETL 任務包含下列步驟：

1. 對於 Apache Kafka 串流來源，建立連接到 Kafka 來源或 Amazon MSK 叢集的 AWS Glue 連線。
2. 為串流來源手動建立 Data Catalog 資料表。
3. 為串流資料來源建立 ETL 任務。定義串流特定的任務屬性，並提供您自己的指令碼，或選擇修改產生的指令碼。

如需詳細資訊，請參閱 AWS Glue 的串流 ETL。

為 Amazon Kinesis Data Streams 建立串流 ETL 任務時，您不需要建立 AWS Glue 連線。但是，如果有連線連接到以 Kinesis Data Streams 做為來源的 AWS Glue 串流 ETL 任務，則需要連到 Kinesis 的 Virtual Private Cloud (VPC) 端點。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的建立介面端點。在另一個帳戶中指定 Amazon Kinesis Data Streams 時，您必須設定角色和政策以允許跨帳戶存取。如需詳細資訊，請參閱範例：從不同帳戶中的 Kinesis 串流讀取。

AWS Glue 串流 ETL 作業可以自動偵測壓縮資料、透明地解壓縮串流資料、在輸入來源上執行例行轉換並載入輸出存放區。

AWS Glue 根據輸入格式，支援以下壓縮類型的自動解壓縮：

壓縮類型	Avro 檔案	Avro 資料	JSON	CSV	Grok
BZIP2	是	是	是	是	是
GZIP	否	是	是	是	是
SNAPPY	是 (原始 Snappy)	是 (框架式 Snappy)	是 (框架式 Snappy)	是 (框架式 Snappy)	是 (框架式 Snappy)
XZ	是	是	是	是	是
ZSTD	是	否	否	否	否
DEFLATE	是	是	是	是	是

主題

為 Apache Kafka 資料串流建立 AWS Glue 連線

若要讀取 Apache Kafka 串流，您必須建立 AWS Glue 連線。

為 Kafka 來源建立 AWS Glue 連線 (主控台)

1. 請在以下位置開啟 AWS Glue 主控台。 https://console.aws.amazon.com/glue/
2. 在導覽窗格中，於 Data catalog (Data Catalog) 下選擇 Connections (連線)。
3. 選擇 Add connection (新增連線)，然後在 Set up your connection’s properties (設定連線的屬性) 頁面上輸入連線名稱。

> **Note**
>
> 如需指定連線屬性的詳細資訊，請參見《AWS Glue 連線屬性。》。

4. 對於連線類型，請選擇 Kafka。
5. 對於 Kafka bootstrap servers URLs (Kafka 引導伺服器 URL)，請為您的 Amazon MSK 叢集或 Apache Kafka 叢集輸入引導代理程式的主機和連接埠號碼。僅使用 Transport Layer Security (TLS) 端點來建立與 Kafka 叢集的初始連線。不支援純文字端點。

   以下是 Amazon MSK 叢集的主機名稱和連接埠號碼對的範例清單。

   ```
   myserver1.kafka.us-east-1.amazonaws.com:9094,myserver2.kafka.us-
   east-1.amazonaws.com:9094,
   myserver3.kafka.us-east-1.amazonaws.com:9094
   ```

   如需取得引導代理程式資訊的詳細資訊，請參閱 Amazon Managed Streaming for Apache Kafka 開發人員指南中的取得 Amazon MSK 叢集的引導代理程式。
6. 如果您想要與 Kafka 資料來源的安全連線，請選取 Require SSL connection (需要 SSL 連線)，以及對於 Kafka private CA certificate location (Kafka 私有 CA 憑證位置)，輸入自訂 SSL 憑證的有效 Amazon S3 路徑。

   對於自我管理的 Kafka 的 SSL 連線，自訂憑證是強制性的。這對於 Amazon MSK 是選用的。

   如需指定 Kafka 之自訂憑證的詳細資訊，請參閱 the section called “SSL 連線屬性”。
7. 使用 AWS Glue Studio 或 AWS CLI 指定 Kafka 用戶端驗證方法。若要存 AWS Glue Studio 取，請AWS Glue從左側導覽窗格的 ETL 功能表中選取。

   如需 Kafka 用戶端身分驗證方法的詳細資訊，請參閱 用於用戶端身分驗證的 AWS Glue Kafka 連連線屬性 。
8. 選用地輸入描述，然後選擇 Next (下一頁)。
9. 對於 Amazon MSK 叢集，請指定其 Virtual Private Cloud (VPC)、子網路和安全群組。VPC 資訊對於自我管理 Kafka 是選用的。
10. 選擇 Next (下一頁) 檢閱所有連線屬性，然後選擇 Finish (完成)。

如需 AWS Glue 連線的詳細資訊，請參閱 連線至資料。

用於用戶端身分驗證的 AWS Glue Kafka 連連線屬性

SASL/GSSAPI (Kerberos) 身分驗證

選擇此身分驗證方法將允許您指定 Kerberos 屬性。

Kerberos Keytab

選擇 Keytab 檔案的位置。Keytab 存放了一個或多個主體的長期金鑰。如需詳細資訊，請參閱 MIT Kerberos 文件：Keytab。

Kerberos krb5.conf 檔案

選擇 krb5.conf 檔案。這包含預設範圍 (類似網域的邏輯網路，可定義相同 KDC 下的一組系統) 和 KDC 伺服器的位置。如需詳細資訊，請參閱 MIT Kerberos 文件：krb5.conf。

Kerberos 主體和 Kerberos 服務名稱

輸入 Kerberos 主體和服務名稱。如需詳細資訊，請參閱 MIT Kerberos 文件：Kerberos 主體。

SASL/SCRAM-SHA-512 身分驗證

選擇此身分驗證方法將允許您指定身分驗證憑證。

AWS Secrets Manager

在搜尋方塊中鍵入名稱或 ARN 來搜尋字符。

直接提供使用者名稱和密碼

請在搜尋方塊中鍵入名稱或 ARN 來搜尋字符。

SSL 用戶端身分驗證

選擇此身分驗證方法將允許您透過瀏覽 Amazon S3 來選取 Kafka 用戶端金鑰存放區的位置。或者，您可以輸入 Kafka 用戶端金鑰存放區密碼和 Kafka 用戶端金鑰密碼。

IAM 身分驗證

此身分驗證方法不需要任何額外的規格，且只有當串流來源是 MSK Kafka 時才適用。

SASL/普通驗證

選擇此驗證方法可讓您指定認證證明資料。

為串流來源建立資料目錄資料表

可針對串流來源手動建立的資料目錄表，此資料目錄表可指定來源資料串流屬性，包含資料結構描述。此資料表做為串流 ETL 任務的資料來源。

如果您不知道來源資料串流中資料的結構描述，您可以建立沒有結構描述的資料表。然後，當您建立串流 ETL 任務時，您可以開啟 AWS Glue 結構描述偵測功能。AWS Glue 會從串流資料判斷結構描述。

使用AWS Glue主控台、 AWS Command Line Interface (AWS CLI) 或 AWS Glue API 來建立資料表。如需使用 AWS Glue 主控台手動建立資料表的詳細資訊，請參閱 the section called “建立資料表”。

> **Note**
>
> 您無法使用主 AWS Lake Formation 控台建立資料表；您必須使用主AWS Glue控台。

另外，請考慮以下 Avro 格式的串流來源或您可以套用 Grok 模式之日誌資料的資訊。

- the section called “Avro 串流來源的注意事項與限制”
- the section called “將 grok 模式套用至串流來源”

主題

- Kinesis 資料來源
- Kafka 資料來源
- AWS Glue 結構描述登錄檔資料表來源

Kinesis 資料來源

建立資料表時，請設定下列串流 ETL 屬性 (主控台)。

來源類型

Kinesis

針對相同帳戶中的 Kinesis 來源：

區域

Amazon Kinesis Data Streams 服務所在的 AWS 區域。區域和 Kinesis 串流名稱會一起翻譯成串流 ARN。

範例：https://kinesis.us-east-1.amazonaws.com

Kinesis 串流名稱

串流名稱，如Amazon Kinesis Data Streams 開發人員指南中的建立串流所述。

如需其他帳戶中的 Kinesis 來源，請參閱此範例，將角色和政策設定為允許跨帳戶存取。設定下列設定：

串流 ARN

消費者註冊的 Kinesis 資料串流的 ARN。如需詳細資訊，請參閱中的 Amazon 資源名稱 (ARN) 和 AWS 服務命名空間。AWS 一般參考

擔任的角色 ARN

要擔任之角色的 Amazon Resource Name (ARN)。

工作階段名稱 (選用)

擔任角色工作階段的識別碼。

當同一個角色由不同委託人或出於不同原因而承擔時，使用角色工作階段名稱來唯一識別工作階段。在跨帳戶案例中，角色工作階段名稱對擁有該角色的帳戶是可見的，而且可以由擁有該角色的帳戶記錄。角色工作階段名稱也用於擔任角色委託人的 ARN。這表示使用臨時安全性登入資料的後續跨帳戶 API 要求會將角色工作階段名稱公開給其 AWS CloudTrail 記錄檔中的外部帳戶。

為 Amazon Kinesis Data Streams 設定串流 ETL 屬性 (AWS Glue API 或 AWS CLI)

- 若要為相同帳戶中的 Kinesis 來源設定串流 ETL 屬性，請在 `CreateTable` API 操作或 `create_table` CLI 命令的 `StorageDescriptor` 結構中指定 `streamName` 和 `endpointUrl` 參數。

```
"StorageDescriptor": {
 "Parameters": {
  "typeOfData": "kinesis",
  "streamName": "sample-stream",
  "endpointUrl": "https://kinesis.us-east-1.amazonaws.com"
 }
 ...
}
```

或者，指定 `streamARN`。

Example

```
"StorageDescriptor": {
```

```
 "Parameters": {
  "typeOfData": "kinesis",
  "streamARN": "arn:aws:kinesis:us-east-1:123456789:stream/sample-stream"
 }
 ...
}
```

- 若要為另一個帳戶中的 Kinesis 來源設定串流 ETL 屬性，請在 `CreateTable` API 操作或 `create_table` CLI 命令的 `StorageDescriptor` 結構中指定 `streamARN`、`awsSTSRoleARN` 和 `awsSTSSessionName` (選用) 參數。

```
"StorageDescriptor": {
 "Parameters": {
  "typeOfData": "kinesis",
  "streamARN": "arn:aws:kinesis:us-east-1:123456789:stream/sample-stream",
  "awsSTSRoleARN": "arn:aws:iam::123456789:role/sample-assume-role-arn",
  "awsSTSSessionName": "optional-session"
 }
 ...
}
```

Kafka 資料來源

建立資料表時，請設定下列串流 ETL 屬性 (主控台)。

來源類型

Kafka

針對 Kafka 來源：

主題名稱

主題名稱依 Kafka 中所指定。

連線

參考 Kafka 來源的 AWS Glue 連線，如 the section called “為 Kafka 資料串流建立連線” 中所述。

AWS Glue 結構描述登錄檔資料表來源

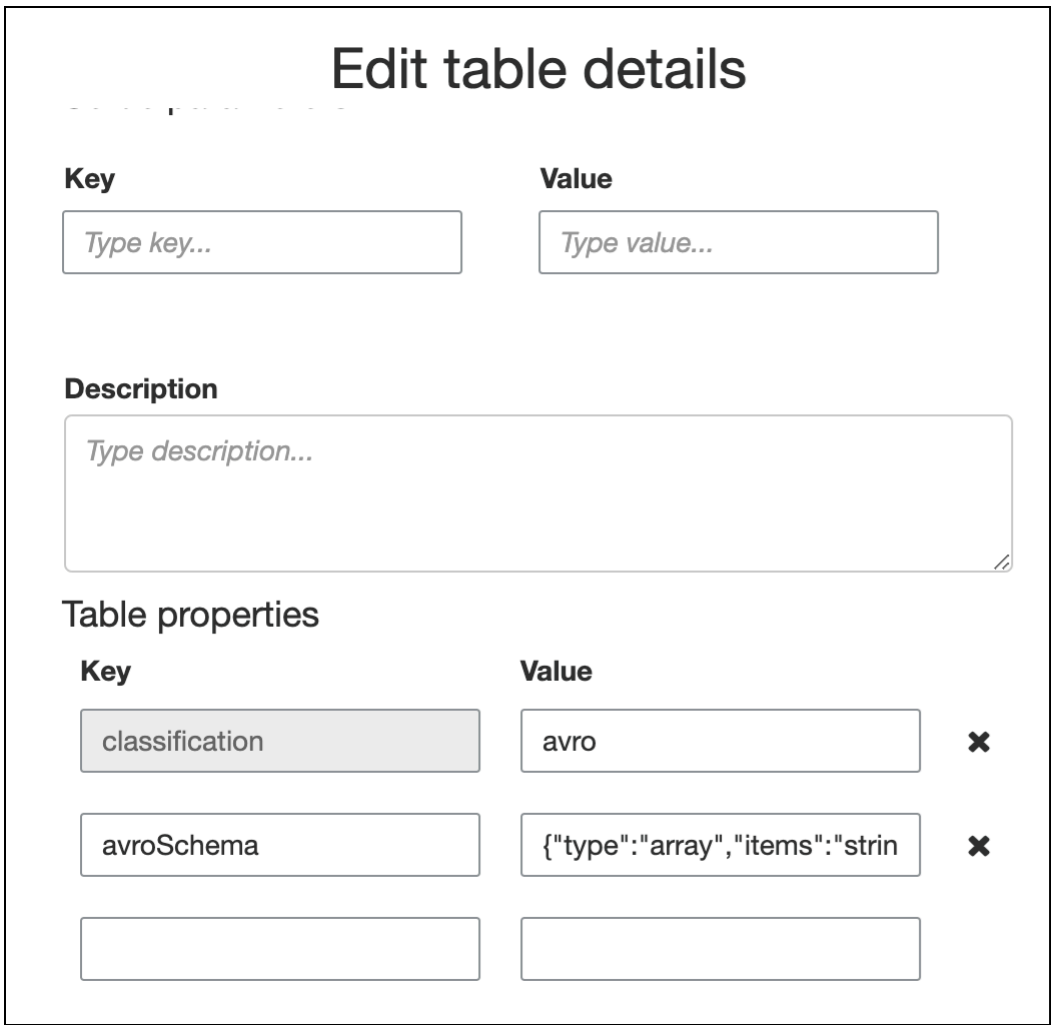
若要使用使用 AWS Glue 結構描述登錄檔串流任務，請依照 使用案例：AWS Glue Data Catalog 中的說明建立或更新結構描述登錄檔資料表。

目前，AWS Glue 串流僅支援 Glue 結構描述登錄檔 Avro 格式，其中結構描述推斷設定為 `false`。

Avro 串流來源的注意事項與限制

下列注意事項和限制適用於 Avro 格式的串流來源：

- 開啟結構描述偵測時，Avro 結構描述必須包含在裝載中。關閉時，裝載應該只包含資料。
- 動態框架不支援某些 Avro 資料類型。在 AWS Glue 主控台的建立資料表精靈 Define a schema (定義結構描述) 頁面中定義結構描述時，您無法指定這些資料類型。在結構描述偵測期間，Avro 結構描述中不支援的類型會轉換成支援的類型，如下所示：
  - `EnumType => StringType`
  - `FixedType => BinaryType`
  - `UnionType => StructType`
- 如果您使用主控台中 Define a schema (定義結構描述) 頁面來定義資料表結構描述，則結構描述的隱含根元素類型為 `record`。如果您想要 `record` 以外的根元素類型，例如 `array` 或 `map`，則無法使用 Define a schema (定義結構描述) 頁面指定結構描述。相反，您必須略過該頁面，並將結構描述指定為資料表屬性或在 ETL 指令碼內指定。
  - 若要在資料表屬性中指定結構描述，請完成建立資料表精靈、編輯資料表詳細資訊，並在 Table properties (資料表屬性) 下新增鍵值對。使用鍵 `avroSchema`，然後輸入值的結構描述 JSON 物件，如下列螢幕擷取畫面所示。



- 若要在 ETL 指令碼中指定結構描述，請修改 `datasource0` 指派陳述式，並新增 `avroSchema` 鍵至 `additional_options` 參數，如下列 Python 和 Scala 範例所示。

  Python

  ```
  SCHEMA_STRING = '{"type":"array","items":"string"}'
  datasource0 = glueContext.create_data_frame.from_catalog(database =
   "database", table_name = "table_name", transformation_ctx = "datasource0",
   additional_options = {"startingPosition": "TRIM_HORIZON", "inferSchema":
   "false", "avroSchema": SCHEMA_STRING})
  ```

  Scala

  ```
  val SCHEMA_STRING = """{"type":"array","items":"string"}"""
  val datasource0 = glueContext.getCatalogSource(database = "database", tableName
   = "table_name", redshiftTmpDir = "", transformationContext = "datasource0",
  ```

```
additionalOptions = JsonOptions(s"""{"startingPosition": "TRIM_HORIZON",
"inferSchema": "false", "avroSchema":"$SCHEMA_STRING"}""")).getDataFrame()
```

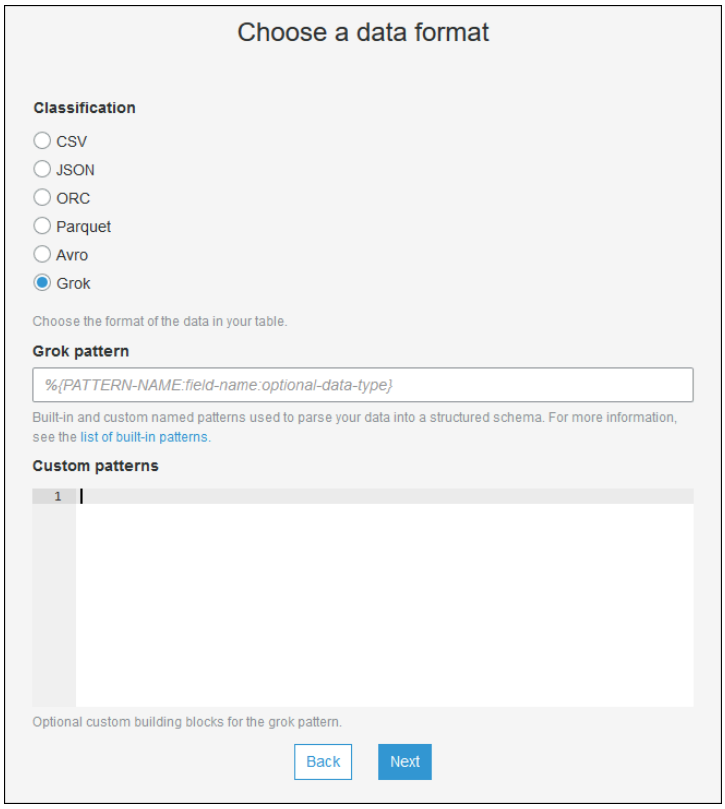
將 grok 模式套用至串流來源

您可以為日誌資料來源建立串流 ETL 任務，並使用 Grok 模式將日誌轉換為結構化資料。ETL 任務接著會將資料當成結構化資料來源處理。您可以指定當您為串流來源建立 Data Catalog 資料表時要套用的 Grok 模式。

如需 Grok 模式和自訂模式字串值的相關資訊，請參閱 撰寫 grok 自訂分類器。

將 grok 模式新增至資料目錄資料表 (主控台)

- 使用建立資料表精靈，並使用 the section called “為串流來源建立資料目錄資料表” 中指定的參數建立資料表。指定資料格式為 Grok，填寫 Grok pattern (Grok 模式) 欄位，並選擇性地在 Custom patterns (optional) (自訂模式 (選用)) 下方新增自訂模式。



  在每個自訂模式之後按 Enter。

將 grok 模式新增至資料目錄資料表 (AWS Glue API 或 AWS CLI)

- 新增 `GrokPattern` 參數，並選擇性地將 `CustomPatterns` 參數新增至 `CreateTable` API 操作或 `create_table` CLI 命令。

```
 "Parameters": {
...
    "grokPattern": "string",
    "grokCustomPatterns": "string",
...
},
```

將 `grokCustomPatterns` 表示為字串，並使用 "\n" 作為模式之間的分隔符。

下列是指定這些參數的範例。

Example

```
"parameters": {
...
    "grokPattern": "%{USERNAME:username} %{DIGIT:digit:int}",
    "grokCustomPatterns": "digit \d",
...
}
```

定義串流 ETL 任務的任務屬性

在 AWS Glue 主控台上定義串流 ETL 任務時，請提供下列串流特定的屬性。如需其他任務屬性的說明，請參閱 定義 Spark 任務的任務屬性。

IAM 角色

指定用於授權資源的 AWS Identity and Access Management (IAM) 角色，這些資源用於執行工作、存取串流來源和存取目標資料存放區。

若要存取 Amazon Kinesis Data Streams，請將`AmazonKinesisFullAccess` AWS 受管政策附加到角色，或附加類似的 IAM 政策以允許更細微的存取。如需政策範例，請參閱使用 IAM 控制對 Amazon Kinesis Data Streams 資源的存取。

如需在 AWS Glue 執行任務之許可的詳細資訊，請參閱 AWS Glue 的身分識別與存取管理。

Type

選擇 Spark streaming (Spark 串流)。

AWS Glue 版本

AWS Glue 版本決定了可用於任務的 Apache Spark 和 Python 或 Scala 版本。選擇一個選項，指定可用於任務的 Python 或 Scala 版本。AWS Glue具有 Python 3 支援的 2.0 版本是串流 ETL 任務的預設值。

Maintenance window (維護時段)

指定可以重新啟動串流工作的視窗。請參閱the section called “維護時段”。

任務逾時

選擇性地輸入持續時間 (以分鐘為單位)。預設值為空白。

- 串流工作的逾時值必須小於 7 天或 10080 分鐘。
- 當值保留空白時，如果您尚未設定維護時段，工作將在 7 天後重新啟動。如果您已設定維護時段，工作將在 7 天後的維護時段中重新啟動。

資料來源

指定您在 the section called “為串流來源建立資料目錄資料表” 中建立的資料表。

資料目標

執行以下任意一項：

- 選擇 Create tables in your data target (在您的資料目標中建立資料表)，然後指定下列資料目標屬性。

  資料存放區

  選擇 Amazon S3 或 JDBC。

  格式

  選擇任何格式。全部支援串流。

- 選擇 Use tables in the data catalog and update your data target (使用 Data Catalog 中的資料表並更新您的資料目標)，然後選擇 JDBC 資料存放區的資料表。

輸出結構描述定義

執行以下任意一項：

- 選擇 Automatically detect schema of each record (自動偵測每筆記錄的結構描述) 以開啟結構描述偵測。AWS Glue 會從串流資料判斷結構描述。
- 選擇 Specify output schema for all records (指定所有記錄的輸出結構描述)，以使用「套用映射」轉換來定義輸出結構描述。

指令碼

選擇性地提供您自己的指令碼或修改產生的指令碼，以執行 Apache Spark 結構化串流引擎支援的操作。如需可用作業的相關資訊，請參閱串流 DataFrames/資料集的作業。

串流 ETL 注意事項和限制

請記住下列注意事項和限制：

- 自動解壓縮 AWS Glue 串流 ETL 作業僅適用於支援的壓縮類型。同時請注意以下各項：
  - 框架式 Snappy 指的是官方 Snappy 框架格式。
  - Glue 3.0 版支援 Deflate，而 Glue 2.0 版不支援。
- 使用結構描述偵測時，您無法執行串流資料的聯結。
- AWS Glue 串流 ETL 任務不支援 Avro 格式 AWS Glue 結構描述登錄檔的 Union 資料類型。
- 您的 ETL 指令碼可以使用 Apache Spark 結構化串流原生的 AWS Glue 內建轉換。如需詳細資訊，請參閱 Apache Spark 網站上的串流 DataFrames/資料集的作業或AWS Glue PySpark 轉換參考。
- AWS Glue 串流 ETL 任務使用檢查點來追蹤已讀取的資料。因此，已停止再重新啟動的任務會從原先在串流中停止的位置接續下去。如果想要重新處理資料，您可以刪除指令碼中參考的檢查點資料夾。
- 不支援任務書籤。
- 若要在任務中使用 Kinesis Data Streams 的強化廣發功能，請參閱 the section called “在 Kinesis 串流任務中使用強化廣發功能”。
- 如果使用從 AWS Glue 結構描述登錄檔建立的 Data Catalog 資料表，則當新的結構描述版本可用時，為了反映新的結構描述，您需要執行以下操作：
  1. 停止與資料表相關聯的任務。
  2. 更新「 Data Catalog 」資料表的結構描述。
  3. 重新啟動與資料表相關聯的任務。

使用 AWS Lake Formation FindMatches 比對記錄

> **Note**
>
> 目前，以下區域在 AWS Glue 主控台中不支援記錄比對：中東 (阿拉伯聯合大公國)、歐洲 (西班牙)(eu-south-2) 和歐洲 (蘇黎世)(eu-central-2)。

AWS Lake Formation 提供機器學習功能，可讓您建立自訂轉換以清理您的資料。目前有一個名為 FindMatches 的可用轉換。FindMatches 轉換可讓您識別資料集中重複或相符的記錄，即使記錄沒有通用的唯一識別符，也沒有欄位完全相符。不需要編寫任何程式碼或了解機器學習的運作方式。FindMatches 在許多不同的問題中非常有用，例如：

- Matching customers (比對客戶)：跨不同客戶資料庫連結客戶記錄，即使許多客戶欄位在資料庫之間不完全相符 (例如：名稱拼字不同、地址差異、資料遺失或不正確等)。
- Matching products (比對產品)：比對目錄中的產品與其他產品來源，例如產品目錄與競爭對手的目錄 (其中的項目結構不同)。
- Improving fraud detection (改善詐騙偵測)：識別重複的客戶帳戶，新建立的帳戶與先前已知的詐騙使用者相符 (或可能相符) 時進行判斷。
- Other matching problems (其他比對問題)：比對地址、電影、零件清單等。一般而言，如果人類可以查看您的資料庫列並判斷它們是否相符，則 FindMatches 轉換很有可能對您有幫助。

當您建立任務時，您可以建立這些轉換。您建立的轉換會以來源資料存放區結構描述與您標記的來源資料集中的範例資料為基礎 (我們將這個程序稱為「教導」轉換)。來源資料集中必須存在您標記的記錄。在這個程序中，我們會產生一個檔案，您可以標記然後重新上傳，轉換會從中學習。在您教導轉換後，您便可以從您的 Spark 式 AWS Glue 任務 (PySpark 或 Scala Spark) 呼叫它，並搭配相容的來源資料存放區，在其他指令碼中使用它。

在建立轉換後，它便會存放在 AWS Glue 中。在 AWS Glue 主控台上，您可以管理您建立的轉換。在資料整合和 ETL 的導覽窗格中，選擇資料分類工具 > 記錄比對，您可以進行編輯並繼續教導您的機器學習轉換。如需在主控台上管理轉換的詳細資訊，請參閱在 AWS Glue 主控台上使用機器學習轉換。

> **Note**
>
> AWS Glue 2.0 版 FindMatches 任務使用 Amazon S3 儲存貯體 `aws-glue-temp-<accountID>-<region>`，在轉換處理資料期間存放暫時檔案。您可以在執行完成後，手動或設定 Amazon S3 生命週期規則來刪除此資料。

機器學習轉換的類型

您可以建立機器學習轉換，來清理您的資料。您可以從 ETL 指令碼呼叫這些轉換。您的資料會在資料結構中從轉換傳遞至轉換，而此資料結構稱為 DynamicFrame，是 Apache Spark SQL `DataFrame` 的延伸。`DynamicFrame` 包含您的資料，而您可以參考其結構描述以處理資料。

下列類型的機器學習轉換可供您使用：

尋找符合項目

在來源資料中尋找重複記錄。您可以透過標記範例資料集，指出哪些資料列相符，來教導此機器學習轉換。您使用範例標記資料教導它的次數越多，機器學習轉換便會學習應將哪些資料列視為相符項目。取決於您設定轉換的方式，輸出會是以下其中一項：

- 輸入資料表的複本，加上 `match_id` 資料行，其中會填入指出相符記錄集的值。`match_id` 資料行是任意識別符。具有相同 `match_id` 的任何記錄已識別為彼此相符。具有不同 `match_id` 的記錄不相符。
- 輸入資料表的複本，其中會移除重複的資料列。若找到多個重複項目，便會保留主索引鍵最小的記錄。

尋找增量改進相符項目

尋找相符項目轉換也可以設定為在現有和增量改進框架中尋找相符項目，並以輸出方式傳回包含每個相符群組唯一 ID 的資料行。

如需詳細資訊，請參閱：尋找增量改進相符項目

使用 FindMatches 轉換

您可以使用 `FindMatches` 轉換來在來源資料中尋找重複記錄。其中會產生或提供標記檔案，來協助教導轉換。

> **Note**
>
> 目前，下列區域不支援使用自訂加密金鑰的 `FindMatches` 轉換：
>
> - 亞太區域 (大阪) – `ap-northeast-3`

若要開始使用 FindMatches 轉換，您可依照以下步驟操作。如需更進階和詳細的範例，請參閱 AWS 大數據部落格：Harmonize data using AWS Glue and AWS Lake Formation FindMatches ML to build a customer 360 view。

開始使用尋找相符項目轉換

遵循這些步驟來開始使用 `FindMatches` 轉換：

1. 在 AWS Glue Data Catalog 中為要清理的來源資料建立資料表。如需如何建立爬蟲程式的資訊，請參閱在 AWS Glue 主控台上使用爬蟲程式。

   若您的來源資料是文字類型的檔案 (例如逗點分隔值 (CSV) 檔案)，請考慮以下事項：

   - 在不同的資料夾中保留您的輸入記錄 CSV 檔案及標記檔案。否則，AWS Glue 爬蟲程式可能會將他們視為相同資料表的多個部分，並以不正確的方式在 Data Catalog 中建立資料表。
   - 除非您的 CSV 檔案只包含 ASCII 字元，否則請確保針對 CSV 檔案使用不包含 BOM (位元組順序標記) 的 UTF-8 編碼。Microsoft Excel 通常會在 UTF-8 CSV 檔案的開頭新增 BOM。若要移除它，請在文字編輯器中開啟 CSV 檔案，然後將檔案重新儲存為 UTF-8 without BOM (不包含 BOM 的 UTF-8)。
2. 在 AWS Glue 主控台上建立任務，然後選擇 Find matches (尋找相符項目) 轉換類型。

   > **Important**
   >
   > 您為任務選擇的資料來源資料表不能包含超過 100 個資料行。

3. 告訴 AWS Glue 產生標籤檔案，方法是選擇 Generate labeling file (產生標籤檔案)。AWS Glue 會先對每個 `labeling_set_id` 的類似記錄進行分組，以便您可以檢閱這些群組。您在 `label` 資料行中標示相符項目。

   - 若您已有標記檔案 (即指出相符資料列的記錄範例)，請將檔案上傳至 Amazon Simple Storage Service (Amazon S3)。如需標記檔案格式的資訊，請參閱標記檔案格式。繼續進行步驟 4。
4. 下載標記檔案並標記檔案，如標記一節所述。
5. 上傳修正後的標記檔案。AWS Glue 會執行任務來教導轉換如何尋找相符項目。

在 Machine learning transforms (機器學習轉換) 清單頁面上，選擇 History (歷史記錄) 標籤。此頁面會指出 AWS Glue 何時執行下列任務：

- Import labels (匯入標籤)
- Export labels (匯出標籤)
- Generate labels (產生標籤)
- Estimate quality (估計品質)

6. 若要建立更佳的轉換，您可以反覆地下載、標記和上傳標記檔案。在初始執行時，會有許多記錄比對錯誤。但是 AWS Glue 會隨著您透過驗證標記檔案繼續教導它而逐步學習。
7. 評估尋找相符項目的效能和結果，來評估和調校您的轉換。如需詳細資訊，請參閱 在 AWS Glue 中調校機器學習轉換。

標記

當 FindMatches 產生標記檔案時，會從您的來源資料表選取記錄。根據先前的培訓結果，FindMatches 會識別價值最高的記錄來進行學習。

標記的動作是編輯標記檔案 (我們建議使用如 Microsoft Excel 的試算表)，並將識別符或標籤新增至識別相符和不相符記錄的 label 資料行。在您資料檔案中擁有清楚且一致的相符項目定義相當重要。FindMatches 會從您指定為相符 (或不相符) 的記錄學習，並使用您的決定來學習如何尋找重複的記錄。

FindMatches 產生標記檔案時，大約會產生 100 筆記錄。這些 100 筆記錄通常會分為 10 個標記組，其中每個標記組會以 FindMatches 產生的唯一 labeling_set_id 辨識。每個標記組應視為是獨立於其他標記組的個別標記任務。您的任務為使用每個標記組識別相符與不相符的記錄。

在試算表中編輯標記檔案的秘訣

在試算表應用程式中編輯標記檔案時，請考慮以下事項：

- 檔案可能會在尚未完全展開資料行欄位的情況下開啟。您可能需要展開 labeling_set_id 和 label 資料行來查看那些資料格中的內容。
- 若主索引鍵資料行是數字 (例如 long 資料類型)，試算表可能會將其解譯為數字並變更值。此索引鍵值必須做為文字處理。若要修正此問題，請將主索引鍵資料行中的所有資料格格式化成 Text data (文字資料)。

標記檔案格式

AWS Glue 產生的標記檔案用以教導您的 `FindMatches` 轉換使用以下格式。如果您為 AWS Glue 產生自己的檔案，該檔案也需遵循此格式：

- 它是一個逗點分隔值 (CSV) 檔案。
- 它必須以 `UTF-8` 編碼。若您使用 Microsoft Windows 編輯檔案，它可能會以 `cp1252` 進行編碼。
- 它必須位在 Amazon S3 位置中，才能傳遞給 AWS Glue。
- 為每個標記任務使用中等數量的行。雖然每個任務可達 2-30 列，但每個任務建議使用 10—20 列。不建議使用超過 50 列的任務，且此類任務可能會導致不佳的結果或系統故障。
- 如果您有已標記的資料 (由標記為「match」(相符) 或「no-match」(不相符) 的記錄對組成)，這也沒問題。這些已標記的記錄對可以表示為大小 2 的標記組。在這種情況下，例如，如果兩項紀錄相符，則可將它們標記為「A」；如果它們不相符，則可將一筆記錄標記為「A」，另一筆標記為「B」。

> **Note**
>
> 因為它包含額外的資料行，標記檔案所擁有的結構描述會與包含您來源資料的檔案不同。請將標記檔案置放在與任何轉換輸入 CSV 檔案不同的資料夾中，以防 AWS Glue 爬蟲程式在 Data Catalog 中建立資料表時將其納入考慮範圍。否則，AWS Glue 爬取程式建立的資料表可能無法正確地代表您的資料。

- 前兩個資料行 (`labeling_set_id`、`label`) 是 AWS Glue 的必要項目。其餘資料行必須與要處理資料的結構描述相符。
- 針對每個 `labeling_set_id`，您可以使用相同的標籤識別所有相符記錄。標籤是置放在 `label` 欄中的唯一字串。我們建議使用包含簡單字元的標籤，例如 A、B、C 等。標籤區分大小寫，並且會在 `label` 資料行中輸入。
- 包含相同 `labeling_set_id` 和相同標籤的資料列會視為是相符的標記。
- 包含相同 `labeling_set_id` 但不同標籤的資料列會視為是不相符的標記。
- 包含不同 `labeling_set_id` 資料會視為未傳達任何資訊或不相符。

以下是標記資料的範例：

labeling_set_id	label	first_name	last_name	生日
ABC123	A	John	Doe	04/01/1980

labeling_set_id	label	first_name	last_name	生日
ABC123	B	Jane	Smith	04/03/1980
ABC123	A	Johnny	Doe	04/01/1980
ABC123	A	Jon	Doe	04/01/1980
DEF345	A	Richard	Jones	12/11/1992
DEF345	A	Rich	Jones	11/12/1992
DEF345	B	Sarah	Jones	12/11/1992
DEF345	C	Richie	Jones Jr.	05/06/2017
DEF345	B	Sarah	Jones-Walker	12/11/1992
GHI678	A	Robert	Miller	1/3/1999
GHI678	A	Bob	Miller	1/3/1999
XYZABC	A	William	Robinson	2/5/2001
XYZABC	B	Andrew	Robinson	2/5/1971

- 在以上的例子中，我們會將 John/Johnny/Jon Doe 識別為相符，並教導系統這些記錄與 Jane Smith 不相符。我們會另外教導系統 Richard 和 Rich Jones 皆為同一人，但這些記錄與 Sarah Jones/Jones-Walker 和 Richie Jones Jr. 不相符。
- 如您所見，標籤的範圍會限制在 `labeling_set_id`。因此，標籤不會超過 `labeling_set_id` 的邊界。例如，位於 `labeling_set_id` 1 中的標籤「A」不會與 `labeling_set_id` 2 中的標籤「A」有任何關聯。
- 若記錄不包含標記組內的任何相符項目，請為其指派一個唯一標籤。例如，Jane Smith 與標記組 ABC123 中的任何記錄，因此它是在有 B 標籤的該標記組中的唯一記錄。
- 標記組「GHI678」表示標記組可以只由授予相同標籤 (以表示其相符) 的兩筆記錄組成。同樣地「XYZABC」表示授予不同標籤給兩筆記錄，以表示其不相符。
- 請注意，有時標記組可能不含任何相符項目 (亦即，您授予標記組中的每筆記錄不同的標籤)，或標記組可能全都「相同」(您授予它們全都相同的標籤)。只要您的標籤組共同包含依您的條件屬於與不屬於「相同」的記錄範例。

> **Important**
>
> 確認您傳遞給 AWS Glue 的 IAM 角色具備包含標記檔案 Amazon S3 儲存貯體的存取權限。按照慣例，AWS Glue 政策會授予許可至名稱字首為 aws-glue- 的 Amazon S3 儲存貯體或資料夾。若您的標記檔案位於不同位置，請在 IAM 角色中將許可新增至該位置。

在 AWS Glue 中調校機器學習轉換

您可以在 AWS Glue 中調校您的機器學習轉換，改善資料清理任務的結果來滿足您的目標。若要改善轉換，您可以產生標記集、新增標籤，然後重複這些步驟多次來教導它，直到您取得所需要的結果為止。您也可以變更一些機器學習參數來進行調校。

如需機器學習轉換的詳細資訊，請參閱使用 AWS Lake Formation FindMatches 比對記錄。

主題

- 機器學習度量
- 在精確度和取回率之間進行選擇
- 在正確性和成本之間進行選擇
- 使用相符項目可信度分數估計相符項目的品質
- 教導尋找相符項目轉換

機器學習度量

若要了解用來調校您機器學習轉換的度量，建議您先熟悉以下術語：

真肯定 (TP)

轉換在資料中正確找到的相符項目，有時候稱為「命中」。

真否定 (TN)

轉換在資料中正確拒絕的不相符項目。

假肯定 (FP)

轉換在資料中不正確地分類為相符項目的不相符項目，有時候稱為「錯誤警示」。

假否定 (FN)

轉換在資料中並未找到的相符項目，有時候稱為「未中」。

如需機器學習中所使用術語的詳細資訊，請參閱 Wikipedia 中的 Confusion matrix。

若要調校您的機器學習轉換，您可以在轉換的 Advanced properties (進階屬性) 中變更下列度量的值。

- Precision (精確度) 會測量轉換在識別為肯定 (真肯定和假肯定) 的記錄總數中，找到真肯定的效果有多好。如需詳細資訊，請參閱 Wikipedia 中的 Precision and recall。
- Recall (取回率) 會測量從來源資料的所有記錄中，轉換找到真肯定的效果有多好。如需詳細資訊，請參閱 Wikipedia 中的 Precision and recall。
- Accuracy (準確性) 會測量轉換找到真肯定和真否定的效果有多好。提高正確性需要更多的機器資源和成本。但也會增加回收率。如需詳細資訊，請參閱 Wikipedia 中的 Accuracy and precision。
- Cost (成本) 會測量執行轉換需要使用多少運算資源 (以及因此產生的金錢成本)。

在精確度和取回率之間進行選擇

每個 `FindMatches` 轉換都包含了 `precision-recall` 參數。您可以使用此參數來指定以下其中一個項目：

- 若您更擔心轉換在兩筆記錄實際上不相符時錯誤地報告成相符，建議您強調「精確度」。
- 如果您更擔心轉換無法偵測到相符的記錄，建議您強調「取回率」。

您可以在 AWS Glue 主控台上進行這項取捨，或是使用 AWS Glue 機器學習 API 操作。

何時應著重在精確度

若您更擔心 `FindMatches` 在兩筆記錄不相符報告成相符的風險，建議您著重在精確度。若要著重在精確度，請選擇「較高」的精確度取回率取捨值。使用更高的值，`FindMatches` 轉換便需要更多證據來判斷兩筆記錄是否相符。轉換會調校成較傾向於將記錄視為不相符。

例如，假設您正在使用 `FindMatches` 偵測影片目錄中的重複項目，並且您為轉換提供了較高的精確度取回率值。若您的轉換不正確地將「星際大戰四部曲：曙光乍現」視為與「星際大戰：帝國反擊戰」相同，想要「星際大戰四部曲：曙光乍現」的客戶便可能會看到「星際大戰：帝國反擊戰」。這會是不佳的客戶體驗。

但是，若轉換無法偵測到「星際大戰四部曲：曙光乍現」與「星際大戰：四部曲 – 曙光乍現」相同，客戶一開始可能會感到困惑，但最後還是會了解到他們是相同的項目。這會是一項錯誤，但不會像先前的案例那樣不佳。

何時應著重在取回率

若您更擔心 FindMatches 轉換可能無法偵測到兩筆記錄實際上是相符項目的風險，建議您著重在取回率。若要著重在取回率，請選擇「較低」的精確度取回率取捨值。使用較低的值，FindMatches 轉換便需要較少證據來判斷兩筆記錄是否相符。轉換會調校成較傾向於將記錄視為相符。

例如，這可能會是安全組織的優先事項。假設您正在將客戶與已知詐騙犯的清單進行比對，此時判斷客戶是否為詐騙犯便非常重要。您正在使用 FindMatches 將詐騙犯清單和客戶清單進行比對。每次 FindMatches 在兩個清單間找到相符項目，便會指派一名稽核人員來驗證該客戶是否確實是詐騙犯。相較於精確度，您的組織可能會偏好選擇取回率。換句話說，您會寧願讓稽核人員手動檢閱並拒絕客戶並非詐騙犯時的一些案例，也不願意在識別確實位於詐騙犯清單上的客戶時失敗。

如何同時著重精確度和取回率

同時改善精確度和取回率的最佳方式是標記更多資料。隨著您標記更多資料，FindMatches 轉換的整體正確性便會獲得改善，進而同時改善精確度和取回率。但是，即使是使用最正確的轉換，仍然還是會有您必須實驗著重精確度或取回率，或是在這兩者之間選擇一個值的灰色地帶。

在正確性和成本之間進行選擇

每個 FindMatches 轉換都包含 accuracy-cost 參數。您可以使用此參數來指定以下其中一個項目：

- 若您更希望轉換能夠正確地報告兩個記錄相符，建議您強調「正確性」。
- 若您更擔心執行轉換的成本或速度，建議您強調「較低成本」。

您可以在 AWS Glue 主控台上進行這項取捨，或是使用 AWS Glue 機器學習 API 操作。

何時應著重在正確性

若您更擔心 find matches 結果不包含任何相符項目的風險，建議您著重在正確性。若要著重在正確性，請選擇「較高」的正確性成本取捨值。使用較高的值，FindMatches 轉換便需要更多時間執行更完整的搜尋，以正確地取得相符記錄。請注意，此參數不會讓錯誤地將不相符兩筆記錄視為相符項目的機率降低。轉換會調校成傾向花費更多時間尋找相符項目。

何時應著重在成本

若您更擔心執行 find matches 轉換的成本，而非找到多少相符項目，建議您著重在成本。若要著重在成本，請選擇「較低」的正確性成本取捨值。使用較低的值，FindMatches 轉換執行時需要的資源便會更少。轉換會調校成傾向尋找較少相符項目。若您可以接受著重在較低成本時的結果，請使用此設定。

如何同時著重正確性及較低成本

檢查更多筆記錄來判斷他們是否為相符項目時，會需要更多的電腦處理時間。若您希望減少成本，卻又不想犧牲品質，以下是一些您可以採取的步驟：

- 消除資料來源中您不在乎是否相符的記錄。
- 從資料來源中消除您確定在判斷相符/不相符時實用性不高的資料行。一個良好的判斷方式便是消除您認為不會在您決定一組記錄是否為「相同」記錄時影響您決策的資料行。

使用相符項目可信度分數估計相符項目的品質

相符項目可信度分數提供 FindMatches 所找到之相符項目的品質估計，以區分機器學習模型中具有高度自信、不確定或不太可能的相符記錄。相符項目可信度分數介於 0 到 1 之間，其中分數越高，表示相似度越高。檢查相符項目可信度分數可讓您區分系統高度可信 (您可能會決定合併) 的相符項目叢集、系統不確定的叢集 (您可能會決定安排人工檢閱)，以及系統認為不太可能的叢集 (可能會決定拒絕)。

如果您看到高相符項目可信度分數，但確定沒有相符項目；或是看到低分數，但實際上確定有相符項目，則可能要調整自己的訓練資料。

存在大規模產業資料集時，可信度分數就特別有用，因為檢閱每個 FindMatches 決定是不切實際的行為。

相符項目可信度分數在 AWS Glue 2.0 或更高版本中推出。

產生相符項目可信度分數

您可以在呼叫 `FindMatches` 或 `FindIncrementalMatches` API 時將 `computeMatchConfidenceScores` 的布林值設定為 True，即可產生相符項目可信度分數。

AWS Glue 將新的 `column match_confidence_score` 新增至輸出。

相符項目評分範例

例如，請考慮下列相符的記錄：

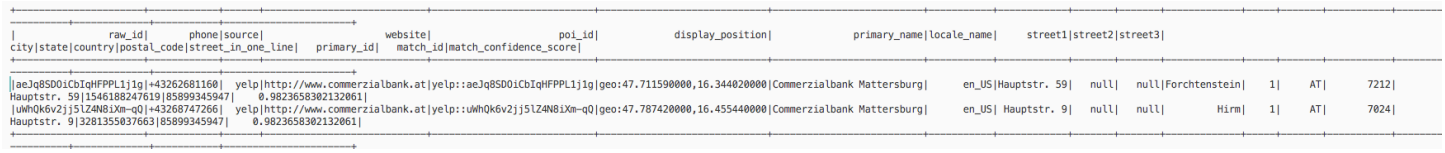
分數 >= 0.9

相符記錄的摘要：

```
primary_id  |   match_id  | match_confidence_score
```

```
3281355037663    85899345947   0.9823658302132061
1546188247619    85899345947   0.9823658302132061
```

詳細資訊:



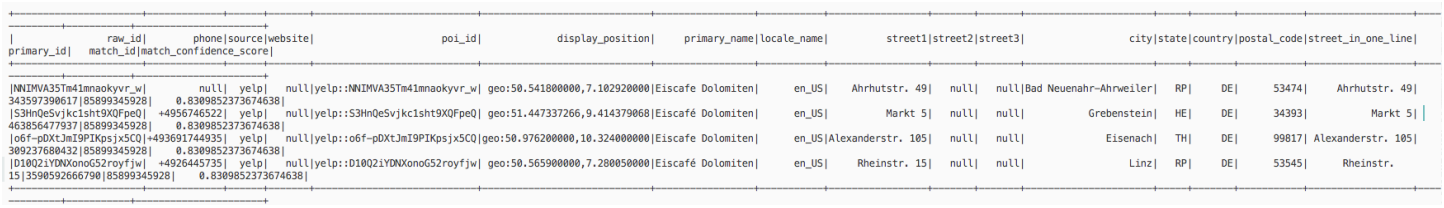
在此範例中，我們可以看到兩條記錄非常相似，共同具有 display_position、primary_name 和 street name。

分數 >= 0.8 和分數 < 0.9

相符記錄的摘要：

```
  primary_id  |   match_id  | match_confidence_score

309237680432     85899345928   0.8309852373674638
3590592666790    85899345928   0.8309852373674638
343597390617     85899345928   0.8309852373674638
249108124906     85899345928   0.8309852373674638
463856477937     85899345928   0.8309852373674638
```

詳細資訊:



在此範例中，我們可以看到這些記錄具有相同的 primary_name 和 country。

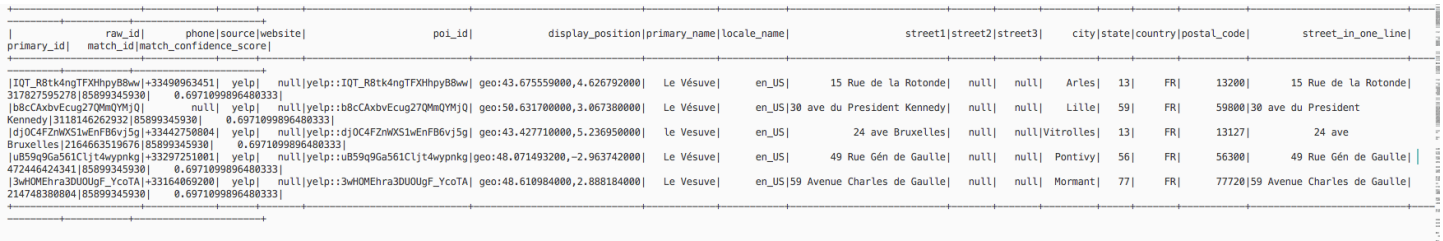
分數 > = 0.6 和分數 < 0.7

相符記錄的摘要：

```
  primary_id  |   match_id  | match_confidence_score

2164663519676    85899345930   0.6971099896480333
```

```
 317827595278    85899345930   0.6971099896480333
 472446424341    85899345930   0.6971099896480333
3118146262932    85899345930   0.6971099896480333
 214748380804    85899345930   0.6971099896480333
```

詳細資訊:



在此範例中，我們可以看到這些記錄僅具有相同的 `primary_name`。

如需詳細資訊，請參閱：

- 步驟 5：使用您的機器學習轉換新增和執行任務
- PySpark：FindMatches 類別
- PySpark：FindIncrementalMatches 類別
- Scala：FindMatches 類別
- Scala：FindIncrementalMatches 類別

教導尋找相符項目轉換

每個 `FindMatches` 轉換都必須經過教導，來了解哪些項目應視為相符項目，哪些項目又應視為不相符的項目。您可以透過新增標籤至檔案，並將您的選擇上傳至 AWS Glue 來教導您的轉換。

您可以在 AWS Glue 主控台中協調此標記，或是使用 AWS Glue 機器學習 API 操作。

我應該新增多少次標籤？ 我需要多少標籤？

這些問題的解答大部分都取決於您。您必須評估 `FindMatches` 是否交付了您所需要的正確性等級，以及您是否認為進行更多標記對您來說確實有價值。決定這一點的最佳方式是查看在 AWS Glue 主控台中選擇 Estimate quality (估計品質) 時可以產生的「Precision (精確度)」、「Recall (取回)」和「Area under the Precision-Recall curve (精確度取回率曲線下方的區域)」指標。在您標記更多組任務後，請重新執行這些指標並驗證他們是否獲得改善。若在標記幾組任務之後，您仍然沒有在您關注的指標上看到改善，表示轉換的品質可能已到達極限。

為何同時需要真肯定和真否定標籤？

FindMatches 轉換需要肯定和否定範例，才能了解您視為相符的項目有哪些。若您標記 FindMatches 產生的培訓資料 (例如使用 I do not have labels (我沒有標籤) 選項)，FindMatches 會嘗試為您產生一組「標籤組 ID」。在每個任務中，您都會將相同的「標籤」給予一些記錄，並將不同的「標籤」給予其他記錄。換句話說，任務通常不是全部相同或是全部不同 (但若特定任務為全部「相同」或全部「不同」也沒什麼問題)。

如果您正在使用 Upload labels from S3 (從 S3 上傳標籤) 選項教導您的 FindMatches 轉換，請嘗試同時包含相符記錄和不相符記錄的範例。您也可以只擁有一個類型。這些標籤可協助您建置更正確的 FindMatches 轉換，但您仍然需要使用 Generate labeling file (產生標記檔案) 選項標記一些您產生的記錄。

我該如何強制讓轉換完全照我教導的方式進行比對？

FindMatches 轉換會從您提供的標籤學習，因此它可能會產生與您所提供標籤不相符的記錄對。若要強制 FindMatches 轉換遵守您的標籤，請選取 FindMatchesParameter 中的 EnforceProvidedLabels。

當機器學習轉換將實際上不相符的項目識別為相符項目時，您可以使用哪些技術？

您可以使用下列技術：

- 將 precisionRecallTradeoff 提高到更高的值。這最後會導致尋找到的相符項目數減少，但應該也會在到達夠高的值時細分您的龐大叢集。
- 擷取對應至不正確結果的輸出資料列，然後將他們重新格式化成標記集 (移除 match_id 資料行並新增 labeling_set_id 和 label 資料行)。若有需要，請分散 (細分) 成多個標記集，確保標記程式能在指派標籤時記住每個標記集。然後，正確地標記相符的資料集、上傳標籤檔案，然後將它附加到您現有的標籤。這可能會足以教導您的轉換程式要了解模式時應尋找的項目為何。
- (進階) 最後，請觀察該資料，查看是否有您可以偵測，但系統並未注意到的模式。使用標準 AWS Glue 函數預先處理該資料，來「正常化」資料。分離您知道對其自身資料行而言重要程度不同的資料，來強調您希望演算法學習的內容。或是從您知道其資料彼此相關的資料行建構合併資料行。

在 AWS Glue 主控台上使用機器學習轉換

您可以使 AWS Glue 用建立可用來清理資料的自訂機器學習轉換。您可以在 AWS Glue 主控台上建立任務時使用這些轉換。

如需如何建立機器學習轉換的資訊，請參閱使用 AWS Lake Formation FindMatches 比對記錄。

主題

- 轉換屬性
- 新增和編輯機器學習轉換
- 檢視轉換詳細資訊
- 使用標籤教導轉換

轉換屬性

若要檢視現有的機器學習轉換，請登入 AWS Management Console，然後在 https://console.aws.amazon.com/glue/ 開啟AWS Glue主控台。在資料整合和 ETL 下的導覽窗格中，選擇資料分類工具 > 記錄比對。

每個轉換的屬性：

轉換名稱

建立轉換時授予它的唯一名稱。

ID

轉換的唯一識別符。

標籤計數

為了協助教導轉換而在標籤檔案中提供的標籤數。

Status

指出轉換為 Ready (準備就緒) 還是 Needs training (需要訓練)。若要在任務中成功執行機器學習轉換，它必須處於就緒。

已建立

建立轉換的日期。

已修改

上次更新轉換的日期。

描述

為轉換提供的描述 (如果已提供)。

AWS Glue 版本

使用的 AWS Glue 版本。

執行識別碼

建立轉換時授予它的唯一名稱。

任務類型

機器學習轉換的類型；例如 Find matching records (尋找相符記錄)。

Status

指出任務執行的狀態。可能的狀態包括：

- 啟動
- 執行中
- 正在停止
- 已停止
- Succeeded
- 失敗
- 逾時

錯誤

如果狀態為「失敗」，會顯示錯誤訊息，說明失敗的原因。

新增和編輯機器學習轉換

您可以在 AWS Glue 主控台上檢視、刪除、設定和教導，或調校轉換。在清單中選取轉換旁邊的核取方塊、選擇 Action (動作)，然後選擇您希望採取的動作。

建立新的 ML 轉換

若要新增機器學習轉換，請選擇建立轉換。依照新增任務精靈中的說明來進行操作。如需詳細資訊，請參閱 使用 AWS Lake Formation FindMatches 比對記錄。

步驟 1. 設定轉換屬性。

1. 輸入名稱和描述 (選用)。
2. 或者，設定安全組態。請參閱使用機器學習轉換的資料加密。
3. 或者，進行任務執行設定。任務執行設定可讓您自訂任務的執行方式。選取工作者類型、工作者數量、任務逾時 (以分鐘為單位)、重試次數及 AWS Glue 版本。

4. 或者設定標籤。標籤是您可以指派給 AWS 資源的標籤。每個標籤皆包含索引鍵與選用值。標籤可用於搜尋和篩選資源或追蹤 AWS 成本。

步驟 2. 選擇資料表和主索引鍵。

1. 選擇 AWS Glue 型錄資料庫和資料表。
2. 從選取的資料表中選擇主索引鍵。主索引鍵資料欄通常包含資料來源中每個記錄的唯一識別碼。

步驟 3。選取調整選項。

1. 對於取回率與精確度，請選擇調整值來調整轉換，以著重於取回率或精確度。依預設，已選取已平衡，但您可以選擇著重於取回率或精確度，或選擇自訂並輸入介於 0.0 到 1.0 (含) 之間的值。
2. 對於低成本與準確度，請選擇調整值以著重於成本或準確度，或選擇自訂並輸入介於 0.0 到 1.0 (含) 之間的值。
3. 對於比對強制執行，如果您想要透過強制輸出比對使用的標籤來教導機器學習轉換，請選擇強制輸出以比對標籤。

步驟 4. 檢閱和建立。

1. 檢閱步驟 1 至 3 的選項。
2. 針對需要修改的任何步驟選擇編輯。選擇建立轉換以完成建立轉換精靈。

使用機器學習轉換的資料加密

將機器學習轉換新增至 AWS Glue，您可以選擇性地指定與資料來源或資料目標相關聯的安全性組態。如果用於存放資料的 Amazon S3 儲存貯體是使用安全組態加密的，請在建立轉換時指定相同的安全組態。

您也可以選擇使用伺服器端加密 AWS KMS (SSE-KMS) 來加密模型和標籤，以防止未經授權的人員檢查模型。如果您選擇此選項，系統會提示您選擇 AWS KMS key 依名稱，或者您可以選擇 [輸入金鑰 ARN]。如果您選擇輸入 KMS 金鑰的 ARN，則會出現第二個欄位，您可以在其中輸入 KMS 金鑰 ARN。

> **Note**
>
> 目前，下列區域不支援使用自訂加密金鑰的機器學習轉換：

- 亞太區域 (大阪)ap-northeast-3

檢視轉換詳細資訊

檢視轉換屬性

轉換屬性頁面包含轉換的屬性。它會顯示轉換定義的詳細資訊，例如以下項目：

- Transform name (轉換名稱) 顯示轉換的名稱。
- Type (類型) 會列出轉換的類型。
- Status (狀態) 顯示轉換是否已準備就緒，可在指令碼或任務中使用。
- Force output to match labels (強制輸出與標籤進行比對) 顯示轉換是否會強制輸出與使用者提供的標籤進行比對。
- Spark version (Spark 版本) 與您在新增轉換時在 Task run properties (任務執行屬性) 中選擇的 AWS Glue 版本相關。AWS Glue 1.0 和 Spark 2.4 是建議大多數客戶使用的版本。如需詳細資訊，請參閱 AWS Glue 版本。

「歷史記錄」、「預估品質」和「標籤」索引標籤

轉換詳細資訊包含您在建立轉換時所定義的資訊。若要查看轉換詳細資訊，請在 Machine learning transforms (機器學習轉換) 清單中選取轉換，然後在下列標籤上檢閱資訊：

- 歷史記錄
- Estimate quality (估計品質)
- 標籤

歷史記錄

History (歷史記錄) 標籤會顯示您的轉換任務執行歷史記錄。為了教導轉換，會執行數種類型的任務。針對每個任務，執行指標包含下列項目：

- Run ID (執行 ID) 是 AWS Glue 在此任務每次執行時所建立的識別符。
- Task type (任務類型) 顯示任務執行的類型。
- Status (狀態) 顯示所列出的每個任務是否成功，其中最上方會列出最近的執行。

- Error (錯誤) 會在執行未成功時，顯示錯誤訊息的詳細資訊。
- Start time (開始時間) 會顯示任務開始的日期和時間 (本地時間)。
- 結束時間會顯示任務結束的日期和時間 (本地時間)。
- Logs (日誌) 會連結至此次任務執行時所寫入 `stdout` 的日誌。

  日誌鏈接將帶您到 Amazon CloudWatch 日誌。您可以在此檢視中建立之表格的詳細資訊，以 AWS Glue Data Catalog 及遇到的任何錯誤。您可以在 CloudWatch 主控台上管理記錄保留期。預設的日誌保留期間為 `Never Expire`。如需有關如何變更保留期的詳細資訊，請參閱 Amazon CloudWatch 日誌使用指南中的變更 CloudWatch 日誌資料保留。
- 標籤檔案會顯示指向所產生標記檔案之 Amazon S3 的連結。

Estimate quality (估計品質)

Estimate quality (估計品質) 標籤會顯示您用來測量轉換品質的指標。透過比較使用標記資料子集轉換比對預測結果與您提供的標籤，來計算預估值。這些預估是近似值。您可以從此標籤呼叫 Estimate quality (估計品質) 任務執行。

Estimate quality (估計品質) 標籤從包括下列屬性的最近一次 Estimate quality (估計品質) 執行顯示指標：

- Area under the Precision-Recall curve (精確度取回率曲線下方的區域) 是一個單一數字，用來估算轉換的整體品質上限。它獨立於為精確度取回率參數所進行的選擇之外。較高的值表示您有較具吸引力的精確度與取回率取捨。
- Precision (精確度) 估算轉換正確預測相符項目的頻率。
- Recall upper limit (取回率上限) 估算針對實際的相符項目，轉換預測為相符的頻率。
- F1 預估介於 0 和 1 之間轉換的準確度，其中 1 表示最佳準確度。如需詳細資訊，請參閱 Wikipedia 中的 F1 score。
- Column importance (欄重要性) 資料表會顯示每個資料行的資料行名稱和重要性分數。欄重要性可協助您了解欄對模型的貢獻方式，藉由識別記錄中的哪些欄比其他欄更重要。此資料可能會提示您新增或變更標籤集，以提高或降低欄的重要性。

  [重要性] 欄會提供每個欄的數值分數，表示為不大於 1.0 的小數點。

如需了解品質估計與真實品質的資訊，請參閱質量估計與 end-to-end （真實）質量。

如需調校您轉換的詳細資訊，請參閱在 AWS Glue 中調校機器學習轉換。

質量估計與 end-to-end （真實）質量

AWS Glue 會透過向內部機器學習模型呈現您為其提供比對標籤，但該模型先前從未看過的數組記錄，來預估轉換品質。這些品質估計是機器學習模型的品質函數，會受到您標記以「教導」轉換的記錄數量影響。或 true 調用 (不會由自動計算ML transform) 也受到篩選機制的影響 end-to-end，該ML transform過濾機制會提出與機器學習的模型有多種可能的匹配。

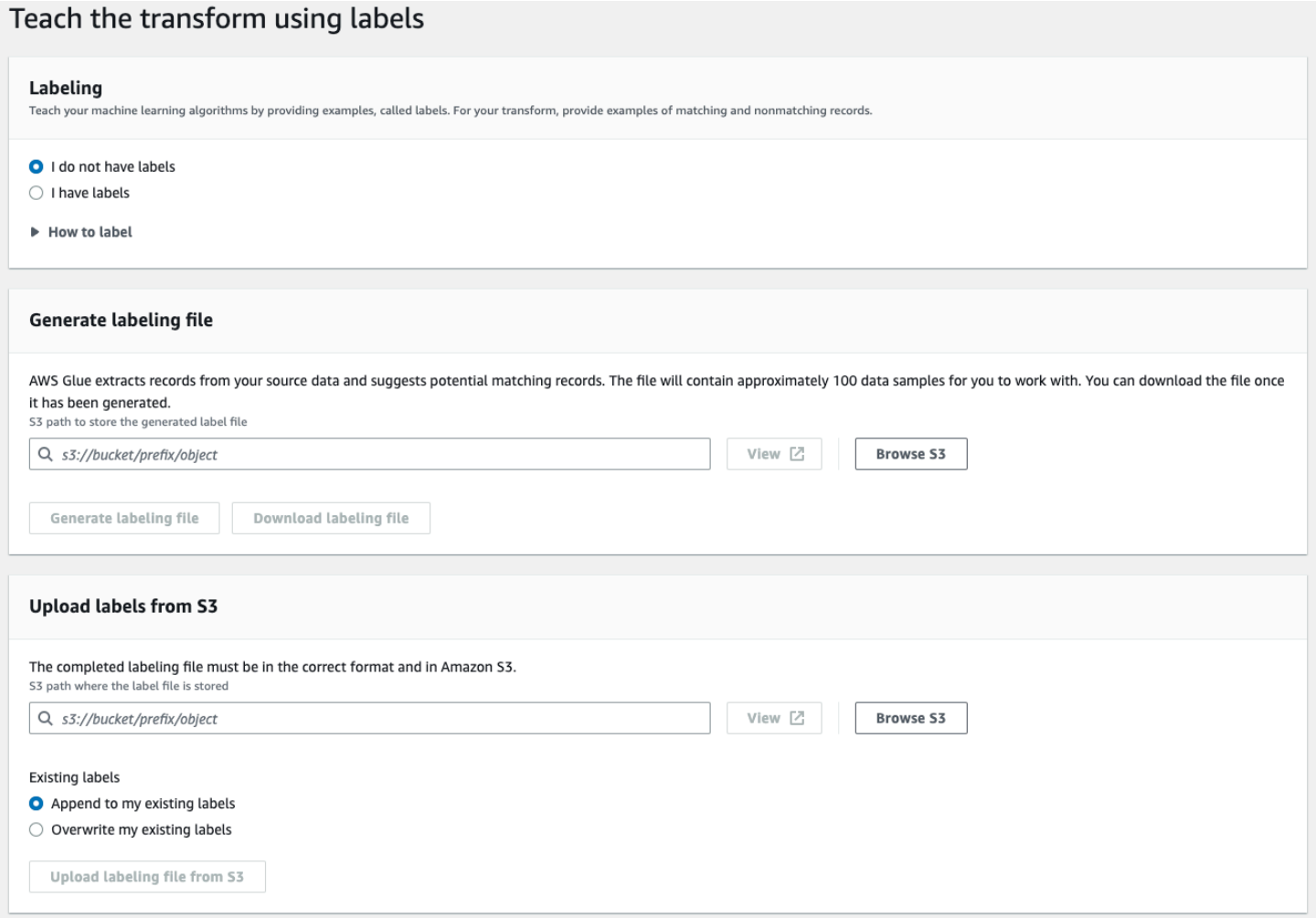
您可以指定較低成本-準確度調整值來重點調整此篩選方法。在您將調整值移向準確度一端時，系統會進行更完整且更詳細的搜尋，以尋找可能是相符項目的記錄組。將更多的記錄對輸入到您的機器學習模型中，而您ML transform的 end-to-end或真實的回收將更接近估計的召回量度。因此，由於比賽的成本/準確度權衡發生變化，比賽 end-to-end 品質的變化通常不會反映在品質估計中。

標籤

標籤是您可以指派給 AWS 資源的標籤。每個標籤皆包含索引鍵與選用值。標籤可用於搜尋和篩選資源或追蹤 AWS 成本。

使用標籤教導轉換

您可以從機器學習轉換詳細資訊頁面選擇教導轉換，以使用標籤 (範例) 教導機器學習轉換。透過提供範例 (稱為標籤) 來教導機器學習演算法時，您可以選擇要使用的現有標籤或建立標記檔案。



- 標記：如果您有標籤，請選擇我有標籤。如果沒有標籤，您仍然可以繼續「產生標記檔案」中的下一步。
- 產生標記檔案：AWS Glue 從來源資料中擷取記錄並建議潛在的相符記錄。您可以選擇 Amazon S3 儲存貯體來存放產生的標籤檔案。選擇產生標記檔案以開始該程序。完成後，選擇下載標記檔案。下載的檔案將有一欄標籤，您可以在其中填寫標籤。
- 從 Amazon S3 上傳標籤：從存放標籤檔案的 Amazon S3 儲存貯體中選擇完成的標記檔案。然後，選擇將標籤附加到現有標籤或覆寫現有標籤。選擇從 Amazon S3 上傳標記檔案。

教學課程：使用 AWS Glue 建立機器學習轉換

本教學會帶您使用 AWS Glue 逐步進行建立及管理機器學習 (ML) 轉換的動作。使用本教學前，建議您先熟悉使用 AWS Glue 主控台新增爬取程式和任務，以及編輯指令碼。您也應熟悉在 Amazon Simple Storage Service (Amazon S3) 主控台上尋找及下載檔案。

在此範例中，您會建立一個 FindMatches 轉換來尋找相符的記錄、教導它如何識別相符及不相符的記錄，並在 AWS Glue 任務中使用它。AWS Glue 任務會寫入一個包含名為 match_id 額外資料行的新 Amazon S3 檔案。

本教學使用的來源資料是名為 dblp_acm_records.csv 的檔案。此檔案是可從原始 DBLP ACM 資料集所取得學術出版物 (DBLP 和 ACM) 的修改版本。dblp_acm_records.csv 檔案是一個逗點分隔值 (CSV) 檔案，其格式為不包含位元組順序標記 (BOM) 的 UTF-8 格式。

第二個檔案 dblp_acm_labels.csv 則是包含相符和不相符記錄的範例標記檔案，會做為本教學的一部分用來教導轉換。

主題

步驟 1：網路爬取來源資料

首先，請先爬取來源 Amazon S3 CSV 檔案來在 Data Catalog 中建立相對應的中繼資料資料表。

> **Important**
>
> 若要指示爬蟲程式只為 CSV 檔案建立資料表，請將 CSV 來源資料存放在與其他檔案不同的 Amazon S3 資料夾中。

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格中，選擇 Crawlers (爬取程式)、Add crawler (新增爬取程式)。
3. 遵循精靈來建立和執行名為 demo-crawl-dblp-acm 的爬取程式，並輸出至 demo-db-dblp-acm 資料庫。執行精靈時，若資料庫尚未存在，請建立 demo-db-dblp-acm 資料庫。選擇 Amazon S3 包含路徑來對目前 AWS 區域中的資料進行採樣。例如，針對 us-east-1，指向來源檔案的 Amazon S3 包含路徑為 s3://ml-transforms-public-datasets-us-east-1/dblp-acm/records/dblp_acm_records.csv。

若作業成功，爬取程式會建立 dblp_acm_records_csv 資料表，其中包含下列資料行：id、標題 (title)、作者 (authors)、地點 (venue)、年份 (year) 及來源 (source)。

步驟 2：建立機器學習轉換

接著，請新增以爬取程式所建立資料來源資料表的結構描述為基礎，名為 demo-crawl-dblp-acm 的機器學習轉換。

1. 在 AWS Glue 主控台的資料整合和 ETL 下的導覽窗格中，選擇資料分類工具 > 記錄比對，然後選擇新增轉換。請遵循精靈來建立包含下列屬性的 Find matches 轉換。

   a. 針對 Transform name (轉換名稱)，請輸入 **demo-xform-dblp-acm**。這是轉換的名稱，用來尋找來源資料中的相符項目。

   b. 針對 IAM role (IAM 角色)，請選擇擁有 Amazon S3 來源資料、標記檔案及 AWS Glue API 操作許可的 IAM 角色。如需詳細資訊，請參閱 AWS Glue 開發人員指南中的為 AWS Glue 建立 IAM 角色。

   c. 對於 Data source (資料來源)，請選擇資料庫 demo-db-dblp-acm 中名為 dblp_acm_csv 的資料表。

   d. 針對 Primary key (主索引鍵)，請選擇資料表的主索引鍵資料行，即 id。

2. 在精靈中，選擇 Finish (完成) 並傳回 ML transforms (機器學習轉換) 清單。

步驟 3：教導您的機器學習轉換

接下來，您會使用教學範例標記檔案來教導您的機器學習轉換。

在其狀態變更為 Ready for use (已準備就緒可供使用) 之前，您無法在擷取、轉換和載入 (ETL) 作業中使用機器學習語言。若要讓您的轉換準備就緒，您必須提供相符和不相符記錄的範例，來教導它如何識別相符與不相符的記錄。若要教導您的轉換，您可以 Generate a label file (產生標記檔案) 及標籤，然後 Upload label file (上傳標記檔案)。在本教學中，您可以使用名為 dblp_acm_labels.csv 的範例標記檔案。如需標記程序的詳細資訊，請參閱 標記。

1. 在 AWS Glue 主控台上的導覽窗格中，選擇記錄比對。
2. 選擇 demo-xform-dblp-acm 轉換，然後選擇 Action (動作)、Teach (教導)。遵循精靈來教導您的 Find matches 轉換。
3. 在轉換屬性頁面上，選擇 I have labels (我擁有標籤)。選擇指向目前 AWS 區域中範例標記檔案的 Amazon S3 路徑。例如，針對 us-east-1，請從 Amazon S3 路徑 s3://ml-transforms-

`public-datasets-us-east-1/dblp-acm/labels/dblp_acm_labels.csv` 使用 overwrite (覆寫) 現有標籤選項來上傳所提供的標記檔案。標記檔案必須位於和 AWS Glue 主控台相同區域內的 Amazon S3 中。

在您上傳標記檔案時，會在 AWS Glue 中啟動任務，來新增或覆寫用來教導轉換如何處理資料來源的標籤。

4. 在精靈的最終頁面上，選擇 Finish (完成)，然後傳回 ML transforms (機器學習轉換) 清單。

步驟 4：估計您機器學習轉換的品質

接著，您可以估計您機器學習轉換的品質。品質會取決於您已完成的標記數。如需估計品質的詳細資訊，請參閱 Estimate quality (估計品質)。

1. 在 AWS Glue 主控台的資料整合和 ETL 下的導覽窗格中，選擇資料分類工具 > 記錄比對。
2. 選擇 `demo-xform-dblp-acm` 轉換，然後選擇 Estimate quality (估計品質) 標籤。此標籤會顯示目前轉換的品質估計 (若可用的話)。
3. 選擇 Estimate quality (估計品質) 來啟動任務，估計轉換的品質。品質估計的正確性會以來源資料的標記為基礎。
4. 導覽至 History (歷史記錄) 標籤。在此窗格中，會為轉換列出任務執行，包括 Estimating quality (估計品質) 任務。如需執行的詳細資訊，請選擇 Logs (日誌)。在完成時，檢查執行狀態是否是 Succeeded (成功)。

步驟 5：使用您的機器學習轉換新增和執行任務

在此步驟中，您會使用機器學習轉換來在 AWS Glue 中新增及執行任務。當 `demo-xform-dblp-acm` 轉換為 Ready for use (已準備就緒可供使用) 時，您便可以在 ETL 任務中使用它。

1. 在 AWS Glue 主控台上，選擇導覽窗格中的 Jobs (任務)。
2. 選擇 Add job (新增任務)，並遵循精靈中的步驟，使用所產生的指令碼來建立 ETL Spark 任務。為您的轉換選擇下列屬性值：

   a. 針對 Name (名稱)，選擇本教學中的範例任務 demo-etl-dblp-acm。

   b. 針對 IAM role (IAM 角色)，選擇具備 Amazon S3 來源資料、標記檔案及 AWS Glue API 操作許可的 IAM 角色。如需詳細資訊，請參閱 AWS Glue 開發人員指南中的為 AWS Glue 建立 IAM 角色。

   c. 針對 ETL language (ETL 語言)，選擇 Scala。這是 ETL 指令碼中的程式語言。

d. 針對 Script file name (指令碼檔案名稱)，選擇 demo-etl-dblp-acm。這是 Scala 指令碼的檔案名稱 (與任務名稱相同)。

e. 針對 Data source (資料來源)，選擇 dblp_acm_records_csv。您選擇的資料來源必須符合機器學習轉換資料來源結構描述。

f. 針對 Transform type (轉換類型)，請選擇 Find matching records (尋找相符記錄) 來使用機器學習轉換建立任務。

g. 清除 Remove duplicate records (移除重複記錄)。由於寫入的輸出記錄包含新增的額外 `match_id` 欄位，因此您不需要移除重複記錄。

h. 針對 Transform (轉換)，選擇任務使用的機器學習轉換 demo-xform-dblp-acm。

i. 針對 Create tables in your data target (在您的資料目標中建立資料表)，選擇使用下列屬性建立資料表：

- 資料存放區類型 — **Amazon S3**
- 格式 – **CSV**
- 壓縮類型 — **None**
- Target path (目標路徑) — 寫入任務輸出的 Amazon S3 路徑 (在目前的主控台 AWS 區域)

3. 選擇 Save job and edit script (儲存任務並編輯指令碼) 來顯示指令碼編輯器頁面。

4. 編輯指令碼以新增陳述式，將指向 Target path (目標路徑) 的任務輸出寫入單一分割區。在執行 `FindMatches` 轉換的陳述式後立即新增此陳述式。陳述式與以下內容相似。

```
val single_partition = findmatches1.repartition(1)
```

您必須修改 `.writeDynamicFrame(findmatches1)` 陳述式，將輸出做為 `.writeDynamicFrame(single_partion)` 寫入。

5. 在您編輯指令碼後，請選擇 Save (儲存)。修改後的指令碼看起來會與以下程式碼相似，但已根據您的環境進行自訂。

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.ml.FindMatches
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
```

```
object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    // @params: [JOB_NAME]
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)
    // @type: DataSource
    // @args: [database = "demo-db-dblp-acm", table_name = "dblp_acm_records_csv",
 transformation_ctx = "datasource0"]
    // @return: datasource0
    // @inputs: []
    val datasource0 = glueContext.getCatalogSource(database = "demo-db-dblp-acm",
 tableName = "dblp_acm_records_csv", redshiftTmpDir = "", transformationContext =
 "datasource0").getDynamicFrame()
    // @type: FindMatches
    // @args: [transformId = "tfm-123456789012", emitFusion = false,
 survivorComparisonField = "<primary_id>", transformation_ctx = "findmatches1"]
    // @return: findmatches1
    // @inputs: [frame = datasource0]
    val findmatches1 = FindMatches.apply(frame = datasource0, transformId
 = "tfm-123456789012", transformationContext = "findmatches1",
 computeMatchConfidenceScores = true)


    // Repartition the previous DynamicFrame into a single partition.
    val single_partition = findmatches1.repartition(1)


    // @type: DataSink
    // @args: [connection_type = "s3", connection_options = {"path": "s3://aws-
glue-ml-transforms-data/sal"}, format = "csv", transformation_ctx = "datasink2"]
    // @return: datasink2
    // @inputs: [frame = findmatches1]
    val datasink2 = glueContext.getSinkWithFormat(connectionType =
 "s3", options = JsonOptions("""{"path": "s3://aws-glue-ml-transforms-
data/sal"}"""), transformationContext = "datasink2", format =
 "csv").writeDynamicFrame(single_partition)
    Job.commit()
  }
}
```

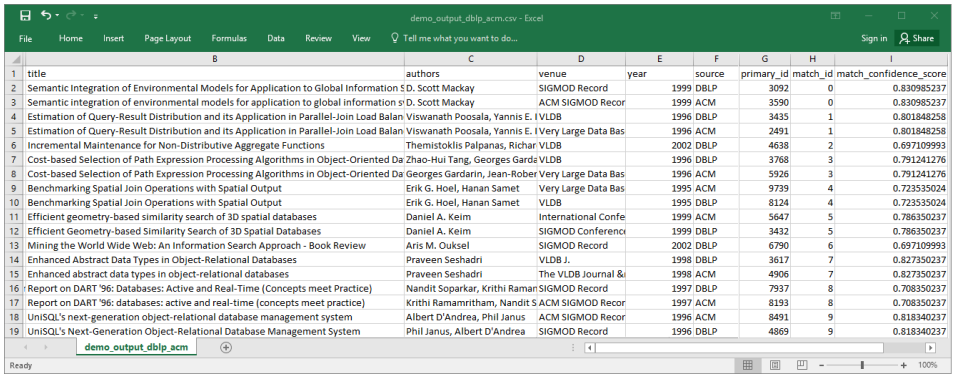
6. 選擇 Run job (執行任務) 來啟動任務執行。在任務清單中檢查任務的狀態。當任務完成時，在 ML transform (ML 轉換)、History (歷史記錄) 索引標籤上，有一個新的 Run ID (執行 ID) 列已加入 ETL job (ETL 任務) 類型。
7. 導覽至 Jobs (任務)、History (歷史記錄) 標籤。在這個窗格中會列出任務執行。如需執行的詳細資訊，請選擇 Logs (日誌)。在完成時，檢查執行狀態是否是 Succeeded (成功)。

步驟 6：驗證 Amazon S3 的輸出資料

在此步驟中，您可以在新增任務時所選擇的 Amazon S3 儲存貯體內檢查任務執行的輸出。您可以將輸出檔案下載到您的本機電腦，並驗證已成功識別相符的記錄。

1. 開啟位於 https://console.aws.amazon.com/s3/ 的 Amazon S3 主控台。
2. 下載任務的目標輸出檔案 `demo-etl-dblp-acm`。在試算表應用程式 (您可能需要為檔案新增 `.csv` 副檔名，才能正常開啟) 中開啟檔案。

   下圖顯示 Microsoft Excel 中的輸出摘要。



	B	C	D	E	F	G	H	I
1	title	authors	venue	year	source	primary_id	match_id	match_confidence_score
2	Semantic Integration of Environmental Models for Application to Global Information S	D. Scott Mackay	SIGMOD Record	1999	DBLP	3092	0	0.830985237
3	Semantic integration of environmental models for application to global information s	D. Scott Mackay	ACM SIGMOD Recor	1999	ACM	3590	0	0.830985237
4	Estimation of Query-Result Distribution and its Application in Parallel-Join Load Balan	Viswanath Poosala, Yannis E. I	VLDB	1996	DBLP	3435	1	0.801848258
5	Estimation of Query-Result Distribution and its Application in Parallel-Join Load Balan	Viswanath Poosala, Yannis E. I	Very Large Data Bas	1996	ACM	2491	1	0.801848258
6	Incremental Maintenance for Non-Distributive Aggregate Functions	Themistoklis Palpanas, Richar	VLDB	2002	DBLP	4638	2	0.697109993
7	Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented Da	Zhao-Hui Tang, Georges Garda	VLDB	1996	DBLP	3768	3	0.791241276
8	Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented Da	Georges Gardarin, Jean-Rober	Very Large Data Bas	1996	ACM	5926	3	0.791241276
9	Benchmarking Spatial Join Operations with Spatial Output	Erik G. Hoel, Hanan Samet	Very Large Data Bas	1995	ACM	9739	4	0.723535024
10	Benchmarking Spatial Join Operations with Spatial Output	Erik G. Hoel, Hanan Samet	VLDB	1995	DBLP	8124	4	0.723535024
11	Efficient geometry-based similarity search of 3D spatial databases	Daniel A. Keim	International Confe	1999	ACM	5647	5	0.786350237
12	Efficient Geometry-based Similarity Search of 3D Spatial Databases	Daniel A. Keim	SIGMOD Conferenc	1999	DBLP	3432	5	0.786350237
13	Mining the World Wide Web: An Information Search Approach - Book Review	Aris M. Ouksel	SIGMOD Record	2002	DBLP	6790	6	0.697109993
14	Enhanced Abstract Data Types in Object-Relational Databases	Praveen Seshadri	VLDB J.	1998	DBLP	3617	7	0.827350237
15	Enhanced abstract data types in object-relational databases	Praveen Seshadri	The VLDB Journal &	1998	ACM	4906	7	0.827350237
16	Report on DART '96: Databases: Active and Real-Time (Concepts meet Practice)	Nandit Soparkar, Krithi Raman	SIGMOD Record	1997	DBLP	7937	8	0.708350237
17	Report on DART '96: databases: active and real-time (concepts meet practice)	Krithi Ramamritham, Nandit S	ACM SIGMOD Recor	1997	ACM	8193	8	0.708350237
18	UniSQL's next-generation object-relational database management system	Albert D'Andrea, Phil Janus	ACM SIGMOD Recor	1996	ACM	8491	9	0.818340237
19	UniSQL's Next-Generation Object-Relational Database Management System	Phil Janus, Albert D'Andrea	SIGMOD Record	1996	DBLP	4869	9	0.818340237

   資料來源和目標檔案都有 4,911 筆記錄。但是，`Find matches` 轉換會新增另一個名為 `match_id` 的資料行，來識別輸出中的相符記錄。`match_id` 相同的資料列會視為相符記錄。`match_confidence_score` 為介於 0 至 1 之間的數字，用來估計 `Find matches` 找到之相符項目的品質。
3. 依據 `match_id` 排序輸出檔案，來輕鬆查看哪些記錄是相符項目。比較其他資料行中的值，查看您是否同意 `Find matches` 轉換的結果。若您不同意其結果，您可以新增更多標籤來繼續教導轉換。

   您也可以依據其他欄位排序檔案 (例如 `title`)，來查看標題相似的記錄是否擁有相同的 `match_id`。

尋找增量改進相符項目

尋找相符項目功能可讓您識別資料集中重複或相符的記錄，即使記錄沒有通用的唯一識別符，也沒有欄位完全相符。尋找相符項目的最初版本轉換單一資料集中識別的相符記錄。將新資料新增至資料集時，您必須將其與現有的乾淨資料集合併，然後針對完整的合併資料集重新執行比對。

增量改進比對功能可讓您更輕鬆地比對現有相符的資料集與增量記錄。假設您想要將潛在客戶資料與現有客戶資料集進行比對。增量改進比對功能可讓您靈活地透過將結果合併至單一資料庫或資料表，將數十萬個新潛在客戶與現有的潛在客戶和客戶資料庫進行比對。藉由僅在新的和現有的資料集之間進行比對，尋找增量改進相符項目最佳化功能可縮短運算時間，同時降低成本。

增量改進比對的用法與尋找相符項目類似，如 教學課程：使用 AWS Glue 建立機器學習轉換 中所述。本主題僅識別與增量改進比對的差異。

如需詳細資訊，請參閱部落格貼文增量改進資料比對。

執行增量改進比對任務

對於下列程序，假設以下情況：

- 您已將現有的資料集網路爬取至資料表 first_records。first_records 資料集必須是相符的資料集，或相符任務的輸出。
- 您已使用 AWS Glue 2.0 版建立並訓練「尋找相符項目」轉換。這是唯一支援增量改進相符項目的 AWS Glue 版本。
- 使用的 ETL 語言是 Scala。請注意，Python 也受到支援。
- 已經產生的模型稱為 `demo-xform`。

1. 將增量改進資料集抓取至 second_records 資料表中。
2. 在 AWS Glue 主控台上，選擇導覽窗格中的 Jobs (任務)。
3. 選擇 Add job (新增任務)，並遵循精靈中的步驟，使用所產生的指令碼來建立 ETL Spark 任務。為您的轉換選擇下列屬性值：

    a. 對於 Name (名稱)，請選擇 demo-etl。

    b. 對於 IAM role (IAM 角色)，請選擇具備 Amazon S3 來源資料、標記檔案及 AWS Glue API 操作許可的 IAM 角色。

    c. 針對 ETL language (ETL 語言)，選擇 Scala。

    d. 對於 Script file name (指令碼檔案名稱)，請選擇 demo-etl。這是 Scala 指令碼的檔案名稱。

e. 對於 Data source (資料來源)，請選擇 first_records。您選擇的資料來源必須符合機器學習轉換資料來源結構描述。

f. 針對 Transform type (轉換類型)，請選擇 Find matching records (尋找相符記錄) 來使用機器學習轉換建立任務。

g. 選取增量改進比對選項，並針對 Data Source (資料來源) 選取名為 second_records 的資料表。

h. 對於 Transform (轉換)，請選擇任務使用的機器學習轉換 demo-xform。

i. 選擇 Create tables in your data target (在您的資料目標中建立資料表) 或 Use tables in the data catalog and update your data target (使用 Data Catalog 中的資料表並更新您的資料目標)。

4. 選擇 Save job and edit script (儲存任務並編輯指令碼) 來顯示指令碼編輯器頁面。

5. 選擇 Run job (執行任務) 來啟動任務執行。

在視覺化任務中使用 FindMatches

若要在 AWS Glue Studio 中使用 FindMatches 轉換，您可使用自訂轉換節點來調用 FindMatches API。如需有關如何使用自訂轉換的詳細資訊，請參閱建立自訂轉換

> **Note**
>
> FindMatches API 目前僅適用於 `Glue 2.0`。若要使用自訂轉換執行任務來調用 FindMatches API，請在任務詳細資訊索引標籤中確認 AWS Glue 版本為 `Glue 2.0`。若 AWS Glue 的版本並非 `Glue 2.0`，則任務在執行期可能會失敗且顯示下列錯誤訊息：「無法從 'awsglueml.transforms' 匯入名稱 'FindMatches'」。

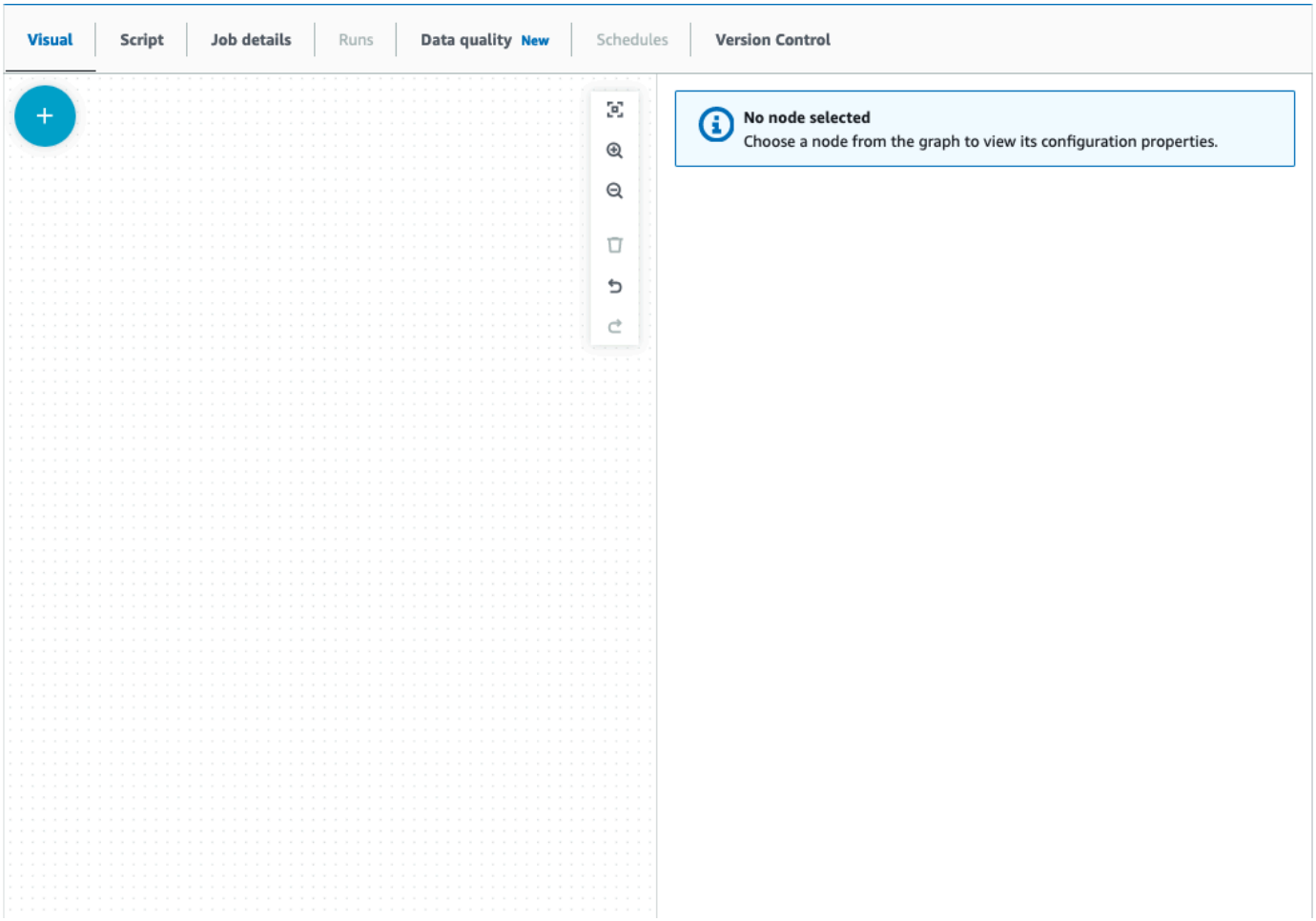
先決條件

- 若要使用 Find Matches 轉換，請在 https://console.aws.amazon.com/gluestudio/ 開啟 AWS Glue Studio 主控台。
- 建立機器學習轉換。建立完成後會產生 transformId。您將需要此 ID 完成以下步驟。如需有關如何建立機器學習轉換的詳細資訊，請參閱 Adding and editing machine learning transforms。

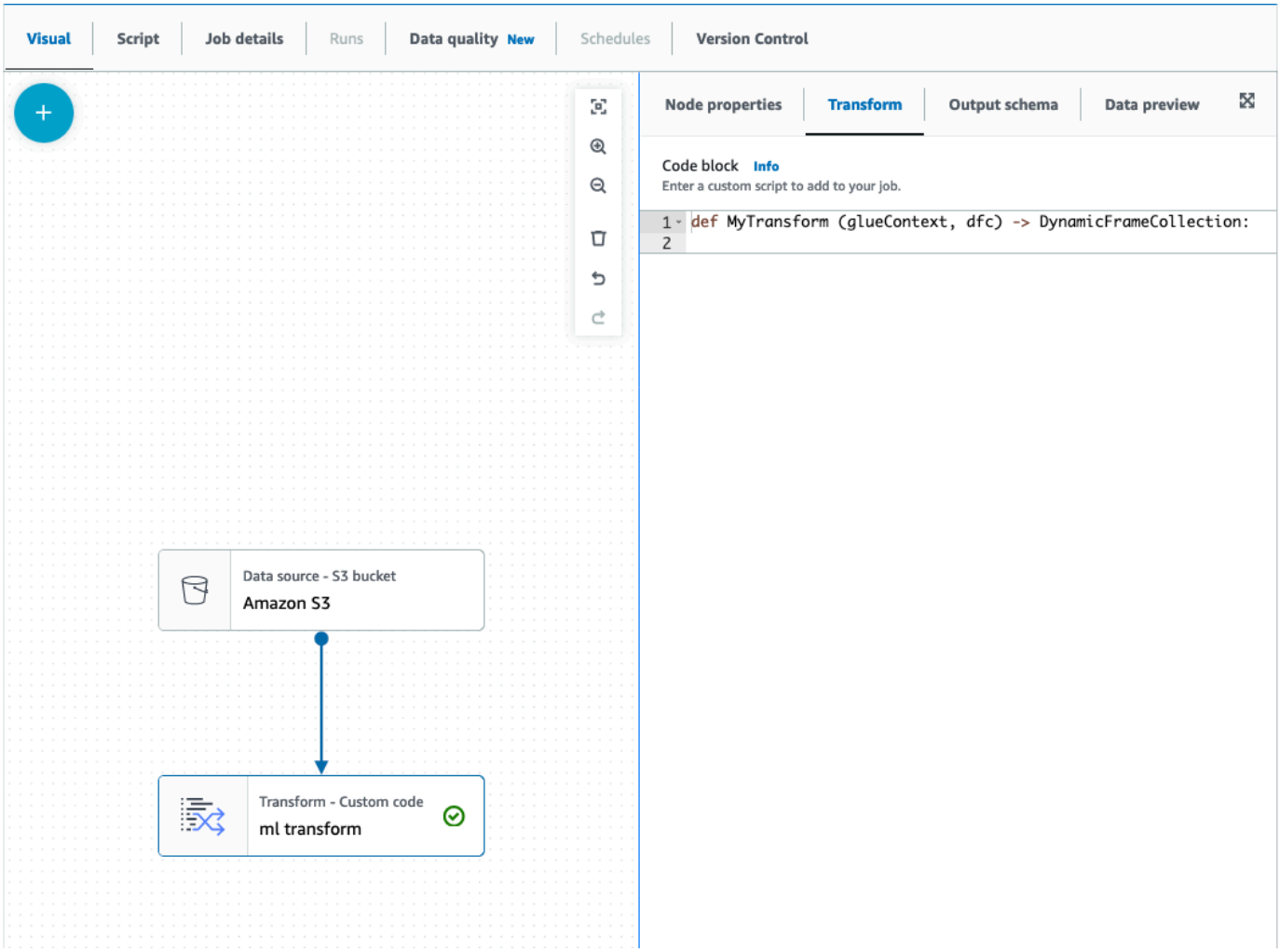
新增 FindMatches 轉換

若要新增 FindMatches 轉換：

1. 在 AWS Glue Studio 任務編輯器中，按一下視覺化任務圖表左上角的十字符號以開啟「資源」面板，然後選擇資料索引標籤以選擇資料來源。此為您要檢查相符項目的資料來源。



2. 選擇資料來源節點，然後按一下視覺化任務圖表左上角的十字符號，開啟「資源」面板並搜尋「自訂轉換」。選擇自訂轉換節點以將其新增至圖表。自訂轉換會連結至資料來源節點。若未連結，則可按一下自訂轉換節點並選擇節點屬性索引標籤，然後在節點父項下方選擇資料來源。
3. 按一下視覺化圖表中的自訂轉換節點，然後選擇節點屬性索引標籤並命名自訂轉換。建議您重新命名轉換，以便在視覺化圖表中輕鬆識別轉換名稱。
4. 選擇轉換索引標籤，您可以在其中編輯程式碼區塊。您可在此處新增程式碼來調用 FindMatches API。



程式碼區塊包含可讓您開始使用的預先填入程式碼。使用以下範本覆寫預先填入的程式碼。此範本具有用於 transformId 的預留位置 (可由您提供)。

```
def MyTransform (glueContext, dfc) -> DynamicFrameCollection:
    dynf = dfc.select(list(dfc.keys())[0])
    from awsglueml.transforms import FindMatches
    findmatches = FindMatches.apply(frame = dynf, transformId = "<your id>")
    return(DynamicFrameCollection({"FindMatches": findmatches}, glueContext))
```

5. 在視覺畫圖表中按一下自訂轉換節點，然後按一下視覺化任務圖表左上角的十字符號，開啟「資源」面板並搜尋「從集合中選取」。由於集合中只有一個 DynamicFrame，無須變更預設選擇。

6. 您可以繼續新增轉換或存放結果，其中現已具有豐富的「尋找相符項目」額外資料欄。若您想在下游轉換中參考這些新資料欄，需要將其新增至轉換輸出結構描述。執行此作業的最簡易方式，是選擇資料預覽索引標籤，然後在結構描述索引標籤中選擇「使用 datapreview 結構描述」。
7. 若要自訂 FindMatchs，您可新增其他參數並傳遞至 'apply' 方法。參閱 FindMatches 類別。

新增 FindMatch 增量轉換

增量比對的程序與新增 FindMatches 轉換相同，但具有下列差異：

- 您需要兩個父節點，而非自訂轉換的父節點。
- 第一個父節點應為資料集。
- 第二個父節點應為增量資料集。

  將 `transformId` 替換為範本程式碼區塊中的 `transformId`：

```
def MyTransform (glueContext, dfc) -> DynamicFrameCollection:
    dfs = list(dfc.values())
    dynf = dfs[0]
    inc_dynf = dfs[1]
    from awsglueml.transforms import FindIncrementalMatches
    findmatches = FindIncrementalMatches.apply(existingFrame = dynf, incrementalFrame
 = inc_dynf,
                                    transformId = "<your id>")
    return(DynamicFrameCollection({"FindMatches": findmatches}, glueContext))
```

- 如需選用參數，請參閱 FindIncrementalMatches 類別。

將 Apache Spark 程式遷移到 AWS Glue

Apache Spark 是在大型資料集上執行的分散式運算工作負載的開放原始碼平台。AWS Glue 利用 Spark 的功能，為 ETL 提供最佳化的體驗。您可以將 Spark 程式遷移到 AWS Glue 以充分利用我們的功能。AWS Glue 提供與 Amazon EMR 上的 Apache Spark 相同的效能增強功能。

執行 Spark 程式碼

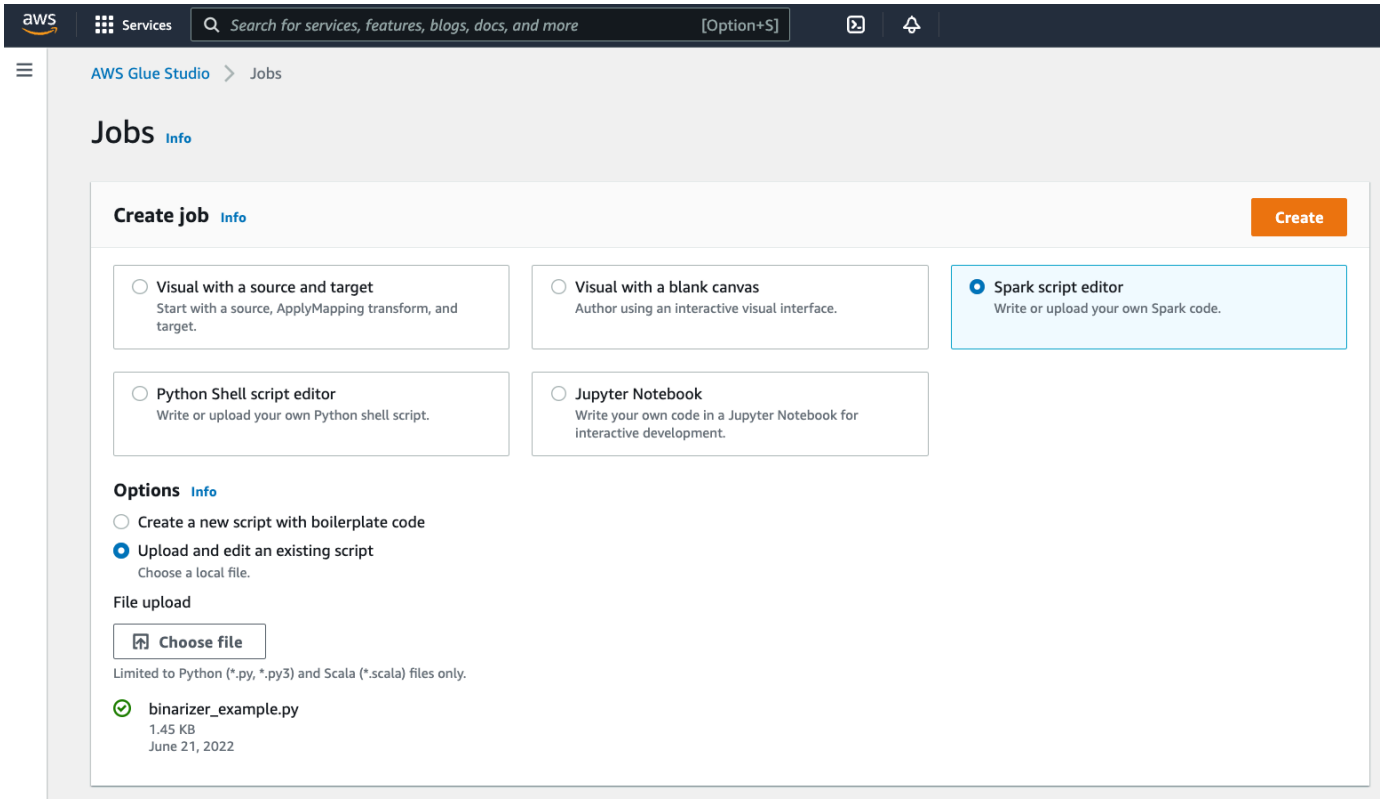
原生 Spark 程式碼可以在 AWS Glue 環境中立即執行。指令碼通常是透過反覆變更一段程式碼來開發，這樣的工作流程很適合互動式工作階段。但是，現有程式碼更適合在 AWS Glue 任務中執行，可讓您安排並持續取得每個指令碼執行的日誌和指標。您可以透過主控台上傳和編輯現有指令碼。

1. 取得指令碼的來源。在此範例中，您將使用 Apache Spark 儲存庫中的範例指令碼。二值化器範例
2. 在 AWS Glue 主控台中，展開左側導覽窗格，然後選取 ETL > Jobs (任務)

   在 Create job (建立任務) 面板中，選取 Spark script editor (Spark 指令碼編輯器)。Options (選項) 區段將會出現。在 Options (選項) 下，選取 Upload and edit an existing script (上傳並編輯現有的指令碼)。

   File upload (檔案上傳) 區段將會出現。在 File upload (檔案上傳) 下，按一下 Choose file (選擇檔案)。您的系統文件選擇器將會出現。導覽到您儲存 `binarizer_example.py` 的位置中，選取它並確認。

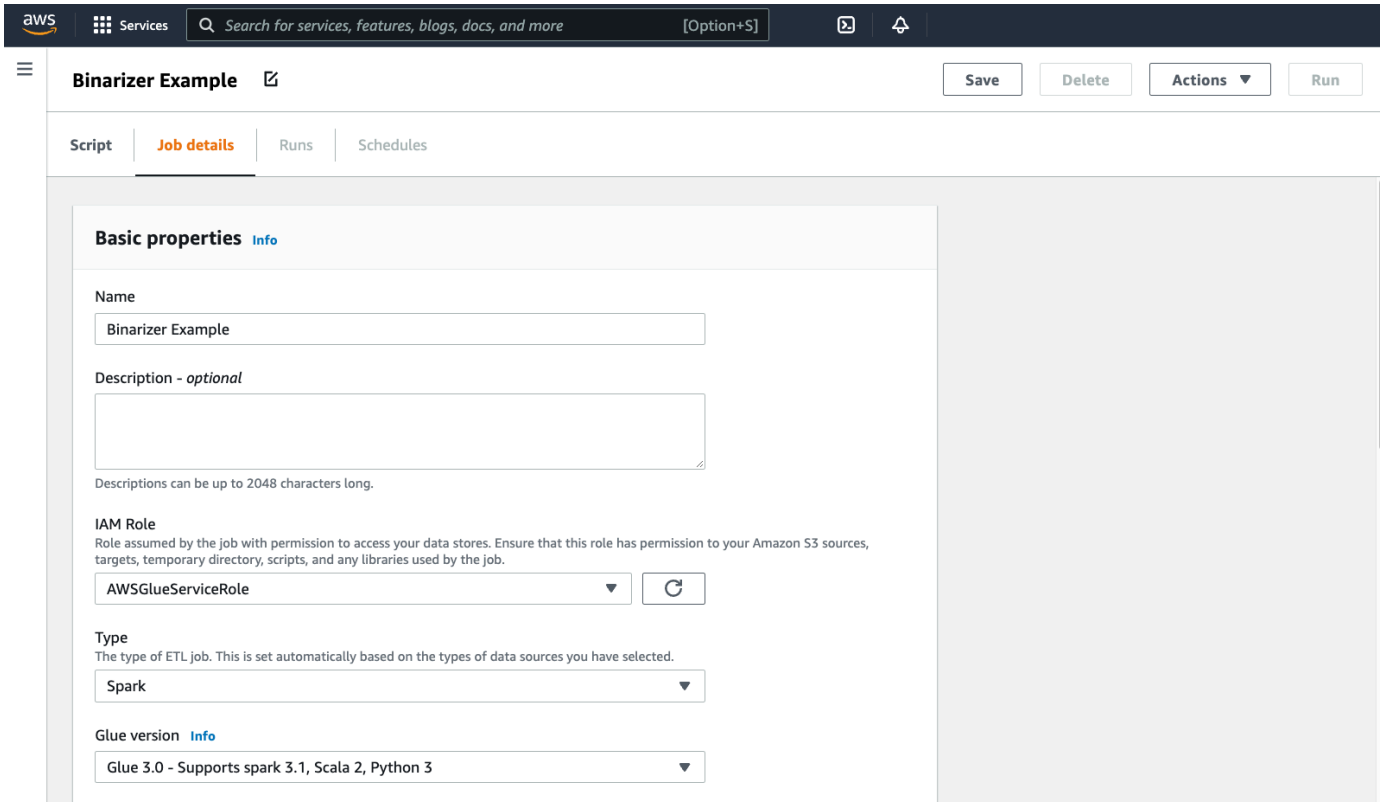
   Create (建立)按鈕會出現在 Create job (建立任務) 面板的標頭。按一下該按鈕。



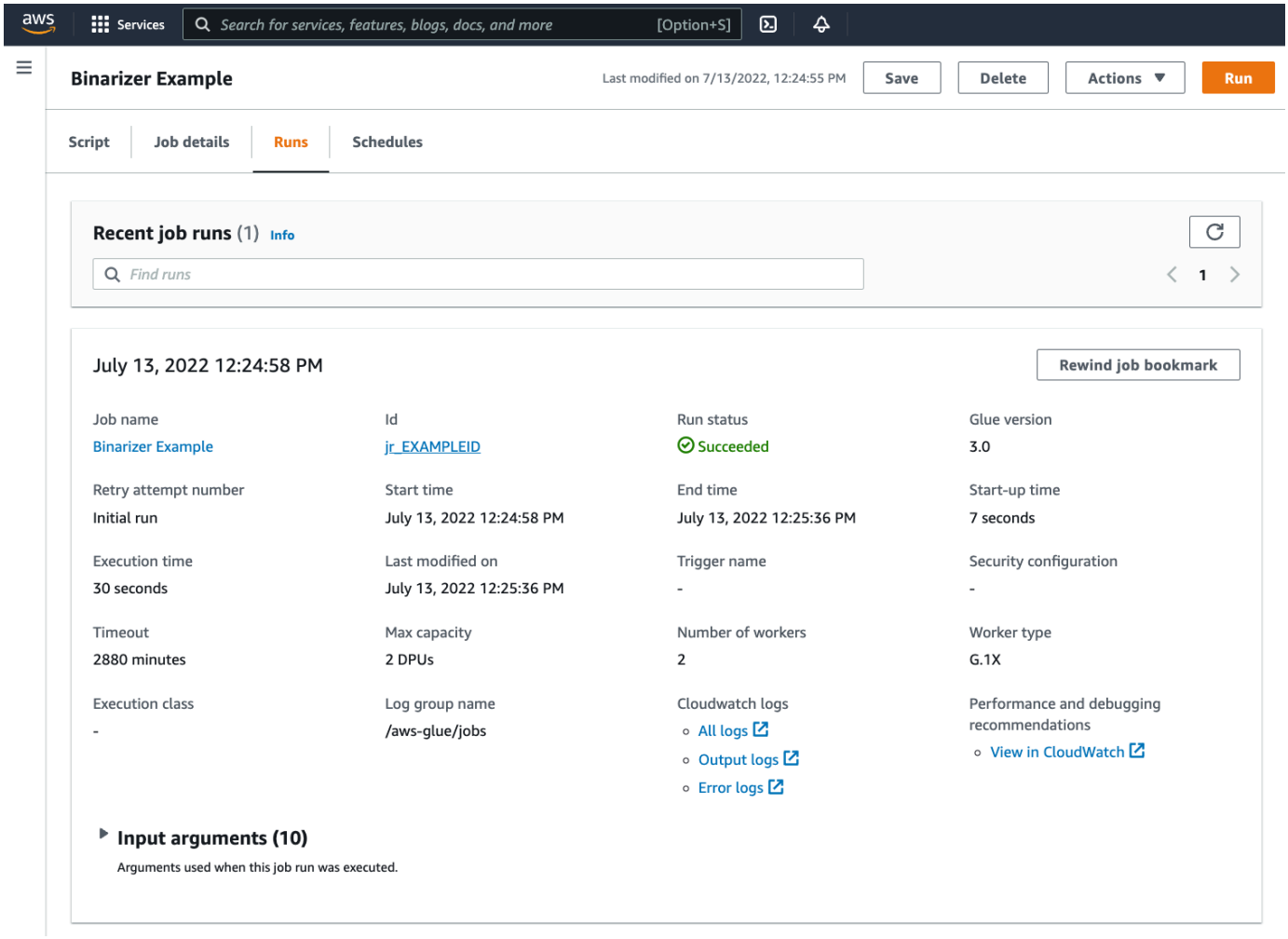
3. 您的瀏覽器將導覽至指令碼編輯器。在標頭上，按一下 Job details (任務詳細資訊) 索引標籤。設定名稱和 IAM 角色。如需 AWS Glue IAM 角色的相關指引，請參閱 the section called “設定 IAM 許可”。

   選用 - 將 Requested number of workers (請求的工作者數目) 設定為 2，並將 Number of retries (重試次數) 設定為 1。這些選項在執行生產任務時非常重要，但是在測試功能時，關閉這些選項可以簡化您的體驗。

   在標題列中，按一下 Save (儲存)，然後點選 Run (執行)

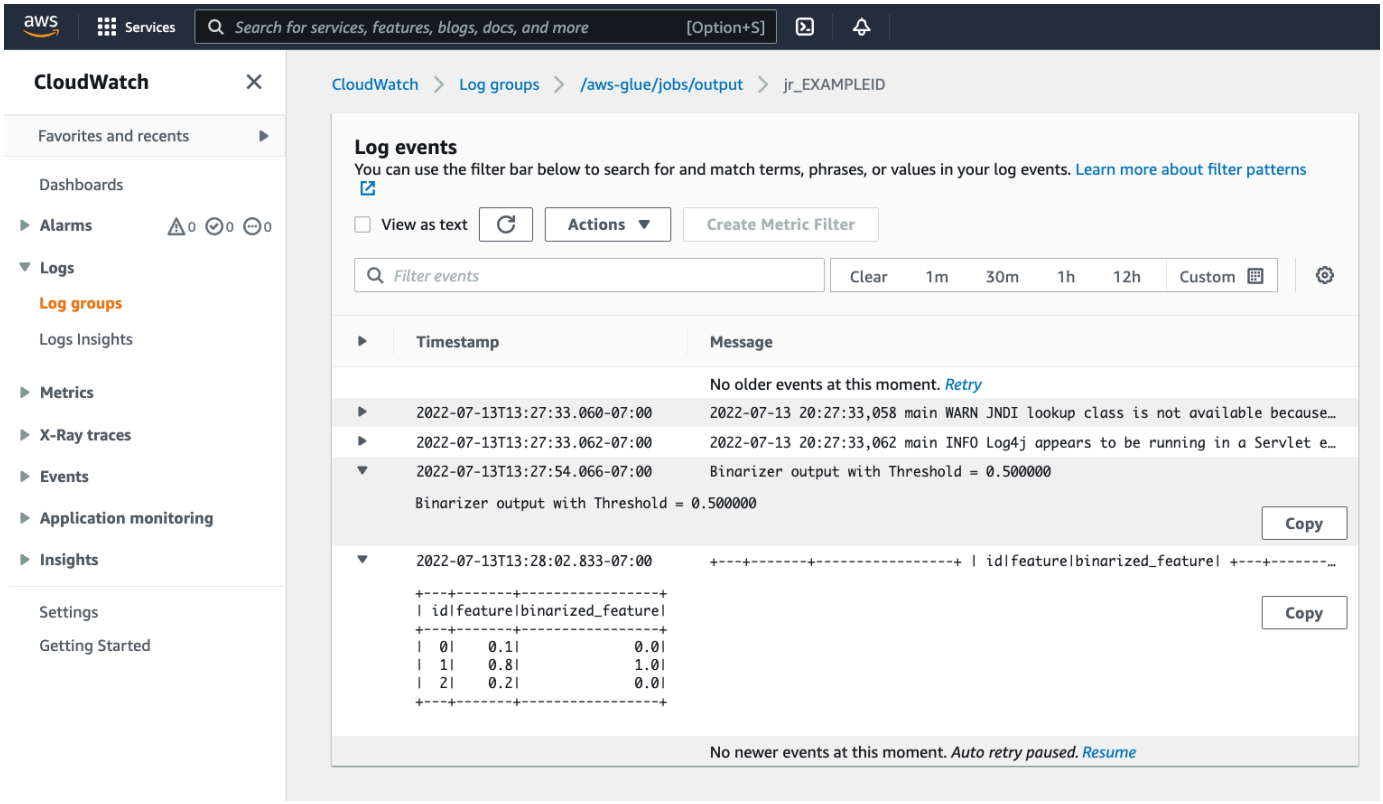


4. 導覽至 Runs (執行) 索引標籤。您會看到與您任務執行對應的面板。請稍候幾分鐘，頁面應該會自動重新整理，並在 Run status (執行狀態) 下顯示 Succeeded (成功)。



5. 您需要檢查輸出，以確認 Spark 指令碼按預期執行。此 Apache Spark 範例指令碼應該將字串寫入至輸出串流。您可以透過導覽至成功任務執行面板中 Cloudwatch logs (CloudWatch 日誌) 下的 Output logs (輸出日誌) 找到該資料。請注意，任務執行運行 ID (在 Id 標籤下產生的 id) 以 `jr_` 開頭 。

   這將開啟 CloudWatch 主控台，設定為可視覺化預設 AWS Glue 日誌群組 `/aws-glue/jobs/output` 的內容，篩選為任務執行 ID 的日誌串流內容。每個工作者都會產生日誌串流，並在 Log streams (日誌串流) 下顯示為資料列。每個工作者應執行請求的程式碼。您需要開啟所有日誌串流來找出正確的工作者。找到正確的工作者之後，您應該會看到指令碼的輸出，如下圖所示：



遷移 Spark 程式所需的常見程序

評估 Spark 版本支援

AWS Glue 發行版本決定了可用於 AWS Glue 任務的 Apache Spark 和 Python 版本。您可以在 the section called “AWS Glue 版本” 找到我們的 AWS Glue 版本和其支援的內容。您可能需要更新 Spark 程式，以便與較新版本的 Spark 相容，才能存取部分 AWS Glue 功能。

包含第三方程式庫

許多現有的 Spark 程式將具有相依性 (在私有和公有成品上)。AWS Glue 支援 Scala 任務的 JAR 樣式相依性和 Python 任務的 Wheel 和來源純 Python 相依性。

Python - 如需 Python 相依性的相關資訊，請參閱 the section called “Python 程式庫”

在 AWS Glue 環境中提供常見的 Python 相依性，包含經常請求的 Pandas 程式庫。相依性包含在 AWS Glue 2.0 以上的版本中。如需有關所提供模組的詳細資訊，請參閱 the section called “AWS Glue 中已提供 Python 模組”。如果您需要提供的任務具有預設包含不同版本的相依性，您可以使用 `--additional-python-modules`。如需任務引數的相關資訊，請參閱 the section called “任務參數”。

您可以為額外的 Python 相依性提供 `--extra-py-files` 任務引數。如果您要從 Spark 程式遷移任務，這個參數就是一個不錯的選擇，因為其在功能上等同於 PySpark 中的 `--py-files` 旗標，並且受到相同限制。如需有關 `--extra-py-files` 參數的詳細資訊，請參閱 the section called “包括具有 PySpark 本地功能的 Python 文件”

對於新任務，您可以使用 `--additional-python-modules` 任務引數來管理 Python 相依性。使用此引數可以獲得更完善的相依性管理體驗。此參數支援 Wheel 樣式相依性，包含具有與 Amazon Linux 2 相容的原生程式碼繫結的相依性。

Scala

您可以為額外的 Scala 相依性提供 `--extra-jars` 任務引數。相依性必須在 Amazon S3 中託管，且引數值應為以逗號分隔的 Amazon S3 路徑清單，並不含空格。先重新綁定相依性再託管和設定相依性，會讓您更輕鬆地管理組態。AWS GlueJAR 相依性包含可以從任何 JVM 語言產生的 Java 位元碼。您可以使用其他 JVM 語言 (例如 Java) 來撰寫自訂相依性。

管理資料來源憑證

現有的 Spark 程式可能帶有複雜或自訂組態，以從其資料來源中提取資料。AWS Glue 連線支援常見的資料來源身分驗證流程。如需有關 AWS Glue 連線的詳細資訊，請參閱 連線至資料。

AWS Glue 連線有助於將您的任務連線到各種類型的資料存放區，這展現在兩種主要方式上：透過對我們程式庫的方法呼叫以及在 AWS 主控台中設定 Additional network connection (其他網路連線)。您也可以從您的任務中呼叫 AWS SDK 以從連線擷取資訊。

方法呼叫 – AWS Glue 連線已與 AWS Glue 資料目錄緊密整合，可讓您彙整資料集的相關資訊，並反映在可用來與 AWS Glue 連線互動的方法。如果您有希望重複使用的現有身分驗證組態，對於 JDBC 連線，您可以透過 `GlueContext` 上的 `extract_jdbc_conf` 方法來存取 AWS Glue 連線組態。如需詳細資訊，請參閱 the section called “extract_jdbc_conf”

主控台組態 – AWS Glue 任務使用關聯 AWS Glue 連線來設定連線至 Amazon VPC 子網路。如果您直接管理安全素材，可能需要在 AWS 主控台中提供 `NETWORK` 類型 Additional network connection (其他網路連線) 來設定路由。如需有關 AWS Glue 連線 API 的詳細資訊，請參閱 the section called “連線”

如果您的 Spark 程式具有自訂或不常見的身分驗證流程，則可能需要以實作方式來管理安全素材。如果 AWS Glue 連線不適合使用，您可以安全地在 Secrets Manager 中託管安全素材，並透過 boto3 或 AWS SDK (在任務中會提供) 執行存取。

設定 Apache Spark

複雜的遷移通常會改變 Spark 組態，以因應其工作負載。現代版本的 Apache Spark 允許執行時間組態與 `SparkSession` 一起設定。AWS Glue3.0 以上的任務提供 `SparkSession`，可以修改以設定執行

時間組態。Apache Spark 組態。調校 Spark 很複雜，AWS Glue 不保證支援所有 Spark 組態的設定。如果您的遷移需要大量 Spark 層級組態，請聯絡支援人員。

設定自訂組態

遷移的 Spark 程式可以設計為採取自訂組態。AWS Glue 允許透過任務引數在任務和任務執行層級上設定組態。如需任務引數的相關資訊，請參閱 the section called “任務參數”。您可以透過我們的程式庫來存取任務內容中的任務引數。AWS Glue 提供公用程式功能，可在任務上設定的引數與任務執行上設定的引數之間提供一致檢視。請在 Python 中參閱 the section called “getResolvedOptions”，並在 Scala 中參閱 the section called “GlueArgParser”。

遷移 Java 程式碼

如同 the section called “第三方程式庫” 中所說明，您的相依性可以包含由 JVM 語言 (例如 Java 或 Scala) 產生的類別。您的相依性可以包含 `main` 方法。您可以使用相依性中的 `main` 方法來作為 AWS Glue Scala 任務的入口點。這可以讓您在 Java 中寫入 `main` 方法，或重複使用封裝至您自身程式庫標準的 `main` 方法。

若要使用來自相依性的 `main` 方法，請執行以下操作：清除提供預設 `GlueApp` 物件的編輯窗格內容。在相依性中提供完全合格的類別名稱，以作為具有金鑰 `--class` 的任務引數。然後，您應該能夠觸發任務執行。

您無法設定引數 AWS Glue 傳遞給 `main` 方法的順序或結構。如果您現有的程式碼需要讀取 AWS Glue 中設定的組態，這可能會導致與先前程式碼的不相容性。如果您使用 `getResolvedOptions`，也無法擁有能夠呼叫此方法的適當位置。考慮直接從 AWS Glue 產生的主要方法來叫用您的相依性。以下展示 AWS Glue ETL 指令碼範例。

```
import com.amazonaws.services.glue.util.GlueArgParser

object GlueApp {
  def main(sysArgs: Array[String]) {
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)

    // Invoke static method from JAR. Pass some sample arguments as a String[], one
 defined inline and one taken from the job arguments, using getResolvedOptions
    com.mycompany.myproject.MyClass.myStaticPublicMethod(Array("string parameter1",
 args("JOB_NAME")))

    // Alternatively, invoke a non-static public method.
    (new com.mycompany.myproject.MyClass).someMethod()
  }
```

```
}
```

在 AWS Glue 中使用 Ray 任務

本節提供有關使用 AWS Glue for Ray 任務的資訊。如需有關撰寫 AWS Glue for Ray 指令碼的詳細資訊，請參閱 the section called “AWS Glue 對於雷” 一節。

主題

- AWS Glue for Ray 入門
- 支援的 Ray 執行期環境
- 計算 Ray 任務中的工作者
- 在 Ray 任務中使用任務參數
- 使用指標監控 Ray 任務

AWS Glue for Ray 入門

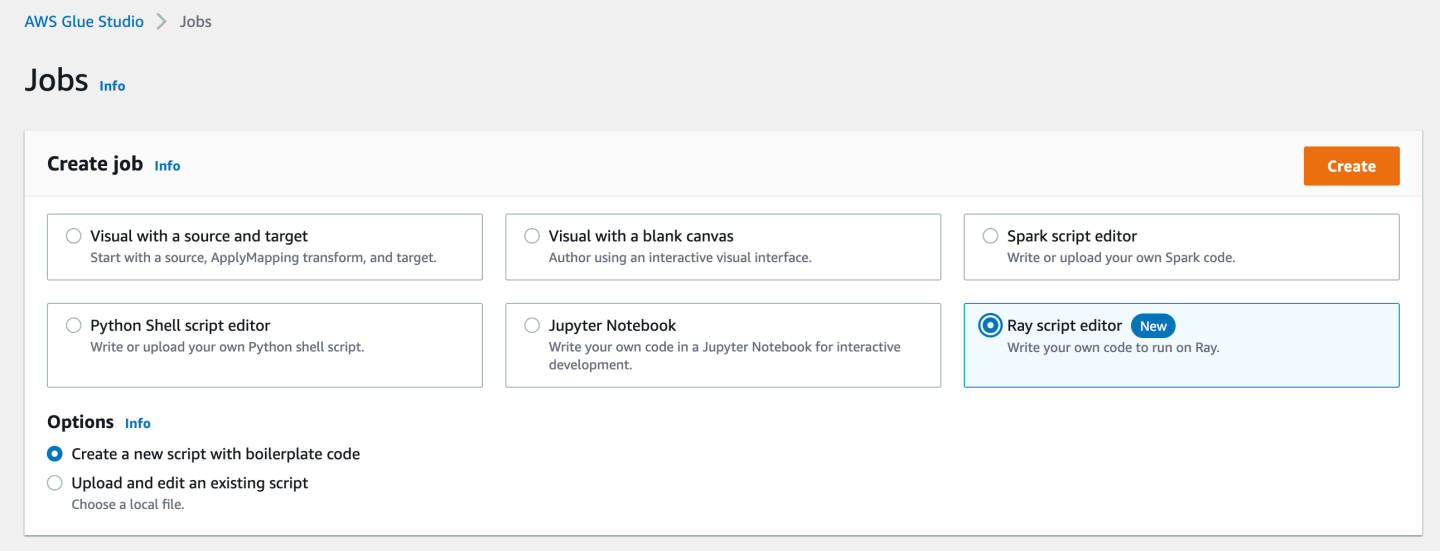
若要使用 AWS Glue for Ray，您可使用與 AWS Glue for Spark 相同的 AWS Glue 任務和互動式工作階段。AWS Glue 任務旨在以週期性節奏執行相同的指令碼，而互動式工作階段則可讓您針對相同的佈建資源，依序執行程式碼片段。

AWS Glue ETL 和 Ray 的底層有所不同，因此在指令碼中，您可以存取不同的工具、功能和組態。作為由 AWS Glue 管理的新運算架構，Ray 具有不同的架構，並使用不同的詞彙來描述其作用。如需詳細資訊，請參閱 Ray 文件中的 Architecture Whitepapers (架構白皮書)。

> **Note**
>
> AWS Glue for Ray 已在美國東部 (維吉尼亞北部)、美國東部 (俄亥俄)、美國西部 (奧勒岡)、亞太區域 (東京) 及歐洲 (愛爾蘭) 推出。

AWS Glue Studio 主控台中的 Ray 任務

在 AWS Glue Studio 主控台的任務頁面上，您可以在 AWS Glue Studio—Ray 指令碼編輯器中建立任務時選擇一個新選項。選擇此選項可在主控台中建立 Ray 任務。如需有關這些任務及其使用方式的詳細資訊，請參閱 使用 AWS Glue Studio 建立視覺化 ETL 任務。



AWS CLI 和 SDK 中的 Ray 任務

AWS CLI 中的 Ray 任務使用與其他任務相同的 SDK 動作及參數。AWS Glue for Ray 為某些參數引入新值。如需有關任務 API 的詳細資訊，請參閱 the section called “任務”。

支援的 Ray 執行期環境

在 Spark 任務中，`GlueVersion` 會決定在 AWS Glue for Spark 任務中可用的 Apache Spark 和 Python 的版本。Python 版本指示針對 Spark 類型任務支援的版本。這不是 Ray 執行期環境的設定方式。

針對 Ray 任務，您應將 `GlueVersion` 設定為 `4.0` (或更高版本)。不過，Ray 任務中可用的 Ray、Python 和其他程式庫的版本由任務定義中的 `Runtime` 欄位決定。

`Ray2.4` 執行期環境將在發布後至少可用 6 個月。隨著 Ray 的快速發展，您將能夠透過未來的執行期環境發行版本合併 Ray 更新和改進。

有效值：`Ray2.4`

執行期值	Ray 和 Python 版本
`Ray2.4` (適用於 AWS Glue 4.0+)	Ray 2.4.0 Python 3.9

其他資訊

- 如需 Ray 發行版本上隨附 AWS Glue 的版本說明，請參閱 the section called “AWS Glue 版本”。
- 如需在執行期環境中提供的 Python 程式庫，請參閱 the section called “Ray 任務隨附的模組”。

計算 Ray 任務中的工作者

AWS Glue 在新的以 Graviton 為基礎的 EC2 工作者類型上執行 Ray 任務，這些工作者類型僅適用於 Ray 任務。為了適當地為工作負載 (專門為其設計了 Ray) 佈建這些工作者，我們提供了與大多數工作者不同的運算資源與記憶體資源比率。為了計算這些資源，我們使用記憶體最佳化資料處理單元 (M-DPU)，而不是標準資料處理單元 (DPU)。

- 一個 M-DPU 對應 4 個 vCPU 和 32 GB 記憶體。
- 一個 DPU 對應 4 個 vCPU 和 16 GB 記憶體。DPU 用於透過 Spark 任務和對應的工作者來計算 AWS Glue 中的資源。

Ray 任務目前擁有一種工作者類型 (即 `Z.2X`) 的存取權。`Z.2X` 工作者映射至 2 個 M-DPU (8 個 vCPU、64 GB 記憶體)，且擁有 128 GB 的磁碟空間。`Z.2X` 機器提供 8 個 Ray 工作者 (每個 vCPU 一個)。

您可以在帳戶中同時使用的 M-DPU 數量取決於服務配額。如需有關 AWS Glue 帳戶限制的詳細資訊，請參閱 AWS Glue 端點和配額。

您可以在任務定義中使用 `--number-of-workers` (`NumberOfWorkers`) 指定可用於 Ray 任務的工作節點數量。如需有關任務 API 中 Ray 值的詳細資訊，請參閱 the section called “任務”。

您可以進一步指定 Ray 任務必須與 `--min-workers` 任務參數一起配置的工作者數量下限。如需有關任務參數的詳細資訊，請參閱 the section called “參考資料”。

在 Ray 任務中使用任務參數

您可以為 AWS Glue Ray 任務設定引數，採用的方式與為 Spark 任務設定 AWS Glue 引數的方式相同。如需 AWS Glue API 的詳細資訊，請參閱the section called “任務”。您可以使用本參考資料中列出的不同引數來設定 AWS Glue Ray 任務。您也可以提供自己的引數。

您可以透過主控台在 Job Parameters (任務參數) 標題下的 Job details (任務詳細資訊) 索引標籤中設定任務。藉由在任務上設定 `DefaultArguments` 或在任務執行上設定 `Arguments`，您也可以透過 AWS CLI 來設定任務。預設引數和任務參數會在多次執行中留在任務內。

例如，下列是執行使用 `--arguments` 來設定特殊參數之任務的語法。

```
$ aws glue start-job-run --job-name "CSV to CSV" --arguments='--scriptLocation="s3://
my_glue/libraries/test_lib.py",--test-environment="true"'
```

設定引數後，您可以透過環境變數從 Ray 任務中存取任務參數。這可為您提供為每個執行設定任務的方法。環境變數的名稱將是不含 -- 字首的任務引數名稱。

例如，在之前的範例中，變數名稱將是 scriptLocation 和 test-environment。然後，您需透過標準程式庫中可用的方法擷取引數：test_environment = os.environ.get('test-environment')。如需有關使用 Python 存取環境變數的詳細資訊，請參閱 Python 文件中的 os module。

設定 Ray 任務產生日誌的方式

依預設，Ray 任務產生的日誌和指標會傳送到 CloudWatch 和 Amazon S3。您可以使用 --logging_configuration 參數來變更日誌的產生方式，目前您可以使用此參數來讓 Ray 任務停止產生各種類型的日誌。此參數採用 JSON 物件，其金鑰對應至您想要變更的日誌/行為。此參數支援下列金鑰：

- CLOUDWATCH_METRICS – 設定可用於將任務運作狀態視覺化的 CloudWatch 指標系列。如需指標的詳細資訊，請參閱 the section called “Ray 任務指標”。
- CLOUDWATCH_LOGS – 設定 CloudWatch 日誌，以提供 Ray 有關任務執行狀態的應用程式層級詳細資料。如需日誌的詳細資訊，請參閱 the section called “對 Ray 錯誤進行疑難排解”。
- S3 – 設定 AWS Glue 寫入 Amazon S3 的內容，主要是與 CloudWatch 日誌類似的資訊，但是以檔案形式而非日誌串流。

若要停用 Ray 日誌行為，請提供值 {\"IS_ENABLED\": \"False\"}。例如，若要停用 CloudWatch 指標和 CloudWatch 日誌，請提供下列設定：

```
"--logging_configuration": "{\"CLOUDWATCH_METRICS\": {\"IS_ENABLED\": \"False\"},
 \"CLOUDWATCH_LOGS\": {\"IS_ENABLED\": \"False\"}}"
```

參考資料

Ray 任務會辨識以下引數名稱，您可以用來為 Ray 任務和任務執行設定指令碼環境：

- --logging_configuration – 用於停止產生 Ray 任務建立的各種日誌。依預設，這些日誌會在所有 Ray 任務上產生。格式：逸出字串 JSON 物件。如需更多詳細資訊，請參閱 the section called “設定 Ray 任務產生日誌的方式”。

- `--min-workers` – 配置給 Ray 任務的工作節點最小數目。工作節點可以執行多個複本，每個虛擬 CPU 一個複本。格式：整數。下限：0。上限：在任務定義的 `--number-of-workers` (`NumberOfWorkers`) 中指定的值。如需有關計算工作節點的詳細資訊，請參閱 the section called "計算 Ray 任務中的工作者"。
- `--object_spilling_config`：AWS Glue for Ray 支援使用 Amazon S3 作為擴充 Ray 物件存放區可用空間的一種方式。若要啟用此行為，您可以使用此參數提供 Ray 一個物件溢出的 JSON 組態物件。如需有關 Ray 物件溢出組態的詳細資訊，請參閱 Ray 文件中的 Object Spilling。格式：JSON 物件。

  AWS Glue for Ray 僅支援溢出到磁碟或一次溢出到 Amazon S3。您可以提供多個溢出位置，只要它們遵守此限制即可。溢出到 Amazon S3 時，您還需要為此儲存貯體新增 IAM 許可至任務。

  使用 CLI 將 JSON 物件作為組態提供時，您必須將其作為字串提供，並對 JSON 物件進行字串逸出。例如，溢出到一個 Amazon S3 路徑的字串值如下所示：`"{\"type\": \"smart_open\", \"params\": {\"uri\":\"s3path\"}}"`。在 AWS Glue Studio 中，將此參數作為 JSON 物件提供，不需要額外的格式化處理。
- `--object_store_memory_head` – 配置給 Ray 前端節點上 Plasma 物件存放區的記憶體。此執行個體會執行叢集管理服務以及工作者複本。此值代表暖啟動後執行個體上可用記憶體的百分比。您使用此參數調整記憶體密集型工作負載，大多數使用案例都可以接受預設值。格式：正整數。下限：1。上限：100。

  如需有關 Plasma 的詳細資訊，請參閱 Ray 文件中的 The Plasma In-Memory Object Store (Plasma 記憶體內物件存放區)。
- `--object_store_memory_worker` – 配置給 Ray 工作節點上 Plasma 物件存放區的記憶體。這些執行個體僅執行工作者複本。此值代表暖啟動後執行個體上可用記憶體的百分比。此參數用於調整記憶體密集型工作負載 – 大多數使用案例都可以接受預設值。格式：正整數。下限：1。上限：100。

  如需有關 Plasma 的詳細資訊，請參閱 Ray 文件中的 The Plasma In-Memory Object Store (Plasma 記憶體內物件存放區)。
- `--pip-install` – 要安裝的一組 Python 套件。您可以使用此引數從 PyPI 安裝套件。格式：以逗號分隔的清單。

  PyPI 套件項目的格式為 `package==version`，其中包含 PyPI 名稱和目標套件的版本。項目使用 Python 版本比對來比對套件和版本。例如 ==，而非單一等於 =。也存在其他的版本比對運算子。如需詳細資訊，請參閱 Python 網站上的 PEP 440。您還可以使用 `--s3-py-modules` 提供自訂模組。

- `--s3-py-modules`：託管 Python 模組分佈的一組 Amazon S3 路徑。格式：以逗號分隔的清單。

  您可以使用它來分發自己的模組到 Ray 任務。您還可以使用 `--pip-install` 提供來自 PyPI 的模組。與 AWS Glue ETL 不同，自訂模組的設定不是透過 pip 進行，而是會傳遞給 Ray 進行分發。如需更多詳細資訊，請參閱 the section called “Ray 任務的其他 Python 模組”。
- `--working-dir`：Amazon S3 中託管之 .zip 檔案的路徑，其中包含要分發到執行 Ray 任務的所有節點的檔案。格式：字串。如需更多詳細資訊，請參閱 the section called “為 Ray 任務提供檔案”。

使用指標監控 Ray 任務

您可以使用 AWS Glue Studio 和 Amazon CloudWatch 來監控 Ray 任務。CloudWatch 可透過 Ray 收集並處理來自 AWS Glue 的原始指標，進而使這些指標可供分析。這些指標會在 AWS Glue Studio 主控台中以視覺化方式呈現，因此您可以在任務執行時監控任務。

如需如何監控 AWS Glue 的一般概觀，請參閱the section called “使用 CloudWatch 指標”。如需如何使用 AWS Glue 發佈的 CloudWatch 指標的一般概觀，請參閱the section called “使用 CloudWatch 進行監控”。

在 AWS Glue 主控台中監控 Ray 任務

在任務執行的詳細資訊頁面的執行詳細資訊區段下方，您可以檢視預先建立的彙總圖表，其中將可用的任務指標以視覺化方式呈現。AWS Glue Studio 會針對每個任務執行傳送任務指標到 CloudWatch。您可以使用這些任務指標建立叢集和工作的設定檔，以及存取有關每個節點的詳細資訊。

如需有關可用指標圖表的詳細資訊，請參閱 the section called “檢視 Ray 工作執行的 Amazon CloudWatch 測量結果”。

CloudWatch 中的 Ray 任務指標概觀

在 CloudWatch 中啟用詳細監控功能時，我們會發佈 Ray 指標。指標會發佈至 `Glue/Ray` CloudWatch 命名空間。

- 執行個體指標

  我們會針對指派給任務的執行個體，發佈 CPU、記憶體和磁碟使用率的指標。這些指標會由 `ExecutorId`、`ExecutorType` 和 `host` 等特性識別。這些指標是標準 Linux CloudWatch 代理程式指標的子集。您可以在 CloudWatch 文件中找到指標名稱和特性的相關資訊。如需詳細資訊，請參閱 CloudWatch 代理程式收集的指標。
- Ray 叢集指標

我們會將執行指令碼的 Ray 程序中的指標轉送至此命名空間，然後提供對您而言最重要的指標。可用的指標可能因 Ray 版本而異。如需有關任務正在執行之 Ray 版本的詳細資訊，請參閱 the section called “AWS Glue 版本”。

Ray 會在執行個體層級收集指標。它還提供了任務和叢集的指標。如需有關 Ray 基礎指標策略的詳細資訊，請參閱 Ray 文件中的指標。

> **Note**
>
> 我們不會將 Ray 指標發佈至僅用於 AWS Glue ETL 任務的 `Glue/Job Metrics/` 命名空間。

設定 Python 殼層工作的工作屬性 AWS Glue

您可以使用 Python shell 任務，在 AWS Glue 中以 shell 的方式執行 Python 指令碼。使用 Python 殼層工作，您可以執行與 Python 3.6 或 3.9 相容的指令碼。

主題

限制

注意 Python Shell 任務的下列限制：

- 您無法使用任務書籤搭配 Python shell 任務。
- 你不能打包任何 Python 庫作為`.egg`文件在 Python 3.9 +。請改用 `.whl`。
- 由於 S3 資料的暫時複本有限制，`--extra-files` 選項無法使用。

為 Python shell 任務定義任務屬性

這些章節描述了在中AWS Glue Studio定義或使用 AWS CLI 的工作屬性。

AWS Glue Studio

當您在 AWS Glue Studio 中定義 Python Shell 任務時，您會提供以下一些屬性：

IAM 角色

指定用於授權用於對用於執行任務和存取資料存放區的資源進行授權的 AWS Identity and Access Management (IAM) 角色。如需在 AWS Glue 執行任務之許可的詳細資訊，請參閱 AWS Glue 的身分識別與存取管理。

Type

選擇 Python shell (Python shell) 來使用名為 `pythonshell` 的任務執行 Python 指令碼。

Python 版本

選擇 Python 版本。預設為 Python 3.9。有效版本為 Python 3.6 和 Python 3.9。

載入常見的分析程式庫 (建議)

選擇此選項可在 Python Shell 中包含 Python 3.9 的常見程式庫。

如果您的程式庫是自訂的，或者與預先安裝的程式庫衝突，您可以選擇不安裝常見程式庫。但是，除了常見程式庫之外，您還可以安裝其他程式庫。

當您選取此選項時，`library-set` 選項設定為 `analytics`。當您取消選取此選項時，`library-set` 選項設定為 `none`。

指令碼檔案名稱和指令碼路徑

指令碼中的程式碼定義了任務的程序性邏輯。您可以在 Amazon Simple Storage Service (Amazon S3) 中提供指令碼名稱和位置。請確認路徑中沒有跟指令碼目錄名稱相同的檔案。若要進一步了解使用指令碼，請參閱 AWS Glue 編程指南。

指令碼

指令碼中的程式碼定義了任務的程序性邏輯。您可以使用 Python 3.6 或 Python 3.9 編寫指令碼。您可以在 AWS Glue Studio 中編輯指令碼。

資料處理單位

可在此任務執行時分配的 AWS Glue 資料處理單位 (DPU) 數目上限。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價。

您可以將值設為 0.0625 或 1。預設為 0.0625。在任一情況下，執行個體的本機磁碟都是 20 GB。

CLI

您也可以使用建立 Python 殼層工作 AWS CLI，如下列範例所示。

```
aws glue create-job --name python-job-cli --role Glue_DefaultRole
    --command '{"Name" :  "pythonshell", "PythonVersion": "3.9", "ScriptLocation" :
"s3://DOC-EXAMPLE-BUCKET/scriptname.py"}'
    --max-capacity 0.0625
```

> **Note**
>
> 您不需要指定的版本，AWS Glue因為參數`--glue-version`不適用於 AWS Glue shell 作業。指定的任何版本都將被忽略。

您建立的工作 AWS CLI 預設值為 Python 3。有效的 Python 版本為 3 (對應於 3.6) 和 3.9。若要指定 Python 3.6，請將此元組新增至 `--command` 參數：`"PythonVersion":"3"`

若要指定 Python 3.9，請將此元組新增至 `--command` 參數：`"PythonVersion":"3.9"`

若要設定 Python shell 任務所用的容量上限，請提供 `--max-capacity` 參數。 無法將 `--allocated-capacity` 參數用於 Python shell 任務。

適用於 Python shell 任務的支援程式庫

在使用 Python 3.9 的 Python Shell 中，您可以選擇程式庫集來使用預先封裝的程式庫集來滿足您的需求。您可以使用 `library-set` 選項來選擇程式庫集。有效值為 `analytics` 和 `none`。

執行 Python shell 任務的環境支援以下程式庫：

Python 版本	Python 3.6	Python 3.9	
程式庫集	無	分析	無
avro		1.11.0	
awscli	116.242	1.23.5	1.23.5
awswrangler		2.15.1	

Python 版本	Python 3.6	Python 3.9	
botocore	1.12.232	1.24.21	1.23.5
boto3	1.9.203	1.21.21	
elasticsearch		8.2.0	
numpy	1.16.2	1.22.3	
pandas	0.24.2	1.4.2	
psycopg2		2.9.3	
pyathena		2.5.3	
PyGreSQL	5.0.6		
PyMySQL		1.0.2	
pyodbc		4.0.32	
pyorc		0.6.0	
redshift-connector		2.0.907	
請求	2.22.0	2.27.1	
scikit-learn	0.20.3	1.0.2	
scipy	1.2.1	1.8.0	
SQLAlchemy		1.4.36	
s3fs		2022.3.0	

您可以在 Python shell 任務中使用 NumPy 程式庫以進行科學運算。如需詳細資訊，請參閱NumPy。下列範例顯示可在 NumPy Python 殼層作業中使用的指令碼。指令碼會列印「Hello world」和多個數學計算的結果。

```
import numpy as np
print("Hello world")

a = np.array([20,30,40,50])
print(a)

b = np.arange( 4 )

print(b)

c = a-b

print(c)

d = b**2

print(d)
```

提供自己的 Python 程式庫

使用 PIP

Python Shell 使用 Python 3.9 允許您在任務層級提供額外的 Python 模組或不同的版本。您可以使用 `--additional-python-modules` 選項與逗號分隔的 Python 模組清單來新增新模組或變更現有模組的版本。使用 Python Shell 任務時，您無法使用此參數來提供在 Amazon S3 上託管的自訂 Python 模組。

例如，要更新或新增新的 `scikit-learn` 模組，請使用以下索引鍵與值：`"--additional-python-modules", "scikit-learn==0.21.3"`。

AWS Glue 使用 Python Package Installer (pip3) 來安裝其他模組。您可以在 `--additional-python-modules` 值內傳遞額外的 pip3 選項。例如 `"scikit-learn==0.21.3 -i https://pypi.python.org/simple/"`。來自 pip3 的任何不相容或限制都適用。

> **Note**
>
> 為避免未來出現不相容，我們建議您使用為 Python 3.9 建置的程式庫。

使用 Egg 或 Whl 文件

您可能已有一或多個 Python 程式庫封裝為 `.egg` 或 `.whl` 檔案。若是如此，您可以使用「`--extra-py-files`」旗標下的 AWS Command Line Interface (AWS CLI)，將它們指定到您的任務，如下列範例所示。

```
aws glue create-job --name python-redshift-test-cli --role role --command '{"Name" :
 "pythonshell", "ScriptLocation" : "s3://MyBucket/python/library/redshift_test.py"}'
    --connections Connections=connection-name --default-arguments '{"--extra-py-
files" : ["s3://DOC-EXAMPLE-BUCKET/EGG-FILE", "s3://DOC-EXAMPLE-BUCKET/WHEEL-FILE"]}'
```

如果您不確定如何從 Python 程式庫建立 `.egg` 或 `.whl` 檔案，請使用以下步驟。此範例適用於 macOS、Linux 和適用於 Linux 的 Windows 子系統 (WSL)。

建立 Python .egg 或 .whl 檔案

1. 在 Virtual Private Cloud (VPC) 建立 Amazon Redshift 叢集並將一些資料新增到資料表。
2. 為用來建立叢集的 VPC SecurityGroup-子網路組合建立連AWS Glue線。測試連線是否成功。
3. 建立名為 `redshift_example` 的目錄，並建立名為 `setup.py` 的檔案。 將以下程式碼貼入 `setup.py`。

   ```
   from setuptools import setup

   setup(
       name="redshift_module",
       version="0.1",
       packages=['redshift_module']
   )
   ```

4. 在 `redshift_example` 目錄中，建立 `redshift_module` 目錄。在 `redshift_module` 目錄中，建立檔案 `__init__.py` 和 `pygresql_redshift_common.py`。
5. 將 `__init__.py` 檔案留空。在 `pygresql_redshift_common.py` 中貼上以下程式碼。在先前的程式碼中，將 *port*、*db_name*、*user* 和 *password_for_user* 取代為 Amazon Redshift 叢集的特定詳細資訊。將 *table-name* 取代為 Amazon Redshift 中的資料表名稱。

   ```
   import pg


   def get_connection(host):
       rs_conn_string = "host=%s port=%s dbname=%s user=%s password=%s" % (
   ```

```
        host, port, db_name, user, password_for_user)

    rs_conn = pg.connect(dbname=rs_conn_string)
    rs_conn.query("set statement_timeout = 1200000")
    return rs_conn


def query(con):
    statement = "Select * from table_name;"
    res = con.query(statement)
    return res
```

6. 如果您尚未這麼做，請變更至 redshift_example 目錄。
7. 執行以下任意一項：

   - 執行下列命令以建立 .egg 檔案。

     ```
     python setup.py bdist_egg
     ```

   - 執行下列命令以建立 .whl 檔案。

     ```
     python setup.py bdist_wheel
     ```

8. 安裝上述命令所需的相依項目。
9. 該命令會在 dist 目錄中建立檔案。

   - 如果您建立了 egg 檔案，會命名為 redshift_module-0.1-py2.7.egg。
   - 如果您建立了 wheel 檔案，會命名為 redshift_module-0.1-py2.7-none-any.whl。

   將此檔案上傳到 Amazon S3。

   在此範例中，上傳的檔案路徑為 *s3://DOC-EXAMPLE-BUCKET/EGG-FILE* 或 *s3://DOC-EXAMPLE-BUCKET/WHEEL-FILE*。

10. 建立要用來做為 AWS Glue 任務之指令碼的 Python 檔案，並將以下程式碼新增到該檔案。

```
from redshift_module import pygresql_redshift_common as rs_common

con1 = rs_common.get_connection(redshift_endpoint)
res = rs_common.query(con1)
```

```
print "Rows in the table cities are: "

print res
```

11. 將上述檔案上傳到 Amazon S3。在此範例中，上傳的檔案路徑為 *s3://DOC-EXAMPLE-BUCKET/scriptname.py*。
12. 使用此指令碼建立 Python shell 任務。在 AWS Glue 主控台的 Job properties (任務屬性) 頁面上，在 Python library path (Python 程式庫路徑) 方塊中指定 `.egg/.whl` 檔案的路徑。如果您有多個 `.egg/.whl` 檔案和 Python 檔案，請在此方塊中提供以逗號分隔的清單。

    修改或重新命名 `.egg` 檔案時，檔案名稱必須使用由「python setup.py bdist_egg」指令產生的預設名稱，或必須承繼 Python 模組命名慣例。如需詳細資訊，請參閱 Python 程式碼樣式指南。

    使用 AWS CLI，使用指令建立工作，如下列範例所示。

    ```
    aws glue create-job --name python-redshift-test-cli --role Role --command
     '{"Name" :  "pythonshell", "ScriptLocation" : "s3://DOC-EXAMPLE-BUCKET/
    scriptname.py"}'
         --connections Connections="connection-name" --default-arguments '{"--extra-
    py-files" : ["s3://DOC-EXAMPLE-BUCKET/EGG-FILE", "s3://DOC-EXAMPLE-BUCKET/WHEEL-
    FILE"]}'
    ```

    當任務執行時，指令碼會列印在 Amazon Redshift 叢集 *table_name* 資料表中建立的列。

AWS CloudFormation 與 Python 殼層工作搭配使用 AWS Glue

您可以在中使 AWS CloudFormation 用 Python 殼層作業AWS Glue。以下是範例：

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Python39Job:
    Type: 'AWS::Glue::Job'
    Properties:
      Command:
        Name: pythonshell
        PythonVersion: '3.9'
        ScriptLocation: 's3://bucket/location'
      MaxRetries: 0
      Name: python-39-job
```

```
      Role: RoleName
```

Amazon CloudWatch 日誌組 Python 外殼任務輸出是/aws-glue/python-jobs/output. 針對錯誤，請參閱日誌群組 /aws-glue/python-jobs/error。

監控 AWS Glue

監控對於維護 AWS Glue 及其他 AWS 解決方案的可靠性、可用性和效能至關重要。AWS 提供監控工具，可讓您監看 AWS Glue、在發現錯誤時回報，並適時自動採取動作：

您可以使用下列自動化監控工具來監看 AWS Glue，並在發生錯誤時進行回報：

- Amazon CloudWatch Events 會提供近乎即時的系統事件串流，說明 AWS 資源的變動情形。CloudWatch Events 能自動化執行事件導向型運算。您可以編寫規則，在其他 AWS 服務內監看特定事件，並在這些事件發生時觸發自動化動作。如需詳細資訊，請參閱 Amazon CloudWatch Events 使用者指南。
- Amazon CloudWatch Logs 可讓您監控、存放及存取來自 Amazon EC2 執行個體、AWS CloudTrail 或其他來源的日誌檔案。CloudWatch Logs 可監控日誌檔內的資訊，並在滿足特定閾值時向您發出通知。您也可以將日誌資料存檔在高耐用性的儲存空間。如需詳細資訊，請參閱 Amazon CloudWatch Logs 使用者指南。
- AWS CloudTrail 可擷取 AWS 帳戶發出或代表發出的 API 呼叫和相關事件，並傳送日誌檔案至您指定的 Amazon S3 儲存貯體。您可以找出呼叫 AWS 的使用者和帳戶、發出呼叫的來源 IP 地址，以及呼叫的發生時間。如需詳細資訊，請參閱《AWS CloudTrail 使用者指南》。

此外，也可以存取 AWS Glue 主控台中的下列深入解析，以協助您針對工作進行偵錯和分析：

- Spark 工作 – 您可以看到所選 CloudWatch 指標系列的視覺化，而較新的工作可以存取 Spark UI。如需更多詳細資訊，請參閱 the section called “監控 Spark 任務”。
- Ray 工作 – 您可以看到所選 CloudWatch 指標系列的視覺化。如需更多詳細資訊，請參閱 the section called “Ray 任務指標”。

主題

- AWS 中的標籤 AWS Glue
- 透過 CloudWatch Events 自動化 AWS Glue
- AWS Glue 資源監控

- 使用 AWS CloudTrail 記錄 AWS Glue API 呼叫

AWS 中的標籤 AWS Glue

為了利於您管理 AWS Glue 資源，您可以選擇將自己的標籤指派至某些 AWS Glue 資源類型。標籤是指派給 AWS 資源的標籤。每個標籤皆包含由您定義的一個金鑰與一個選用值。您可以在 AWS Glue 中使用標籤，方便您整理和識別資源。標籤可以用來建立成本會計報告，以及限制資源的存取情況。如果您使用的是 AWS Identity and Access Management，您可以控制 AWS 帳戶中哪些使用者有權建立、編輯或刪除標籤。除了呼叫標籤相關 API 的許可外，還需要呼叫連線上標記 API 的 `glue:GetConnection` 許可，以及呼叫資料庫上標記 API 的 `glue:GetDatabase` 許可。如需詳細資訊，請參閱 用 Glue AWS。

在 AWS Glue 中，您可以標記下列資源：

- 連線
- 資料庫
- 爬蟲程式
- 互動式會話
- 開發端點
- 任務
- 觸發條件
- 工作流程
- 藍圖
- 機器學習轉換
- 資料品質規則集
- 串流結構描述
- 串流結構描述登錄檔

> **Note**
>
> 最佳實務的做法為允許標記這些 AWS Glue 資源，隨時將 `glue:TagResource` 動作包含至您的政策當中。

在 AWS Glue 上使用標籤時，請考慮下列事項。

- 每個實體最多支援使用 50 個標籤。
- 在 AWS Glue 中，您以 `{"string": "string" ...}` 格式將標籤指定為金鑰值對清單。
- 當您在物件上建立標籤時，需要標籤索引鍵，而標籤值為選用。
- 標籤索引鍵和標籤值皆區分大小寫。
- 標籤索引鍵與標籤值不得包含 aws 字首。這類標籤不可執行任何操作。
- 最大標籤索引鍵長度為 128 個 UTF-8 形式的 Unicode 字元。標籤索引鍵不得為空或 null。
- 最大標籤值長度為 256 個 UTF-8 形式的 Unicode 字元。標籤索引鍵可以為空或 null。

AWS Glue 連接的標記支持

您可以限制 `CreateConnection`、`UpdateConnection`、`GetConnection` 和 `DeleteConnection` 動作許可 (根據資源標籤)。這可讓您對具有需要從「資料目錄」擷取 JDBC 連線資訊的 JDBC 資料來源的 AWS Glue 工作實作最低權限存取控制。

範例使用方式

創建與標籤 [「 AWS Glue 連接類」，「開發測試」] 的連接。

為 IAM 政策中的 `GetConnection` 動作指定標籤條件。

```
{
   "Effect": "Allow",
   "Action": [
     "glue:GetConnection"
     ],
   "Resource": "*",
   "Condition": {
     "ForAnyValue:StringEquals": {
       "aws:ResourceTag/tagKey": "dev-test"
     }
   }
 }
```

範例

以下範例會使用指派的標籤來建立任務。

AWS CLI

```
aws glue create-job --name job-test-tags --role MyJobRole --command
 Name=glueetl,ScriptLocation=S3://aws-glue-scripts//prod-job1
--tags key1=value1,key2=value2
```

AWS CloudFormation JSON

```
{
  "Description": "AWS Glue Job Test Tags",
  "Resources": {
    "MyJobRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "glue.amazonaws.com"
                ]
              },
              "Action": [
                "sts:AssumeRole"
              ]
            }
          ]
        },
        "Path": "/",
        "Policies": [
          {
            "PolicyName": "root",
            "PolicyDocument": {
              "Version": "2012-10-17",
              "Statement": [
                {
                  "Effect": "Allow",
                  "Action": "*",
                  "Resource": "*"
                }
              ]
            }
          }
```

```
        ]
      }
    },
    "MyJob": {
      "Type": "AWS::Glue::Job",
      "Properties": {
        "Command": {
          "Name": "glueetl",
          "ScriptLocation": "s3://aws-glue-scripts//prod-job1"
        },
        "DefaultArguments": {
          "--job-bookmark-option": "job-bookmark-enable"
        },
        "ExecutionProperty": {
          "MaxConcurrentRuns": 2
        },
        "MaxRetries": 0,
        "Name": "cf-job1",
        "Role": {
           "Ref": "MyJobRole",
           "Tags": {
                "key1": "value1",
                "key2": "value2"
           }
        }
      }
    }
  }
}
```

AWS CloudFormation YAML

```
Description: AWS Glue Job Test Tags
Resources:
  MyJobRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
```

```
                - glue.amazonaws.com
            Action:
              - sts:AssumeRole
      Path: "/"
      Policies:
        - PolicyName: root
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action: "*"
                Resource: "*"
  MyJob:
    Type: AWS::Glue::Job
    Properties:
      Command:
        Name: glueetl
        ScriptLocation: s3://aws-glue-scripts//prod-job1
      DefaultArguments:
        "--job-bookmark-option": job-bookmark-enable
      ExecutionProperty:
        MaxConcurrentRuns: 2
      MaxRetries: 0
      Name: cf-job1
      Role:
        Ref: MyJobRole
        Tags:
           key1: value1
           key2: value2
```

如需詳細資訊，請參閱 AWS 標記策略。

如需有關如何使用標籤控制存取權的詳細資訊，請參閱用 Glue AWS。

透過 CloudWatch Events 自動化 AWS Glue

您可以使用 Amazon CloudWatch Events 來自動化您的 AWS 服務，並自動回應應用程式可用性問題或資源變更等系統事件。AWS 服務的事件會以接近即時的方式傳送到 CloudWatch Events。您可編寫簡單的規則，來指示您在意的事件，以及當事件符合規則時所要自動執行的動作。可以自動觸發的動作如下：

- 呼叫 AWS Lambda 函數
- 呼叫 Amazon EC2 執行命令

- 將事件轉傳至 Amazon Kinesis Data Streams
- 啟動 AWS Step Functions 狀態機器
- 通知 Amazon SNS 主題或 Amazon SQS 佇列

下列是 CloudWatch Events 與 AWS Glue 搭配使用的部分範例：

- 在 ETL 任務成功時啟動 Lambda 函數
- 在 ETL 任務失敗時通知 Amazon SNS 主題

下列 CloudWatch Events 由 AWS Glue 產生。

- `"detail-type":"Glue Job State Change"` 的事件是針對SUCCEEDED、FAILED、TIMEOUT 和 STOPPED 產生。
- `"detail-type":"Glue Job Run Status"` 的事件是針對 RUNNING、STARTING 和 STOPPING 任務執行產生 (在超過任務延遲通知閾值時)。您必須設定任務延遲通知閾值屬性才能接收這些事件。

  超過任務延遲通知閾值時，每個任務執行狀態只會產生一個事件。
- `"detail-type":"Glue Crawler State Change"` 的事件是針對 Started、Succeeded 和 Failed 產生。
- `"detail-type":"Glue Data Catalog Database State Change"` 的事件是針對 CreateDatabase、DeleteDatabase、CreateTable、DeleteTable 和 BatchDeleteTable 產生。例如，建立或刪除資料表時，便會將通知傳送到 CloudWatch Events。請注意，您無法撰寫會根據通知事件的順序或存在的程式，因為這些事件可能會被移出序列或遺漏。盡可能發出事件。在通知詳細資訊中：
  - typeOfChange 會包含 API 操作的名稱。
  - databaseName 會包含受影響資料庫的名稱。
  - changedTables 包含每個通知高達 100 個受影響資料表的名稱。資料表名稱太長時，可能會建立多個通知。
- `"detail-type":"Glue Data Catalog Table State Change"` 的事件是針對 UpdateTable、CreatePartition、BatchCreatePartition、UpdatePartition、DeleteParti 和 BatchDeletePartition 產生。例如，更新資料表或分割區時，便會將通知傳送到 CloudWatch Events。請注意，您無法撰寫會根據通知事件的順序或存在的程式，因為這些事件可能會被移出序列或遺漏。盡可能發出事件。在通知詳細資訊中：
  - typeOfChange 會包含 API 操作的名稱。

- databaseName 包含內含受影響資源之資料庫的名稱。
- tableName 會包含受影響資料表的名稱。
- changedPartitions 會在一個通知中指定高達 100 個受影響的分割區。 分割區名稱太長時，可能會建立多個通知。

  例如，如果有兩個分割區金鑰 (Year 和 Month)，則 "2018,01", "2018,02" 會修改 "Year=2018" and "Month=01" 的分割區和 "Year=2018" and "Month=02" 的分割區。

```
{
    "version":"0",
    "id":"abcdef00-1234-5678-9abc-def012345678",
    "detail-type":"Glue Data Catalog Table State Change",
    "source":"aws.glue",
    "account":"123456789012",
    "time":"2017-09-07T18:57:21Z",
    "region":"us-west-2",
    "resources":["arn:aws:glue:us-west-2:123456789012:database/default/foo"],
    "detail":{
        "changedPartitions": [
            "2018,01",
            "2018,02"
        ],
        "databaseName": "default",
        "tableName": "foo",
        "typeOfChange": "BatchCreatePartition"
        }
}
```

如需詳細資訊，請參閱 Amazon CloudWatch Events 使用者指南。如需 AWS Glue 特有的事件，請參閱 AWS Glue 事件。

AWS Glue 資源監控

AWS Glue 設有服務限制，可保護客戶免於意外過度佈建，以及避免意圖增加帳單的惡意動作。這些限制也會保護服務。客戶可以登入 AWS Service Quota 主控台，檢視其目前的資源限制，並要求增加 (在適當情況下)。

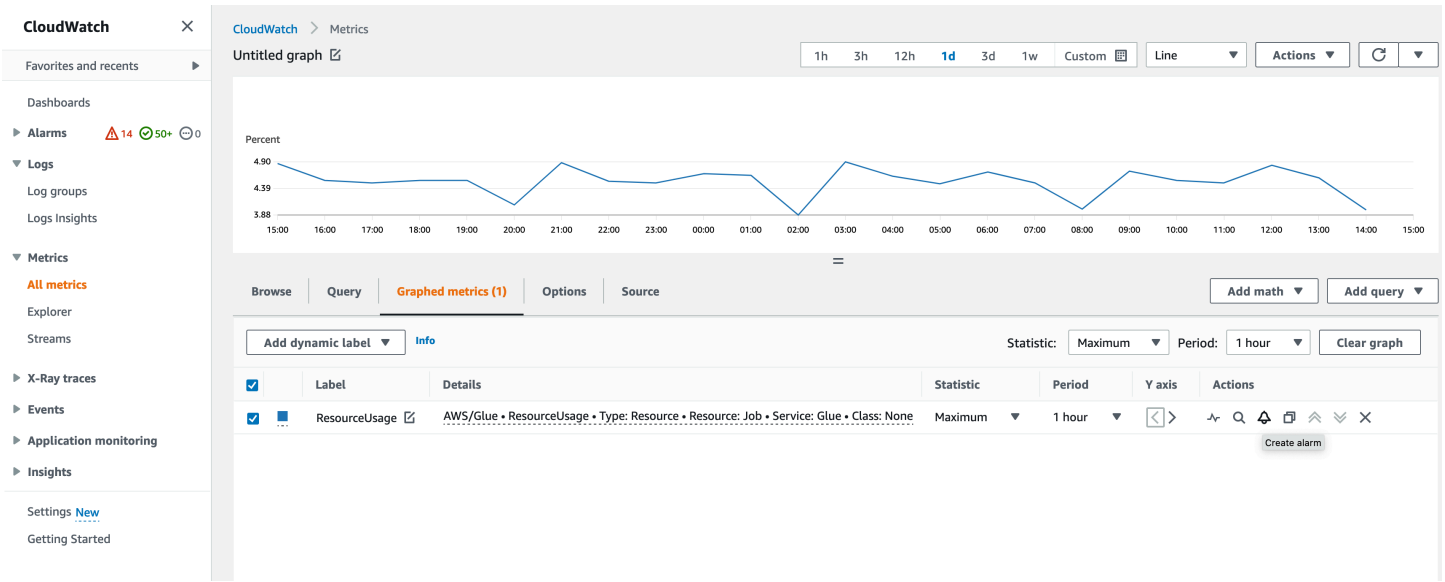
AWS Glue 可讓您在 Amazon CloudWatch 中以百分比形式檢視服務的資源使用量，並在其上設定 CloudWatch 警示以監控使用量。Amazon CloudWatch 可為 AWS 資源和在 Amazon 基礎設施上執行的客戶應用程式提供監控功能。這些指標免費提供給您。支援下列指標：

- 每個帳戶的工作流程數量
- 每個帳戶的觸發程式數量
- 每個帳戶的任務數量
- 每個帳戶的並行執行任務數量
- 每個帳戶的藍圖數量
- 每個帳戶的互動式工作階段數量

設定和使用資源指標

若要使用此功能，您可以前往 Amazon CloudWatch 主控台檢視指標並設定警示。這些指標位於 AWS/Glue 命名空間下，且是實際資源使用量計數除以資源配額所得的百分比。CloudWatch 指標會傳送至您的帳戶，您無需支付任何費用。例如，如果您建立了 10 個工作流程，而您的服務配額允許您最多擁有 200 個工作流程，則使用量為 10/200 = 5%，而在圖表中，您將會看到百分比為 5 的資料點。更具體地說：

```
Namespace: AWS/Glue
Metric name: ResourceUsage
Type: Resource
Resource: Workflow (or Trigger, Job, JobRun, Blueprint, InteractiveSession)
Service: Glue
Class: None
```



若要在 CloudWatch 主控台中建立指標的警示：

1. 找到指標後，請前往圖表化指標。
2. 按一下動作下的建立警示。
3. 視需要設定警示。

每當您的資源使用量發生變化 (例如增加或減少) 時，我們都會發出指標。但是，如果您的資源使用量未變動，我們會每小時發出指標，以便您擁有連續的 CloudWatch 圖表。為避免遺失資料點，我們不建議您設定少於 1 小時的時段。

您也可以使用 AWS CloudFormation 來設定警示，如以下範例所示。在此範例中，一旦工作流程資源使用量達到 80%，即會觸發警示，將訊息傳送至現有 SNS 主題，您可在此訂閱該主題以取得通知。

```
{
 "Type": "AWS::CloudWatch::Alarm",
 "Properties": {
  "AlarmName": "WorkflowUsageAlarm",
  "ActionsEnabled": true,
  "OKActions": [],
  "AlarmActions": [
   "arn:aws:sns:af-south-1:085425700061:Default_CloudWatch_Alarms_Topic"
  ],
  "InsufficientDataActions": [],
  "MetricName": "ResourceUsage",
  "Namespace": "AWS/Glue",
  "Statistic": "Maximum",
  "Dimensions": [{
    "Name": "Type",
    "Value": "Resource"
   },
   {
    "Name": "Resource",
    "Value": "Workflow"
   },
   {
    "Name": "Service",
    "Value": "Glue"
   },
   {
    "Name": "Class",
    "Value": "None"
   }
  ],
```

```
  "Period": 3600,
  "EvaluationPeriods": 1,
  "DatapointsToAlarm": 1,
  "Threshold": 80,
  "ComparisonOperator": "GreaterThanThreshold",
  "TreatMissingData": "notBreaching"
 }
}
```

使用 AWS CloudTrail 記錄 AWS Glue API 呼叫

AWS Glue 已與 AWS CloudTrail 整合，這項服務可提供由使用者、角色或 AWS Glue 中的 AWS 服務所採取之動作的記錄。CloudTrail 會將 AWS Glue 的所有 API 呼叫擷取為事件。擷取的呼叫包括從 AWS Glue 主控台進行的呼叫，以及針對 AWS Glue API 操作的程式碼呼叫。如果您建立追蹤，就可以將 CloudTrail 事件持續交付到 Amazon S3 儲存貯體，包括 AWS Glue 的事件。即使您未設定追蹤，依然可以透過 CloudTrail 主控台中的 Event history (事件歷史記錄) 檢視最新事件。您可以利用 CloudTrail 所收集的資訊來判斷向 AWS Glue 發出的請求，以及發出請求的 IP 地址、人員、時間和其他詳細資訊。

若要進一步了解 CloudTrail，請參閱《AWS CloudTrail 使用者指南》。

CloudTrail 中的 AWS Glue 資訊

當您建立帳戶時，系統即會在 AWS 帳戶中啟用 CloudTrail。當 AWS Glue 中發生活動時，該活動會記錄在 CloudTrail 事件中，其他 AWS 服務事件則記錄於 Event history (事件歷史記錄) 中。您可以檢視、搜尋和下載 AWS 帳戶的最新事件。如需詳細資訊，請參閱使用 CloudTrail 事件歷史記錄檢視事件。

如需您 AWS 帳戶中正在進行事件的記錄 (包含 AWS Glue 的事件)，請建立線索。追蹤能讓 CloudTrail 將日誌檔交付至 Amazon S3 儲存貯體。根據預設，當您在主控台建立線索時，線索會套用到所有 AWS 區域。該追蹤會記錄來自 AWS 分割區中所有區域的事件，並將日誌檔案交付到您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中所收集的事件資料。如需詳細資訊，請參閱下列內容：

- 建立 AWS 帳戶的追蹤
- CloudTrail 支援的服務和整合
- 設定 CloudTrail 的 Amazon SNS 通知
- 從多個區域接收 CloudTrail 日誌檔案，以及從多個帳戶接收 CloudTrail 日誌檔案

CloudTrail 會記錄所有 AWS Glue 動作，並記錄在 AWS Glue API 中。例如，對 CreateDatabase、CreateTable 和 CreateScript 動作發出的呼叫會在 CloudTrail 記錄檔案中產生項目。

每一筆事件或日誌項目都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是否使用根或 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暫時安全憑證。
- 該請求是否由另一項 AWS 服務提出。

如需詳細資訊，請參閱 CloudTrail userIdentity 元素。

然而，CloudTrail 不會記錄所有與呼叫相關的資訊。例如，特定敏感資訊就不會記錄，像是在連線要求中使用的 ConnectionProperties，而下列 API 傳回的回應則會記錄為 null。

```
BatchGetPartition       GetCrawlers          GetJobs          GetTable
CreateScript            GetCrawlerMetrics    GetJobRun        GetTables
GetCatalogImportStatus  GetDatabase          GetJobRuns       GetTableVersions
GetClassifier           GetDatabases         GetMapping       GetTrigger
GetClassifiers          GetDataflowGraph     GetObjects       GetTriggers
GetConnection           GetDevEndpoint       GetPartition     GetUserDefinedFunction
GetConnections          GetDevEndpoints      GetPartitions    GetUserDefinedFunctions
GetCrawler              GetJob               GetPlan
```

了解 AWS Glue 日誌檔案項目

追蹤是一種組態，可讓事件以日誌檔案的形式交付至您指定的 Amazon S3 儲存貯體。CloudTrail 日誌檔案包含一或多個日誌項目。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。CloudTrail 日誌檔並非依公有 API 呼叫的堆疊追蹤排序，因此不會以任何特定順序出現。

以下範例顯示的是展示 DeleteCrawler 動作的 CloudTrail 日誌項目。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/johndoe",
    "accountId": "123456789012",
```

```
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "johndoe"
  },
  "eventTime": "2017-10-11T22:29:49Z",
  "eventSource": "glue.amazonaws.com",
  "eventName": "DeleteCrawler",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "72.21.198.64",
  "userAgent": "aws-cli/1.11.148 Python/3.6.1 Darwin/16.7.0 botocore/1.7.6",
  "requestParameters": {
    "name": "tes-alpha"
  },
  "responseElements": null,
  "requestID": "b16f4050-aed3-11e7-b0b3-75564a46954f",
  "eventID": "e73dd117-cfd1-47d1-9e2f-d1271cad838c",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

以下範例顯示的是展示 `CreateConnection` 動作的 CloudTrail 日誌項目。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/johndoe",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "johndoe"
  },
  "eventTime": "2017-10-13T00:19:19Z",
  "eventSource": "glue.amazonaws.com",
  "eventName": "CreateConnection",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "72.21.198.66",
  "userAgent": "aws-cli/1.11.148 Python/3.6.1 Darwin/16.7.0 botocore/1.7.6",
  "requestParameters": {
    "connectionInput": {
      "name": "test-connection-alpha",
      "connectionType": "JDBC",
      "physicalConnectionRequirements": {
        "subnetId": "subnet-323232",
```

```
        "availabilityZone": "us-east-1a",
        "securityGroupIdList": [
          "sg-12121212"
        ]
      }
    }
  },
  "responseElements": null,
  "requestID": "27136ebc-afac-11e7-a7d6-ab217e5c3f19",
  "eventID": "e8b3baeb-c511-4597-880f-c16210c60a4a",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

AWS Glue 任務執行狀態

您可在 AWS Glue 擷取、轉換和載入 (ETL) 任務執行中或停止之後，檢視其狀態。您可以使用AWS Glue主控台、AWS Command Line Interface (AWS CLI) 或 AWS Glue API 中的GetJobRun動作來檢視狀態。

可能的任務執行狀態為
STARTING、RUNNING、STOPPING、STOPPED、SUCCEEDED、FAILED、ERROR、WAITING 和 TIMEOUT。

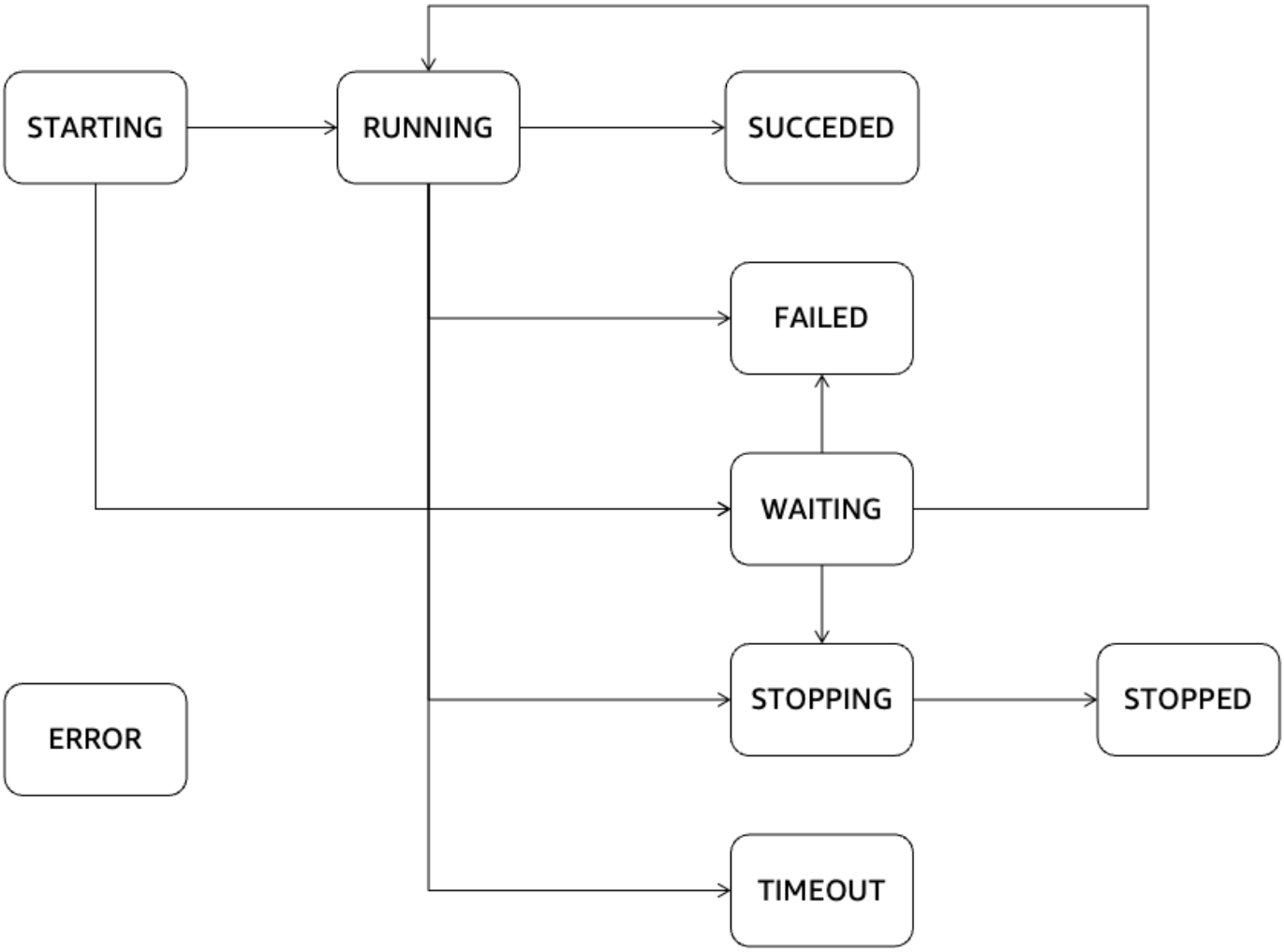
下表列出指出異常任務終止的狀態。

任務執行狀態	描述
FAILED	任務超過其允許並行執行的上限，或以未知的結束代碼終止。
ERROR	嘗試執行已刪除任務的工作流程、排程觸發程序或事件觸發程序。
TIMEOUT	任務執行時間超過其指定的逾時值。

WAITING 狀態表示任務執行正在等待資源。下表說明不同任務類別的等待行為。

工作類型	Behavior (行為)
Spark 任務 (標準)	尚未根據 `maxRetries` 組態設定重試的任務，可能會進入 WAITING 狀態。如果服務無法取得足夠的資源來開始執行，則新的任務執行將處於 WAITING 狀態。此狀態發生的可能原因，在於帳戶的服務配額或區域的容量限制發生下列其中一種錯誤案例： • 超過每個帳戶的並行任務執行上限 • 超過每個任務的並行任務執行上限 (包括帳戶層級服務配額，以及您針對任務使用 `MaxConcurrentRuns` 所指定的限制) • 超過並行運算 (DPU 用量) 上限 • 資源無法使用 如需有關 AWS Glue 服務配額的詳細資訊，請參閱 AWS Glue 端點和配額。AWS Glue 等待資源的時間可能會根據情況而有所不同。當任務嘗試取得資源時，可能會在非終端狀態之間轉換。最終，如果任務無法取得資源，將會轉換為 `FAILED` 狀態。AWSGlue 的重試上限為 15 分鐘或 10 次嘗試 (以先發生者為準)。
Spark 任務 (彈性)	如果服務無法取得足夠的資源來開始執行，則新的任務執行將處於 WAITING 狀態，這會執行延遲開始。執行將處於 WAITING 狀態，最多 20 分鐘 (逾時由服務控制)。15 分鐘後，服務將嘗試強制開始，並且，根據可用容量，執行可能會開始或失敗，並顯示適合的錯誤消息。
Python shell 任務	與使用 Spark 的標準任務相同的行為。

下列狀態圖概述了 AWS Glue 工作生命週期中預期的狀態轉換。此資訊適用於所有工作類型。


STARTING
RUNNING
SUCCEDED
FAILED
WAITING
STOPPING
STOPPED
ERROR
TIMEOUT

AWS Glue 流媒體

AWS Glue 串流是其中的一個元件 AWS Glue，可讓您以近乎即時的速度有效率地處理串流資料，從而使您能夠執行資料擷取、處理和機器學習等關鍵任務。透過 Apache Spark 串流架構， AWS Glue 串流提供可大規模處理串流資料的無伺服器服務。 AWS Glue 在 Apache Spark 之上提供各種最佳化功能，例如無伺服器基礎架構、auto-scaling、視覺化工作開發、串流作業的即時啟動筆記型電腦，以及其他效能改進。

串流使用案例

AWS Glue 串流的一些常見使用案例包括：

N 資ear-real-time 料處理： AWS Glue 串流可讓組織以近乎即時的方式處理串流資料，讓他們得以獲得深入解析，並根據最新資訊做出及時決策。

詐騙偵測：您可以利用 AWS Glue 串流進行即時分析串流資料，使其對偵測詐騙活動非常重要，例如信用卡詐騙、網路入侵或線上詐騙。持續處理和分析傳入的資料，可讓您快速找出可疑的模式或異常情況。

社交媒體分析： AWS Glue 流媒體可以處理實時社交媒體數據，例如推文，帖子或評論，使組織能夠實時監控趨勢，情感分析並管理品牌聲譽。

物聯網 (IoT) 分析： AWS Glue 串流適用於處理和分析 IoT 裝置、感應器和連網機械所產生的高速資料串流。可進行即時監控、異常偵測、預測性維護和其他 IoT 分析使用案例。

點擊流分析： AWS Glue 流媒體可以處理和分析來自網站或移動應用程序的實時點擊流數據。這可協助企業深入了解使用者行為、打造個人化使用者體驗，並根據即時點擊流資料將行銷活動最佳化。

日誌監控和分析： AWS Glue 串流可以即時持續處理和分析來自伺服器、應用程式或網路裝置的日誌資料。這有助於偵測異常、疑難排解問題，以及監控系統運作狀態和效能。

推薦系統： AWS Glue 串流可以即時處理使用者活動資料，並動態更新推薦模型。這可讓系統根據使用者的行為和偏好即時提供個人化的建議。

這些是可以應用 AWS Glue 流式傳輸的各種用例的一些示例。它與 AWS 生態系統和託管服務的整合，使其成為雲端中即時串流處理和分析的便利選擇。

使用 AWS Glue 串流有什麼好處？

使用 AWS Glue 串流的好處如下：

- 無伺服器： AWS Glue 串流是無伺服器的，無需管理基礎架構。此設計可減少營運成本，讓使用者專注於資料處理和分析工作，無須分神管理基礎設施。
- 自動調度資源： AWS Glue 串流提供自動調度資源功能，可根據工作負載動態調整處理容量。此功能會自動擴展或縮減以應付資料量的波動，確保最佳效能和資源使用率。
- 視覺化開發：串流工作開發可能很複雜。 AWS Glue 串流可提供視覺化製作工具 AWS Glue Studio 來解決這項挑戰。 AWS Glue Studio 可簡化串流工作流程的建立程序，並讓開發人員以視覺化的方式設計和管理串流應用程式，減少學習曲線並提高生產力。
- 符合成本效益：作為一項無伺服器服務， AWS Glue Streaming 可免除佈建和維護基礎架構的需求，進而提高成本效益。系統會根據執行串流任務期間所耗用的資源向使用者計費，以便根據實際使用情況進行成本最佳化和擴展。
- 處理複雜的工作負載： AWS Glue 串流專為處理複雜的串流工作負載而設 它可以處理和分析大量即時資料、支援進階轉換，並與其他 AWS 服務整合，進而實現複雜的串流資料管道和分析工作流程。
- 不受限制： AWS Glue 流媒體提供了靈活性並避免了供應商鎖定。使用者可以利用 AWS Glue 串流作為更廣泛 AWS 生態系統的一部分，將其與其他 AWS 服務無縫整合。這樣可以輕鬆地與現有的資料來源、應用程式和服務整合，而無須與特定技術或平台綁定在一起。

何時使用 AWS Glue 串流？

說到串流使用案例，您可以有很多選擇。我們建議在下列情況下進行 AWS Glue 串流。

1. 如果您已經在使用 AWS Glue 或 Spark 進行批量處理，則 AWS Glue 流式傳輸是您的理想選擇。它可讓您順利轉換至建置串流任務，而無須學習新的語言或框架。 AWS Glue Streaming 運用您現有的知識和基礎架構，可簡化工作開發程序，並讓您輕鬆將資料處理能力擴充至即時串流案例。
2. 如果您需要統一服務或產品來處理批次、串流和事件驅動的工作負載， AWS Glue 串流是您的最佳解決方案。透過 AWS Glue 串流，您可以將資料處理需求整合到單一架構中，免除管理多個系統的複雜性。這樣可讓您有效地開發和維護各種資料工作流程，同時確保不同工作負載類型的一致性和相容性。
3. AWS Glue 串流非常適合涉及極大型串流資料量和複雜轉換的案例，例如串流或關聯式資料庫之間的聯結。它可以有效地處理和分析大量資料串流，讓您能夠輕鬆處理高需求的工作負載。無論是高速資料擷取還是複雜的資料操作， AWS Glue Streaming 的可擴充性和先進的處理能力都能確保最佳效能和準確的結果。
4. 如果您希望使用視覺化方法來建立串流工作，請 AWS Glue 提供 AWS Glue Studio，您可以使用它以視覺化方式設計和管理串流應用程式，從而簡化開發程序。此工具的直覺式介面可讓開發人員使用視覺化介面來建立、設定和監控串流工作流程，進而減少學習曲線並提高生產力。

5. AWS Glue 對於嚴格的 SLA (服務等級協定) 超過 10 秒的 near-real-time 使用案例，串流是絕佳的選擇。
6. 如果您要使用 Apache 冰山、Apache Hudi 或三角洲湖來建置交易式資料湖， AWS Glue 串流會為這些開放式資料表格式提供原生支援。這種無縫整合可讓您直接從這些交易資料湖處理串流資料，以確保資料的一致性、完整性和相容性。
7. 需要擷取各種資料目標的串流資料時： AWS Glue 串流可為各種資料目標 (例如 Amazon Redshift、亞馬遜 RDS、亞馬 Amazon Aurora、甲骨文、SQL 伺服器和其他目標) 提供原生目標。

支援的資料來源

AWS Glue 串流支援下列資料來源：

- Amazon Kinesis
- Amazon MSK (Managed Streaming for Apache Kafka)
- 自我管理的 Apache Kafka

支援的資料目標

AWS Glue 串流支援多種資料目標，例如：

- 資料目錄支援的 AWS Glue 資料目標
- Amazon S3
- Amazon Redshift
- MySQL
- PostgreSQL
- Oracle
- Microsoft SQL Server
- Snowflake
- 任何可使用 JDBC 連接的資料庫
- Apache Iceberg、Delta 和 Apache Hudi
- AWS Glue Marketplace 連接器

教學課程：使用 AWS Glue Studio 建置您的第一個串流工作負載

在本教學課程中，您將會學到如何使用 AWS Glue Studio 建立串流任務。AWS GlueStudio 是建立 AWS Glue 任務的視覺化界面。

您可以建立串流擷取、轉換和載入 (ETL) 任務，讓它連續執行，並使用來自 Amazon Kinesis Data Streams、Apache Kafka 和 Amazon Managed Streaming for Apache Kafka (Amazon MSK) 等串流來源的資料。

必要條件

若要遵循本教學課程，您需要一位具有 AWS 主控台權限的使用者才能使用AWS Glue、Amazon Kinesis、Amazon S3、Amazon Athena、AWS CloudFormation、AWS Lambda 和 Amazon Cognito。

使用 Amazon Kinesis 的串流資料

主題

- 使用 Kinesis 資料產生器產生模擬資料
- 使用 AWS Glue Studio 建立 AWS Glue 串流任務
- 執行轉換並將轉換的結果儲存在 Amazon S3

使用 Kinesis 資料產生器產生模擬資料

您可以使用 Kinesis 資料產生器 (KDG) 以 JSON 格式合成產生範例資料。您可以在工具文件中找到完整的說明和詳細資訊。

1. 若要開始使用，請按一下

   以在您的 AWS 環境中執行 AWS CloudFormation 範本。

   > **Note**
   >
   > 因為您的 AWS 帳戶中已存在某些資源 (例如 Kinesis 資料產生器的 Amazon Cognito 使用者)，因此您可能會遇到 CloudFormation 範本失敗。這可能是因為您已經在其他教學課程或部落格中進行過設定。若要解決這個問題，您可以嘗試在新的 AWS 帳戶使用範本以重新

來過，或探索不同的 AWS 區域。這些選項可讓您執行教學課程，而不會與現有資源發生衝突。

此範本會為您佈建 Kinesis 資料串流和 Kinesis 資料產生器帳戶。也會建立 Amazon S3 儲存貯體來保存資料，並建立具有本教學課程所需權限的 Glue 服務角色。

2. 輸入 KDG 將用於驗證的使用者名稱和密碼。記下使用者名稱和密碼以供進一步使用。
3. 選取下一步一直到最後一步。確認 IAM 資源的建立。檢查畫面頂端是否顯示任何錯誤，例如密碼不符合最低需求，然後部署範本。
4. 導覽至堆疊的輸出索引標籤。完成部署範本後，範本會顯示產生的屬性 KinesisDataGeneratorUrl。按一下該網址。
5. 輸入您記下的使用者名稱和密碼。
6. 選取您所使用的區域，然後選取「Kinesis 串流」`GlueStreamTest-{AWS::AccountId}`
7. 輸入下列範本：

```
{
    "ventilatorid": {{random.number(100)}},
    "eventtime": "{{date.now("YYYY-MM-DD HH:mm:ss")}}",
    "serialnumber": "{{random.uuid}}",
    "pressurecontrol": {{random.number(
        {
            "min":5,
            "max":30
        }
    )}},
    "o2stats": {{random.number(
        {
            "min":92,
            "max":98
        }
    )}},
    "minutevolume": {{random.number(
        {
            "min":5,
            "max":8
        }
    )}},
    "manufacturer": "{{random.arrayElement(
        ["3M", "GE","Vyaire", "Getinge"]
```

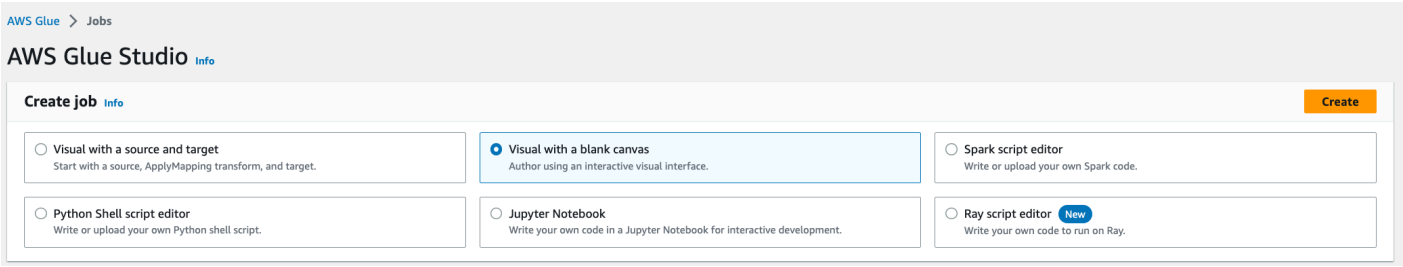
```
    )}}"
}
```

現在，您可以使用測試範本查看模擬資料，並使用傳送資料將模擬資料擷取至 Kinesis。

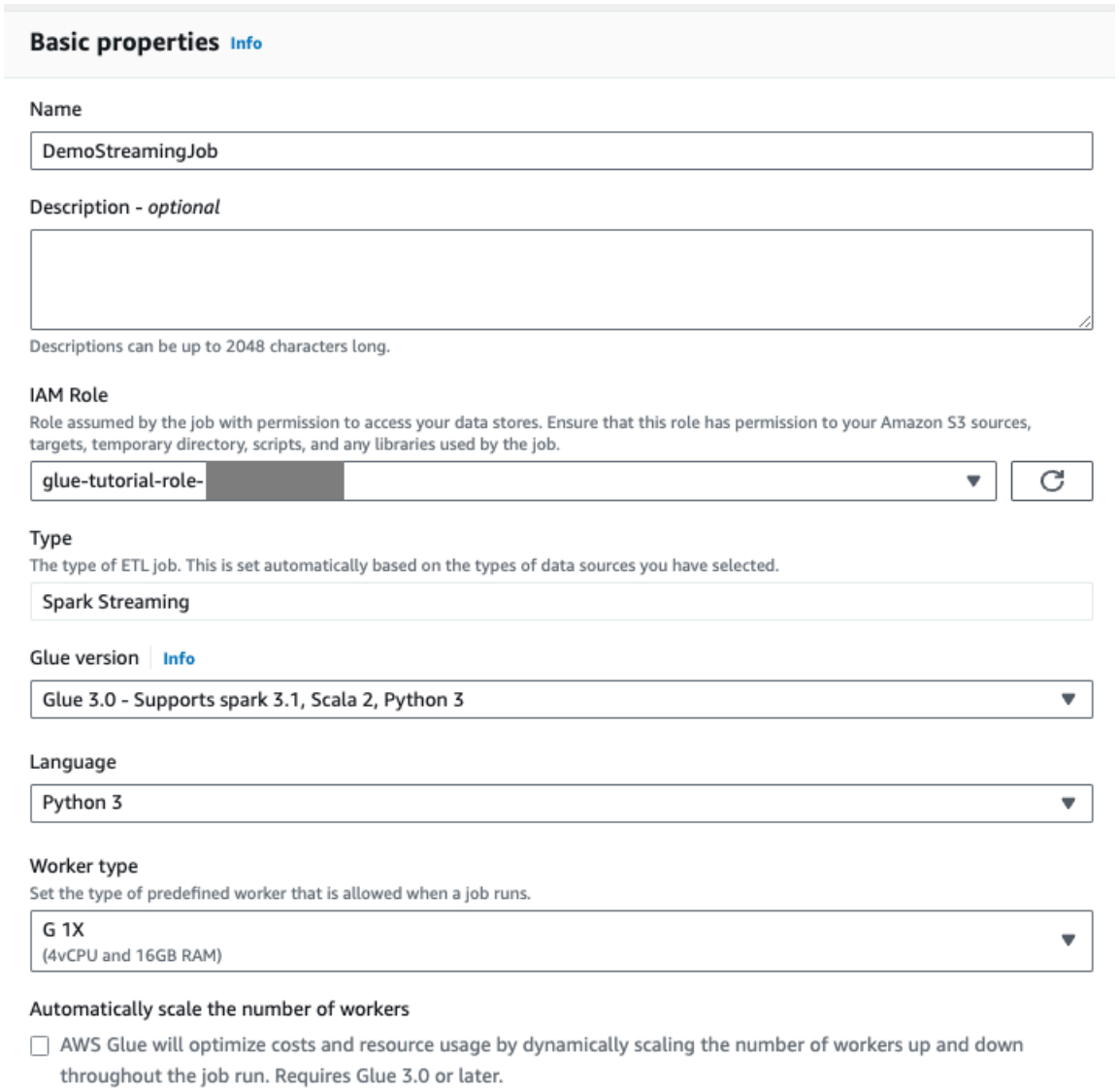
8. 按一下傳送資料，產生 5-10K 的記錄給 Kinesis。

使用 AWS Glue Studio 建立 AWS Glue 串流任務

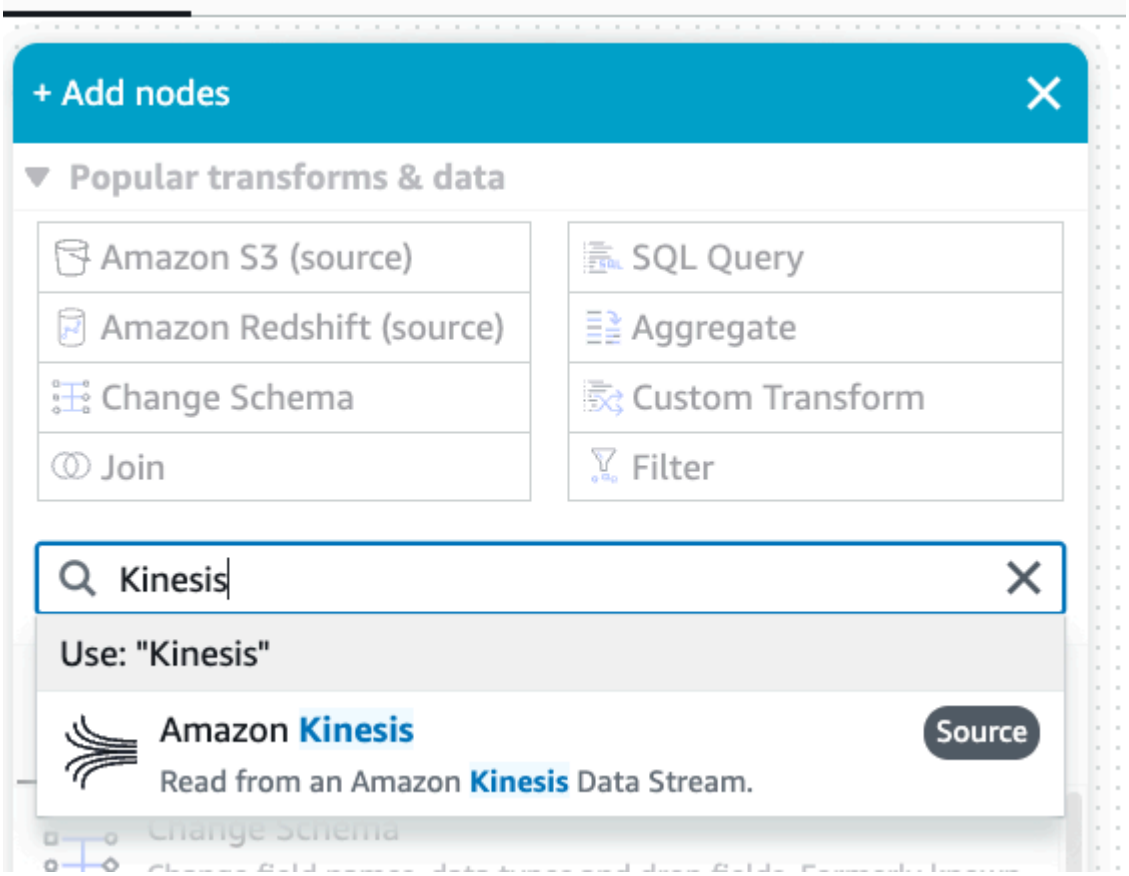
1. 在控制台導覽至同一地區中的 AWS Glue。
2. 在資料整合和 ETL 下的左側導覽列下，選取 ETL 任務。
3. 使用空白畫布透過視覺化建立 AWS Glue 任務。



4. 導覽至任務詳細資訊索引標籤。
5. 對於 AWS Glue 任務名稱，請輸入 `DemoStreamingJob`。
6. 對於 IAM 角色，請選取由 CloudFormation 範本佈建的角色 `glue-tutorial-role-${AWS::AccountId}`。
7. 對於Glue 版本，請選取 Glue 3.0。將其他所有選項保留為預設值。



8. 導覽至視覺化索引標籤。
9. 按一下加號圖示。在搜尋列中，輸入 Kinesis。選取 Amazon Kinesis 資料來源。



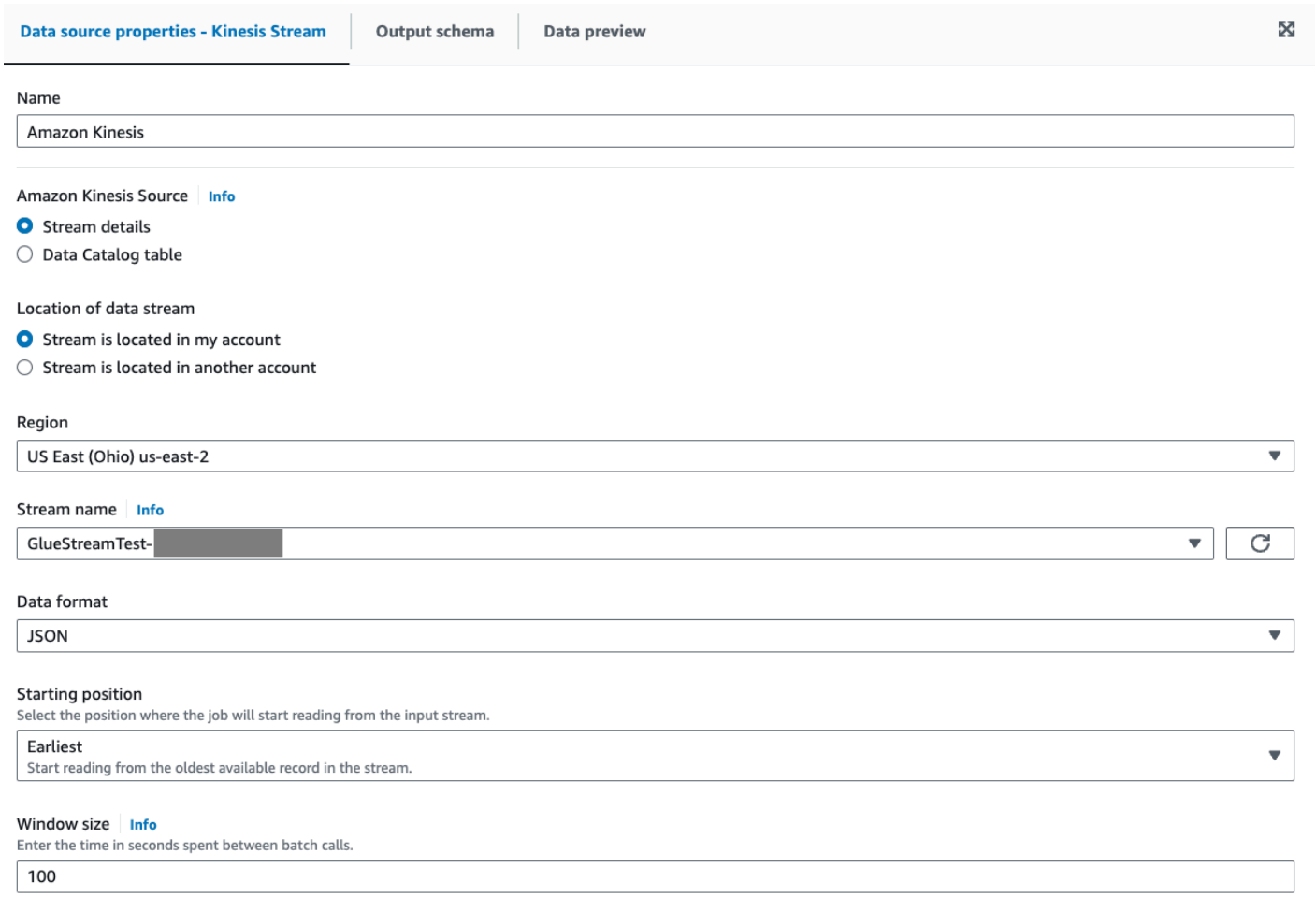
10.在資料來源屬性 - Kinesis 串流索引標籤下，為 Amazon Kinesis 來源選取串流詳細資料。

11.為資料串流位置選取串流位於我的帳戶。

12.選擇您正在使用的區域。

13.選取 `GlueStreamTest-{AWS::AccountId}` 串流。

14.將其他所有設定保留為預設值。

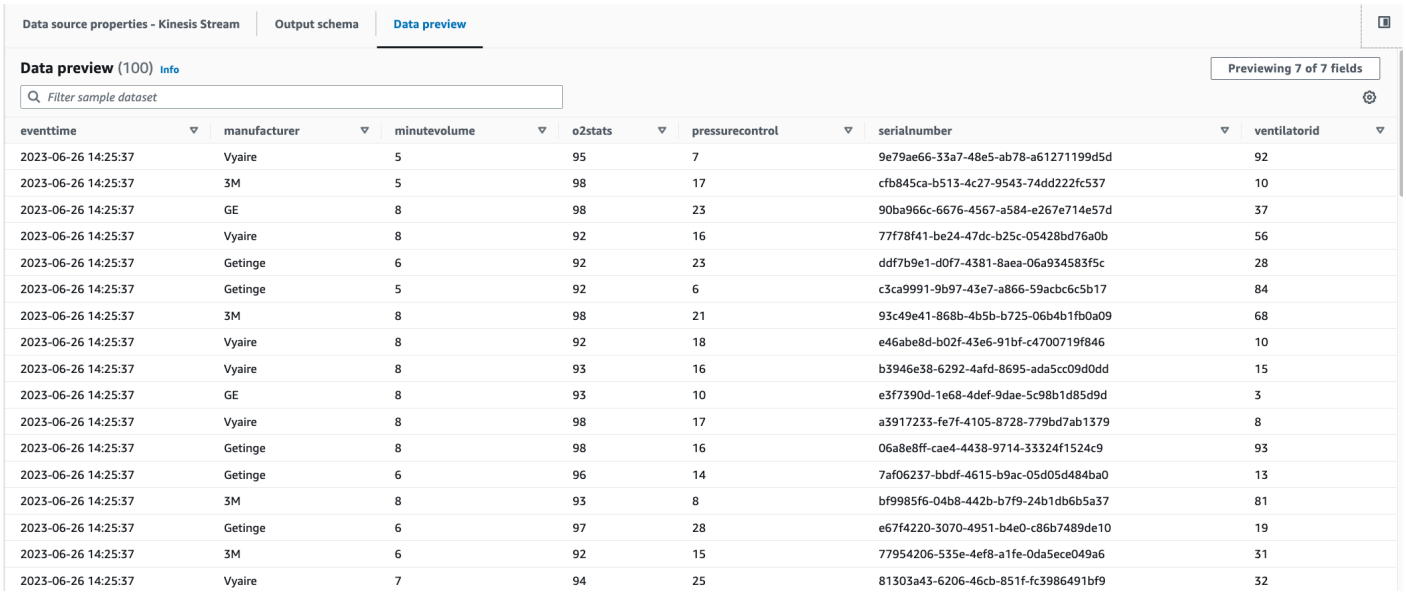


15導覽至資料預覽索引標籤。

16按一下開始資料預覽工作階段，即可預覽 KDG 產生的模擬資料。選擇您先前為 AWS Glue 串流任務建立的「Glue 服務角色」。

預覽資料需要 30-60 秒才會顯示。如果顯示沒有可顯示的資料，請按一下齒輪圖示，然後將要取樣的列數變更為 100。

您可以看到如下所示的範例資料：



Data source properties - Kinesis Stream | Output schema | Data preview

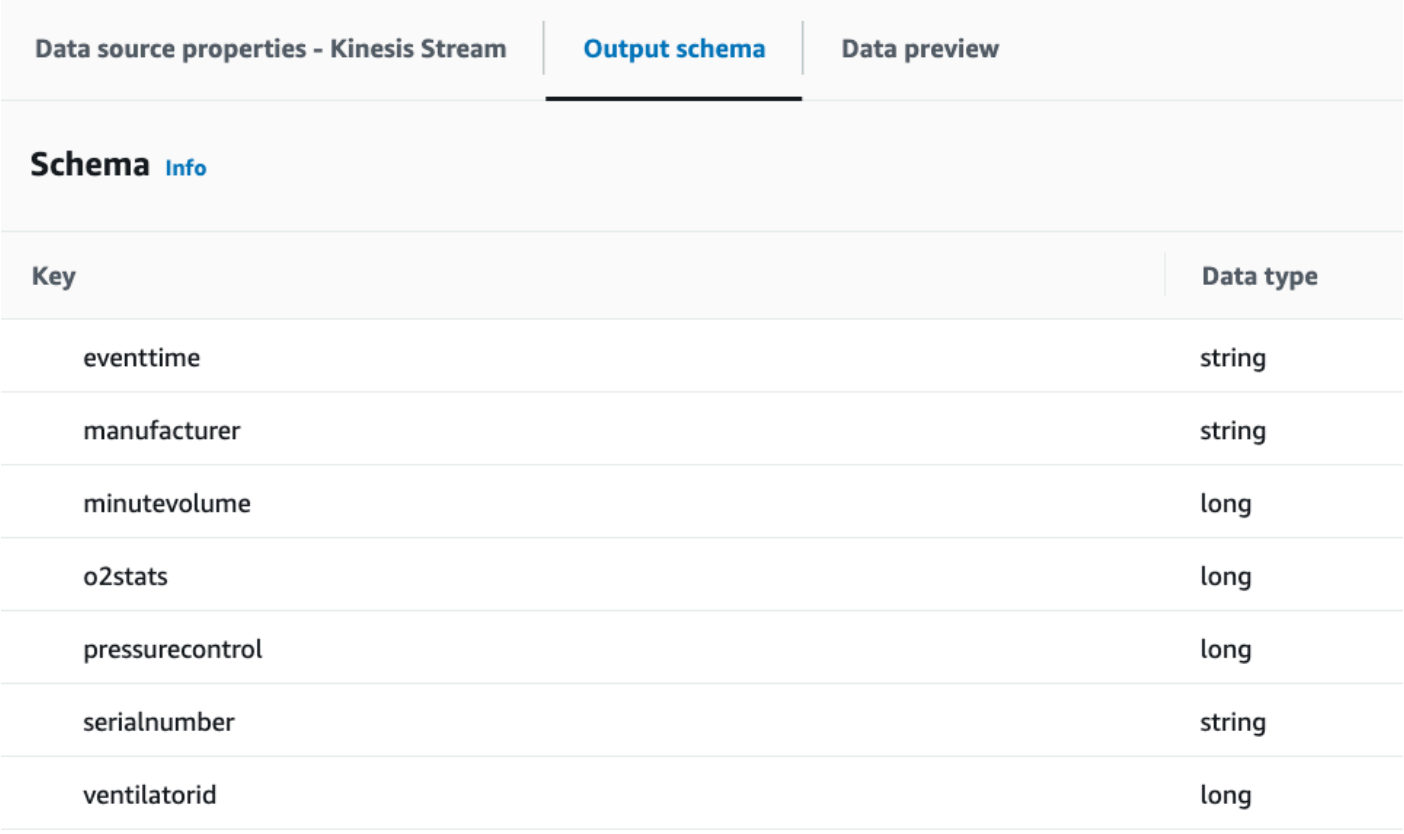
Data preview (100) Info

Previewing 7 of 7 fields

Filter sample dataset

eventtime	manufacturer	minutevolume	o2stats	pressurecontrol	serialnumber	ventilatorid
2023-06-26 14:25:37	Vyaire	5	95	7	9e79ae66-33a7-48e5-ab78-a61271199d5d	92
2023-06-26 14:25:37	3M	5	98	17	cfb845ca-b513-4c27-9543-74dd222fc537	10
2023-06-26 14:25:37	GE	8	98	23	90ba966c-6676-4567-a584-e267e714e57d	37
2023-06-26 14:25:37	Vyaire	8	92	16	77f78f41-be24-47dc-b25c-05428bd76a0b	56
2023-06-26 14:25:37	Getinge	6	92	23	ddf7b9e1-d0f7-4381-8aea-06a934583f5c	28
2023-06-26 14:25:37	Getinge	5	92	6	c3ca9991-9b97-43e7-a866-59acbc6c5b17	84
2023-06-26 14:25:37	3M	8	98	21	93c49e41-868b-4b5b-b725-06b4b1fb0a09	68
2023-06-26 14:25:37	Vyaire	8	92	18	e46abe8d-b02f-43e6-91bf-c4700719f846	10
2023-06-26 14:25:37	Vyaire	8	93	16	b3946e38-6292-4afd-8695-ada5cc09d0dd	15
2023-06-26 14:25:37	GE	8	93	10	e3f7390d-1e68-4def-9dae-5c98b1d85d9d	3
2023-06-26 14:25:37	Vyaire	8	98	17	a3917233-fe7f-4105-8728-779bd7ab1379	8
2023-06-26 14:25:37	Getinge	8	98	16	06a8e8ff-cae4-4438-9714-33324f1524c9	93
2023-06-26 14:25:37	Getinge	6	96	14	7af06237-bbdf-4615-b9ac-05d05d484ba0	13
2023-06-26 14:25:37	3M	8	93	8	bf9985f6-04b8-442b-b7f9-24b1db6b5a37	81
2023-06-26 14:25:37	Getinge	6	97	28	e67f4220-3070-4951-b4e0-c86b7489de10	19
2023-06-26 14:25:37	3M	6	92	15	77954206-535e-4ef8-a1fe-0da5ece049a6	31
2023-06-26 14:25:37	Vyaire	7	94	25	81303a43-6206-46cb-851f-fc3986491bf9	32

您也可以在輸出結構描述索引標籤查看推斷的結構描述。



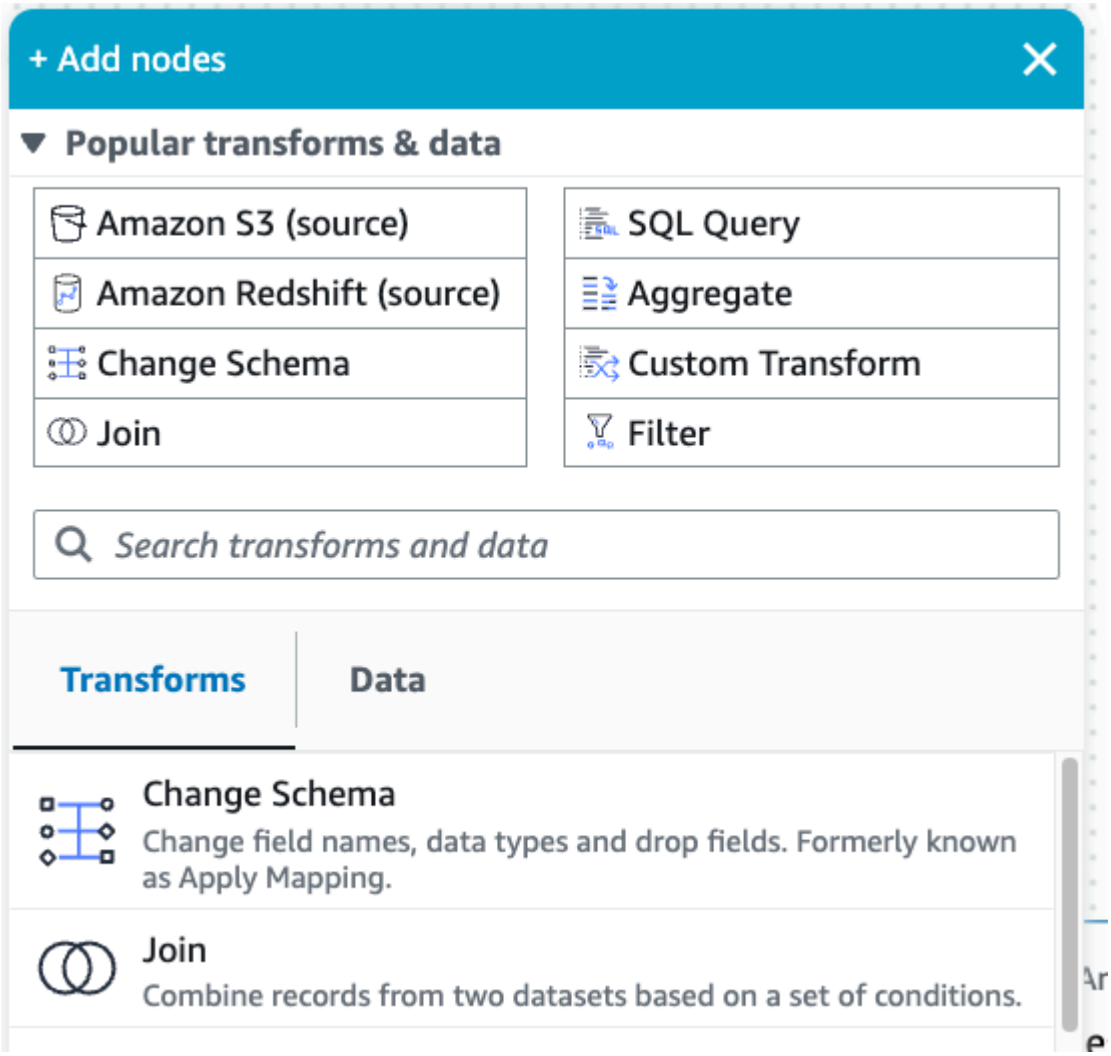
Data source properties - Kinesis Stream | Output schema | Data preview

Schema Info

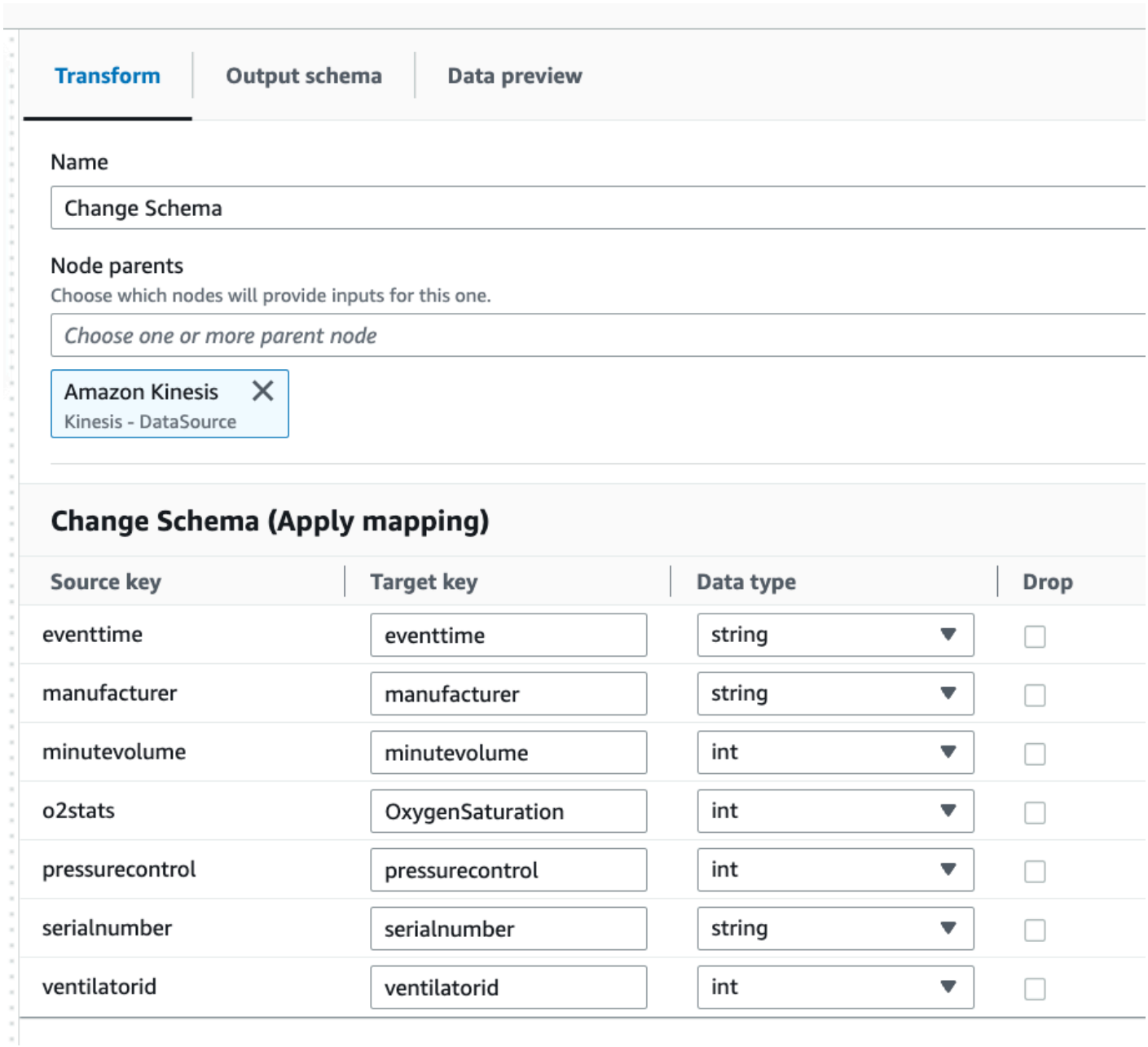
Key	Data type
eventtime	string
manufacturer	string
minutevolume	long
o2stats	long
pressurecontrol	long
serialnumber	string
ventilatorid	long

執行轉換並將轉換的結果儲存在 Amazon S3

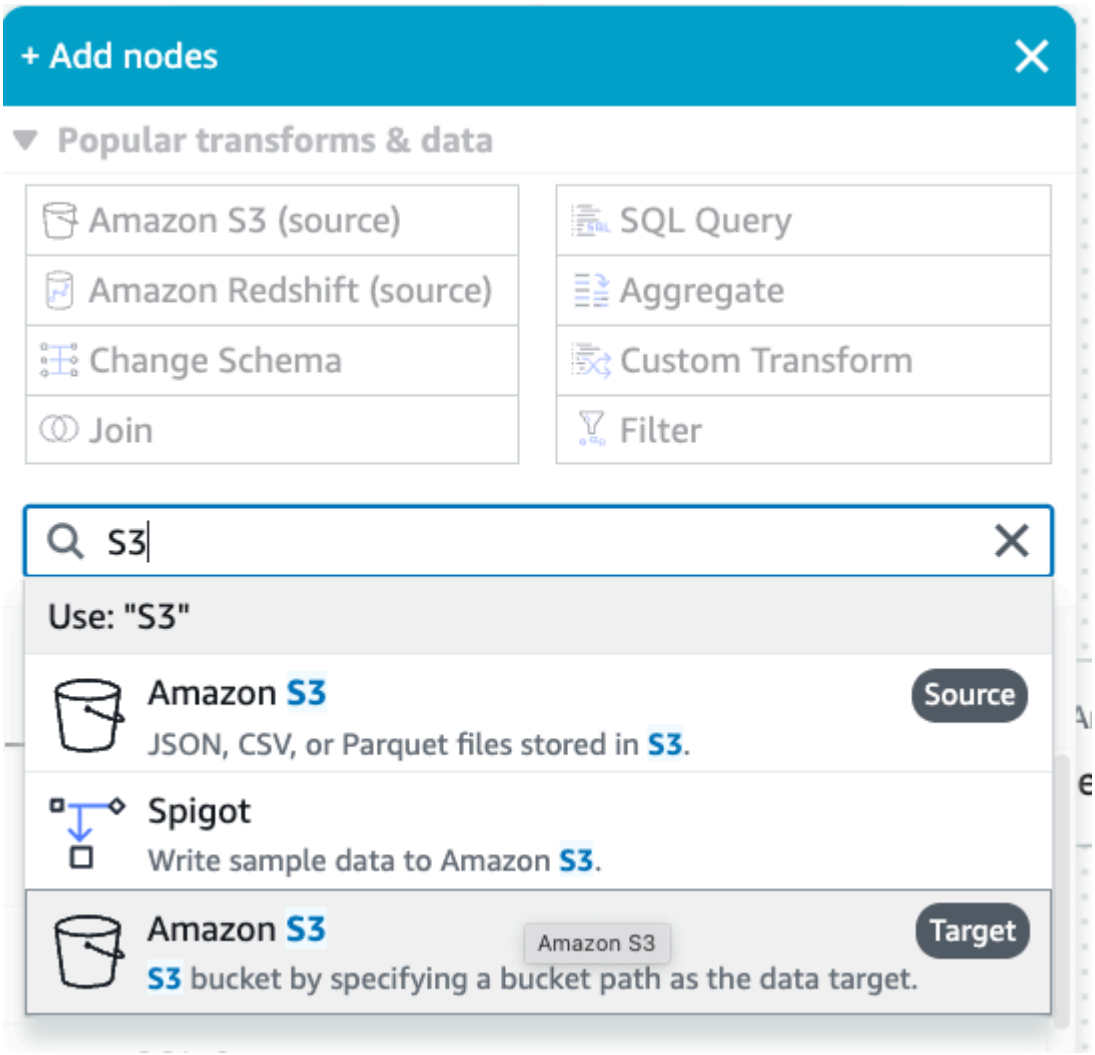
1. 選取來源節點後，按一下左上角的加號圖示以新增轉換步驟。
2. 選取變更結構描述步驟。



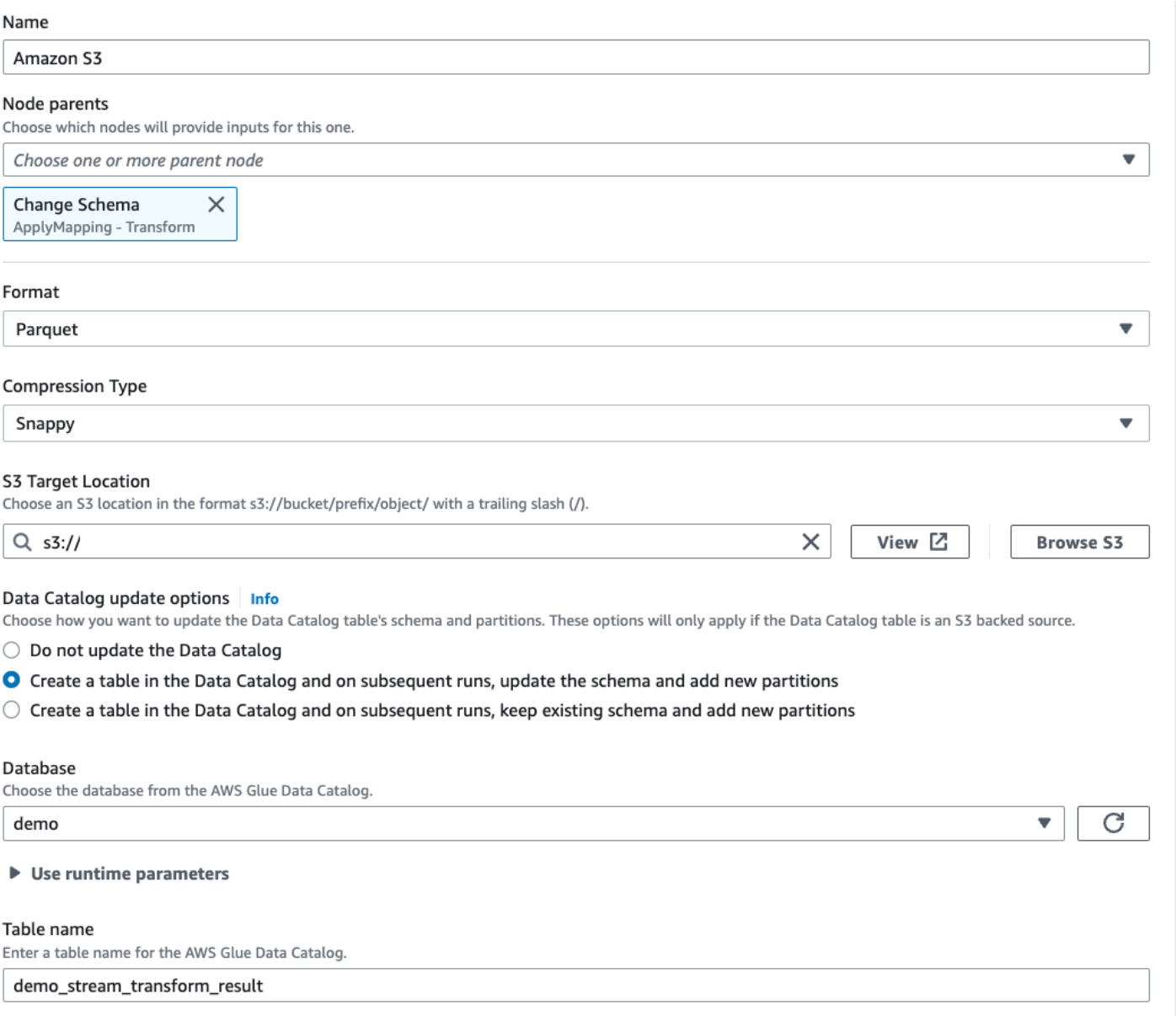
3. 您可以在此步驟重新命名欄位，並轉換欄位的資料類型。將 `o2stats` 欄位重新命名為 `OxygenSaturation`，並將所有 `long` 資料類型轉換為 `int`。



4. 按一下加號圖示以新增 Amazon S3 目標。在搜尋方塊中輸入 S3，然後選取 Amazon S3 - 目標轉換步驟。



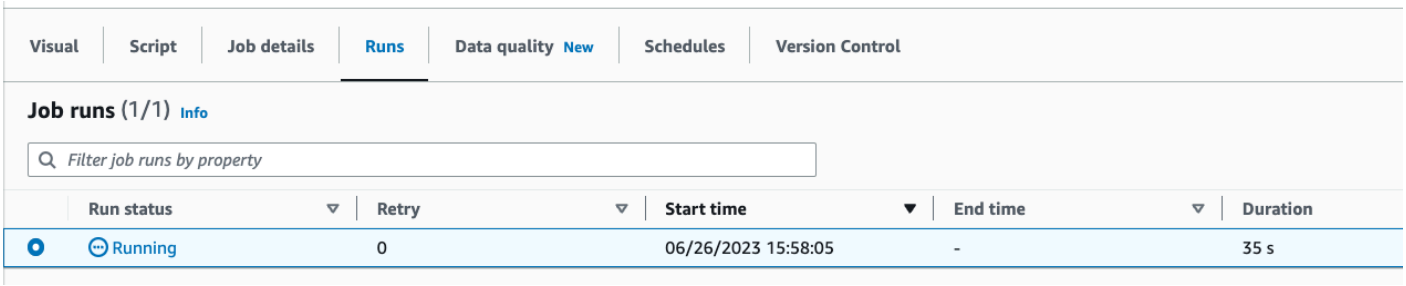
5. 選取 Parquet 作為目標檔案格式。
6. 選取 Snappy 作為壓縮類型。
7. 輸入 CloudFormation 範本 `streaming-tutorial-s3-target-{AWS::AccountId}` 建立的 S3 目標位置。
8. 選取以便在資料目錄建立資料表，並在後續執行時，更新結構描述和新增分割區。
9. 輸入目標資料庫和資料表名稱，以儲存 Amazon S3 目標資料表的結構描述。



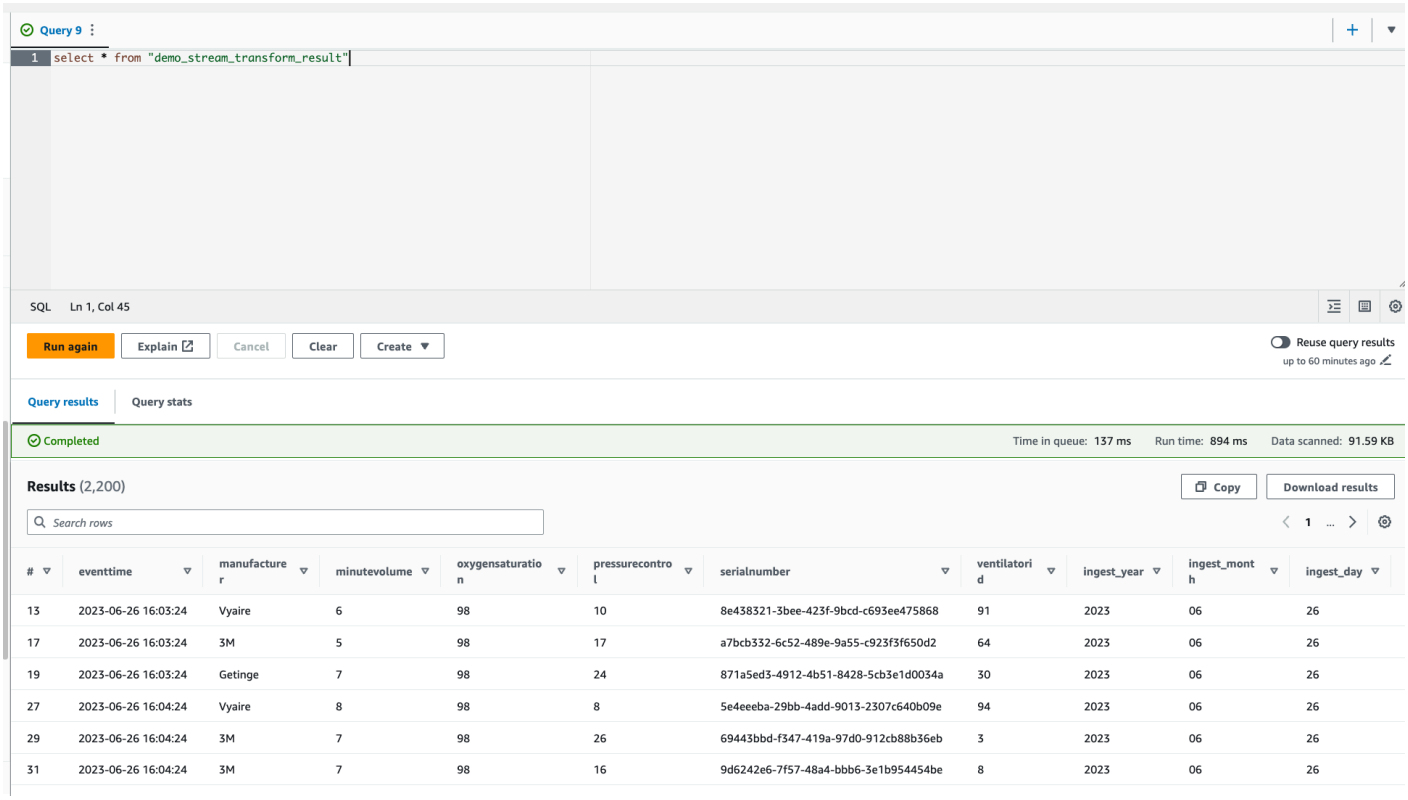
10按一下指令碼索引標籤以查看產生的程式碼。

11按一下右上角的儲存以儲存 ETL 程式碼，然後按一下執行以啟動 AWS Glue 串流任務。

您可以在執行索引標籤找到執行狀態。讓任務執行 3-5 分鐘，然後停止任務。



12驗證在 Amazon Athena 建立的新資料表。



#	eventtime	manufacturer	minutevolume	oxygensaturation	pressurecontrol	serialnumber	ventilatorid	ingest_year	ingest_month	ingest_day
13	2023-06-26 16:03:24	Vyaire	6	98	10	8e438321-3bee-423f-9bcd-c693ee475868	91	2023	06	26
17	2023-06-26 16:03:24	3M	5	98	17	a7bcb332-6c52-489e-9a55-c923f3f650d2	64	2023	06	26
19	2023-06-26 16:03:24	Getinge	7	98	24	871a5ed3-4912-4b51-8428-5cb3e1d0034a	30	2023	06	26
27	2023-06-26 16:04:24	Vyaire	8	98	8	5e4eeeba-29bb-4add-9013-2307c640b09e	94	2023	06	26
29	2023-06-26 16:04:24	3M	7	98	26	69443bbd-f347-419a-97d0-912cb88b36eb	3	2023	06	26
31	2023-06-26 16:04:24	3M	7	98	16	9d6242e6-7f57-48a4-bbb6-3e1b954454be	8	2023	06	26

教學課程：使用 AWS Glue Studio 筆記本建置您的第一個串流工作負載

在本教學課程中，我們將帶您探索如何運用 AWS Glue Studio 筆記本以互動方式建置和優化 ETL 任務，以進行近乎即時的資料處理。無論您是使用 AWS Glue 的新手或是想增強技能，本指南都將引導您完成整個程序，讓您能夠充分利用 AWS Glue 互動式工作階段筆記本的所有潛力。

透過 AWS Glue 串流，您可以建立串流擷取、轉換和載入 (ETL) 任務，讓它連續執行，並使用來自 Amazon Kinesis Data Streams、Apache Kafka 和 Amazon Managed Streaming for Apache Kafka (Amazon MSK) 等串流來源的資料。

必要條件

若要遵循本教學課程，您需要一位具有 AWS 主控台權限的使用者才能使用AWS Glue、Amazon Kinesis、Amazon S3、Amazon Athena、AWS CloudFormation、AWS Lambda 和 Amazon Cognito。

使用 Amazon Kinesis 的串流資料

主題

使用 Kinesis 資料產生器產生模擬資料

> **Note**
>
> 如果您已經完成之前的 教學課程：使用 AWS Glue Studio 建置您的第一個串流工作負載，您的帳戶中便已安裝 Kinesis 資料產生器，您可以略過以下步驟 1-8，繼續前往 使用 AWS Glue Studio 建立 AWS Glue 串流任務 一節。

您可以使用 Kinesis 資料產生器 (KDG) 以 JSON 格式合成產生範例資料。您可以在工具文件中找到完整的說明和詳細資訊。

1. 若要開始使用，請按一下

   Launch Stack

   以在您的 AWS 環境中執行 AWS CloudFormation 範本。

   > **Note**
   >
   > 因為您的 AWS 帳戶中已存在某些資源 (例如 Kinesis 資料產生器的 Amazon Cognito 使用者)，因此您可能會遇到 CloudFormation 範本失敗。這可能是因為您已經在其他教學課程或部落格中進行過設定。若要解決這個問題，您可以嘗試在新的 AWS 帳戶使用範本以重新來過，或探索不同的 AWS 區域。這些選項可讓您執行教學課程，而不會與現有資源發生衝突。

   此範本會為您佈建 Kinesis 資料串流和 Kinesis 資料產生器帳戶。

2. 輸入 KDG 將用於驗證的使用者名稱和密碼。記下使用者名稱和密碼以供進一步使用。

3. 選取下一步一直到最後一步。確認 IAM 資源的建立。檢查畫面頂端是否顯示任何錯誤，例如密碼不符合最低需求，然後部署範本。
4. 導覽至堆疊的輸出索引標籤。完成部署範本後，範本會顯示產生的屬性 KinesisDataGeneratorUrl。按一下該網址。
5. 輸入您記下的使用者名稱和密碼。
6. 選取您所使用的區域，然後選取「Kinesis 串流」GlueStreamTest-{AWS::AccountId}
7. 輸入下列範本：

```
{
    "ventilatorid": {{random.number(100)}},
    "eventtime": "{{date.now("YYYY-MM-DD HH:mm:ss")}}",
    "serialnumber": "{{random.uuid}}",
    "pressurecontrol": {{random.number(
        {
            "min":5,
            "max":30
        }
    )}},
    "o2stats": {{random.number(
        {
            "min":92,
            "max":98
        }
    )}},
    "minutevolume": {{random.number(
        {
            "min":5,
            "max":8
        }
    )}},
    "manufacturer": "{{random.arrayElement(
        ["3M", "GE","Vyaire", "Getinge"]
    )}}"
}
```

現在，您可以使用測試範本查看模擬資料，並使用傳送資料將模擬資料擷取至 Kinesis。

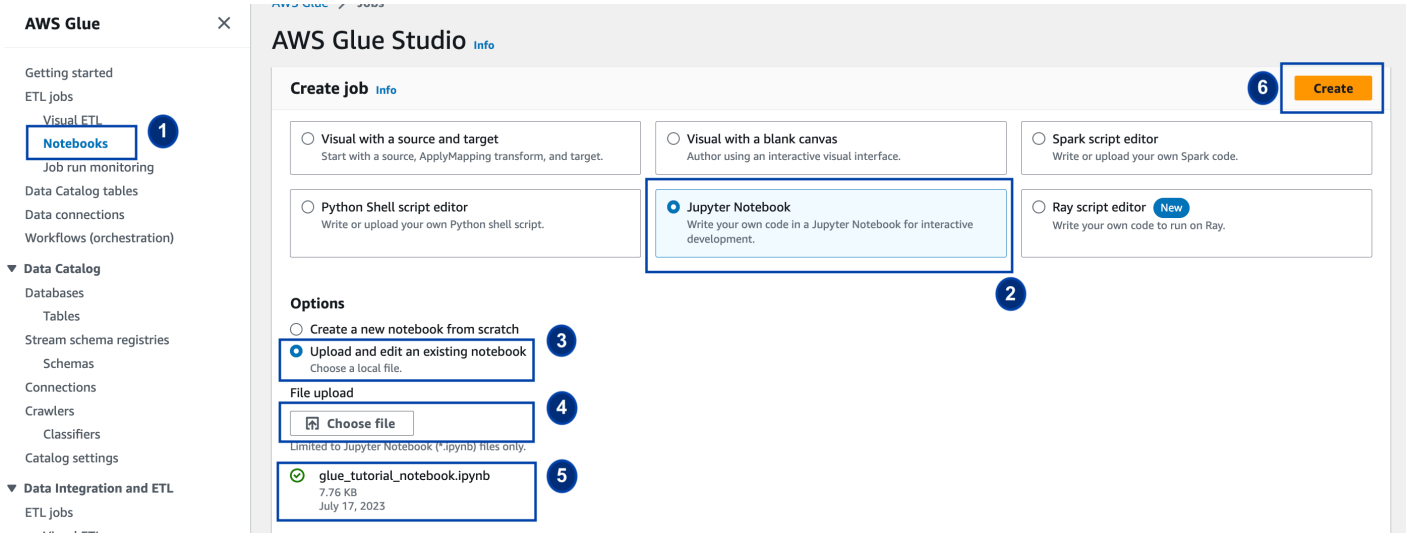
8. 按一下傳送資料，產生 5-10K 的記錄給 Kinesis。

使用 AWS Glue Studio 建立 AWS Glue 串流任務

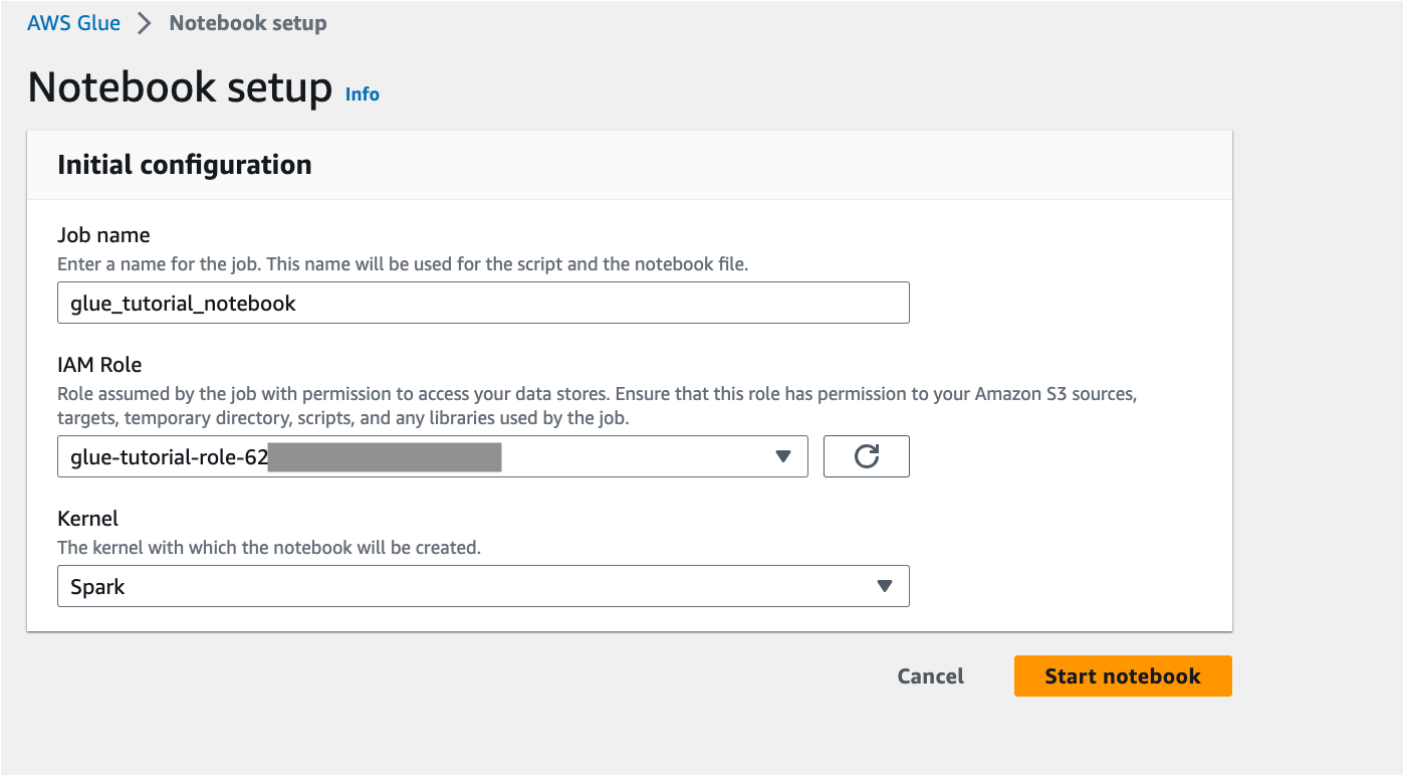
AWS Glue Studio 視覺化介面可簡化設計、協調和監控資料整合管道的程序。它可讓使用者無須編寫大量的程式碼即可構建資料轉換管道。除了視覺化的任務編寫體驗，AWS Glue Studio 還包括由 AWS Glue 互動式工作階段支援的 Jupyter 筆記本，供您在本教學課程的後續內容中使用。

設定 AWS Glue 串流互動式工作階段任務

1. 下載提供的筆記本檔案，並將其儲存到本機目錄
2. 開啟 AWS Glue 主控台，然後在左側窗格按一下筆記本 > Jupyter 筆記本 > 上傳並編輯現有的筆記本。在上一個步驟上傳記事本，然後按一下建立。



3. 為工作提供名稱、角色，然後選取預設 Spark 核心。然後按一下啟動筆記本。對於 IAM 角色，請選取由 CloudFormation 範本佈建的角色。您可以在 CloudFormation 的輸出索引標籤看到這個角色。



此筆記本有繼續本教學課程的所有必要指示。您可以依筆記本中的指示執行，也可以按照本教學課程繼續進行任務開發。

執行筆記本儲存格

1. (可選) 第一個程式碼儲存格，`%help` 列出了所有可用的筆記本魔法。您可以暫時先略過此儲存格，之後再隨時返回探索。
2. 從下一個程式碼區塊 `%streaming` 開始。這項魔法會將任務類型設定為串流，讓您開發、偵錯和部署 AWS Glue 串流 ETL 任務。
3. 執行下一個儲存格以建立 AWS Glue 互動式工作階段。輸出儲存格有訊息可確認工作階段建立。

Run this cell to set up and start your interactive session.

```
[1]: %glue_version 3.0

import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue import DynamicFrame
from datetime import datetime
from pyspark.sql.types import StructType, StructField, StringType, LongType
from pyspark.sql.functions import lit,col,from_json
import boto3

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
```

```
Setting Glue version to: 3.0
Authenticating with environment variables and user-defined glue_role_arn: arn:aws:iam::62          0:role/glue-tutorial-role
Trying to create a Glue session for the kernel.
Worker Type: G.1X
Number of Workers: 5
Session ID:                                af
Job Type: gluestreaming
Applying the following default arguments:
--glue_kernel_version 0.37.3
--enable-glue-datacatalog true
Waiting for session 4                                  to get into ready status...
Session 48                                 has been created.
```

4. 下一個儲存格可定義變數。將值取代為適合您任務的值，然後執行儲存格。例如：

```
output_database_name="default"
output_table_name="test_stream_001"

account_id = boto3.client("sts").get_caller_identity()["Account"]
region_name=boto3.client('s3').meta.region_name
stream_arn_name = "arn:aws:kinesis:{}:{}:stream/GlueStreamTest-{}".format(region_name,account_id,account_id)
s3_bucket_name = "streaming-tutorial-s3-target-{}".format(account_id)

output_location = "s3://{}/streaming_output/".format(s3_bucket_name)
checkpoint_location = "s3://{}/checkpoint_location/".format(s3_bucket_name)
```

5. 由於資料已經串流至 Kinesis Data Streams，因此下一個儲存格會使用串流中的結果。執行下一個儲存格。因為沒有列印陳述式，此儲存格並沒有預期的輸出。

6. 在下列儲存格中，您可以取得範例集並列印其結構描述和實際資料來探索傳入串流。例如：

Sample and print the incoming records

the sampling is for debugging purpose. You may comment off the entire code cell below, before deploying the actual code

[4]:
```
options = {
    "pollingTimeInMs": "20000",
    "windowSize": "5 seconds"
}
sampled_dynamic_frame = glueContext.getSampleStreamingDynamicFrame(data_frame, options, None)

count_of_sampled_records = sampled_dynamic_frame.count()

print(count_of_sampled_records)

sampled_dynamic_frame.printSchema()

sampled_dynamic_frame.toDF().show(10,False)
```

```
100
root
|-- eventtime: string
|-- manufacturer: string
|-- minutevolume: long
|-- o2stats: long
|-- pressurecontrol: long
|-- serialnumber: string
|-- ventilatorid: long

+--------------------+------------+------------+-------+---------------+------------------------------------+------------+
|eventtime           |manufacturer|minutevolume|o2stats|pressurecontrol|serialnumber                        |ventilatorid|
+--------------------+------------+------------+-------+---------------+------------------------------------+------------+
|2023-07-18 10:20:11|3M          |6           |92     |24             |a3e860ba-24b9-41c4-bc10-91c6b35e1406|6           |
|2023-07-18 10:20:11|Vyaire      |6           |95     |6              |96101dca-3e88-457f-b390-e3291df48a81|26          |
|2023-07-18 10:20:12|Getinge     |8           |96     |24             |18f3d448-1dee-4c80-835b-1a0daa818915|22          |
|2023-07-18 10:20:12|Getinge     |7           |98     |30             |25f425cd-b978-4953-9a03-4d607a639364|91          |
|2023-07-18 10:20:12|GE          |5           |93     |25             |2cd7cdc2-f5f5-4ff2-ae32-45e5a8922d53|93          |
```

7. 下一步，定義實際資料轉換邏輯。此儲存格包含每個微批次中觸發的 `processBatch` 方法。執行儲存格。在高階程序中，我們對輸入串流執行以下操作：

   a. 選取輸入欄子集。

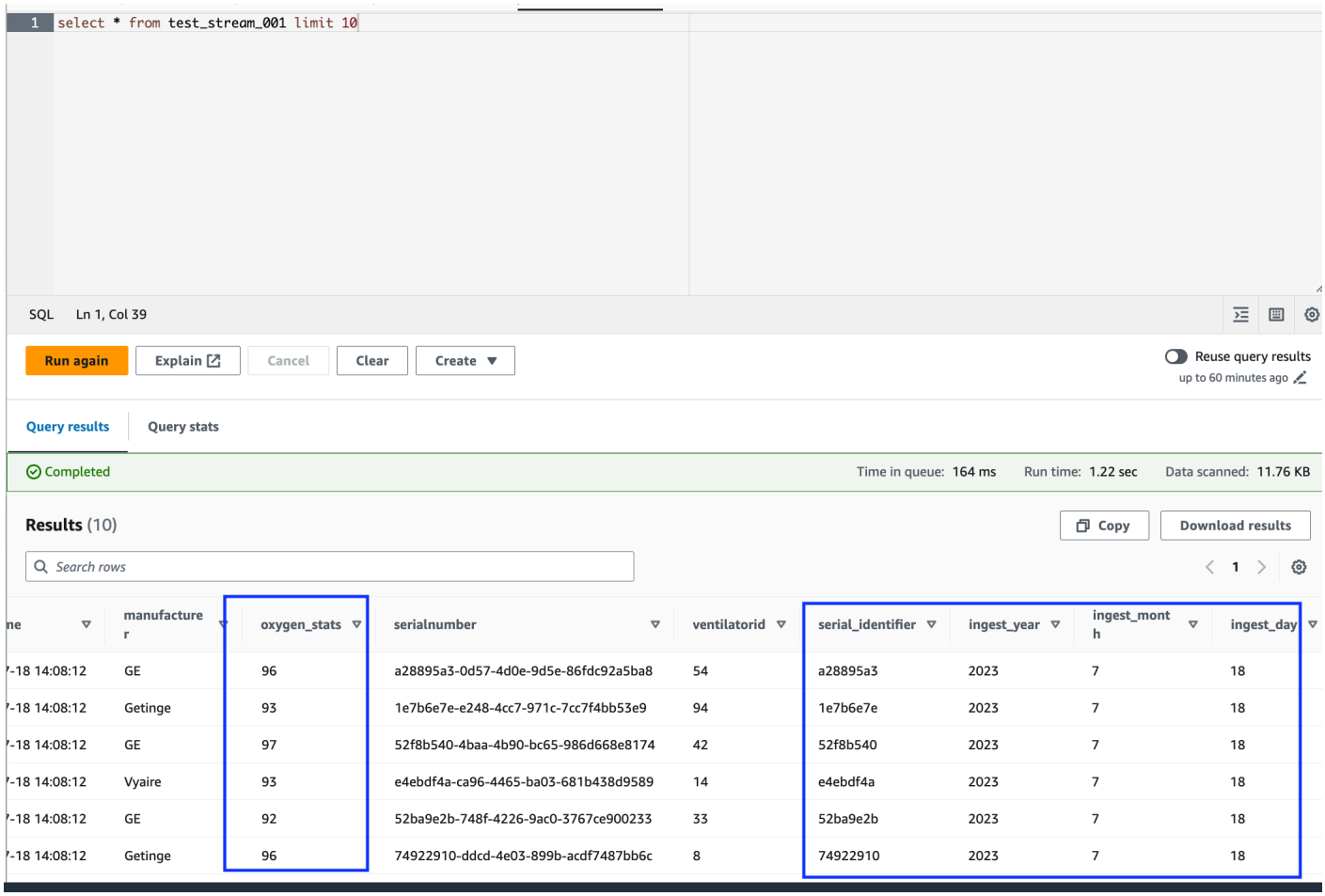
   b. 重新命名欄位 (將 o2stats 改為 oxygen_stats)。

   c. 衍生新欄位 (serial_identifier、ingest_year、ingest_month 和 ingest_da)。

   d. 將結果儲存在 Amazon S3 儲存貯體中，並建立分割的 AWS Glue 目錄資料表

8. 在最後一個儲存格中，每 10 秒觸發一次處理批次。執行儲存格並等待大約 30 秒，讓儲存格填入 Amazon S3 儲存貯體和 AWS Glue 目錄資料表。

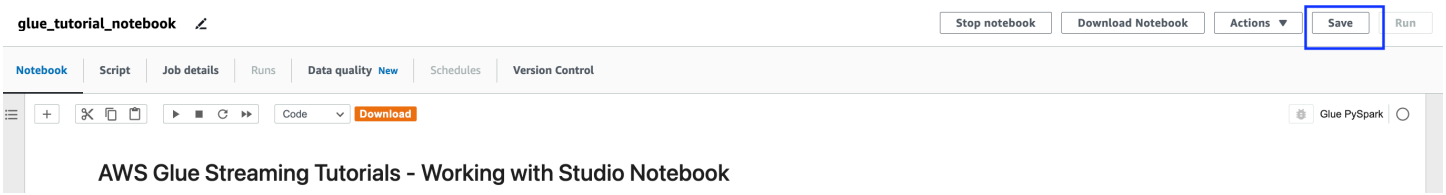
9. 最後，使用 Amazon Athena 查詢編輯器瀏覽儲存的資料。您可以看到重新命名的欄位和新的分割區。



此筆記本有繼續本教學課程的所有必要指示。您可以依筆記本中的指示執行，也可以按照本教學課程繼續進行任務開發。

儲存並執行 AWS Glue 任務

使用互動式工作階段筆記本完成應用程式的開發與測試後，按一下筆記本介面頂端的儲存。儲存後，您還可以將應用程式作為任務執行。



清除

為避免對您的帳戶產生額外費用，請停止您依指示啟動的串流任務。您可以停止筆記本來結束工作階段，便可停止串流任務。清空 Amazon S3 儲存貯體，並刪除您之前佈建的 AWS CloudFormation 堆疊。

結論

在本教學課程中，我們示範了如何使用 AWS Glue Studio 筆記本執行下列操作

- 使用筆記本撰寫串流 ETL 任務
- 預覽傳入資料串流
- 無須發佈 AWS Glue 任務即可撰寫程式碼和修正問題
- 查看端對端可行程式碼，移除任何偵錯，並從筆記本列印陳述式或儲存格
- 將程式碼發佈為 AWS Glue 任務

本教學課程的目標是提供您使用 AWS Glue 串流和互動式工作階段的實際操作經驗。建議您將本課程作為個別 AWS Glue 串流使用案例的參考資料。如需更多詳細資訊，請參閱 AWS Glue 互動式工作階段入門。

AWS Glue 串流概念

以下章節將提供 AWS Glue 串流概念的相關資訊。

主題

AWS Glue 串流任務的剖析

AWS Glue 串流任務根據 Spark 串流範例運作，並利用 Spark 架構中的結構化串流。串流任務會以特定時間間隔持續輪詢串流資料來源，以擷取記錄作為微批次。以下各節將詳細說明 AWS Glue 串流任務的各個部分。

forEachBatch

`forEachBatch` 方法是 AWS Glue 串流任務執行的進入點。AWS Glue 串流任務會使用 `forEachBatch` 方法來輪詢資料，其功能類似於在串流任務生命週期期間保持作用中的反覆運算器，並定期輪詢串流來源以取得新資料，並以微批次處理最新資料。

```
glueContext.forEachBatch(
    frame=dataFrame_AmazonKinesis_node1696872487972,
    batch_function=processBatch,
    options={
        "windowSize": "100 seconds",
        "checkpointLocation": args["TempDir"] + "/" + args["JOB_NAME"] + "/
checkpoint/",
    },
)
```

設定 forEachBatch 的 frame 屬性以指定串流來源。在此範例中，您在建立任務時在空白畫布建立的來源節點會填入任務的預設 DataFrame。將 batch_function 屬性設定為您決定針對每個微批次作業呼叫的 function。您必須定義函數來處理傳入資料的批次轉換。

來源

在 processBatch 函數的第一個步驟，程式會驗證您定義作為 forEachBatch 框架屬性的 DataFrame 的記錄計數。程式會將擷取時間戳記附加至非空白的 DataFrame。data_frame.count()>0 子句會判斷最新的微批次是否並非空白，並且已準備好進行進一步處理。

```
def processBatch(data_frame, batchId):
    if data_frame.count() >0:
        AmazonKinesis_node1696872487972 = DynamicFrame.fromDF(
            glueContext.add_ingestion_time_columns(data_frame, "hour"),
            glueContext,
            "from_data_frame",
        )
```

映射

該程式的下一部分是套用映射。Spark DataFrame 上的 Mapping.apply 方法可讓您定義關於資料元素的轉換規則。一般來說，您可以重新命名和變更資料類型，或在來源資料欄上套用自訂函數，並將這些函數映射至目標欄。

```
    #Script generated for node ChangeSchema
    ChangeSchema_node16986872679326 = ApplyMapping.apply(
        frame =  AmazonKinesis_node1696872487972,
        mappings = [
            ("eventtime", "string", "eventtime", "string"),
            ("manufacturer", "string", "manufacturer", "string"),
            ("minutevolume", "long", "minutevolume", "int"),
            ("o2stats", "long", "OxygenSaturation", "int"),
            ("pressurecontrol", "long", "pressurecontrol", "int"),
            ("serialnumber", "string", "serialnumber", "string"),
            ("ventilatorid", "long", "ventilatorid", "long"),
            ("ingest_year", "string", "ingest_year", "string"),
            ("ingest_month", "string", "ingest_month", "string"),
            ("ingest_day", "string", "ingest_day", "string"),
            ("ingest_hour", "string", "ingest_hour", "string"),
```

```
        ],
        transformation_ctx="ChangeSchema_node16986872679326",
    )
        )
```

接收

在這個部分，來自串流來源的傳入資料集會儲存在目標位置。在此範例中，我們會將資料寫入 Amazon S3 位置。AmazonS3_node_path 屬性詳細資料會根據您在建立任務期間所使用的設定從畫布預先填入。您可以根據您的使用案例來設定 updateBehavior，並決定不更新資料目錄表格，或在後續執行時建立資料目錄並更新資料目錄結構描述，或是建立目錄表格而不在後續執行時更新結構描述定義。

partitionKeys 屬性定義了存儲分割區選項。預設行為是根據在來源區段中提供的 ingestion_time_columns 來分割資料。compression 屬性可讓您設定要在目標寫入期間套用的壓縮演算法。您可以選擇將壓縮技術設定為 Snappy、LZO 或 GZIP。enableUpdateCatalog 屬性可控制是否需要更新 AWS Glue 目錄表格。此屬性可用的選項為 True 或 False。

```
#Script generated for node Amazon S3
AmazonS3_node1696872743449 = glueContext.getSink(
    path =  AmazonS3_node1696872743449_path,
    connection_type = "s3",
    updateBehavior = "UPDATE_IN_DATABASE",
    partitionKeys = ["ingest_year", "ingest_month", "ingest_day", "ingest_hour"],
    compression = "snappy",
    enableUpdateCatalog = True,
    transformation_ctx = "AmazonS3_node1696872743449",
)
```

AWS Glue 目錄連接器

任務的此一區段控制 AWS Glue 目錄表格更新行為。根據 AWS Glue 目錄資料庫名稱以及與您正在設計的 AWS Glue 任務相關的表格名稱設定 catalogDatabase 和 catalogTableName 屬性。您可以透過 setFormat 屬性定義目標資料的檔案格式。在這個例子中，我們會以 Parquet 格式儲存資料。

參考本教學課程設定並執行 AWS Glue 串流任務後，在 Amazon Kinesis Data Streams 產生的串流資料會以 Parquet 格式儲存在 Amazon S3 位置，並進行 Snappy 壓縮。成功執行串流任務後，您將可透過 Amazon Athena 查詢資料。

```
    AmazonS3_node1696872743449 = setCatalogInfo(
        catalogDatabase =  "demo", catalogTableName = "demo_stream_transform_result"
    )
    AmazonS3_node1696872743449.setFormat("glueparquet")
    AmazonS3_node1696872743449.writeFormat("ChangeSchema_node16986872679326")
    )
```

Kafka 連線

指定連接到 Kafka 叢集或 Amazon Managed Streaming for Apache Kafka 叢集的連線。

您可以使用存儲在「數據目錄」表中的信息或提供信息直接訪問數據流來讀取和寫入卡夫卡數據流。您可以從卡夫卡讀取信息到火花 DataFrame，然後將其轉換為 Glue。 AWS DynamicFrame您可以以 JSON 格式寫信給 DynamicFrames 卡夫卡。如果您直接存取資料串流，則請使用這些選項來提供如何存取資料串流的相關資訊。

如果您使用getCatalogSource或create_data_frame_from_catalog使用卡夫卡流源中的記錄，getCatalogSink或者write_dynamic_frame_from_catalog將記錄寫入卡夫卡，並且該作業具有數據目錄數據庫和表名信息，則可以使用該信息來獲取從卡夫卡流源讀取一些基本參數。如果使用getSource、、getCatalogSink、或getSourceWithFormat，createDataFrameFromOptions或 getSinkWithFormat create_data_frame_from_optionswrite_dynamic_frame_from_catalog，則必須使用此處描述的連接選項來指定這些基本參數。

您可以使用類中的指定方法下面的參數指定卡夫卡的連接選項。GlueContext

- Scala
  - connectionOptions：與 getSource、createDataFrameFromOptions、getSink 搭配使用
  - additionalOptions：與 getCatalogSource、getCatalogSink 搭配使用。
  - options：與 getSourceWithFormat、getSinkWithFormat 搭配使用。
- Python
  - connection_options：與 create_data_frame_from_options、write_dynamic_frame_from_options 搭配使用。

- additional_options：與 create_data_frame_from_catalog、write_dynamic_frame_from_catalog 搭配使用。
- options：與 getSource、getSink 搭配使用。

如需有關串流 ETL 任務的注意事項和限制，請參閱 the section called “串流 ETL 注意事項和限制”。

設定 Kafka

連接到通過互聯網可用的卡夫卡流沒有 AWS 先決條件。

您可以建立 AWS Glue Kafka 連線來管理您的連線認證。如需詳細資訊，請參閱 the section called “為 Kafka 資料串流建立連線”。在您的 AWS Glue 工作組態中，提供 *ConnectionName* 做為其他網路連線，然後在您的方法呼叫中，提供####給參數。connectionName

在某些情況下，您需要設定其他先決條件：

- 如果搭配 IAM 身分驗證使用 Amazon Managed Streaming for Apache Kafka，您會需要適當的 IAM 組態。
- 如果搭配 Amazon VPC 使用 Amazon Managed Streaming for Apache Kafka，您會需要適當的 Amazon VPC 組態。您必須建立可提供 Amazon VPC 連線資訊的 AWS Glue 連線。您需要工作組態，才能將 AWS Glue 連線納入為其他網路連線。

如需有關串流 ETL 任務先決條件的詳細資訊，請參閱 the section called “串流 ETL 任務”。

範例：從 Kafka 串流讀取

搭配 the section called “forEachBatch” 使用。

Kafka 串流來源範例：

```
kafka_options =
    { "connectionName": "ConfluentKafka",
      "topicName": "kafka-auth-topic",
      "startingOffsets": "earliest",
      "inferSchema": "true",
      "classification": "json"
    }
```

```
data_frame_datasource0 =
 glueContext.create_data_frame.from_options(connection_type="kafka",
 connection_options=kafka_options)
```

範例：寫入卡夫卡串流

寫信給卡夫卡的例子：

使用該getSink方法的示例：

```
data_frame_datasource0 =
glueContext.getSink(
 connectionType="kafka",
 connectionOptions={
  JsonOptions("""{
   "connectionName": "ConfluentKafka",
   "classification": "json",
   "topic": "kafka-auth-topic",
   "typeOfData": "kafka"}
  """)},
 transformationContext="dataframe_ApacheKafka_node1711729173428")
 .getDataFrame()
```

使用該write_dynamic_frame.from_options方法的示例：

```
kafka_options =
    { "connectionName": "ConfluentKafka",
      "topicName": "kafka-auth-topic",
      "classification": "json"
    }
data_frame_datasource0 =
 glueContext.write_dynamic_frame.from_options(connection_type="kafka",
 connection_options=kafka_options)
```

Kafka 連線選項參考

閱讀時，請使用以下連接選項"connectionType": "kafka"：

- "bootstrap.servers" (必要) 自舉伺服器 URL 的清單，例如 b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094。此選項必須在 API 呼叫中指定，或在 Data Catalog 的資料表中繼資料中定義。

- "security.protocol"(必要) 用來與代理程式通訊的協定。可能的值為 "SSL" 或 "PLAINTEXT"。
- "topicName" (必要) 要訂閱的主題清單 (以逗號分隔)。您必須指定 "topicName"、"assign" 或 "subscribePattern" 其中一個。
- "assign"：(必要) JSON 字串，指定要消耗的特定 TopicPartitions。您必須指定 "topicName"、"assign" 或 "subscribePattern" 其中一個。

  範例：'{"topicA":[0,1],"topicB":[2,4]}'
- "subscribePattern"：(必要) 識別要訂閱的主題清單的 Java regex 字串。您必須指定 "topicName"、"assign" 或 "subscribePattern" 其中一個。

  範例：'topic.*'
- "classification" (必要) 記錄中資料使用的檔案格式。除非透過資料型錄提供，否則為必要。
- "delimiter" (選用) 當 classification 為 CSV 時使用的值分隔符號。預設值為 ","。
- "startingOffsets"：(選用) 要從中讀取資料的 Kafka 主題的起始位置。可能的值為 "earliest" 或 "latest"。預設值為 "latest"。
- "startingTimestamp": (選用，僅適用於 AWS Glue 4.0 版或更新版本) Kafka 主題中要讀取資料的記錄時間戳記。可能的值是 yyyy-mm-ddTHH:MM:SSZ 模式中 UTC 格式的時間戳記字串 (其中 Z 代表以 +/- 表示的 UTC 時區偏移。例如："2023-04-04T08:00:00-04:00")。

  注意： AWS Glue 串流指令集的「連線選項」清單中只能有一個「開始偏移」或「開始時間戳記」，包括這兩個屬性會導致工作失敗。
- "endingOffsets"：(選用) 批次查詢結束時的終點。可能值為 "latest" 或指定每個 TopicPartition 結束偏移的 JSON 字串。

  對於 JSON 字串，格式為 {"topicA":{"0":23,"1":-1},"topicB":{"0":-1}}。值 -1 作為偏移代表 "latest"。
- "pollTimeoutMs"：(選用) 在 Spark 任務執行器中從 Kafka 輪詢資料的逾時 (以毫秒為單位)。預設值為 512。
- "numRetries"：(選用) 擷取 Kafka 位移失敗之前，要重試的次數。預設值為 3。
- "retryIntervalMs"：(選用) 重試擷取 Kafka 偏移量之前等待的時間 (毫秒)。預設值為 10。
- "maxOffsetsPerTrigger"：(選用) 每個觸發間隔所處理之偏移數目上限的速率限制。指定的偏移總數會按比例跨 topicPartitions 或不同磁碟區而分割。預設值為 null，這表示消費者讀取所有偏移，直到已知的最新偏移。

- "minPartitions"：(選用) 從 Kafka 讀取所需的分割區最小數量。預設值為 null，這表示 Spark 分割區的數量等於 Kafka 分割區的數量。
- "includeHeaders"：(選用) 是否包含 Kafka 標頭。當選項設定為「true」時，資料輸出將包含一個名為「glue_streaming_kafka_headers」的額外欄，其類型為 Array[Struct(key: String, value: String)]。預設值為 "false"。此選項能在 AWS Glue 3.0 版或更新版中使用。
- "schema"：(當 InferSchema 設定為 false 時為必要) 用於處理承載的架構。如果分類為 avro，提供的架構必須採用 Avro 架構格式。如果分類不是 avro，提供的架構必須採用 DDL 架構格式。

  以下是架構範例。

  Example in DDL schema format

  ```
  'column1' INT, 'column2' STRING , 'column3' FLOAT
  ```

  Example in Avro schema format

  ```
  {
  "type":"array",
  "items":
  {
  "type":"record",
  "name":"test",
  "fields":
  [
    {
      "name":"_id",
      "type":"string"
    },
    {
      "name":"index",
      "type":
      [
        "int",
        "string",
        "float"
      ]
    }
  ]
  }
  }
  ```

- "inferSchema"：(選用) 預設值為 'false'。如果設為 'true'，將在執行時間時從 foreachbatch 承載偵測架構。
- "avroSchema"：(已棄用) 使用 Avro 格式時，用於指定 Avro 資料架構的參數。此參數現已棄用。使用 schema 參數。
- "addRecordTimestamp"：(選用) 當此選項設定為 'true' 時，資料輸出將包含一個名為 "__src_timestamp" 的額外資料欄，其指示主題收到相應記錄的時間。預設值為 'false'。在 AWS Glue 4.0 版或更新版中支援此選項。
- "emitConsumerLagMetrics": (選擇性) 當選項設為 'true' 時，對於每個批次，它會發出主題接收到的最舊記錄到達時間之間的持續時間的AWS Glue指標。 CloudWatch該指標的名稱是「膠合. 驅動程序. maxConsumerLagInMs」. 預設值為 'false'。在 AWS Glue 4.0 版或更新版中支援此選項。

寫入時，請使用以下連接選項"connectionType": "kafka"：

- "connectionName"（必要）用於連接到卡夫卡集群的 AWS Glue 連接名稱（類似於卡夫卡源）。
- "topic"(必要) 如果主題欄存在，則除非設定了主題組態選項，否則在將指定資料列寫入 Kafka 時，會使用其值作為主題。也就是說，組topic態選項會覆寫主題欄。
- "partition"(選擇性) 如果指定了有效的分割區編號，partition將在傳送記錄時使用。

  如果沒有指定分區，但存key在一個分區，將使用密鑰的散列來選擇一個分區。

  如果key既不存在partition也不存在，則當至少為分區產生 batch .size 字節時，將根據粘性分區這些更改選擇分區。
- "key"(選擇性) 如partition果為 null，則用於分割。
- "classification"(選擇性) 記錄中資料使用的檔案格式。我們只支持 JSON，CSV 和阿夫羅。

  使用 Avro 格式，我們可以提供自定義的 AvroSchema 進行序列化，但請注意，這也需要在源代碼上提供反序列化。否則，默認情況下它使用 Apache AvroSchema 進行序列化。

此外，您可以根據需要通過更新卡夫卡生產者配置參數微調卡夫卡水槽。請注意，連接選項上沒有允許列出，所有鍵值對都保留在接收器上。

但是，有一個小的拒絕列表的選項不會生效。如需詳細資訊，請參閱 Kafka 特定組態。

Kinesis 連線

您可以使用儲存在 Data Catalog 資料表中的資訊，或提供資訊以直接存取資料串流，藉此從Amazon Kinesis Data Streams 中讀取和寫入。您可以從 Kinesis 讀取信息到火花 DataFrame，然後將其轉換為

AWS Glue DynamicFrame。您可以 DynamicFrames 使用 JSON 格式寫入 Kinesis。如果您直接存取資料串流，則請使用這些選項來提供如何存取資料串流的相關資訊。

如果您使用 `getCatalogSource` 或 `create_data_frame_from_catalog` 來取用來自 Kinesis 串流來源的記錄，則該任務具有 Data Catalog 資料庫和資料表名稱資訊，並可以使用它來獲取一些從 Kinesis 串流來源讀取的基本參數。如果使用 `getSource`、`getSourceWithFormat`、`createDataFrameFromOptions` 或 `create_data_frame_from_options`，則您必須使用此處描述的連線選項來指定這些基本參數。

您可以使用 `GlueContext` 類別中指定方法的下列引數來指定 Kinesis 的連線選項。

- Scala
  - `connectionOptions`：與 `getSource`、`createDataFrameFromOptions`、`getSink` 搭配使用
  - `additionalOptions`：與 `getCatalogSource`、`getCatalogSink` 搭配使用。
  - `options`：與 `getSourceWithFormat`、`getSinkWithFormat` 搭配使用。
- Python
  - `connection_options`：與 `create_data_frame_from_options`、`write_dynamic_frame_from_options` 搭配使用。
  - `additional_options`：與 `create_data_frame_from_catalog`、`write_dynamic_frame_from_catalog` 搭配使用。
  - `options`：與 `getSource`、`getSink` 搭配使用。

如需有關串流 ETL 任務的注意事項和限制，請參閱 the section called “串流 ETL 注意事項和限制”。

設定 Kinesis

若要連線到 AWS Glue Spark 工作中的 Kinesis 資料串流，您需要一些先決條件：

- 如果讀取， AWS Glue 工作必須具有 Kinesis 資料串流的讀取存取權層級 IAM 權限。
- 如果要寫入， AWS Glue 工作必須具有 Kinesis 資料串流的寫入存取權層級 IAM 權限。

在某些情況下，您需要設定其他先決條件：

- 如果您的 AWS Glue 任務設定了「其他網路連線」(通常是用來連接到其他資料集)，而其中一個連線提供 Amazon VPC 網路選項，則會引導您的任務透過 Amazon VPC 進行通訊。在這種情況下，您還需要將 Kinesis 資料串流設定為透過 Amazon VPC 進行通訊。為此，您可以建立 Amazon VPC 與 Kinesis 資料串流之間的介面 VPC 端點。如需詳細資訊，請參閱 Using Kinesis Data Streams with Interface VPC Endpoints。
- 在另一個帳戶中指定 Amazon Kinesis Data Streams 時，您必須設定角色和政策以允許跨帳戶存取。如需詳細資訊，請參閱範例：從不同帳戶中的 Kinesis 串流讀取。

如需有關串流 ETL 任務先決條件的詳細資訊，請參閱 the section called “串流 ETL 任務”。

從 Kinesis 讀取

範例：從 Kinesis 串流讀取

搭配 the section called “forEachBatch” 使用。

Amazon Kinesis 串流來源範例：

```
kinesis_options =
   { "streamARN": "arn:aws:kinesis:us-east-2:777788889999:stream/fromOptionsStream",
     "startingPosition": "TRIM_HORIZON",
     "inferSchema": "true",
     "classification": "json"
   }
data_frame_datasource0 =
 glueContext.create_data_frame.from_options(connection_type="kinesis",
 connection_options=kinesis_options)
```

寫入 Kinesis

範例：寫入 Kinesis 串流

搭配 the section called “forEachBatch” 使用。您 DynamicFrame 將以 JSON 格式寫入資料流。如果任務在數次重試後仍無法寫入，便會失敗。依預設，每 DynamicFrame 筆記錄都會個別傳送至 Kinesis 串流。您可以使用 `aggregationEnabled` 和關聯的參數來設定此行為。

從串流任務寫入 Amazon Kinesis 的範例：

Python

```
glueContext.write_dynamic_frame.from_options(
```

```
    frame=frameToWrite
    connection_type="kinesis",
    connection_options={
        "partitionKey": "part1",
        "streamARN": "arn:aws:kinesis:us-east-1:111122223333:stream/streamName",
    }
)
```

Scala

```
glueContext.getSinkWithFormat(
                connectionType="kinesis",
                options=JsonOptions("""{
                    "streamARN": "arn:aws:kinesis:us-
east-1:111122223333:stream/streamName",
                    "partitionKey": "part1"
                }"""),
            )
            .writeDynamicFrame(frameToWrite)
```

Kinesis 連線參數

指定 Amazon Kinesis Data Streams 的連線選項。

針對 Kinesis 串流資料來源使用下列的連線選項：

- "streamARN" (必要) 用於讀取/寫入。Kinesis 資料串流的 ARN。
- "classification" (讀取時為必要) 用於讀取。記錄中資料使用的檔案格式。除非透過資料型錄提供，否則為必要。
- "streamName" – (選用) 用於讀取。要從中讀取的 Kinesis 資料串流名稱。與 endpointUrl 搭配使用。
- "endpointUrl" – (選用) 用於讀取。預設："https://kinesis.us-east-1.amazonaws.com"。Kinesis 串流的 AWS 端點。除非您要連線到特殊區域，否則無需變更此設定。
- "partitionKey" – (選用) 用於寫入。在產生記錄時使用的 Kinesis 分割區索引鍵。
- "delimiter" (選用) 用於讀取。當 classification 為 CSV 時使用的值分隔符號。預設值為 ","。
- "startingPosition"：(選用) 用於讀取。Kinesis 資料串流中要從中讀取資料的起始位置。可能的值包括 "latest"、"trim_horizon"、"earliest" 或 yyyy-mm-

ddTHH:MM:SSZ 模式中 UTC 格式的時間戳記字串 (其中 Z 代表以 +/- 表示的 UTC 時區偏移。例如："2023-04-04T08:00:00-04:00")。預設值為 "latest"。注意：只有 AWS Glue 4.0 版或更新版本"startingPosition"才支援 UTC 格式的時間戳記字串。

- "failOnDataLoss"：(選用) 如果有任何作用中的碎片遺失或過期，則任務失敗。預設值為 "false"。
- "awsSTSRoleARN"：(選用) 用於讀取/寫入。角色的 Amazon 資源名稱 (ARN) 假定使用 AWS Security Token Service (AWS STS)。此角色必須具有描述或讀取 Kinesis 資料串流記錄操作的許可。存取不同帳戶中的資料串流時，您必須使用此參數。搭配 "awsSTSSessionName" 使用。
- "awsSTSSessionName"：(選用) 用於讀取/寫入。使用 AWS STS擔任角色之工作階段的識別符。存取不同帳戶中的資料串流時，您必須使用此參數。搭配 "awsSTSRoleARN" 使用。
- "awsSTSEndpoint": (選擇性) 以假定角色連線到 Kinesis 時要使用的 AWS STS 端點。這允許在 VPC 中使用地區 AWS STS 端點，而預設全域端點無法執行此操作。
- "maxFetchTimeInMs"：(選用) 用於讀取。工作執行程式從 Kinesis 資料串流讀取目前批次記錄所花費的時間上限，以毫秒 (毫秒) 為單位。在這段時間內可能會進行多個 GetRecords API 呼叫。預設值為 1000。
- "maxFetchRecordsPerShard"：(選用) 用於讀取。每個微批次在 Kinesis 資料串流中，每個碎片可擷取的最大記錄數。注意：如果串流工作已讀取 Kinesis 的額外記錄 (在相同的 Get-record 呼叫中)，用戶端可能會超過此限制。如果maxFetchRecordsPerShard需要嚴格，那麼它需要是maxRecordPerRead. 預設值為 100000。
- "maxRecordPerRead"：(選用) 用於讀取。要從每個 getRecords 操作的 Kinesis 資料串流中擷取的記錄數量上限。預設值為 10000。
- "addIdleTimeBetweenReads"：(選用) 用於讀取。增加兩個連續 getRecords 操作之間的時間延遲。預設值為 "False"。此選項僅在 Glue 2.0 及以上版本上才可設定。
- "idleTimeBetweenReadsInMs"：(選用) 用於讀取。兩個連續 getRecords 操作的最小延遲時間，以毫秒為單位指定。預設值為 1000。此選項僅在 Glue 2.0 及以上版本上才可設定。
- "describeShardInterval"：(選用) 用於讀取。指令碼考慮重新分片的兩個 ListShards API 呼叫之間的最小時間間隔。如需詳細資訊，請參閱 Amazon Kinesis Data Streams 開發人員指南中的重新分片的策略。預設值為 1s。
- "numRetries"：(選用) 用於讀取。Kinesis Data Streams API 請求的重試數上限。預設值為 3。
- "retryIntervalMs"：(選用) 用於讀取。重試 Kinesis Data Streams API 呼叫之前的冷卻時間期間 (以毫秒為單位)。預設值為 1000。
- "maxRetryIntervalMs"：(選用) 用於讀取。Kinesis Data Streams API 呼叫之兩次重試之間的最大冷卻時間期間 (以毫秒為單位)。預設值為 10000。

- "avoidEmptyBatches"：(選用) 用於讀取。避免建立空白微批次任務，方法是在批次開始之前檢查 Kinesis 資料串流中是否有未讀取的資料。預設值為 "False"。
- "schema"：(在 inferSchema 設定為 false 時為必要) 用於讀取。用於處理承載的結構描述。如果分類為 avro，提供的架構必須採用 Avro 架構格式。如果分類不是 avro，提供的架構必須採用 DDL 架構格式。

  以下是架構範例。

  Example in DDL schema format

  ```
  `column1` INT, `column2` STRING , `column3` FLOAT
  ```

  Example in Avro schema format

  ```
  {
    "type":"array",
    "items":
    {
      "type":"record",
      "name":"test",
      "fields":
      [
        {
          "name":"_id",
          "type":"string"
        },
        {
          "name":"index",
          "type":
          [
            "int",
            "string",
            "float"
          ]
        }
      ]
    }
  }
  ```

- "inferSchema"：(選用) 用於讀取。預設值為 'false'。如果設為 'true'，將在執行時間時從 foreachbatch 承載偵測架構。

- "avroSchema"：(已棄用) 用於讀取。使用 Avro 格式時，用於指定 Avro 資料架構的參數。此參數現已棄用。使用 schema 參數。
- "addRecordTimestamp"：(選用) 用於讀取。當此選項設定為 'true' 時，資料輸出將包含一個名為 "__src_timestamp" 的額外資料欄，其指示串流收到相應記錄的時間。預設值為 'false'。在 AWS Glue 4.0 版或更新版中支援此選項。
- "emitConsumerLagMetrics"：(選用) 用於讀取。當該選項設置為 'true' 時，對於每個批次，它將發出流接收到的最舊記錄到達時間之間的持續時間的AWS Glue指標。 CloudWatch該度量標準的名稱是「膠合. 驅動程序. maxConsumerLagInMs」. 預設值為 'false'。在 AWS Glue 4.0 版或更新版中支援此選項。
- "fanoutConsumerARN"：(選用) 用於讀取。Kinesis 串流取用者的 ARN，適用於 streamARN 中指定的串流。用於啟用 Kinesis 連線的強化廣發功能模式。如需有關使用強化廣發功能使用 Kinesis 串流的詳細資訊，請參閱 the section called “在 Kinesis 串流任務中使用強化廣發功能”。
- "recordMaxBufferedTime" – (選用) 用於寫入。預設：1000 (毫秒)。在等待寫入時，記錄受到緩衝的最長時間。
- "aggregationEnabled" – (選用) 用於寫入。預設：true。指定是否應在將記錄傳送至 Kinesis 前先彙整記錄。
- "aggregationMaxSize" – (選用) 用於寫入。預設：51200 (位元組)。若記錄大於此限制，則其會略過彙整工具。請注意，Kinesis 會強制執行 50 KB 的記錄大小限制。若您將此值設定為超過 50 KB，Kinesis 將會拒絕過大的記錄。
- "aggregationMaxCount" – (選用) 用於寫入。預設：4294967295。要匯入彙整記錄的項目數量上限。
- "producerRateLimit" – (選用) 用於寫入。預設：150 (%)。作為後端限制的百分比來限制從單一生產者 (例如您的任務) 傳送的每個碎片輸送量。
- "collectionMaxCount" – (選用) 用於寫入。預設：500。要打包到 PutRecords 請求中的最大項目數。
- "collectionMaxSize" – (選用) 用於寫入。預設：5242880 (位元組)。與 PutRecords 請求一起發送的最大數據量。

AWS Glue 串流選項

指定連接到 Kafka 叢集或 Amazon Managed Streaming for Apache Kafka 叢集的連線。

您可以使用存儲在「數據目錄」表中的信息或提供信息直接訪問數據流來讀取和寫入卡夫卡數據流。您可以從卡夫卡讀取信息到火花 DataFrame，然後將其轉換為 Glue。 AWS DynamicFrame您可以以

JSON 格式寫信給 DynamicFrames 卡夫卡。如果您直接存取資料串流，則請使用這些選項來提供如何存取資料串流的相關資訊。

如果您使用getCatalogSource或create_data_frame_from_catalog使用卡夫卡流源中的記錄，getCatalogSink或者write_dynamic_frame_from_catalog將記錄寫入卡夫卡，並且該作業具有數據目錄數據庫和表名信息，則可以使用該信息來獲取從卡夫卡流源讀取一些基本參數。如果使用getSource、、getCatalogSink、或getSourceWithFormat，createDataFrameFromOptions或 getSinkWithFormat create_data_frame_from_optionswrite_dynamic_frame_from_catalog，則必須使用此處描述的連接選項來指定這些基本參數。

您可以使用類中的指定方法下面的參數指定卡夫卡的連接選項。GlueContext

- Scala
  - connectionOptions：與 getSource、createDataFrameFromOptions、getSink 搭配使用
  - additionalOptions：與 getCatalogSource、getCatalogSink 搭配使用。
  - options：與 getSourceWithFormat、getSinkWithFormat 搭配使用。
- Python
  - connection_options：與 create_data_frame_from_options、write_dynamic_frame_from_options 搭配使用。
  - additional_options：與 create_data_frame_from_catalog、write_dynamic_frame_from_catalog 搭配使用。
  - options：與 getSource、getSink 搭配使用。

如需有關串流 ETL 任務的注意事項和限制，請參閱 the section called “串流 ETL 注意事項和限制”。

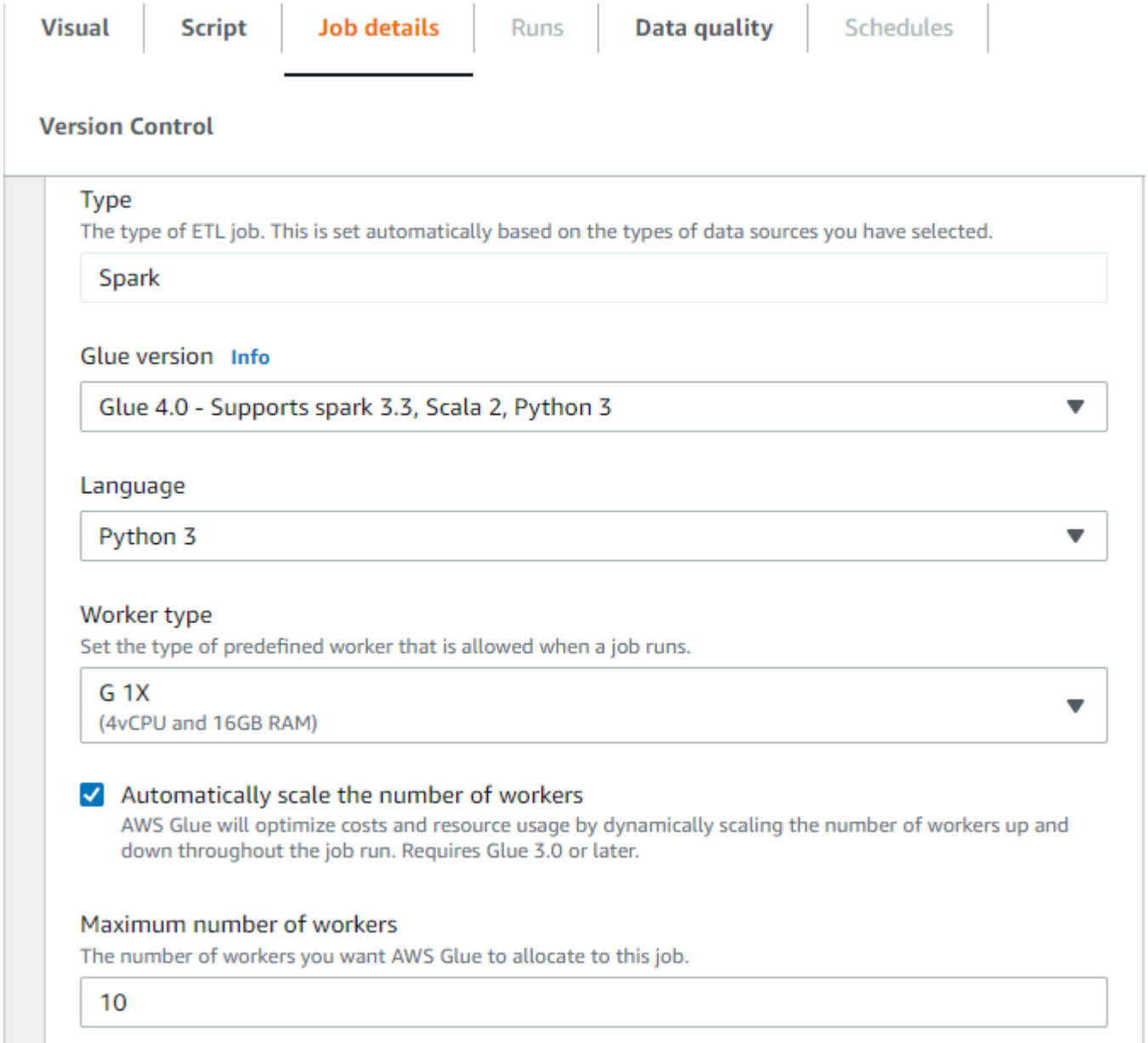
AWS Glue 串流自動調整

以下章節將提供 AWS Glue 串流自動調整的相關資訊

在 AWS Glue Studio 中啟用 Auto Scaling

在 AWS Glue Studio 中的 Job details (任務詳細資訊) 索引標籤上，選擇 Spark 或 Spark Streaming 類型，並將 Glue version (Glue 版本) 選為 **`Glue 3.0`** 或 **`Glue 4.0`**。然後，一個核取方塊將顯示在 Worker type (工作者類型) 下方。

- 選取 Automatically scale the number of workers (自動擴展工作者數量) 選項。
- 設定 Maximum number of workers (工作者數上限) 以定義可提供給任務執行的工作者數上限。



透過 AWS CLI 或開發套件啟用 Auto Scaling

要從 AWS CLI 為任務執行啟用 Auto Scaling，請使用下列組態執行 `start-job-run`：

```
{
    "JobName": "<your job name>",
    "Arguments": {
        "--enable-auto-scaling": "true"
    },
    "WorkerType": "G.2X", // G.1X and G.2X are allowed for Auto Scaling Jobs
    "NumberOfWorkers": 20, // represents Maximum number of workers
    ...other job run configurations...
}
```

在 ETL 任務執行完成後，您也可以呼叫 `get-job-run` 以檢查任務執行的實際資源使用情況 (以 DPU 秒為單位)。注意：新欄位 DPUSeconds 只會在已啟用 Auto Scaling 功能的 AWS Glue 3.0 或更高版本上針對批任務顯示。此欄位不支援串流任務。

```
$ aws glue get-job-run --job-name your-job-name --run-id jr_xx --endpoint https://
glue.us-east-1.amazonaws.com --region us-east-1
{
    "JobRun": {
        ...
        "GlueVersion": "3.0",
        "DPUSeconds": 386.0
    }
}
```

您還可以使用具有相同組態的 AWS Glue SDK 為任務執行設定 Auto Scaling。

運作方式

跨微批次進行調整

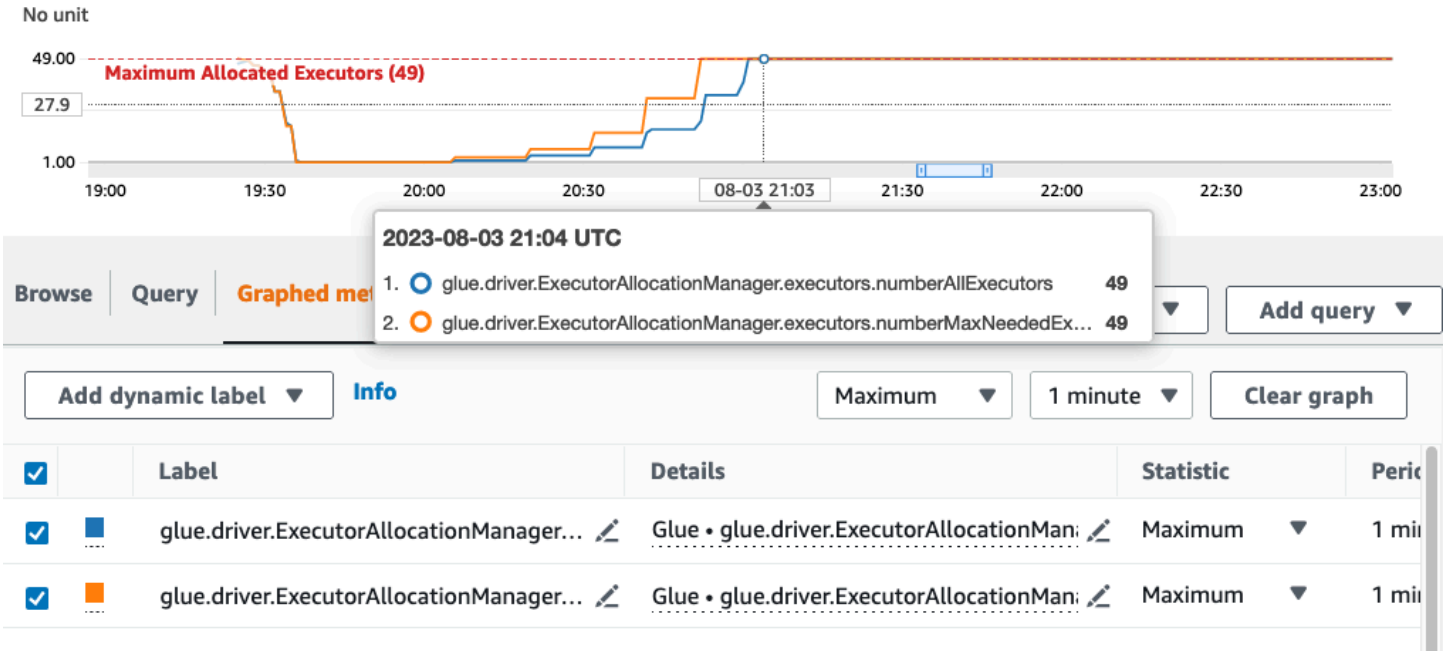
以下範例可說明自動調整的運作方式。

- 您有一個以 50 個 DPU 開始的 AWS Glue 任務。
- 自動調整功能已啟用。

在此範例中，查AWS Glue看幾個微型批次的 batchProcessingTime InMs「量度，並確定您的工作是否在您已建立的視窗大小內完成。如果任務提前完成，則視任務提前完成的時間多寡，AWS Glue 可能會

縮減規模。可以在中Amazon CloudWatch監視使用 numberAllExecutors「」繪製的指標，以查看自動調度資源的工作原理。

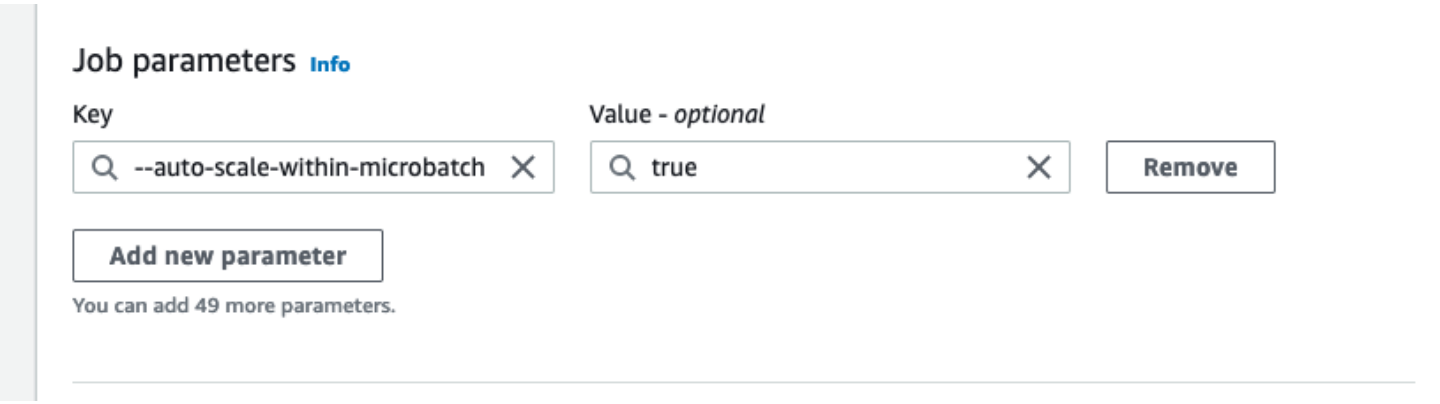
只有在每個微批次完成後，執行器的數量才會呈指數級擴展或縮減。正如您在 Amazon CloudWatch 監控日誌中所看到的，AWS Glue 會查看所需的執行器數量 (橘線)，並自動調整執行器數量 (藍線) 以與所需數量相符。



若 AWS Glue 縮減執行器數量後觀察到資料量增加，因而使微批次處理時間變長，AWS Glue 最多會擴展到指定的上限數量 50 個 DPU。

在微批次內調整

在上述範例中，系統會監控一些已完成的微批次，以決定要擴展還是縮減數量。較長的窗口需要自動調度，以便在微批次內更快地響應，而不是等待幾個微批次。在這些情況下，您可以將其他設定 `--auto-scaIe-within-microbatch` 設定為 `true`。您可以在 AWS Glue Studio 將此新增至 AWS Glue 的任務屬性，如下所示。



AWS Glue 串流的維護時段

AWS Glue 定期執行維護活動。在這些維護時段期間， AWS Glue 將需要重新啟動您的串流工作。您可以指定維護時段來控制重新啟動工作的時間。在本節中，我們概述了您可以在何處設置維護時段以及應考慮的特定行為。

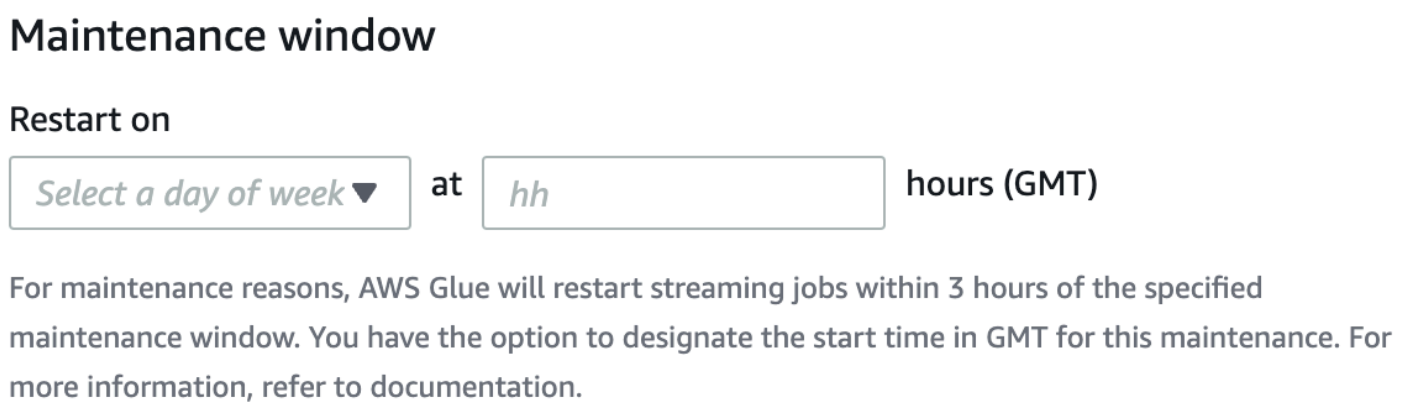
主題

設定維護時段

您可以使用 AWS Glue 工作室或 API 來設定維護視窗。

在 AWS Glue 工作室中設置維護窗口

您可以在「 AWS Glue 串流」Job 的「工作詳細資訊」頁面中指定維護時段。您可以在 GMT 中指定日期和時間。 AWS Glue 將在指定的時間範圍內重新啟動作業。



在 API 中設定維護時段

您也可以在建立 Job API 中設定維護時段。以下是透過 API 設定維護時段的範例。

```
aws glue create-job –name jobName –role roleArnForTheJob –command
 Name=gluestreaming,ScriptLocation=s3-path-to-the-script --maintenance-window="Sun:10"
```

範例命令如下：

```
aws glue create-job –name testMaintenance –role arn:aws:iam::012345678901:role/
Glue_DefaultRole –command Name=gluestreaming,ScriptLocation=s3://glue-example-test/
example.py –maintenance-window="Sun:10
```

維護視窗行為

AWS Glue 執行一系列步驟來決定何時重新啟動作業：

1. 啟動新的串流工作時， AWS Glue 首先檢查工作執行是否有關聯的逾時。逾時可讓您設定工作的結束時間。如果逾時少於 7 天，則不會重新啟動工作。
2. 如果逾時超過 7 天，則 AWS Glue 檢查工作是否已設定維護時段。如果是，則會選取該窗口，並將窗口分配給作業運行。 AWS Glue 將在指定維護時間的 3 小時內重新啟動工作。例如，如果您在格林威治標準時間星期一上午 10:00 設定維護時段，您的作業將在格林威治標準時間上午 10:00 至下午 1:00 之間重新啟動。
3. 如果未設定維護時段， AWS Glue 會自動將重新啟動時間設定為超過工作執行初始化時間的 7 天。舉例來說，如果您在格林威治標準時間 7/1/2024 12:00 啟動工作，但未指定維護時段，您的工作將設定為在 2024 年 7 月 8 日上午 12:00 格林威治標準時間重新啟動。

> **Note**
>
> 如果您已經在執行串流工作，這項變更將會影響您從 2024 年 7 月 1 日開始。您將有時間，直到 6 月 30 日配置您的維護時段。7 月 1 日之後，您啟動的任何串流工作都會根據此文件重新啟動。如果您需要任何其他 Support，可以聯繫 AWS 支持。

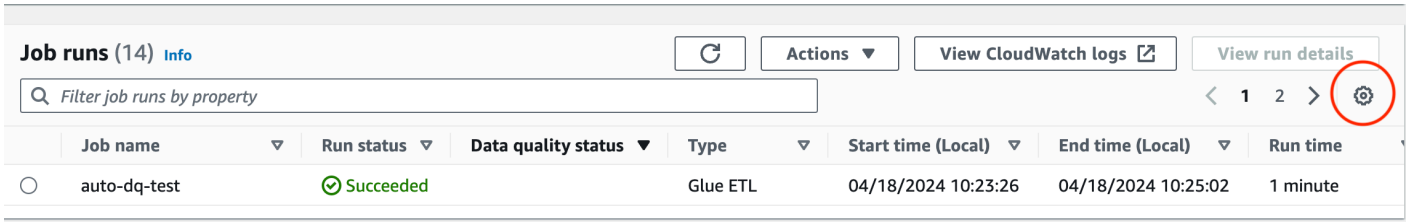
4. 有時候， AWS Glue 可能無法重新啟動工作，尤其是當未處理正在進行的微批次時。在這些情況下，工作將不會中斷。在這些情況下， AWS Glue 將在 14 天後重新啟動工作，在此情況下，不會遵循維護時段。

Job 監控

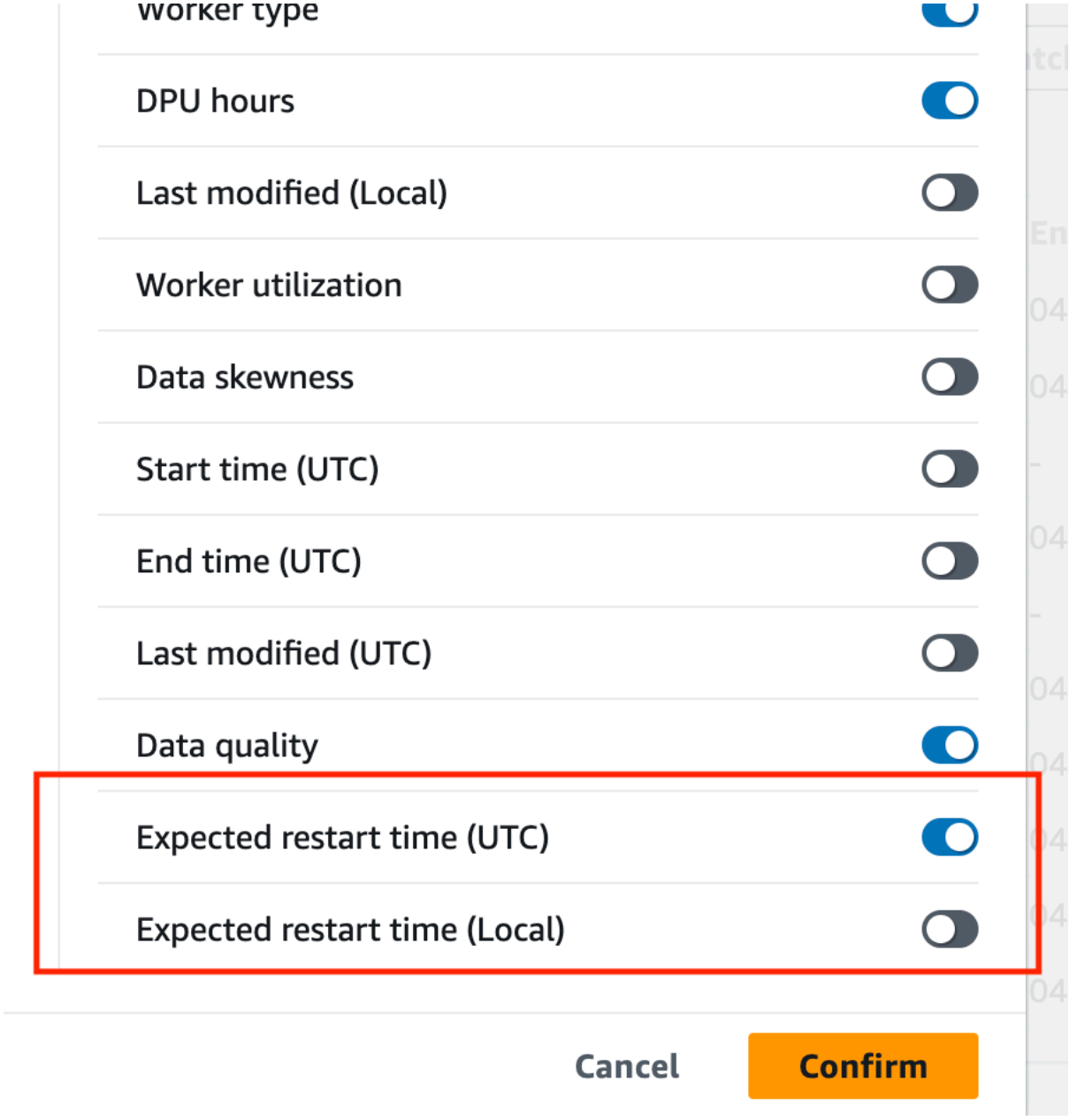
您可以在 [AWS Glue Studio 監視] 頁面中監視工作。

若要查看串流 Job 的預期下次重新啟動時間，請在 [監督] 頁面的 [工作執行] 表格中顯示資料欄。

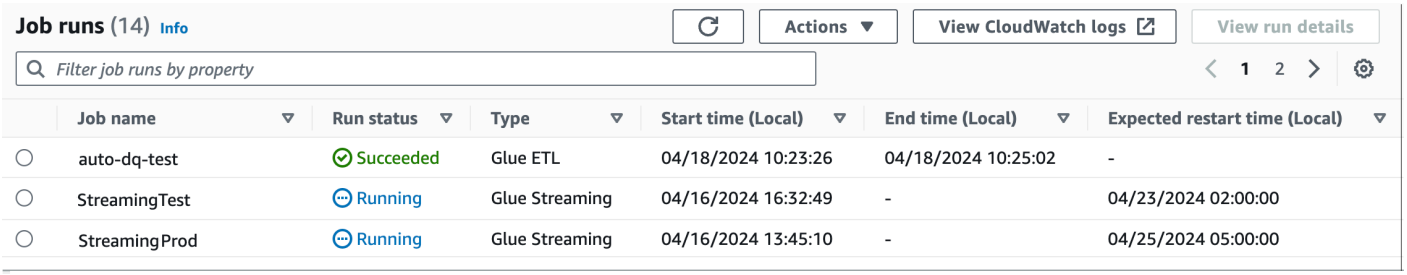
1. 按一下表格右上角的齒輪圖示。



2. 向下捲動，然後開啟預期的重新啟動時間欄。UTC 和當地時間選項均可用。



3. 然後，您可以檢視表格中的欄。



原始工作的狀態為「已過期」，而新的工作執行處理則為「RUNNING」狀態。重新啟動的新作業執行將具有作業執行 ID 作為初始作業執行 ID 的串連，加上代表重新啟動計數的前置詞「restart_」。例如，如果初始作業執行 ID 為jr_1234，則重新啟動的工作執行將具有第一次重新啟動的 ID jr1234_restart_1。第二次重新啟動將jr1234_restart_2用於第二次重啟，依此類推。

您的重試嘗試不會因為重新啟動而受到影響。如果運行失敗，並且由於自動重試而啟動了新的運行，則重新啟動計數器將再次從 1 開始。例如，如果執行失敗jr_1234_attempt_3_restart_5，則自動重試將以 ID 開始新的執行：，jr_id1_attempt_4而當此嘗試在 7 天後重新啟動時，新的執行 ID 將會是jr_id1_attempt_4_restart_1。

資料遺失處理

在維護重新啟動期間， AWS Glue Streaming 會遵循一個程序，以確保上一個工作執行與重新啟動的工作執行之間的資料完整性和一致 請注意， AWS Glue 不能保證工作重新啟動之間的資料完整性和一致性，我們建議您考量架構以處理串流作業中的重複資料。

1. 偵測維護重新啟動狀況： AWS Glue 串流會監控指出何時觸發維護重新啟動的狀況，例如 7 天後到達維護時段，或者 14 天後需要硬重新啟動。
2. 呼叫正常終止：符合維護重新啟動條件時，「 AWS Glue 串流」會針對目前執行中的工作啟動正常終止程序。此程序包含下列步驟：
   a. 停止擷取新資料：串流工作會停止使用來自輸入來源的新資料 (例如，Kafka 主題、Kinesis 串流或檔案)。
   b. 處理擱置資料：工作會繼續處理任何已存在於其內部緩衝區或佇列中的資料。
   c. 提交偏移和檢查點：工作會將最新的偏移或檢查點提交至外部系統 (例如 Kafka、Kinesis 或 Amazon S3)，以確保重新啟動的任務可從先前工作中斷的位置繼續執行。
3. 重新啟動工作：正常終止程序完成後，「 AWS Glue 串流」會使用保留狀態和檢查點重新啟動工作。重新啟動的工作會從上次認可的偏移量或檢查點中挑選處理，確保不會遺失或複製任何資料。
4. 繼續資料處理：重新啟動的工作會從前一個工作中止的時間點繼續資料處理。它會繼續從輸入來源擷取新資料，從上次認可的偏移量或檢查點開始，並根據定義的 ETL 邏輯處理資料。

進階 AWS Glue 串流概念

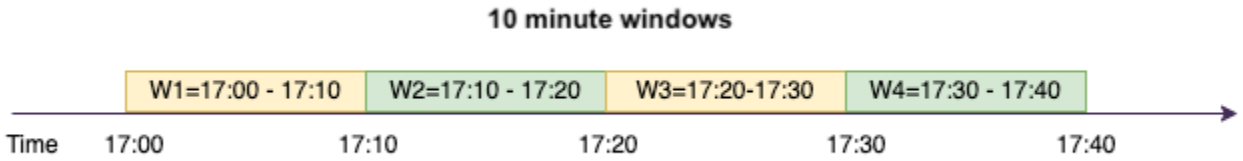
在當代以資料驅動的應用程式中，資料的重要性隨著時間推移而逐漸式微，其價值也從提供預測轉變為做出反應。因此，客戶希望能即時處理資料，以便更快地做出決策。處理即時資料饋送 (例如來自 IoT 感應器) 時，資料可能不會依序送達，或在擷取時由於網路延遲和其他來源相關的故障而導致處理延

遲。作為 AWS Glue 平台的一部分，AWS Glue 串流以這些功能為基礎，以提供可擴展的無伺服器串流 ETL，並由 Apache Spark 結構化串流提供支援，讓使用者能夠進行即時資料處理。

在本主題中，我們將探討 AWS Glue 串流的進階串流概念和功能。

處理串流時的時間考量

處理串流時有四種時間概念：



- Event-time：事件發生的時間。在大多數情況下，此欄位會內嵌在來源的 event-data 本身。
- E vent-time-window — 兩個事件時間之間的時間範圍。如上圖所示，W1 是一個 event-time-window 從下午 5 點到 17 點 10 分。每個都 event-time-window 是由多個事件組成的群組。
- Trigger-time：觸發時間可控制資料處理和更新結果的頻率。這是微批次處理開始的時間。
- Ingestion-time：將串流資料擷取至串流服務的時間。如果事件時間未嵌入至事件本身，在某些情況下，可將此時間用於衡量事件時段。

衡量事件時段

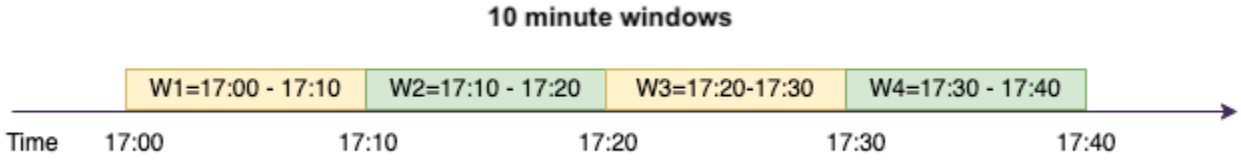
窗口是一種技術，您可以通過對多個事件進行分 event-time-window組和聚合。我們將透過以下範例探討衡量事件時段的好處，以及運用此技術的時機。

根據業務使用案例，Spark 支援三種時段類型。

- 翻滾窗口-一系列非重疊的固定大小 event-time-windows 在其上聚合。
- 滑動時段：從「長度固定」這點來看與輪轉時段類似，但只要滑動的時段小於事件時段本身，則時段可以重疊或滑動。
- 工作階段時段：從輸入資料事件開始，只要在間隙或閒置時接收到輸入資料，此時段就會持續延長。視資料輸入而定，工作階段時段的時段長度可以是靜態或動態。

輪轉時段

翻滾窗口是一系列非重疊的固定大小， event-time-windows 在其上聚合。讓我們以真實的範例來說明這一點。



ABC 汽車公司想要為一款新品牌的跑車進行行銷活動。他們想挑一個跑車迷人數最多的城市。為了達到這個目標，他們在網站上展示了一段介紹該車款的 15 秒簡短廣告。所有的「點擊」和相應的「城市」被記錄並流式傳輸到 Amazon Kinesis Data Streams。我們想算出在 10 分鐘的時段中的點擊次數，並依城市進行分組，以了解哪個城市的需求最高。以下為彙整的輸出結果。

window_start_time	window_end_time	城市	total_clicks
2023-07-10 17:00:00	2023-07-10 17:10	達拉斯	75
2023-07-10 17:00:00	2023-07-10 17:10	芝加哥	10
2023-07-10 17:20	2023-07-10 17:30	達拉斯	20
2023-07-10 17:20	2023-07-10 17:30	芝加哥	50

如上所述，這 event-time-windows 些與觸發時間間隔不同。舉例來說，即使觸發時間是每分鐘一次，輸出結果也只會顯示 10 分鐘的不重疊彙總時段。為了優化，最好將觸發間隔與 event-time-window.

在上表中，達拉斯在 17:00-17:10 這個時段共獲得 75 次點擊，芝加哥則有 10 次點擊。此外，任何城市在 17:10 - 17:20 這個時段都沒有資料，因此系統略過了這個時段。

現在，您可以透過下游分析應用程式對此資料進行進一步分析，以確定最適合進行行銷活動的城市。

在 AWS Glue 使用輪轉時段

1. 創建一個 Amazon Kinesis Data Streams DataFrame 並從中讀取。範例：

```
parsed_df = kinesis_raw_df \
                .selectExpr('CAST(data AS STRING)') \
```

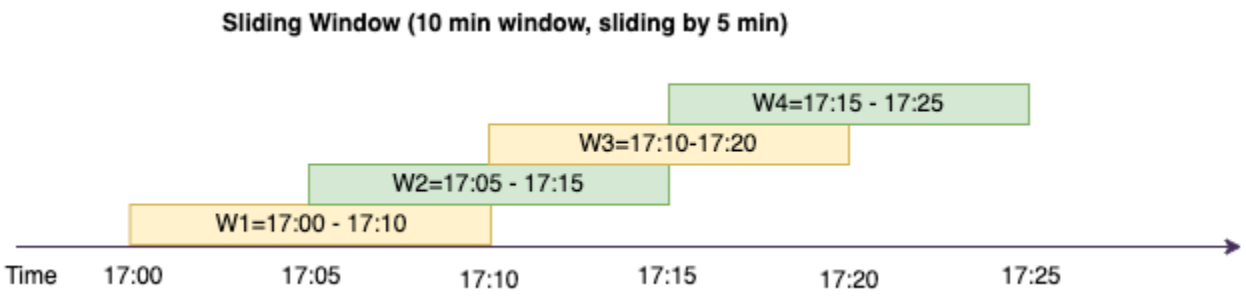
```
                .select(from_json("data", ticker_schema).alias("data")) \
                .select('data.event_time','data.ticker','data.trade','data.volume',
 'data.price')
```

2. 處理輪轉時段的資料。在下面的範例中，資料會根據 10 分鐘輪轉時段的輸入欄位「event_time」分組，並將輸出寫入到 Amazon S3 資料湖中。

```
grouped_df = parsed_df \
                .groupBy(window("event_time", "10 minutes"), "city") \
                .agg(sum("clicks").alias("total_clicks"))

summary_df = grouped_df \
                .withColumn("window_start_time", col("window.start")) \
                .withColumn("window_end_time", col("window.end")) \
                .withColumn("year", year("window_start_time")) \
                .withColumn("month", month("window_start_time")) \
                .withColumn("day", dayofmonth("window_start_time")) \
                .withColumn("hour", hour("window_start_time")) \
                .withColumn("minute", minute("window_start_time")) \
                .drop("window")

write_result = summary_df \
                .writeStream \
                .format("parquet") \
                .trigger(processingTime="10 seconds") \
                .option("checkpointLocation", "s3a://bucket-stock-stream/stock-
stream-catalog-job/checkpoint/") \
                .option("path", "s3a://bucket-stock-stream/stock-stream-catalog-
job/summary_output/") \
                .partitionBy("year", "month", "day") \
                .start()
```

滑動時段

從「長度固定」這點來看，滑動時段與輪轉時段類似，但只要滑動的時段小於事件時段本身，則時段可以重疊或滑動。由於可滑動的性質，輸入資料可以綁定至多個時段。



為了更清楚地了解這一點，讓我們舉個例子，來看一家想要找出疑似信用卡詐騙的銀行。串流應用程式可以監控連續串流的信用卡交易。這些交易可以每 10 分鐘的時段進行彙總，每 5 分鐘，時段就會向前滑動，將最久之前的 5 分鐘的資料消除，新增最近 5 分鐘的新資料。在每一個時段，可依國家/地區將交易分組，檢查是否存在可疑模式，例如在美國進行交易後，緊接著立即在澳洲進行另一筆交易。為了簡單起見，當總交易金額大於 $100 美元，我們就將該交易歸類為詐騙。一旦偵測到這類模式，系統就會發出疑似詐騙訊號，並可能將該信用卡凍結。

信用卡處理系統會針對每個信用卡識別碼和國家/地區，將一連串交易事件傳送給 Kinesis。 AWS Glue 工作會執行分析並產生下列彙總輸出。

window_start_time	window_end_time	card_last_four	國家/地區	total_amount
2023-07-10 17:00:00	2023-07-10 17:10	6544	美國	85
2023-07-10 17:00:00	2023-07-10 17:10	6544	澳洲	10
2023-07-10 17:05:45	2023-07-10 17:15:45	6544	美國	50
2023-07-10 17:10:45	2023-07-10 17:20:45	6544	美國	50
2023-07-10 17:10:45	2023-07-10 17:20:45	6544	澳洲	150

根據以上彙總資料，您可以看到 10 分鐘的時段每 5 分鐘滑動一次，並依交易金額進行加總。系統在 17:10-17:20 這個時段偵測到異常，該時段出現極端值，也就是一筆澳洲 150 美元的交易。AWS Glue

可偵測到此異常，並使用 boto3 將帶有違規密鑰的警報事件推送到 SNS 主題。此外，Lambda 函數可以訂閱此主題並採取行動。

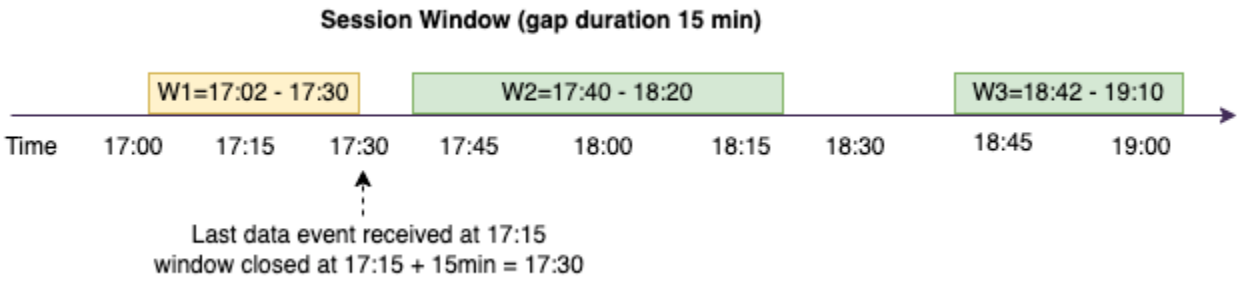
處理滑動時段的資料

group-by 子句和範圍函數可用於實作滑動時段，如下圖所示。

```
grouped_df = parsed_df \
            .groupBy(window(col("event_time"), "10 minute", "5 min"), "country",
 "card_last_four") \
            .agg(sum("tx_amount").alias("total_amount"))
```

工作階段時段

和前述兩種時段皆有固定的長度不同，工作階段時段的時段長度視資料輸入而定，可以是靜態或動態。工作階段時段從輸入資料事件開始，只要在間隙或閒置時接收到輸入資料，此時段就會持續延長。



讓我們舉個例子。ABC 飯店想要找出一星期中最繁忙的時間，以便為客人提供更好的優惠。一旦來賓簽入，會話窗口啟動和 spark 保持與聚合的狀態。 event-time-window每次客座端登記入住時，都會產生一個事件並將其傳送至 Amazon Kinesis Data Streams。酒店做出決定，如果在 15 分鐘內沒有辦理入住手續，則 event-time-window 可以關閉。當有新的入住手續時，下一個 event-time-window 將重新開始。輸出的資料如下所示。

window_start_time	window_end_time	城市	total_checkins
2023-07-10 17:02	2023-07-10 17:30	達拉斯	50
2023-07-10 17:02	2023-07-10 17:30	芝加哥	25
2023-07-10 17:40	2023-07-10 18:20	達拉斯	75

window_start_time	window_end_time	城市	total_checkins
2023-07-10 18:50:45	2023-07-10 19:15:45	達拉斯	20

第一次登記入住的事件時間發生在 17:02，聚合 event-time-window 將從 17:02 開始。只要系統在 15 分鐘內接收到事件，此彙總就會持續進行。在上面的例子中，最後收到的事件是在 17:15，接下來的 15 分鐘都沒有事件。其結果是，星火 event-time-window 在 17：15 + 15 分鐘 = 下午 5 點 30 分關閉，並將其設置為 17:02-17：30。當它收到新的簽入數據事件時，它 event-time-window 在 17:47 開始了一個新的。

處理工作階段時段的資料

`group-by` 子句和範圍函數可用於實作滑動時段。

```
grouped_df = parsed_df \
            .groupBy(session_window(col("event_time"), "10 minute"), "city") \
            .agg(count("check_in").alias("total_checkins"))
```

輸出模式

輸出模式是將無邊界表格的結果寫入外部連接器的一種模式，共有三種模式可用。在以下範例中，您要在每個微批次串流和處理資料行時，計算某個字的出現次數。

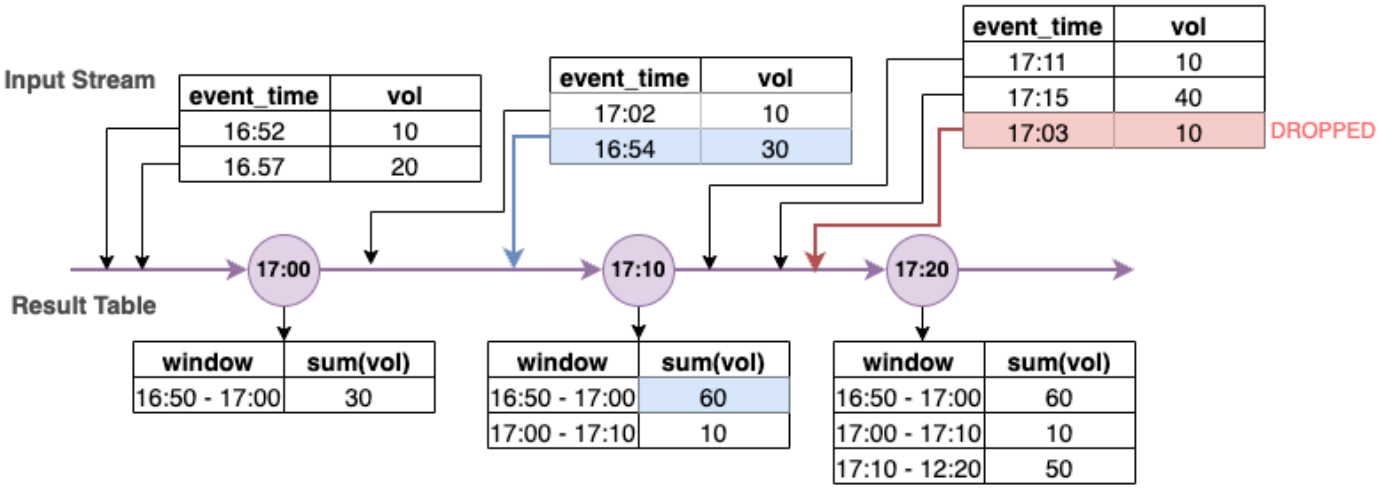
- 完整模式 — 即使目前未更新字數統計，每次微批次處理後，整個結果表仍會寫入接收器 event-time-window。
- 附加模式：此為預設模式。在此模式中，只有自上次觸發後新增至結果表格的新單字和/或新列才會寫入連接器。此模式適用於地圖、flatMap、篩選條件等查詢的無狀態串流。
- 更新模式：只有自上次觸發後更新或新增至結果表格的文字和/或列才會寫入連接器。

> **Note**
>
> 工作階段時段不支援「更新」輸出模式。

處理延遲資料和浮水印

使用即時資料時，由於網路延遲和上游故障，資料到達可能會有延遲，我們需要一種機制來對遺漏的資料再次執行彙總 event-time-window。但是，要做到這一點，需要維持狀態。同時，較舊的資料必須清理掉，以限制狀態的大小。Spark 2.1 版新增了對浮水印功能的支援，該功能可維持狀態，並可讓用戶指定延遲資料的閾值。

參考上面的股票代碼範例，讓我們假設允許的延遲資料閾值為不超過 10 分鐘。為了簡化情況，我們假設使用輪轉時段，股票代碼為 AMZ，交易為買入。



在上圖中，我們在計算 10 分鐘輪轉時段的總量。在 17:00、17:10 和 17:20 皆有觸發事件。在時間軸箭頭上方，顯示的是輸入資料串流，下方則是無邊界結果表。

在第一個 10 分鐘輪轉時段中，我們根據 event_time 進行彙總，計算出的 total_volume 為 30。在第二個 event-time-window，火花得到了事件時間 = 17：02 的第一個數據事件。由於這是 Spark 迄今為止可查看的 event_time 上限，因此浮水印閾值設為往回追溯 10 分鐘 (也就是 watermark_event_time = 16:52)。任何 event_time 在 16:52 之後的資料事件都會被視為有時限的彙總，而任何在此時間之前的資料事件則會被捨棄。這可讓 Spark 可以多維持 10 分鐘的中繼狀態，以容納延遲的資料。在時鐘時間大約 17:08 時，Spark 收到 event_time = 16:54 的事件，此時間正好落在閾值內。因此，火花重新計算了「16：50-17：00」 event-time-window ，總體積從 30 更新為 60。

但是，在 17:20 這個觸發時間，當 Spark 收到 event_time = 17:15 的事件時，其設定的 watermark_event_time = 17:05。因此，event_time = 17:03 的延遲資料事件便被認定為「延遲太久」，而被系統忽略。

```
Watermark Boundary = Max(Event Time) - Watermark Threshold
```

在 AWS Glue 使用浮水印

在超過浮水印邊界之前，Spark 不會將資料發出或寫入至外部連接器。若要在 AWS Glue 實作浮水印，請參閱以下範例。

```
grouped_df = parsed_df \
                .withWatermark("event_time", "10 minutes") \
                .groupBy(window("event_time", "5 minutes"), "ticker") \
                .agg(sum("volume").alias("total_volume"))
```

監控 AWS Glue 串流任務

監控串流任務是建置 ETL 管道的極為關鍵的一環。除了使用 Spark UI 之外，您還可以使 CloudWatch 用 Amazon 監視指標。以下是 AWS Glue 架構發出的串流指標清單。如需所有AWS Glue指標的完整清單，請參閱AWS Glue使用 Amazon CloudWatch 指標進行監控。

AWS Glue 使用結構化串流架構來處理輸入事件。您可以直接在程式碼中使用 Spark API，也可使用發佈這些指標的 `GlueContext` 所提供的 `ForEachBatch`。若要了解這些指標，我們必須先了解何謂 `windowSize`。

windowSize：`windowSize`是指您提供的微批次間隔。如果您指定的時間長度為 60 秒，AWS Glue 串流任務便會等待 60 秒 (如果 60 秒後上一個批次尚未完成，則會等待更長時間)，然後才會從串流來源讀取批次中的資料並套用 `ForEachBatch` 提供的轉換。此間隔也稱為「觸發間隔」。

讓我們更深入地查看指標，以了解運作狀態和效能特性。

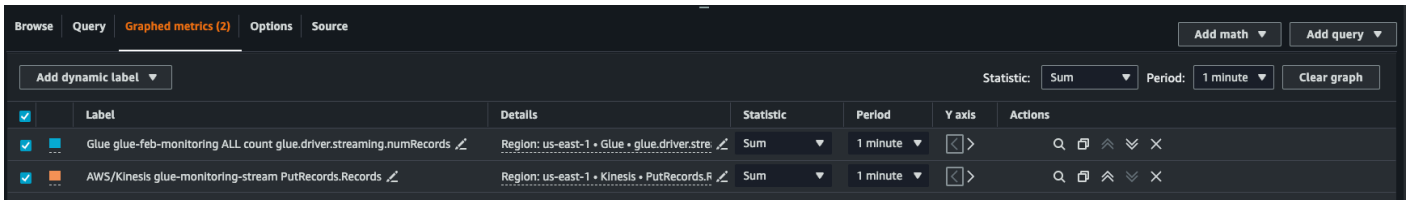
> **Note**
>
> 這些指標每 30 秒發出一次。如果您的 `windowSize` 小於 30 秒，則回報的指標會是彙總資料。舉例來說，假設您的 `windowSize` 是 10 秒，每個微批次固定處理 20 筆記錄。在這個案例中，為 numRecords 發出的指標值會是 60。
> 如果沒有可用的資料，則不會發出指標。此外，如果是取用者延遲指標，則必須啟用該功能才能取得其指標。

視覺化指標

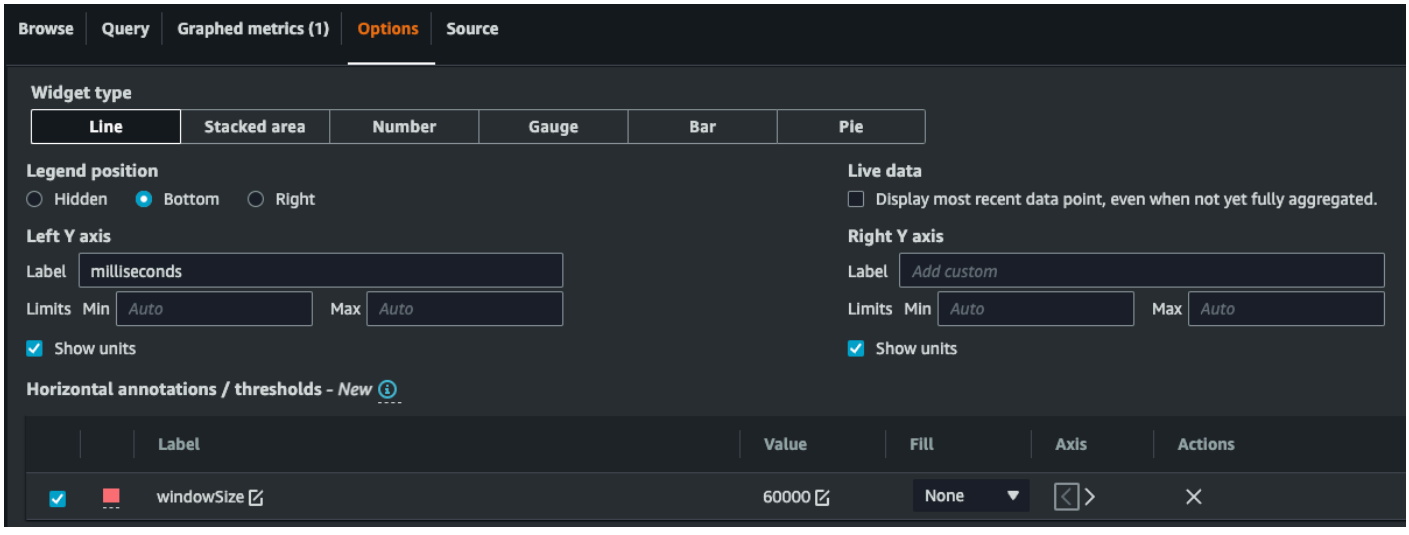
若要繪製視覺化指標：

1. 轉到 Amazon CloudWatch 控制台中的指標，然後選擇瀏覽選項卡。接著在「自訂命名空間」下選擇 Glue。

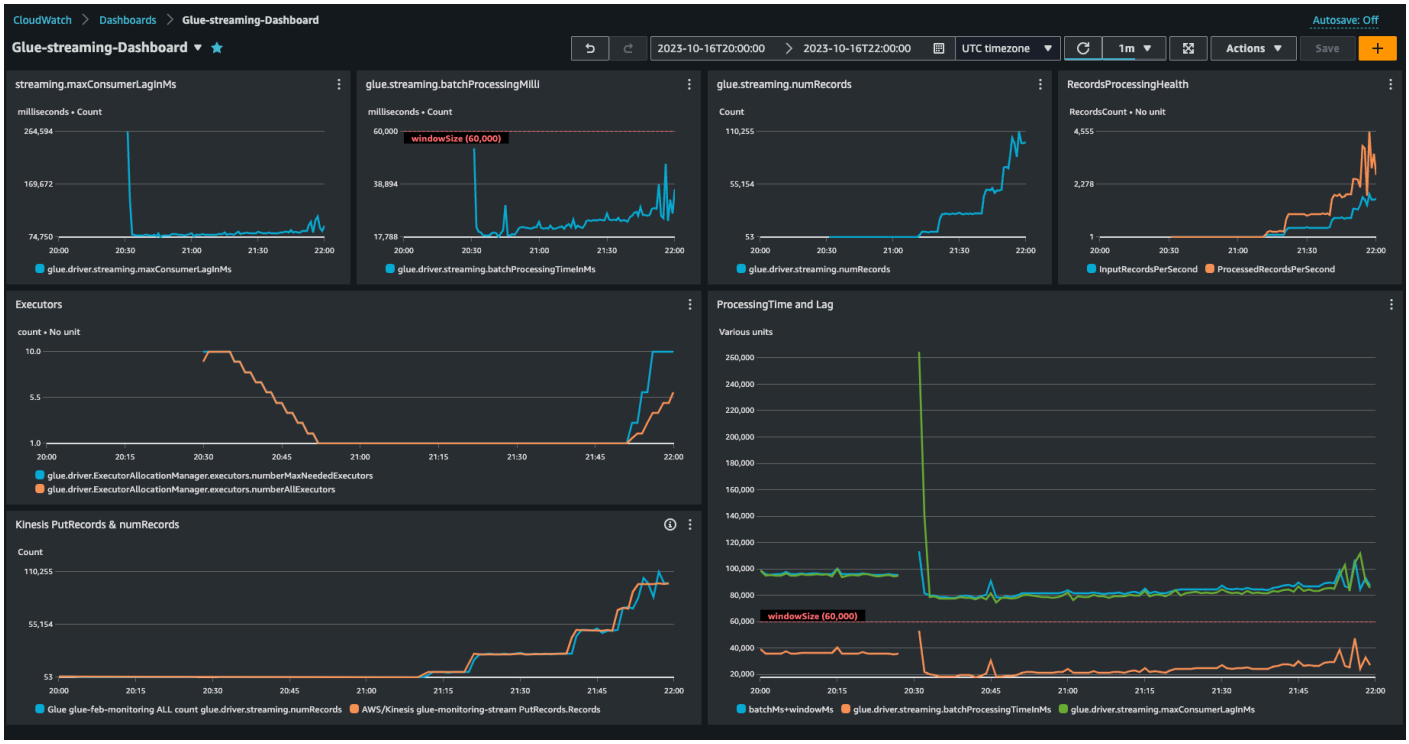
2. 選擇任務指標以顯示所有任務的指標。
3. 根據您的 JobName =glue-feb-monitoring ，然後 JobRunId =ALL 篩選指標。您可以按一下「+」符號 (如下圖所示)，將其新增至搜尋篩選條件。
4. 針對您感興趣的指標，選取其核取方塊。在下圖中，我們選取了 `numberAllExecutors` 和 `numberMaxNeededExecutors`。



5. 選取這些指標後，您可以前往圖表化指標索引標籤套用您的統計資料。
6. 由於指標每分鐘發出一次，您可以大於一分鐘的間隔為 `batchProcessingTimeInMs` 和 `maxConsumerLagInMs` 套用「平均值」。若為 `numRecords`，則可每分鐘套用一次「總和」。
7. 您可以使用選項索引標籤將 `windowSize` 水平註釋新增至圖表。



8. 選取指標後，請建立並新增儀表板。以下是範例儀表板。

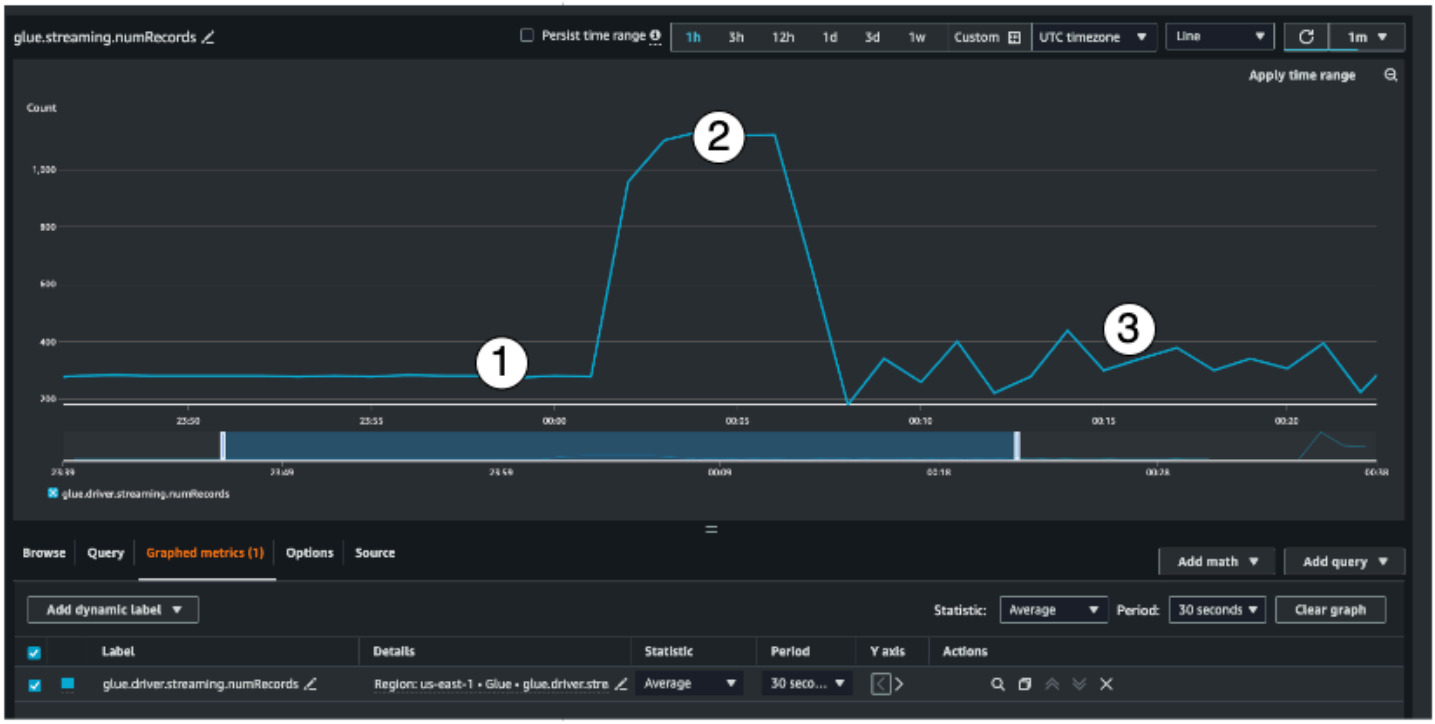


深入了解指標

本節說明各項指標，以及各指標間相互的關聯。

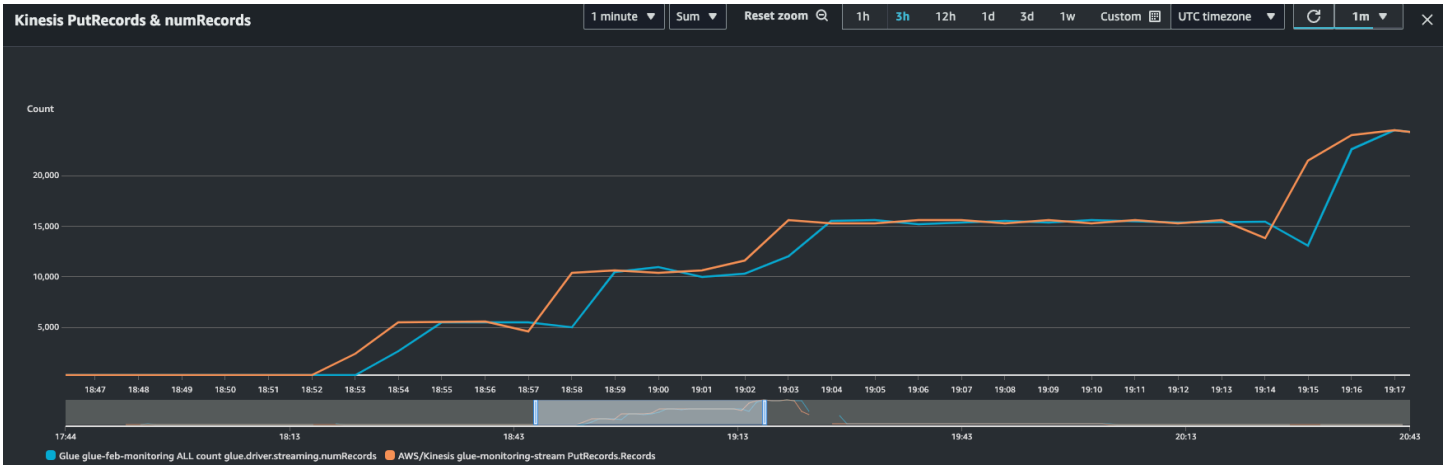
記錄數目 (指標：streaming.numRecords)

此指標指出正在處理的記錄數目。



此串流指標可讓您了解在一段時間內處理的記錄數目。除了正在處理的記錄數目外，這項指標還能幫助您了解輸入流量的行為。

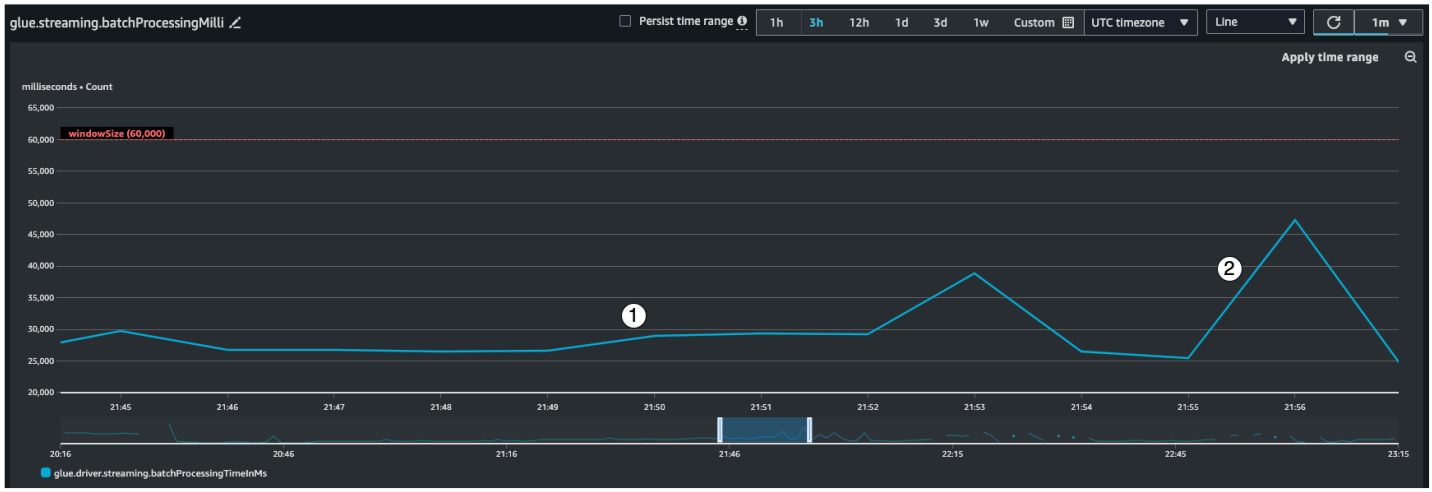
- 指標 #1 顯示了沒有任何暴增流量的穩定流量範例。一般來說，這會是像 IoT 感應器這類的應用程式，IoT 感應器會定期收集資料，並將資料傳送到串流來源。
- 指標 #2 顯示了在穩定的負載中突然出現暴增流量的範例。在遇到像黑色星期五之類的行銷活動，而出現點擊次數暴增時，點擊流應用程式就可能會發生這種情況
- 指標 #3 顯示了無法預測的流量範例。不可預知的流量並不意味著存在問題。這只是輸入資料的本質問題。回到 IoT 感應器的例子，您可以想像有數百個感應器正在向串流來源傳送天氣變化事件。由於天氣變化是不可預測的，資料當然也無法預測。了解流量模式是調整執行器大小的關鍵。如果輸入流量會在瞬間達到高峰，您可以考慮使用自動調整功能 (之後會詳細說明)。



您可以將此量度與 Kinesis PutRecords 量度結合使用，以確保擷取的事件數目和讀取的記錄數幾乎相同。當您試著要了解延遲情況時，這個做法更是格外有用。隨著擷取速率提高，AWS Glue 讀取的 numRecords 也會增加。

Batch 處理時間 (量度：串流。 batchProcessingTimeInMs)

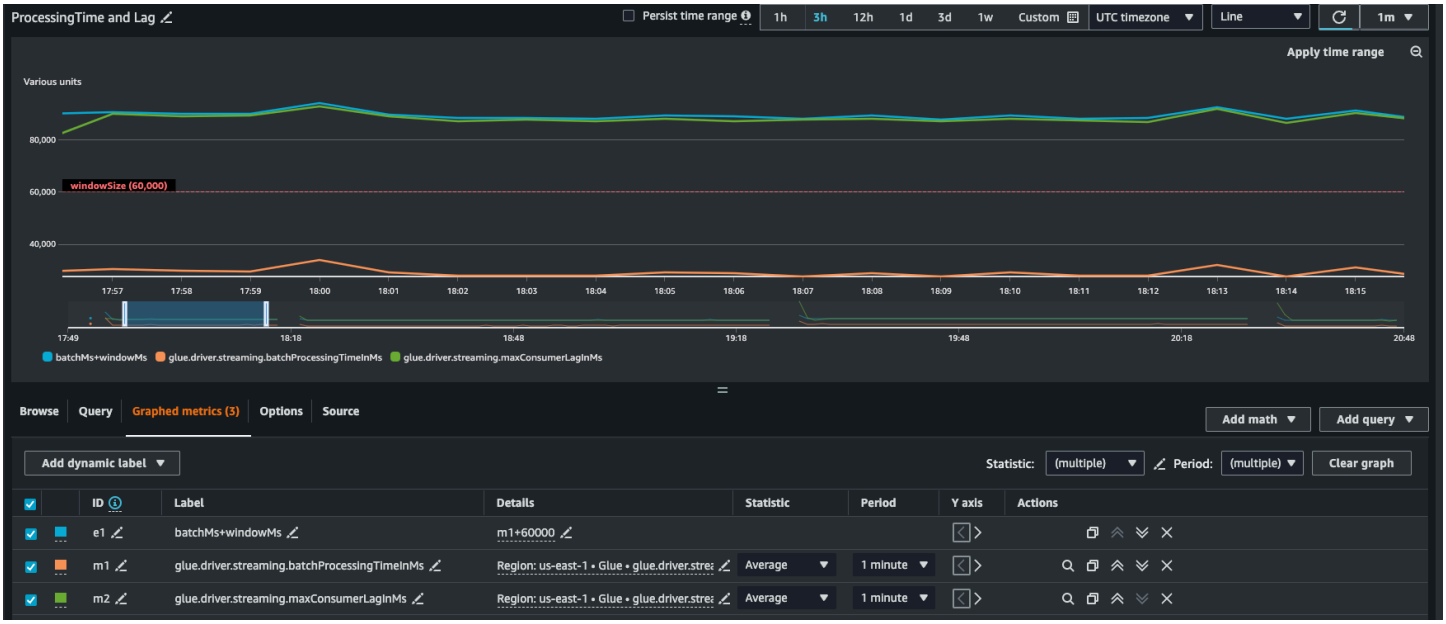
「批次處理時間」指標可協助您判斷叢集是佈建不足或過度佈建。



這項指標表示處理每個微批次記錄所需的毫秒數。這項指標的主要目標是監控時間，以確保時間小於 windowSize 間隔。只要 batchProcessingTimeInMs 在之後的時段間隔恢復正常，暫時的升高並沒有關係。指標 #1 顯示處理任務所花費較穩定或較不穩定的時間. 但是，如果輸入記錄的數量增加，則處理任務所需的時間也會增加 (如指標 #2 所示)。如果 numRecords 沒有增加，但處理時間卻上升，那麼您需要更深入地查看執行器所處理的任務。設定閾值和警示是個不錯的做法，可確保 batchProcessingTimeInMs 高出 120% 的時間不會超過 10 分鐘。如需有關設定鬧鐘的詳細資訊，請參閱使用 Amazon CloudWatch 警示。

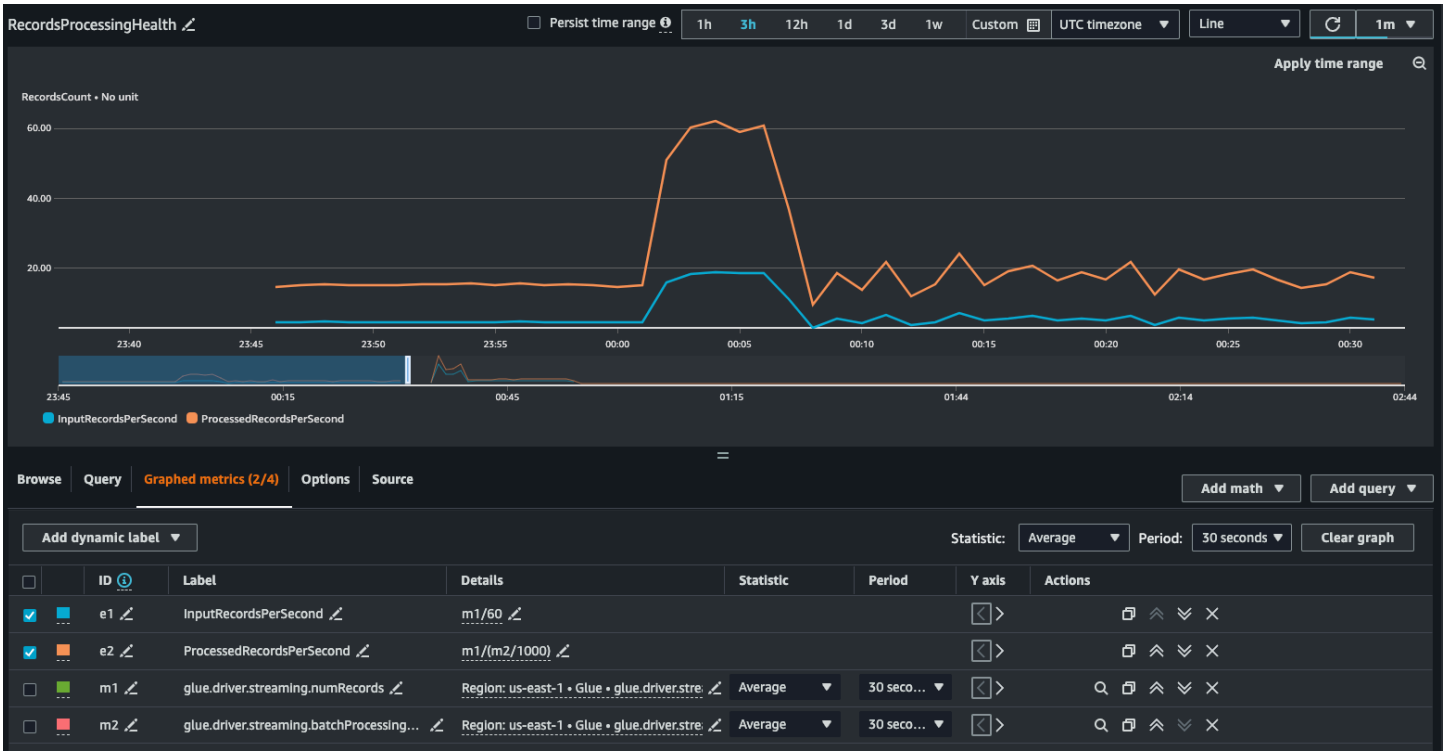
消費者滯後 (量度：串流。 maxConsumerLagInMs)

取用者延遲指標可協助您了解事件處理是否有延遲。如果延遲時間過高，即使您的 windowSize 正確，也可能會錯過業務所依賴的處理中 SLA。您必須使用 `emitConsumerLagMetrics` 連線選項明確啟用這項指標。如需詳細資訊，請參閱KinesisStreamingSourceOptions。



衍生指標

若要獲得更深入的見解，您可以建立衍生指標，以深入了解 Amazon 中的串流任務 CloudWatch。



您可以使用衍生指標建立圖形，以決定是否需要使用更多 DPU。雖然自動調整功能可協助您自動執行此作業，但您可以使用衍生指標來判斷自動調整功能是否有效運作。

- InputRecordsPerSecond表示您取得輸入記錄的速率。它派生如下：輸入記錄的數量（膠水驅動程序流。數字記錄）/。 WindowSize
- ProcessingRecordsPerSecond指出處理記錄的速率。它派生如下：輸入記錄的數量（膠水驅動程序流。數字記錄）/。 batchProcessingTime InMs

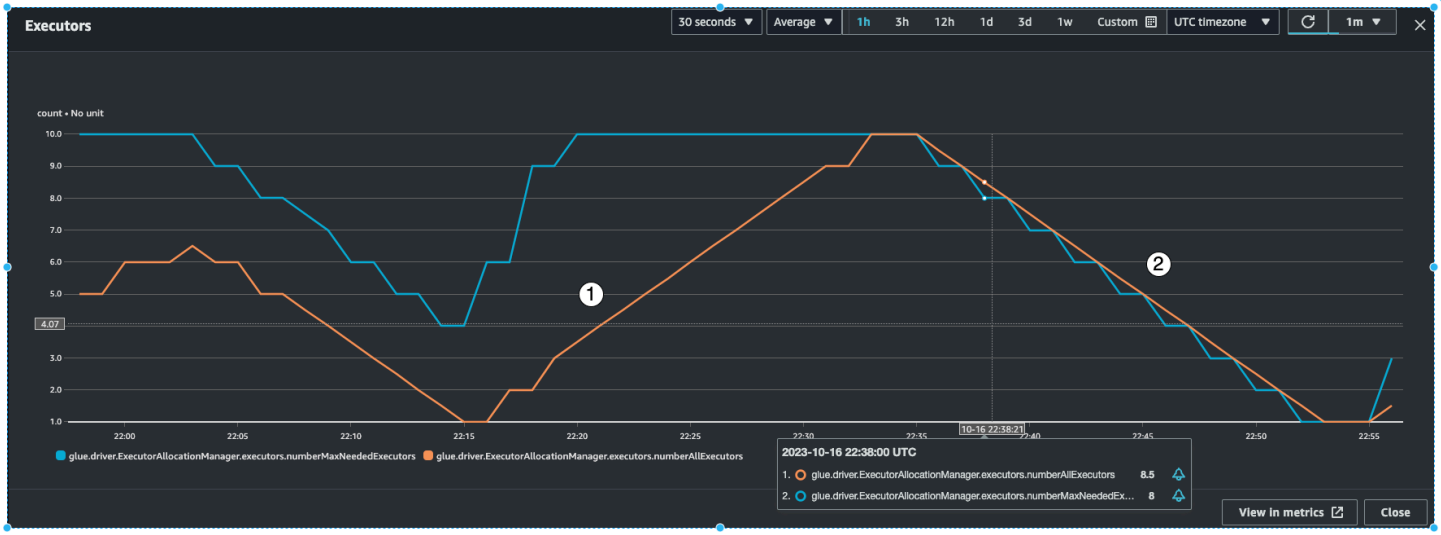
如果輸入速率高於處理速率，則可能需要新增更多容量來處理任務或增加平行處理。

自動調整指標

若您的輸入流量會突然衝高，您應該考慮啟用自動調整功能並指定工作者數目上限。啟用此功能後，您會額外獲得兩個指標，numberAllExecutors 和 numberMaxNeededExecutors。

- numberAllExecutors是主動執行工作執行者的數目
- numberMaxNeededExecutors 是滿足當前負載所需的最大數量（正在運行和待處理）作業執行程序的數量。

這兩個指標可協助您了解自動調整功能是否正常運作。



AWS Glue 會監控幾個微批次的 `batchProcessingTimeInMs` 指標，並執行以下其中一個動作。如果 `batchProcessingTimeInMs` 接近 `windowSize`，便會擴展執行器；如果 `batchProcessingTimeInMs` 相對來說低於 `windowSize`，則會縮減執行器。此外，它會使用演算法來逐步調整執行器。

- 指標 #1 顯示運作中的執行器如何擴展以趕上所需的執行器數目上限，以便處理負載。
- 指標 #2 顯示當 `batchProcessingTimeInMs` 過低，運作中的執行器會如何縮減。

您可以使用這些指標來監控目前的執行器層級平行處理原則，並據此調整自動調整組態中的工作者數目上限。

如何獲得最佳效能

Spark 會嘗試在 Amazon Kinesis 串流中為每個碎片建立一個任務以進行讀取。每個碎片中的資料即成為一個分割區。然後，它會根據每個工作者的核心數量 (每個工作者的核心數量取決於您選擇的工作者類型，`G.025X`、`G.1X` 等)，將這些任務分配給執行器/工作者。然而，任務的分配方式並無法確定。所有任務都會在各自的核心上平行執行。如果碎片超過可用執行器核心的數量，則會將任務排入佇列。

您可以合併使用上述指標和碎片數量，以佈建執行器提供穩定的負載，為突發流量預留空間。建議您為任務執行幾次反覆運算，以判斷工作者的大致數目。對於不穩定/突然達到尖峰的工作負載，您可以設定自動調整功能和工作者數目上限來完成相同的操作。

根據您業務的 SLA 要求來設定 `windowSize`。舉例來說，如果您的業務要求處理過的資料過時時間不能超過 120 秒，則您應將 `windowSize` 設定為至少 60 秒，以確保平均取用者延遲不超過 120 秒 (請參閱上述「取用者延遲」一節)。根據 `numRecords` 和碎片數量，您可以從這裡規劃 DPU 的容量，以確保您的 `batchProcessingTimeInMs` 在大多數時間皆小於 `windowSize` 的 70%。

> **Note**
>
> 熱碎片可能會導致資料扭曲，這表示有部分碎片/分割區比其他碎片大得多。這可能會導致某些平行執行的任務花費更長的時間，因而導致任務落後。因此，在前一批次的所有任務完成之前，無法開始下一批次，這將會使 `batchProcessingTimeInMillis` 和最大延遲受到影響。

AWS Glue 資料品質

AWS Glue 資料品質可讓您測量和監控資料品質，以便做出正確的業務決策。 AWS Glue 資料品質以開放原始碼 DeeQu 架構為基礎，提供受管理的無伺服器體驗。 AWS Glue 資料品質與資料品質定義語言 (DQDL) 搭配使用，此語言是您用來定義資料品質規則的網域特定語言。若要進一步了解 DQDL 和支援的規則類型，請參閱 資料品質定義語言 (DQDL) 參考。

如需了解產品詳細資訊和定價，請參閱 AWS Glue Data Quality 的服務頁面。

優點和重要功能

AWS Glue 資料品質的優點和主要功能包括：

- 無伺服器：無需安裝、修補或維護。
- 快速入門 — AWS Glue 資料品質可快速分析您的資料，並為您建立資料品質規則。只要按兩下即可開始使用：「建立資料品質規則 → 建議規則」。
- 偵測資料品質問題 — 使用機器學習 (ML) 偵測異常和 hard-to-detect 資料品質問題。
- 即興創作您的規則 — 從 25 個以上的 out-of-the-box DQ 規則開始，您可以建立符合您特定需求的規則。
- 評估品質並做出自信的業務決策：評估規則後，即可取得資料品質分數供您了解資料運作狀態。使用資料品質分數做出自信的業務決策。
- 零損壞資料 — 資 AWS Glue 料品質可協助您識別導致品質分數下降的確切記錄。輕鬆識別、隔離並修復這些記錄。
- 按用量付費 — 您不需要年度授權即可使用 AWS Glue 資料品質。
- 不受限制 — AWS Glue 資料品質建立在開放原始碼之上 DeeQu，可讓您以開放的語言保留正在編寫的規則。
- 資料品質檢查 — AWS Glue 資料品質您可以對Data Catalog和 AWS Glue ETL 管道執行資料品質檢查，讓您管理靜態和傳輸中的資料品質。
- 以 ML 為基礎的資料品質偵測 — 使用機器學習 (ML) 偵測異常和 hard-to-detect 資料品質問題。

運作方式

「 AWS Glue 資料品質」有兩個進入點： AWS Glue Data Catalog 和 AWS Glue ETL 工作。本節提供每個進入點支援的使用案例和 AWS Glue 功能的概觀。

資料品質 AWS Glue Data Catalog

AWS Glue 數據質量評估存儲在「 AWS Glue Data Catalog 它」中的對象為非編碼人員提供了一種簡單的方法來設置數據質量規則。這些人員角色包括資料管理員和業務分析師。

您可以針對下列使用案例選擇此選項：

- 您想要對已在 AWS Glue Data Catalog中分類的資料集執行資料品質任務。
- 您致力於資料控管，且需要持續識別或評估資料湖中的資料品質問題。

您可以使用下列介面來管理資料型錄的資料品質：

- AWS Glue 管理主控台
- AWS Glue API

若要開始使用「 AWS Glue 資料品質」， AWS Glue Data Catalog 請參閱開始使用適用於 Data Catalog 的 AWS Glue Data Quality。

AWS Glue ETL 工作的資料品質

AWS Glue AWS Glue ETL 任務的資料品質可讓您執行主動式資料品質工作。主動式任務可協助您在將資料集載入資料湖之前，識別並篩選出錯誤資料。

影片：介紹 ETL 管線的 AWS Glue 資料品質

您可以針對下列使用案例選擇適用於 ETL 任務的資料品質：

- 您想要將資料品質任務納入 ETL 任務
- 您想要撰寫在 ETL 指令碼中定義資料品質任務的程式碼
- 您想要管理在視覺化資料管道中流動的資料品質

您可以使用下列介面來管理適用於 ETL 任務的資料品質：

- AWS Glue Studio、 AWS Glue Studio 筆記本和 AWS Glue 互動式工作階段
- AWS Glue ETL 指令碼的程式庫
- AWS Glue API

若要開始使用適用於 ETL 任務的資料品質，請參閱《AWS Glue Studio 使用者指南》中的教學課程：開始使用 Data Quality。

比較資料型錄的資料品質與 ETL 任務的資料品質

此表格提供「 AWS Glue 資料品質」每個進入點支援的功能概觀。

功能	適用於資料型錄的資料品質	適用於 ETL 任務的資料品質
資料來源	Amazon S3、Amazon Redshift、與資料型錄相容的 JDBC 來源，以及 Apache Iceberg、Apache Hudi 和 Delta Lake 等交易資料湖格式。請注意，如果表是 AWS Lake Formation 受管理的，則不支援冰山、三角洲和 HUDI 資料表。 Amazon Athena 不支援在中編目 AWS Glue Data Catalog 的檢視表。	支援的所有資料來源 AWS Glue，包括自訂連接器和協力廠商連接器。
資料品質規則建議	支援	不支援
撰寫並執行 DQDL 規則	支援	支援
自動擴展	不支援	支援
AWS Glue 彈性支援	不支援	支援
排程	評估資料品質規則和使用 Step Functions 時支援。	使用 Step Functions 和工作流程時支援。
識別未通過資料品質檢查的記錄。	不支援	支援
整合 Amazon Eventbridge	支援	支援
與 AWS 雲觀察整合	支援	支援

功能	適用於資料型錄的資料品質	適用於 ETL 任務的資料品質
將資料品質結果寫入 Amazon S3	支援	支援
增量資料品質	透過下推述詞支援	透過 AWS Glue 書籤支援
AWS CloudFormation 支持	支援	支援
以 ML 為基礎的異常偵測	不支援	預覽版
動態規則	不支援	支援

考量事項

在使用「資 AWS Glue 料品質」之前，請考慮下列項目：

- 資料品質規則無法評估巢狀或清單類型的資料來源。請參閱壓平合併巢狀結構。

術語

下列清單定義了與 AWS Glue 資料品質相關的術語。

資料品質定義語言 (DQDL)

可用來撰寫 AWS Glue 資料品質規則的網域特定語言。

若要進一步了解 DQDL，請參閱 資料品質定義語言 (DQDL) 參考 指南。

資料品質

描述資料集如何達成其特定用途。 AWS Glue 「資料品質」會根據資料集評估規則，以測量資料品質。每個規則都會檢查特定特性，例如資料更新狀態或完整性。若要量化資料品質，您可以使用資料品質分數。

資料品質分數

當您使用「資料品質」評估規則集時，通過 (結果為 true) 的 AWS Glue 資料品質規則百分比。

規則

此即 DQDL 運算式，會檢查資料是否有特定特性並傳回布林值。如需詳細資訊，請參閱 規則結構。

analyzer

收集資料統計資料的 DQDL 表達式。分析器會收集 ML 演算法用來偵測異常和 hard-to-detect 資料品質問題的資料統計資料。

規則集

包含一組 AWS Glue 資料品質規則的資源。規則集必須與 AWS Glue Data Catalog中的資料表建立關聯。儲存規則集時， AWS Glue 會向規則集指派 Amazon Resource Name (ARN)。

資料品質分數

當您使用 AWS Glue Data Quality 評估規則集時，通過 (結果為 true) 的資料品質規則百分比。

觀察

AWS Glue 透過分析一段時間內從規則和分析器收集的資料統計資料，而產生的未經證實的洞察。

限制

AWS Glue 資料品質服務限制：

- 您可以在規則集中擁有 2000 個規則。如果您的規則集較大，我們建議您拆分為多個規則集。
- 規則集的大小為 65KB。如果您的規則集較大，我們建議您拆分為多個規則集。

AWS Glue 資料品質的版本說明

本主題說明「 AWS Glue 資料品質」中引入的功能。

正式推出：新功能

下列新功能適用於「 AWS Glue 資料品質」的正式推出：

- 現在支援識別哪些記錄失敗的資料品質檢查的功能 AWS Glue Studio
- 全新的資料品質規則類型，例如驗證兩個資料集之間的資料參照完整性、比較兩個資料集之間的資料，以及資料類型檢查
- 改善中的使用者體驗 AWS Glue Data Catalog

- 支援 Apache Iceberg、Apache Hudi 和 Delta Lake
- 支援 Amazon Redshift
- 使用 Amazon 簡化通知 EventBridge
- AWS CloudFormation 支援建立規則集
- 性能改進：ETL 中的緩存選項以及 AWS Glue Studio 在評估數據質量時更快的性能

2023 年 11 月 27 日 (預覽)

- 採用 ML 的異常偵測功能現在可在 AWS Glue ETL 和 AWS Glue Studio 中使用。有了這個功能，您現在可以偵測異常和 hard-to-detect 資料品質問題。
- 動態規則可讓您提供動態閾值 (例如：`RowCount> avg(last(10))`)。

2024 年 3 月 12 日

- DQDL 改善功能
  - Support 諸如空，空白，空白 _ 僅限關鍵字
  - 用於指定 AWS Glue 資料品質必須如何處理複合規則的選項
  - ColumnValues 規則類型將不允許在比較期間傳遞 NULL 值
  - Support DQDL 中的非運算子

2024年6月26日

- DQDL 改善功能
  - DQDL 現在支持 where 子句，以便您可以在應用 DQ 規則之前過濾數據

AWS Glue Data Quality 中的異常偵測

> **Note**
>
> AWS Glue Data Quality 可用於下列區域的預覽：
>
> - 美國東部 (俄亥俄，維吉尼亞北部)
> - 美國西部 (奧勒岡)

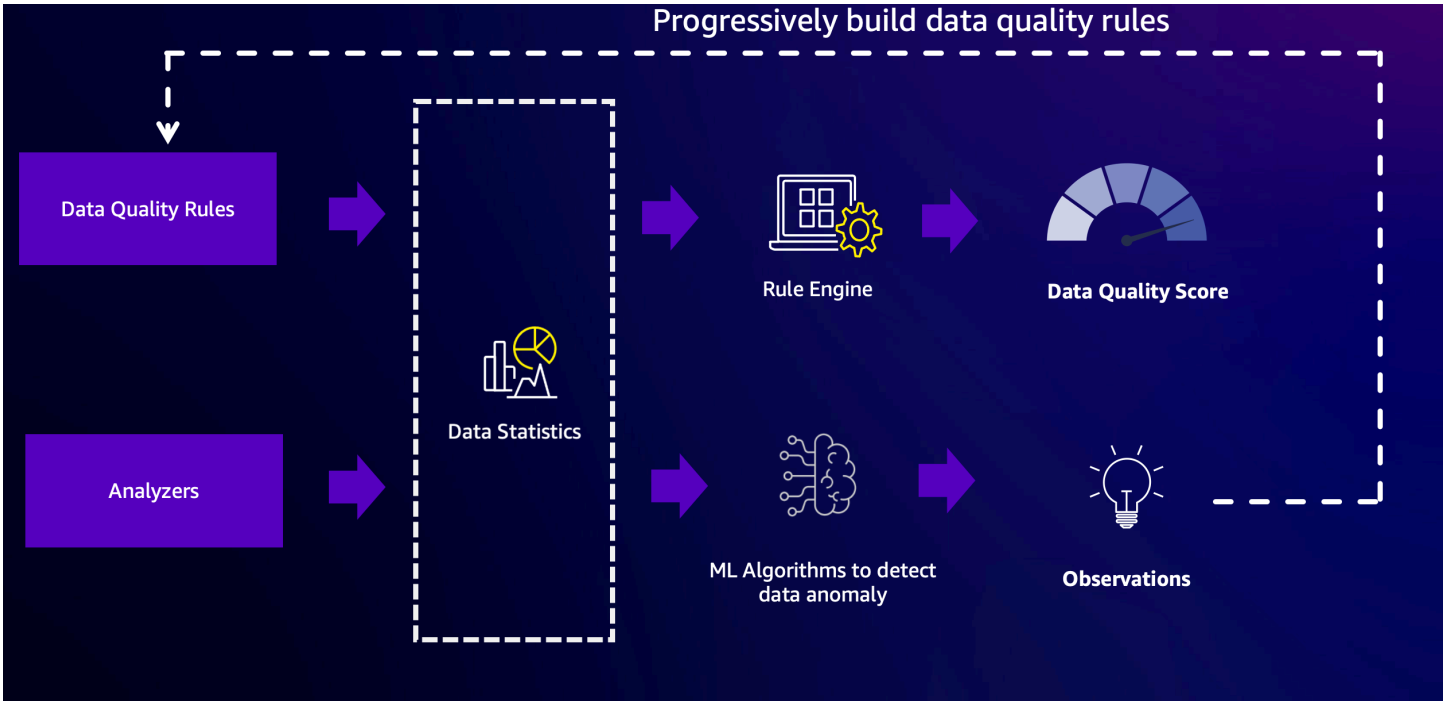
- 亞太區域 (東京)
- 歐洲 (愛爾蘭)

AWS Glue Data Quality 中的異常偵測功能會將機器學習 (ML) 演算法套用於一段時間內的資料統計資料，以偵測難以透過規則偵測的異常模式和隱藏的資料品質問題。目前，異常檢測功能僅適用於 AWS Glue 4.0。此功能目前僅適用於 AWS Glue Studio Visual ETL 和 AWS Glue ETL，此功能不適用於 AWS Glue Studio 筆記本、AWS Glue 資料目錄、AWS Glue 互動式工作階段和 AWS Glue 資料預覽。

運作方式

評估資料品質規則時，AWS Glue 會擷取判斷資料是否符合規則所需的資料統計資料。例如，Data Quality 會計算資料集中不同值的數目，然後將該值與期望值進行比較。

資料品質規則引擎會比較統計值與定義的閾值，並評估品質需求。由於會隨著時間的推移收集這些統計資料，因此您可以在 ETL 管道上啟用異常偵測功能，以便 AWS Glue 從過去的統計資料中學習，並將隱藏的模式報告為「觀察」。觀察是 AWS Glue ML 演算法識別的未經證實的洞察。其中隨附建議的資料品質規則，而您可以將這些規則套用至規則集來監控探索到的模式。我們建議定期 (例如每小時和每日) 執行作業。未定期執行可能會產生品質不佳的洞察。



使用分析器檢查資料

有時候，您可能沒有時間撰寫資料品質規則。這便是分析器發揮作用的時候。分析器是規則集的一部分，設定方式非常簡單。例如，您可以在規則集中寫入以下內容：

```
Analyzers = [
      RowCount,
      Completeness “AllColumns”
]
```

此時分析器會收集以下統計資料：

- 整個資料集的資料列計數
- 資料集中每個資料欄的完整性

我們建議您使用分析器，因為如此就不必擔心閾值問題。您可以執行資料管道，在執行三次之後，AWS Glue Data Quality 會在發現任何異常時開始產生觀察和規則建議。您可以檢閱觀察、相關統計資料，也可以輕鬆地將規則建議合併到規則集中。若要開始使用，請參閱 設定異常偵測功能並產生洞察 。請注意，分析器不會影響資料品質分數。隨著時間的推移，您可以對分析器產生的統計資料進行分析，從而產生觀察。

使用規 DetectAnomaly 則

有時候，您希望作業在偵測到異常時失敗。若要強制執行某個條件，您必須設定規則。分析器不會停止作業，而是會收集統計資料並分析資料。在規則集的規則區段中設定 `DetectAnomaly` 規則，將確認 DQ 掃描會報告：作業無法通過掃描中的所有規則。

異常偵測功能的優點和使用案例

工程師可以在任何給定時間管理數百個資料管道。每個管道都可以從不同來源擷取資料並將資料載入資料湖中。由於每個管道都可能從不同的來源擷取資料並將資料載入資料湖中，因此很難取得對資料的即時回饋：無論資料形狀是否有重大變化，還是其偏離了現有的趨勢。

過去，上游資料來源在資料工程團隊沒有警告的情況下發生變更，因此在此程序中引入了 hard-to-track「資料錯誤」。將 Data Quality 節點新增至作業可讓工作變得更加輕鬆，因為發現問題時作業就會失敗。但是，此舉並不會消除資料團隊擔心的所有失敗模式，這為其他資料錯誤的出現敞開大門。

一種失敗模式與資料磁碟區有關。隨著公司資料存放區隨著時間的推移而增長，資料管道產生的記錄數量可能會呈指數成長。資料團隊每週可能需要手動更新 ETL 作業來增加每個資料品質規則，這樣的規則會對擷取的資料列數目設下限制。

另一種失敗模式是，為適應交易量因星期幾而異的事實，某些資料品質規則限制非常寬鬆。週末幾乎沒有交易，週一的交易量比其他工作日多三倍。資料團隊有兩種選擇：實作邏輯以根據當天即時變更規則集，或者設定非常廣泛的期望值。

最後，資料團隊對也會擔憂定義不夠明確的資料錯誤。模型的訓練是針對具有特定特性的資料進行，如果這些模型開始以意想不到的方式出現偏差，團隊會想了解情況。例如，某公司可能會在二月份擴大到蒙大拿州，因此開始包含 MT 代碼的交易會更頻繁地出現。這可能會中斷 ML 推斷，導致模型錯誤地將每一筆蒙大拿州交易都預測為欺詐性交易。

此時，Data Quality 異常偵測功能有助於解決這些問題。Data Quality 異常偵測功能的優點包括：

- 以排程、事件驅動或手動方式掃描資料。
- 偵測可能表示意外事件、季節性事件或統計異常的異常情況。
- 提供規則建議，以依據 Data Quality 異常偵測功能發現的觀察採取行動。

這在以下情況下非常有用：

- 想在不需要寫入資料品質規則的情況下自動偵測資料異常。
- 想要捕捉資料中僅依靠資料品質規則無法發現的潛在問題。
- 想要自動化一些會隨著時間演進的任務，例如限制為資料品質監控而擷取的資料列數。

為 AWS Glue Data Quality 設定 IAM 許可

本主題提供的資訊將協助身為 IAM 管理員的您了解可在 AWS Glue Data Quality 的 AWS Identity and Access Management (IAM) 政策中使用的動作和資源。其中還包含範例 IAM 政策，具有搭配使用 AWS Glue Data Quality 與 AWS Glue Data Catalog 所需的最低許可。

如需 AWS Glue 中有關安全的額外資訊，請參閱 AWS Glue 中的安全性。

AWS Glue Data Quality 的 IAM 許可

下表列出使用者執行特定 AWS Glue Data Quality 操作所需的許可。若要設定 AWS Glue Data Quality 的精細授權，您可以在 IAM 政策陳述式的 `Action` 元素中指定這些動作。

AWS Glue Data Quality 動作

動作	描述	資源類型
glue:CreateDataQualityRuleset	准許建立資料品質規則集。	::dataQualityRuleset/<name>
glue:DeleteDataQualityRuleset	准許刪除資料品質規則集。	::dataQualityRuleset/<name>
glue:GetDataQualityRuleset	准許擷取資料品質規則集。	::dataQualityRuleset/<name>
glue:ListDataQualityRulesets	准許擷取所有資料品質規則集。	::dataQualityRuleset/*
glue:UpdateDataQualityRuleset	准許更新資料品質規則集。	::dataQualityRuleset/<name>
glue:GetDataQualityResult	准許擷取資料品質任務執行結果。	::dataQualityRuleset/<name>
glue:ListDataQualityResults	准許擷取所有資料品質任務執行結果。	::dataQualityRuleset/*
glue:CancelDataQualityRuleRecommendationRun	准許停止進行中的資料品質建議任務執行。	::dataQualityRuleset/*
glue:GetDataQualityRuleRecommendationRun	准許擷取資料品質建議任務執行。	::dataQualityRuleset/*
glue:ListDataQualityRuleRecommendationRuns	准許擷取所有資料品質建議任務執行。	::dataQualityRuleset/*
glue:StartDataQualityRuleRecommendationRun	准許啟動資料品質建議任務執行。	::dataQualityRuleset/*

動作	描述	資源類型
glue:CancelDataQualityRulesetEvaluationRun	准許停止進行中的資料品質任務執行。	::dataQualityRuleset/*
glue:GetDataQualityRulesetEvaluationRun	准許擷取資料品質任務執行。	::dataQualityRuleset/*
glue:ListDataQualityRulesetEvaluationRuns	准許擷取所有資料品質任務執行。	::dataQualityRuleset/*
glue:StartDataQualityRulesetEvaluationRun	准許啟動資料品質任務執行。	::dataQualityRuleset/<name>
glue:PublishDataQuality	准許發佈資料品質結果	::dataQualityRuleset/<name>

排程評估執行所需的 IAM 設定

IAM 許可

若要執行排程的資料品質評估執行，您必須將 IAM:PassRole 動作新增至許可政策。

AWS EventBridge 排程器所需許可

動作	描述	資源類型
iam:PassRole	授予 IAM 許可，以允許使用者傳遞核准的角色。	用於呼叫 StartDataQualityRulesetEvaluationRun 之角色的 ARN

如果沒有這些許可，會發生下列錯誤：

```
"errorCode": "AccessDenied"
```

```
"errorMessage": "User: arn:aws:sts::account_id:assumed-role/AWSGlueServiceRole is not
authorized to perform: iam:PassRole on resource: arn:aws:iam::account_id:role/service-
role/AWSGlueServiceRole
because no identity-based policy allows the iam:PassRole action"
```

IAM 受信任的實體

需要在受信任的實體中列出 AWS Glue 和 AWS EventBridge 排程器服務，才能建立和執行排程的 `StartDataQualityEvaluationRun`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "scheduler.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

範例 IAM 政策

AWS Glue Data Quality 的 IAM 角色需要下列類型的許可：

- AWS Glue Data Quality 操作許可，這樣您就可以取得建議的資料品質規則，並針對 AWS Glue Data Catalog 中的資料表執行資料品質任務。本節中的範例 IAM 政策包括 AWS Glue Data Quality 操作所需的最低許可。
- 授予 Data Catalog 資料表和基礎資料存取權的許可。這些許可視乎您的使用案例而有所差異。例如，針對您在 Amazon S3 中編目的資料，許可應包括對 Amazon S3 的存取權。

> **Note**
>
> 除了在本節中描述的許可之外，您還必須設定 Amazon S3 許可。

取得建議資料品質規則的最低許可

此範例政策包括產生建議資料品質規則所需的許可。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGlueRuleRecommendationRunActions",
      "Effect": "Allow",
      "Action": [
        "glue:GetDataQualityRuleRecommendationRun",
        "glue:PublishDataQuality",
        "glue:CreateDataQualityRuleset"
      ],
      "Resource": "arn:aws:glue:us-east-1:111122223333:dataQualityRuleset/*"
    },
 {
     "Sid": "AllowCatalogPermissions",
     "Effect": "Allow",
     "Action": [
        "glue:GetPartitions",
        "glue:GetTable"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "AllowS3GetObjectToRunRuleRecommendationTask",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::aws-glue-*"
    },
 {   // Optional for Logs
```

```
        "Sid": "AllowPublishingCloudwatchLogs",
        "Effect": "Allow",
        "Action": [
          "logs:CreateLogStream",
          "logs:CreateLogGroup",
          "logs:PutLogEvents"
        ],
        "Resource": "*"
    },
  ]
}
```

執行資料品質任務的最低許可。

此範例政策包括執行資料品質評估任務所需的許可。

視乎您的使用案例，以下政策陳述式為選用：

- `AllowCloudWatchPutMetricDataToPublishTaskMetrics` - 如果您想要將資料品質執行指標發佈至 Amazon CloudWatch，則為必要項目。
- `AllowS3PutObjectToWriteTaskResults` - 如果您想要將資料品質執行結果寫入 Amazon S3，則為必要選項。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGlueGetDataQualityRuleset",
      "Effect": "Allow",
      "Action": [
        "glue:GetDataQualityRuleset"
      ],
      "Resource": "arn:aws:glue:us-east-1:111122223333:dataQualityRuleset/<YOUR-
RULESET-NAME>"
    },
    {
      "Sid": "AllowGlueRulesetEvaluationRunActions",
      "Effect": "Allow",
      "Action": [
        "glue:GetDataQualityRulesetEvaluationRun",
        "glue:PublishDataQuality"
      ],
```

```
      "Resource": "arn:aws:glue:us-east-1:111122223333:dataQualityRuleset/*"
    },
 {
      "Sid": "AllowCatalogPermissions",
      "Effect": "Allow",
      "Action": [
        "glue:GetPartitions",
        "glue:GetTable"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "AllowS3GetObjectForRulesetEvaluationRun",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::aws-glue-*"
    },
    {
      "Sid": "AllowCloudWatchPutMetricDataToPublishTaskMetrics",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "cloudwatch:namespace": "Glue Data Quality"
        }
      }
    },
    {
      "Sid": "AllowS3PutObjectToWriteTaskResults",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject*"
      ],
      "Resource": "arn:aws:s3:::<YOUR-BUCKET-NAME>/*"
    }
 ]
```

```
}
```

開始使用適用於 Data Catalog 的 AWS Glue Data Quality

此 入門區段提供的說明可協助您開始在 AWS Glue 主控台上使用 AWS Glue Data Quality。您將學習如何完成基本任務，例如產生資料品質規則建議，以及根據資料評估規則集。

主題

必要條件

在使用 AWS Glue Data Quality 之前，您應該熟悉在 AWS Glue 中使用 Data Catalog 和爬蟲程式。使用 AWS Glue Data Quality 時，您可以評估 Data Catalog 資料庫中資料表的品質。您也需要下列項目：

- Data Catalog 中的資料表，用來評估資料品質規則集。
- 您在產生規則建議或執行資料品質任務時提供的 AWS Glue IAM 角色。此角色必須有權存取多個 AWS Glue Data Quality 程序代表您執行時所需的資源。這些資源AWS Glue包括 Amazon S3 和 CloudWatch. 若要檢視包括 AWS Glue Data Quality 最低許可的範例政策，請參閱 範例 IAM 政策。

  若要進一步了解 AWS Glue 的 IAM 角色，請參閱 Create an IAM policy for the AWS Glue service 和 Create an IAM role for the AWS Glue service。您也可以檢視 AWS Glue Data Quality 動作授權中資料品質專屬的所有 AWS Glue 許可的清單。
- 包含至少一個資料表的資料庫，其中包含各種資料。本教學課程中使用的資料表以 `yyz-tickets` 命名，內含資料表 `tickets`。此資料是多倫多市停車罰單公開提供的資訊集合。如果您建立自己的資料表，請確定已填入各種有效資料，以取得最佳建議規則集。

S 的tep-by-step 例子

如需範 step-by-step 例資料集的範例，請參閱 AWSGlue 資料品質部落格文章。

產生規則建議

規則建議可讓您輕鬆開始使用資料品質工具，而無需撰寫程式碼。您可以透過 AWS Glue Data Quality 分析資料、識別規則，並建立可在資料品質任務中評估的規則集。系統會在 90 天後自動刪除建議執行。

產生資料品質規則建議

1. 開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格中，選擇 Tables (資料表)。然後選擇您要為其產生資料品質規則建議的資料表。
3. 在資料表詳細資訊頁面上，選取資料品質索引標籤，以存取資料表的 AWS Glue Data Quality 規則和設定。
4. 在資料品質索引標籤上，選擇新增規則並監控資料品質。
5. 在規則集建置器頁面上，如果沒有規則建議執行，頁面頂端的提醒將提示您開始建議任務。
6. 選擇建議規則以開啟模態並輸入建議任務的參數。
7. 選擇可存取 AWS Glue 的 IAM 角色。此角色必須有權存取多個 AWS Glue Data Quality 程序代表您執行時所需的資源。
8. 根據偏好設定填寫欄位後，選擇建議規則以開始建議任務執行。如果建議執行正在進行中或已完成，您可以在此提醒中管理執行。您可能需要重新整理提醒，才能檢視狀態變更。已完成和進行中的建議任務執行會顯示在執行歷史記錄頁面中，其中會列出所有過去 90 天內的建議執行。

建議規則的含義

AWS Glue Data Quality 會根據輸入資料表中每一欄的資料產生規則。它使用規則來識別可能的邊界，在這些邊界中可以篩選資料以維持品質要求。下列已產生規則的清單包含的範例有助於了解規則的含義，以及在將規則套用至資料時可能會執行的動作。

如需產生之資料品質定義語言 (DQDL) 規則類型的完整清單，請參閱 DQDL rule type reference。

- `IsComplete "SET_FINE_AMOUNT"`：`IsComplete` 規則會驗證是否已針對任何指定資料列填入資料欄。使用此規則可在資料中將資料欄標記為非選用資料欄。

- `Uniqueness "TICKET_NUMBER" > 0.95`：Uniqueness 規則會驗證資料欄中的資料是否符合某些唯一性閾值。在此範例中，填入 "TICKET_NUMBER" 任何指定資料列的資料被判定為與所有其他資料列的內容最多有 95% 相同，如此便建議使用此規則。
- `ColumnValues "PROVINCE" in ["ON", "QC", "AB", "NY",...]`：ColumnValues 規則會根據現有資料欄內容定義資料欄的有效值。在此範例中，每一資料列的資料都是州或省的 2 個字母的車牌。
- `ColumnLength "INFRACTION_DESCRIPTION" between 15 and 31`：ColumnLength 規則會對資料欄的資料強制執行長度限制。此規則是根據字串資料欄的記錄長度下限和上限，從範例資料產生。

監控規則建議

執行資料品質規則建議時，新增規則並監控資料品質頁面會在頂端列中顯示資訊和您可以採取的其他動作。

規則建議正在進行時，您可以選擇在建議任務完成之前停止執行。任務正在進行中時，您會看到進行中狀態，以及執行開始的日期和時間。

規則建議完成時，規則建議列會顯示建議的規則數量、上次建議執行的狀態，以及完成的日期和時間戳記。

您可以選擇插入規則建議來新增建議的規則。若要檢視先前建議的規則，請選取特定日期。若要執行新建議，請選擇其他動作，然後選擇建議的規則。

選擇管理使用者設定以進行預設設定。您可以為 Amazon S3 設定預設路徑來存放規則集，或設定預設角色以執行資料型錄。

編輯建議的規則集

由於 AWS Glue Data Quality 會根據您現有的可用資料產生規則，您可能會在自動建議中看到一些非預期或不想要的規則。為了充分利用建議的規則集，您需要對其進行評估和修改。在本教學課程的此步驟中，您需採用上一步中產生的規則並對其進行調整，以對某些資料強制執行更嚴格的限制。您也可以放寬其他規則，以確保稍後可以新增正確、唯一的資料。

編輯建議的規則集

1. 在 AWS Glue 主控台中，選擇資料型錄，然後在導覽窗格中選擇資料庫資料表。選擇 `tickets` 資料表。

2. 在資料表詳細資料頁面上，選擇資料品質索引標籤，以存取資料表的 AWS Glue Data Quality 規則和設定。
3. 在規則集區段中，選取在 產生規則建議 中產生的規則集。
4. 選擇動作，然後在主控台視窗中選擇編輯。規則集編輯器會在主控台中載入。其中包括規則的編輯窗格和 DQDL 的快速參考。
5. 移除指令碼的第 2 行。如此可以放寬要求，即將資料庫大小限制在特定數量的資料列內。編輯後，檔案的第 1 至 3 行中應包含以下內容：

```
Rules = [
    IsComplete "TAG_NUMBER_MASKED",
    ColumnLength "TAG_NUMBER_MASKED" between 6 and 9,
```

6. 移除指令碼的第 25 行。如此放寬了 96% 的記錄省份為 ON 的要求。編輯後，檔案從第 24 行至規則集結尾應包含以下內容：

```
ColumnValues "PROVINCE" in ["ON", "QC", "AB", "NY", "AZ", "NS", "BC", "MI", "PQ",
 "MB", "PA", "FL", "SK", "NJ", "OH", "NB", "IL", "MA", "CA",
    "VA", "TX", "NF", "MD", "PE", "CT", "NC", "GA", "IN", "OR", "MN", "TN", "WI",
 "KY", "MO", "WA", "NH", "SC", "CO", "OK", "VT", "RI", "ME", "AL",
    "YT", "IA", "DE", "AR", "LA", "XX", "WV", "MT", "KS", "NT", "DC", "NV", "NE",
 "UT", "MS", "NM", "ID", "SD", "ND", "AK", "NU", "GO", "WY", "HI"],
ColumnLength "PROVINCE" = 2
]
```

7. 將第 14 行變更為以下內容：

```
IsComplete "TIME_OF_INFRACTION",
```

   如此可以將資料庫限制為僅包含記錄違規時間的票證，藉此加強對資料欄的要求。您應始終將缺少記錄違規時間的票證視為此資料集中的無效資料。這與分割或轉換可能更適合進一步使用或檢查資料以確定規則品質的情況不同。

8. 選擇主控台頁面底部的更新規則集。

建立新的規則集

規則集是您根據資料評估的一組資料品質規則。在 AWS Glue 主控台中，您可以使用資料品質定義語言 (DQDL) 撰寫自訂規則集。

建立資料品質規則集

1. 在 AWS Glue 主控台中，依序選擇資料型錄、資料庫，然後在導覽窗格中選擇資料表。選取資料表 `tickets`。
2. 開啟 Data quality (資料品質) 索引標籤。
3. 在規則集區段中，選擇建立規則集。DQDL 編輯器會在主控台中啟動。它具有可供直接編輯的文字區域，以及 DQDL 規則和資料表結構描述的快速參考。
4. 開始將規則新增至 DQDL 編輯器的文字區域。您可以直接從本教學課程撰寫規則，也可以使用資料品質規則編輯器的 DQDL 規則建置器功能進行撰寫。

> **Note**
>
> 如何使用 DQDL 規則建置器
>
> 1. 從清單中選取規則類型，然後選取加號，將範例語法插入編輯器窗格。
> 2. 使用您自己的欄名稱替換預留位置欄名稱。資料表中的資料欄名稱位於結構描述索引標籤中。
> 3. 視需要更新運算式參數。如需 DQDL 支援的表達式完整清單，請參閱表達式。

例如，下列規則是 `tickets` 資料表中 `ticket_number` 資料欄資料驗證的限制條件。若要新增下列規則，請使用 DQDL 規則建置器或直接編輯規則集：

```
IsComplete "ticket_number",
IsUnique "ticket_number",
ColumnValues "ticket_number" > 9000000000
```

5. 在規則集名稱欄位中輸入新規則集的名稱。
6. 選擇儲存規則集。

跨多個資料集評估資料品質

您可以使用 ReferentialIntegrity和 DatasetMatch 規則集，跨多個資料集設定資料品質規則。ReferentialIntegrity檢查主要資料集中的資料是否存在於其他資料集中。

若要新增參考資料集，請選擇結構描述索引標籤，然後選擇更新參考資料表。系統將提示您選取資料庫和資料表。您可以新增資料表，然後設定資料品質規則。規則類型 AggregateMatch，例如

RowCountMatch、 ReferentialIntegrity SchemaMatch、和 DatasetMatch 支援跨多個資料集執行資料品質檢查的能力。

執行規則集以評估資料品質

當您執行資料品質任務時，AWS Glue Data Quality 會根據資料評估規則集，並計算資料品質分數。此分數代表針對輸入內容通過的資料品質規則百分比。

執行資料品質任務

1. 在 AWS Glue 主控台中，依序選擇資料型錄、資料庫，然後在導覽窗格中選擇資料表。選取資料表 `tickets`。
2. 選擇資料品質索引標籤。
3. 在規則集清單中，選取您要根據資料表評估的規則集。在此步驟中，我們建議使用已撰寫或修改的規則集，而不是產生的規則。選擇執行。
4. 在模態中選擇 IAM 角色。此角色必須有權存取多個 AWS Glue Data Quality 程序代表您執行時所需的資源。您可以將 IAM 角色儲存為預設角色，或前往預設設定頁面進行修改。
5. 在「資料品質動作」 下，選擇是否要將指標發佈到 Amazon CloudWatch。選取此選項時，AWS Glue Data Quality 會發布指標，指出通過的規則數量和失敗的規則數量。若要對以這種方式儲存的指標採取動作，您可以使用 CloudWatch 警示。關鍵指標也會發布至 Amazon EventBridge，供您設定提醒。如需詳細資訊，請參閱設定提醒、部署和排程。
6. 在執行頻率中，選擇視需求執行或排程規則集。排程規則集時，系統會提示您輸入任務名稱。將在 Amazon EventBridge 中建立排程。您可以在 Amazon EventBridge 中編輯排程。
7. 若要將資料品質結果儲存在 Amazon S3 中，請選取資料品質結果位置。您先前為此任務選取的 IAM 角色必須具有此位置的寫入存取權。
8. 在其他組態下，輸入您想要 AWS Glue 為資料品質任務分配的請求的工作者數量。
9. 您可以選擇在資料來源中設定篩選條件。如此可協助您減少讀取的資料量。您也可以選取分割區資訊並透過 API 呼叫將增量驗證作為參數傳遞，以使用篩選條件執行增量驗證。您可以提供分割區述詞來改善效能。
10. 選擇執行。您應該會在 Data quality task runs (資料品質任務執行) 清單中看到新任務。當任務的執行狀態資料欄顯示為已完成時，您可以檢視品質分數結果。您可能需要重新整理主控台視窗，才能正確更新狀態。
11. 若要檢視資料品質結果詳細資訊的資料欄，請選擇 "+" 圖示以展開規則集。結果會顯示在評估中通過和失敗的規則，以及觸發規則失敗的原因。

檢視資料品質分數和結果

查看對所有已建立規則集的最新執行

1. 在 AWS Glue 主控台的導覽窗格中，選擇 Tables (資料表)。然後選擇您要為其執行資料品質任務的資料表。
2. 選擇資料品質索引標籤。
3. 資料品質快照顯示一段時間內執行的整體趨勢。依預設，會顯示對所有規則集最後 10 次的執行。若要依規則集篩選，請從下拉式清單中選取所需規則集。如果執行少於 10 次，則會顯示所有可用的已完成執行。
4. 在資料品質資料表中，會顯示每個具有最新執行 (如有) 的規則集及其分數。展開規則集會顯示該規則集中的規則，以及該執行的規則結果。

查看特定規則集上的最新執行

1. 在 AWS Glue 主控台的導覽窗格中，選擇 Tables (資料表)。然後選擇您要為其執行資料品質任務的資料表。
2. 選擇資料品質索引標籤。
3. 在資料品質資料表中，選擇特定規則集。
4. 在規則集詳細資訊頁面上，選擇執行歷史記錄索引標籤。

   此索引標籤內的資料表中會列出此特定規則集的所有評估執行。您可以查看分數的歷史記錄和執行狀態。
5. 若要查看特定執行的詳細資訊，請選擇執行 ID，以前往評估執行詳細資訊頁面。您可以在此頁面上查看有關執行的詳細資訊，以及個別規則結果狀態的更多詳細資訊。

相關主題

- DQDL 規則類型參考
- 資料品質定義語言 (DQDL) 參考

使用 AWS Glue Studio 評估資料品質

AWS Glue Data Quality 會根據您定義的規則來評估和監控資料品質。這可以讓您很容易地識別需要採取行動的資料。在 AWS Glue Studio 中，您可以將資料品質節點新增至視覺化任務，以便在資料目錄的資料表上建立資料品質規則。然後，您可以監控和評估資料集隨著時間推移發生的變化。如需有關如何在 AWS Glue Studio 中使用 AWS Glue Data Quality 的概觀，請參閱下列影片。

以下是如何使用 AWS Glue Data Quality 的高階步驟：

1. 建立資料品質規則：透過選擇您設定的內建規則集，使用 DQDL 建置器建立一組資料品質規則。
2. 設定資料品質任務：根據資料品質結果和輸出選項定義動作。
3. 儲存並執行資料品質任務：建立並執行任務。儲存任務將儲存您為任務建立的規則集。
4. 監控和檢閱資料品質結果：在任務執行完成後檢閱資料品質結果。您可以選擇將任務排定在未來日期。

優勢

資料分析師、資料工程師和資料科學家可以使用 AWS Glue Studio 中的評估資料品質節點來分析、設定、監控和改善視覺化任務編輯器中的資料品質。使用資料品質節點的好處包括：

- 偵測資料品質問題：您可以透過建立可檢查資料集特性的規則來檢查問題。
- 易於上手：您可以從預先建置的規則和動作開始。
- 緊密整合：您可以在 AWS Glue Studio 中使用資料品質節點，因為 AWS Glue Data Quality 在 AWS Glue Data Catalog 上執行。

在 AWS Glue Studio 中評估 ETL 任務的資料品質

在本教學課程中，您可開始使用 AWS Glue Studio 中的 AWS Glue Data Quality。您將學到如何：

- 使用資料品質定義語言 (DQDL) 規則建置器建立規則。
- 指定資料品質動作、要輸出的資料，以及資料品質結果的輸出位置。
- 檢閱資料品質結果。

若要以範例進行練習，請檢閱部落格文章 Getting started with AWS Glue Data Quality for ETL pipelines。

步驟 1：將「評估資料品質」轉換節點新增至視覺化任務

在此步驟中，您要將評估資料品質節點新增到視覺化任務編輯器。

新增資料品質節點

1. 在 AWS Glue Studio 主控台中，從建立任務區段選擇具有來源和目標的視覺化，然後選擇建立。
2. 選擇您要套用資料品質轉換的節點。一般而言，這將是轉換節點或資料來源。
3. 透過選擇 "+" 圖示開啟左側的資源面板。然後在搜尋列中輸入評估資料品質，接著從搜尋結果中選擇評估資料品質。
4. 視覺化任務編輯器將顯示從您選取的節點分支的評估資料品質轉換節點。在主控台右側，轉換索引標籤將自動開啟。如果需要變更父節點，請選擇節點屬性索引標籤，然後從下拉式選單中選擇父節點。

   選擇新的父節點時，會在父節點與評估資料品質節點之間建立新的連線。移除任何不需要的父節點。只有一個父節點可以連接到一個評估資料品質節點。
5. 「評估資料品質」轉換支援多個父項，因此您可以驗證多個資料集的資料品質規則。支援多個資料集的規則包括 ReferentialIntegrity、DatasetMatch、SchemaMatch、RowCountMatch 和 AggregateMatch。

   當您將多個輸入新增至「評估資料品質」轉換時，您需要選取「主要」輸入。您的主要輸入是您想要驗證資料品質的資料集。所有其他節點或輸入都被視為參考。

   您可以使用「評估資料品質」轉換來識別未通過資料品質檢查的特定記錄。建議您選擇主資料集，因為標示錯誤記錄的新資料欄會新增至主資料集。
6. 您可以為輸入資料來源指定別名。當您使用 ReferentialIntegrity 規則時，別名會提供另一種參考輸入來源的方式。因為只能將一個資料來源指定為主要來源，您新增的每個其他資料來源都需要別名。

   在下列範例中，ReferentialIntegrity 規則會依別名指定輸入資料來源，並與主要資料來源執行一對一的比較。

```
Rules = [
 ReferentialIntegrity “Aliasname.name” = 1
]
```

步驟 2：使用 DQDL 建立規則

在此步驟中，您要使用 DQDL 建立規則。在本教學課程中，您將使用完整度規則類型建立單一規則。此規則類型會根據指定運算式檢查資料欄中完整 (非 Null) 值的百分比。如需有關使用 DQDL 的詳細資訊，請參閱 DQDL。

1. 在轉換索引標籤中，選擇插入按鈕以新增規則類型。這會將規則類型新增至規則編輯器，您可以在其中輸入規則的參數。

   > **Note**
   >
   > 編輯規則時，請確保規則位於括號內，且以逗號分隔規則。例如，完整的規則運算式如下所示：
   >
   > ```
   > Rules= [
   >     Completeness "year">0.8, Completeness "month">0.8
   > ]
   > ```
   >
   > 此範例指定名為 'year' 和 'month' 資料欄的完整度參數。這些資料欄必須大於 80%「完整」，或各相應資料欄中 80% 以上的執行個體都有資料，才能通過規則。

   在此範例中，搜尋並插入完整度規則類型。這會將規則類型新增至規則編輯器。此規則類型的語法如下：`Completeness <COL_NAME> <EXPRESSION>`。

   大多數規則類型都需要您提供運算式作為參數，以建立布林值回應。如需有關支援的 DQDL 運算式的詳細資訊，請參閱 DQDL 運算式。接著新增資料欄名稱。
2. 在 DQDL 規則建置器中，按一下結構描述索引標籤。使用搜尋列尋找輸入結構描述中的欄名稱。輸入結構描述會顯示欄名稱和資料類型。
3. 在規則編輯器中按一下規則類型右側，將游標插入要插入資料欄的位置。或者，您也可以在規則中輸入資料欄的名稱。

   例如，在輸入結構描述清單中的資料欄清單中，選擇資料欄 (在此範例中為年) 旁的插入按鈕。這會將資料欄新增至規則。
4. 然後，在規則編輯器中新增運算式來評估規則。由於完整度規則類型會根據指定運算式檢查欄中完整 (非 Null) 值的百分比，請輸入運算式，例如 `> 0.8`。此規則將檢查資料欄是否大於 80% 的完整 (非 Null) 值。

步驟 3：設定資料品質輸出

建立資料品質規則後，即可選取其他選項以指定資料品質節點輸出：

1. 在資料品質轉換輸出中，從以下選項中選擇：

   - 原始資料：選擇輸出原始輸入資料。當您選擇此選項時，任務中會新增一個新的子節點 “rowLevelOutcomes”。其結構描述與作為輸入傳遞至轉換的主資料集結構描述相符。如果您只想在品質問題發生時傳遞資料並讓任務失敗，則此選項非常有用。

     另一個使用案例是當您想要偵測未通過資料品質檢查的錯誤記錄時。若要偵測錯誤記錄，請選擇新增資料欄以指出資料品質錯誤選項。此動作會將四個新資料欄新增至 “rowLevelOutcomes” 轉換的結構描述。

     - DataQualityRulesPass (字串陣列)：提供通過資料品質檢查的規則陣列。
     - DataQualityRulesFail (字串陣列)：提供未通過資料品質檢查的規則陣列。
     - DataQualityRulesSkip (字串陣列)：提供已略過的規則陣列。下列規則無法識別錯誤記錄，因為它們是在資料集層級套用。
       - AggregateMatch
       - ColumnCount
       - ColumnExists
       - ColumnNamesMatchPattern
       - CustomSql
       - RowCount
       - RowCountMatch
       - StandardDeviation
       - Mean
       - ColumnCorrelation
     - DataQualityEvaluationResult：在資料列層級提供「已通過」或 「未通過」狀態。請注意，您的整體結果可能是未通過，但某個記錄可能會通過。例如，RowCount 規則可能已失敗，但所有其他規則可能已成功。在這種情況下，此欄位狀態為「已通過」。
2. 資料品質結果：選擇輸出已設定的規則及其通過或失敗狀態。如果您想要將結果寫入 Amazon S3 或其他資料庫，此選項非常有用。
3. 資料品質輸出設定 (選用)：選擇資料品質輸出設定，以顯示資料品質結果位置欄位。然後選擇瀏覽，以搜尋要設定為資料品質輸出目標的 Amazon S3 位置。

步驟 4. 設定資料品質動作

動作可讓您根據特定條件將指標發布至 CloudWatch 或停止任務。動作只有在您建立規則之後才可用。當您選擇此選項時，相同的指標也會發布至 Amazon EventBridge。您可以使用這些選項建立通知警示。

- 規則集失敗時：如果規則集在任務執行時失敗，您可以選擇該怎麼做。如果您希望任務在資料品質失敗時失敗，請選取下列其中一個選項來選擇任務失敗的時間。預設不會選取此動作，即使資料品質規則失敗，任務也會完成執行。
  - 無：如果選擇 無 (預設值)，即使規則集失敗，任務也不會失敗，而會繼續執行。
  - 將資料載入目標後任務失敗：任務失敗且不儲存任何資料。若要儲存結果，請選擇要儲存資料品質結果的 Amazon S3 位置。
  - 任務失敗而不載入至目標資料：發生資料品質錯誤時，此選項會立即讓任務失敗。它不會載入任何資料目標，包括資料品質轉換的結果。

步驟 5：檢視資料品質結果

執行任務後，選擇資料品質索引標籤來檢視資料品質結果。

1. 檢視每個任務執行的資料品質結果。每個節點都會顯示資料品質狀態和狀態詳細資訊。選擇節點以檢視所有規則和每個規則的狀態。
2. 選擇下載結果以下載 CSV 檔案，其中包含有關任務執行和資料品質結果的資訊。
3. 如果您執行多個具有資料品質結果的任務，則可以依日期和時間範圍篩選結果。選擇依日期和時間範圍篩選以展開篩選條件視窗。
4. 選擇相對範圍或絕對範圍。如要使用絕對範圍，請使用行事曆選取日期，並輸入開始時間和結束時間的值。完成時，請選擇套用。

Data Quality 規則建置器

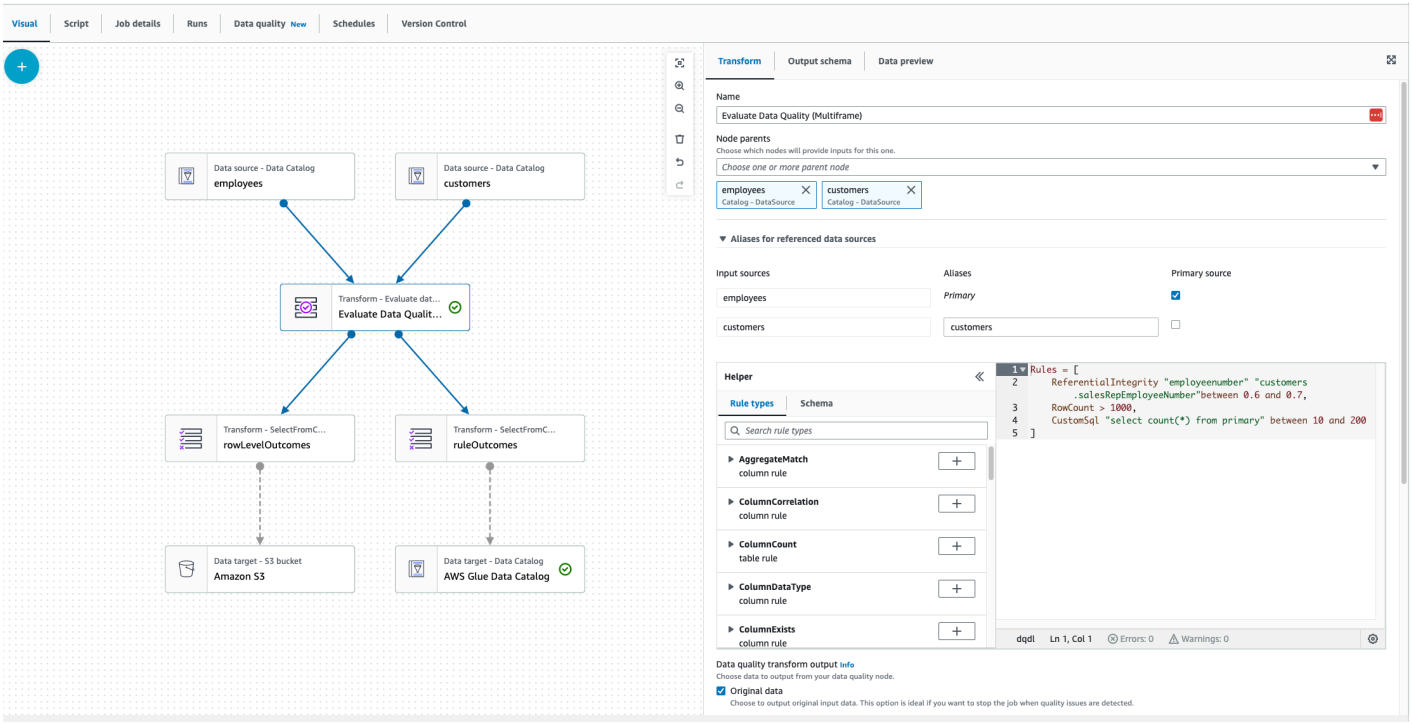
使用資料品質定義語言 (DQDL) 規則建置器，您可以建立資料品質規則來評估資料。先選取規則類型，然後在規則編輯器中指定參數。規則編輯器也會在您建立規則時顯示任何錯誤和警告。

DQDL 指南提供有關如何使用 DQDL 語法、內建規則類型和範例來建構規則的完整文件。

評估資料品質節點

使用評估資料品質轉換節點和 DQDL 規則建置器時，您可以展開工作空間。

- 若要展開轉換索引標籤以填滿整個畫面，請選擇節點詳細資訊面板右上角的展開圖示。
- 若要展開 DQDL 規則編輯器，請選擇 << 圖示以展開規則編輯器及收合規則類型和結構描述索引標籤。



元件

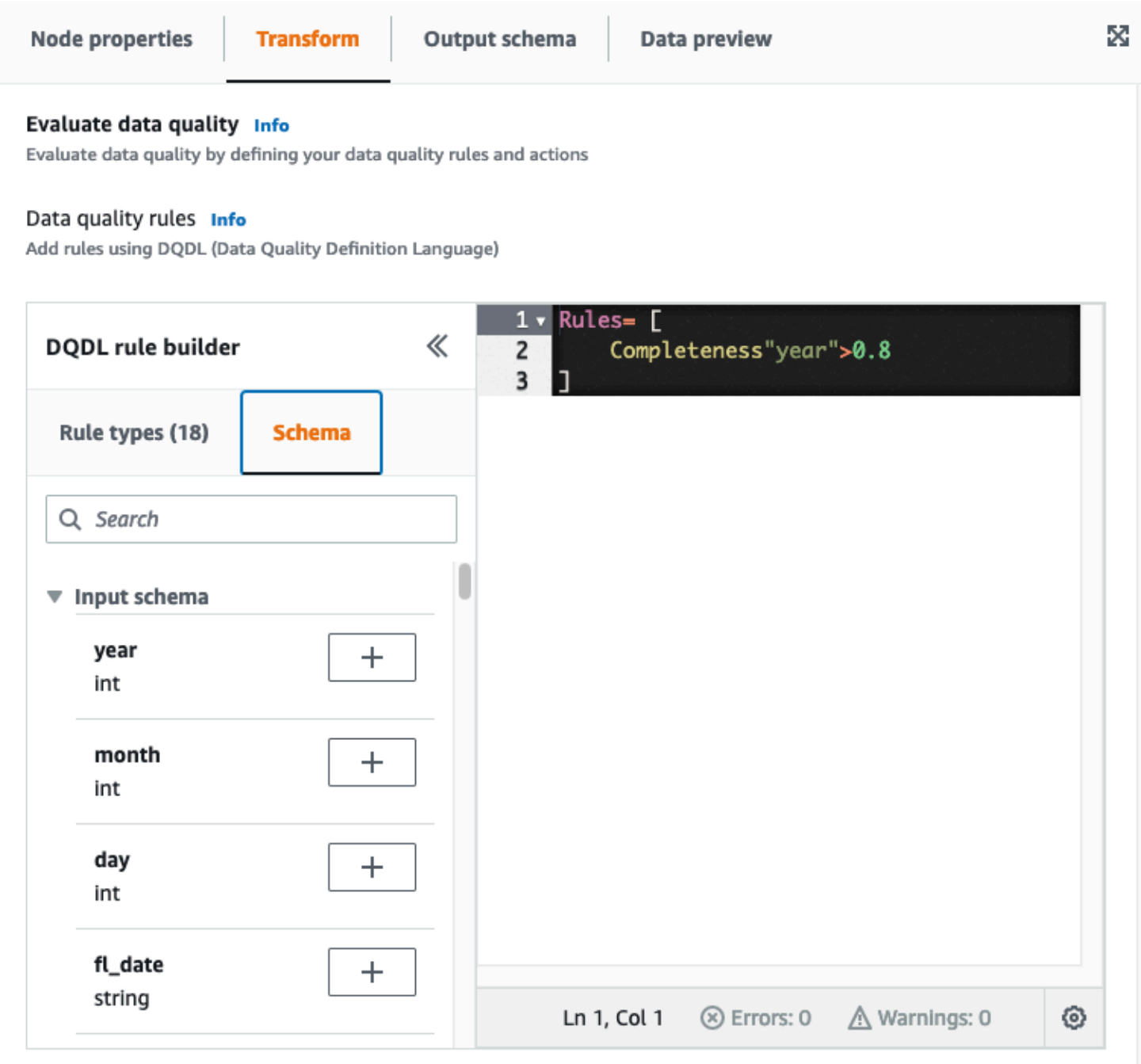
AWS Glue Studio 內建了 26 種規則類型。每種規則類型都有描述和使用方式的範例。

資料品質規則類型

AWS Glue Studio 提供內建規則類型，以便於建立規則。如需規則類型的詳細資訊，請參閱 DQDL rule type reference (DQDL 規則類型參考)。

結構描述

Schema (結構描述) 索引標籤會顯示父節點的欄名稱和資料類型。隨即會顯示多個節點的結構描述。您可以檢視輸入結構描述、依欄名稱搜尋，以及將欄插入規則編輯器。



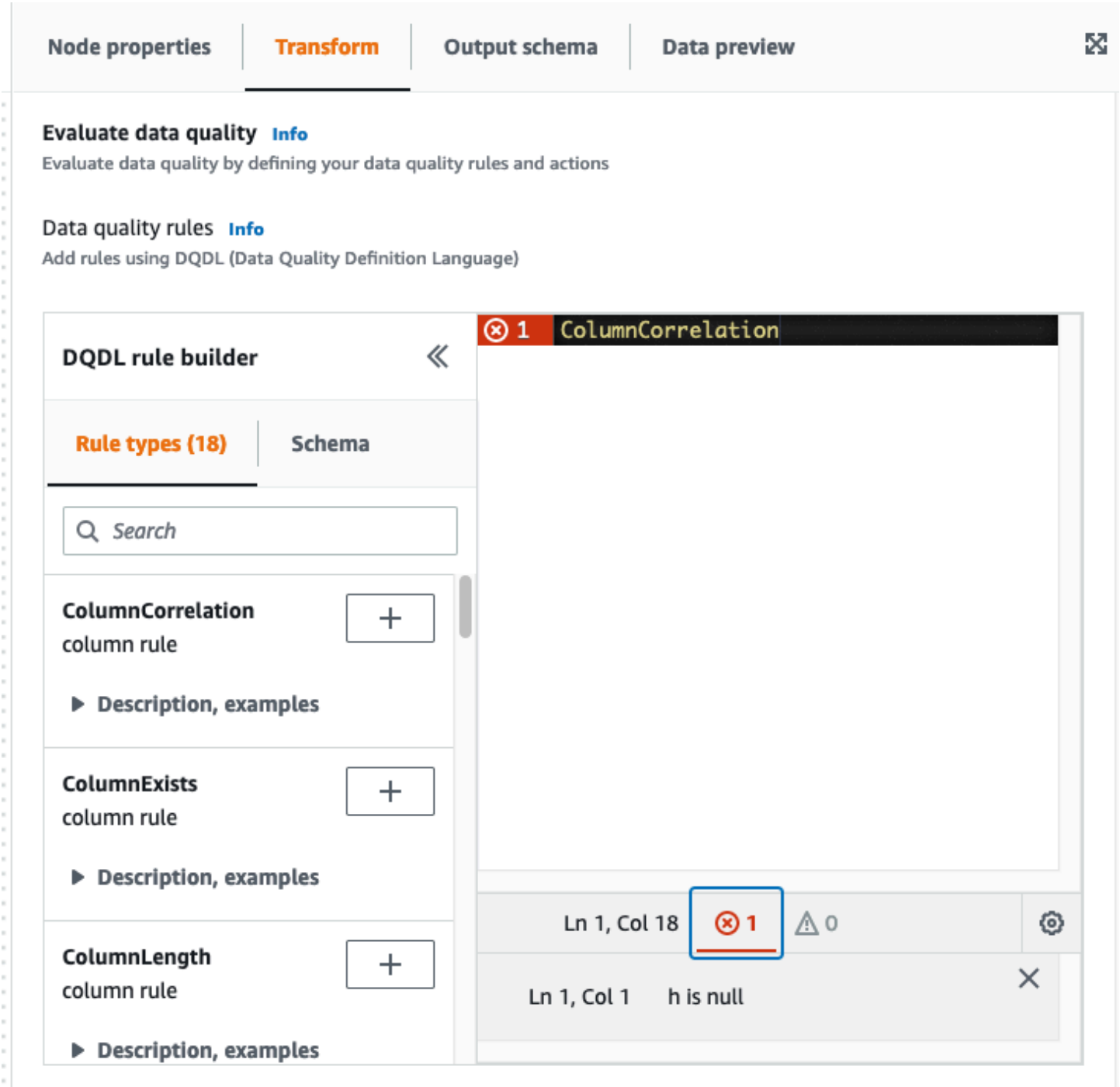
規則編輯器

規則編輯器是一個文字編輯器，您可以在其中編寫和編輯規則。如果您從 DQDL 規則建置器中選取規則類型，則規則類型會新增至規則編輯器。然後，您可以隨需透過修改文字來指定參數、新增規則和編輯規則。AWS Glue Studio 驗證規則編輯器中的規則，並顯示錯誤和警告 (如果有)。

錯誤和警告

如果規則不遵循 DQDL 規則語法，則規則編輯器會顯示數個視覺化指標，表示發生錯誤：

- 規則編輯器在有錯誤的行上顯示紅色的錯誤圖示。
- 規則編輯器會在紅色錯誤圖示旁顯示錯誤的數量。
- 如果選擇包含錯誤的行，錯誤的描述和位置 (行和資料欄) 會顯示在規則編輯器底部。



資料品質動作

依預設不會選取此動作，即使資料品質規則失敗，任務也會完成其執行。

在下列動作之間進行選擇。您可使用動作來根據特定條件將結果發布至 CloudWatch 或停止任務。動作只有在您建立規則之後才可用。

- 將結果發布至 CloudWatch：執行任務時，將結果新增至 CloudWatch。
- 資料品質失敗時任務失敗：如果資料品質規則失敗，任務也會因此失敗。

資料品質轉換輸出

- 原始資料：選擇輸出原始輸入資料。如果您想在偵測到品質問題時停止任務，此選項則是理想的選擇。
- 資料品質指標：選擇輸出已設定的規則及其通過或失敗狀態。如果您想要採取自訂動作，此選項非常實用。

資料品質輸出設定

將 Amazon S3 位置指定為資料品質輸出目標，從而設定資料品質結果位置。

設定異常偵測功能並產生洞察

AWS Glue Data Quality (DQ) 會根據您撰寫的資料品質規則評估資料，並提供一段時間內與資料有關的洞察和觀察，以便您立即採取行動。由於 DQ 會掃描資料，因此 DQ 會計算資料列計數、最大值或最小值等統計指標，然後將其與閾值表達式進行比較。

Data Quality 異常偵測功能的一些優點包括：

- 持續自動掃描資料
- 偵測可能表示意外事件或統計異常的異常狀況
- 提供規則建議，以依據 Data Quality 異常偵測功能發現的觀察採取行動

這在以下情況下非常有用：

- 想在不需要寫入資料品質的情況下自動偵測資料異常
- 想要分析資料並檢視資料外觀的視覺化呈現
- 想要追蹤資料在一段時間變更的方式

我可以檢視哪些與資料有關的觀察？

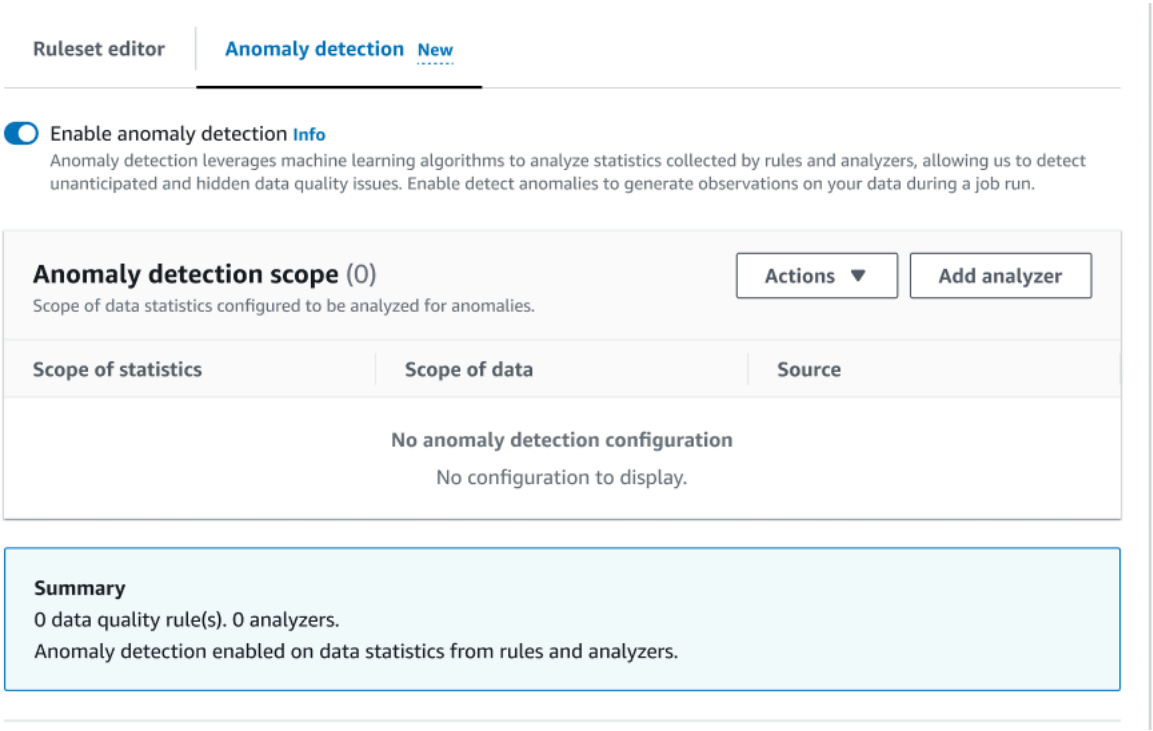
DQ 可識別收集的資料統計資料中的異常值、資料格式變更、資料漂移和結構描述變更。根據觀察，DQ 會建議使用者可以輕鬆操作的資料品質規則。統計數據包括完整性，唯一性，平均值 StandardDeviation，總和，熵， DistinctValuesCount和。 UniqueValueRatio

在 AWS Glue Studio 中啟用異常偵測功能

若要啟用異常偵測功能，您可以開啟 AWS Glue Studio 作業並開啟「啟用異常偵測」。開啟此選項可透過分析一段時間內的資料，並提供與您可採取行動之資料和觀察有關的資料統計資料，以便對資料進行異常偵測。

若要在 AWS Glue Studio 中啟用異常偵測功能：

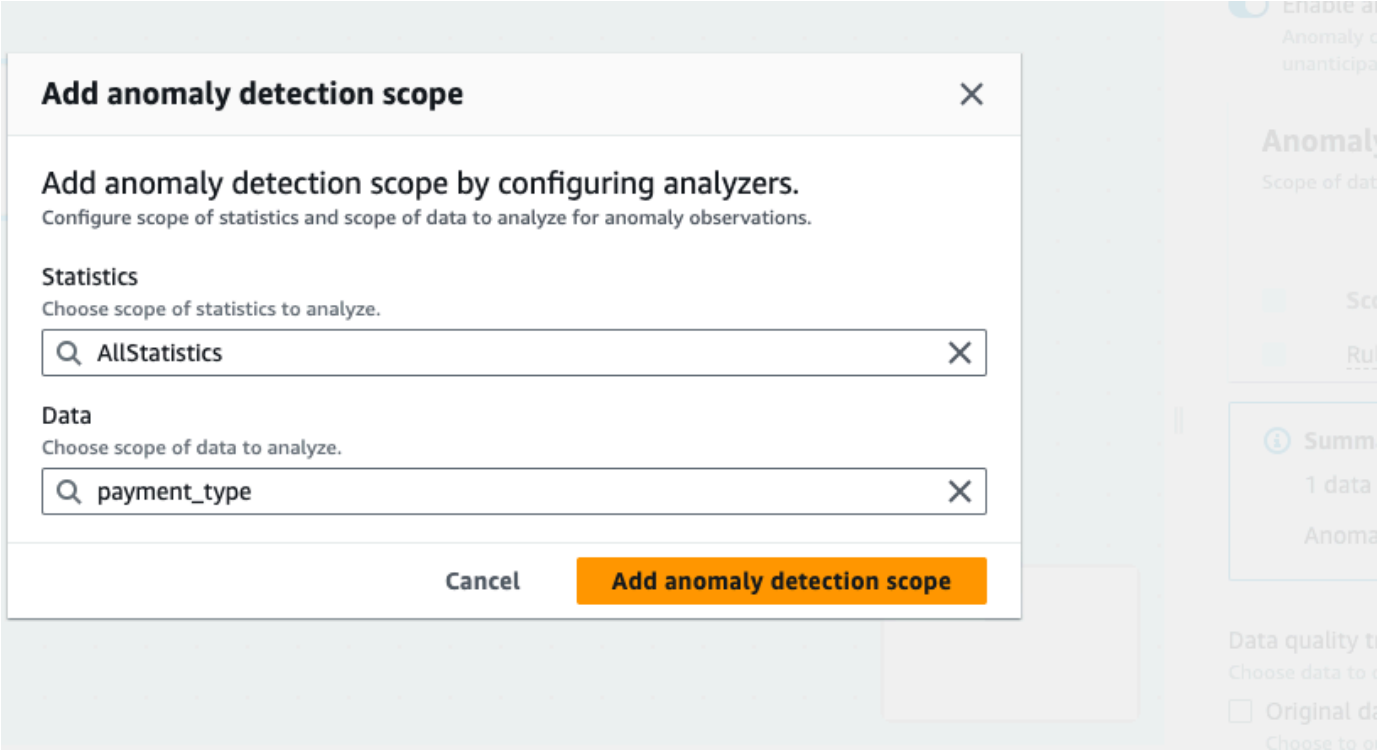
1. 選擇作業中的 Data Quality 節點，然後選擇異常偵測索引標籤。開啟「啟用異常偵測」。



2. 選擇新增分析器，定義要監視異常的資料。您可以填入的欄位有兩個：「統計資料」和「資料」。

   統計資料是與資料形狀和其他屬性有關的資訊。您可以一次選擇一或多個統計資料，或選擇「所有統計資料」。統計數據包括：完整性，唯一性，平均值 StandardDeviation，總和，熵， DistinctValuesCount和。 UniqueValueRatio

   資料是資料集中的資料欄。您可以選擇所有資料欄或個別資料欄。



3. 選擇新增異常偵測範圍來儲存變更。建立分析器後，您就可以在異常偵測範圍區段中看到那些分析器。

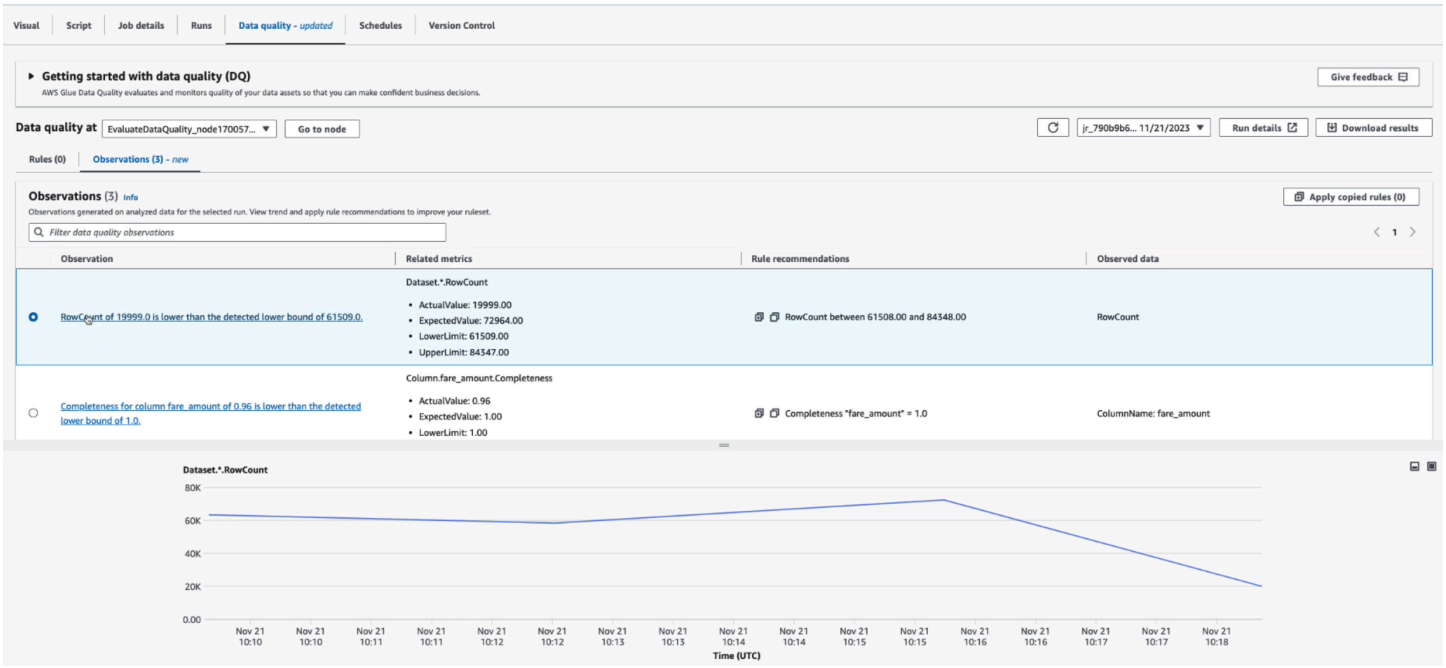
   您也可以使用動作功能表來編輯分析器，或選擇規則集編輯器索引標籤，然後直接在規則集編輯器記事本中編輯分析器。您會在建立的任何規則下方看到所儲存的分析器。

```
Rules = [
]

Analyzers = [
    Completeness “id”
]
```

   透過更新的規則集和分析器，Data Quality 會持續監控傳入的資料、透過警示來發出異常訊號或根據設定停止作業。

> **Note**
>
> 在資料集中觀察到每個資料統計資料至少有三個值時，就會產生觀察。如果沒有看到觀察，則表明 Data Quality 沒有足夠的資料可產生觀察。執行多次作業後，Data Quality 可提供資料的洞察，並在「觀察」區段中顯示那些洞察。

分析器透過偵測資料中的異常來產生觀察，並提供逐步建置規則的建議。您可以選擇「資料品質」索引標籤來檢視觀察。觀察是每個作業執行特有的。您可以檢視「觀察」區段頂端的特定 Data Quality 節點和作業執行。選擇新的節點或作業執行可檢視該節點和作業特定的觀察。



觀察：每個洞察都是以您指定的規則集和分析器設定的特定作業執行為基礎。

相關指標：產生觀察時，「相關指標」資料欄會顯示規則、實際值、預期值、下限和上限。

規則建議：AWS Glue 接著也會建議規則來解決這個問題。您可以按一下複製圖示來複製建議的每個規則。您可以按一下每個規則旁邊的複製圖示，然後按一下套用複製的規則，來複製所有建議的規則。

監控的資料：「監控的資料」資料欄會提供受監視且觸發觀察的資料欄或資料列。

將建議規則套用至 Data Quality 節點

產生觀察並提供建議的規則後，您可以將該規則套用至 Data Quality 節點。若要執行此作業：

1. 按一下每個規則建議旁邊的複製圖示。這會將規則建議新增到記事本中，以便您稍後檢索。
2. 按一下套用規則建議。此操作會開啟記事本，可供您在其中檢視先前複製的規則。
3. 選擇複製規則。
4. 選擇套用至規則集編輯器。此操作會開啟規則集編輯器，可供您在其中貼上複製的規則。
5. 將複製的規則貼至規則集編輯器。

AWS Glue Studio 筆記型電腦中 ETL 工作的資料品質

在本教學課程中，您將了解如何在 AWS Glue Studio 筆記本中使用 AWS Glue Data Quality 進行擷取、轉換和載入 (ETL) 任務。

您可使用 AWS Glue Studio 中的筆記本來編輯任務指令碼並檢視輸出，而不需執行完整任務。您還可以新增 Markdown，並將筆記本儲存為 `.ipynb` 檔案和任務指令碼。請注意，您可以直接啟動筆記本，而無需在本機安裝軟體或管理伺服器。當您完成程式碼的撰寫時，您可以使用 AWS Glue Studio 輕鬆將筆記本轉換為 AWS Glue 任務。

您在此範例中使用的資料集包含從兩個 Data.CMS.gov 資料集下載的美國聯邦醫療保險 (Medicare) 供應商付款資料：「住院患者預期付款系統供應商前 100 大診斷相關群組摘要 - FY2011」和「住院患者費用資料 FY 2011」。

下載資料之後，我們修改了資料集，以在檔案結尾處引入幾個錯誤記錄。上述經修改的檔案位於公有 Amazon S3 儲存貯體，位置在 `s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv`。

必要條件

- 具有 Amazon S3 許可的 AWS Glue 角色，可寫入目的地 Amazon S3 儲存貯體
- 新的筆記本 (請參閱 AWS Glue Studio 中的筆記本入門)

在 AWS Glue Studio 中建立 ETL 任務

建立 ETL 任務

1. 將工作階段版本變更為 AWS Glue 3.0。

   為此，請使用以下魔術命令移除所有樣板程式碼儲存格並執行儲存格。請注意，建立新筆記本時，第一個儲存格中會自動提供此樣板程式碼。

```
%glue_version 3.0
```

2. 複製以下程式碼並在儲存格中執行它。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
```

3. 在下一個儲存格中，匯入評估 AWS Glue Data Quality 的 `EvaluateDataQuality` 類別。

```
from awsgluedq.transforms import EvaluateDataQuality
```

4. 在下一個儲存格中，使用存放在公有 Amazon S3 儲存貯體中的 .csv 檔案讀入來源資料。

```
medicare = spark.read.format(
"csv").option(
"header", "true").option(
"inferSchema", "true").load(
's3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv')
medicare.printSchema()
```

5. 將資料轉換為 AWS Glue DynamicFrame.

```
from awsglue.dynamicframe import DynamicFrame
medicare_dyf = DynamicFrame.fromDF(medicare,glueContext,"medicare_dyf")
```

6. 使用資料品質定義語言 (DQDL) 建立規則集。

```
EvaluateDataQuality_ruleset = """
    Rules = [
```

```
        ColumnExists "Provider Id",
        IsComplete "Provider Id",
        ColumnValues  " Total Discharges " > 15
]
    ]
"""
```

7. 根據規則集驗證資料集。

```
EvaluateDataQualityMultiframe = EvaluateDataQuality().process_rows(
    frame=medicare_dyf,
    ruleset=EvaluateDataQuality_ruleset,
    publishing_options={
        "dataQualityEvaluationContext": "EvaluateDataQualityMultiframe",
        "enableDataQualityCloudWatchMetrics": False,
        "enableDataQualityResultsPublishing": False,
    },
    additional_options={"performanceTuning.caching": "CACHE_NOTHING"},
)
```

8. 檢閱結果。

```
ruleOutcomes = SelectFromCollection.apply(
    dfc=EvaluateDataQualityMultiframe,
    key="ruleOutcomes",
    transformation_ctx="ruleOutcomes",
)

ruleOutcomes.toDF().show(truncate=False)
```

輸出：

```
---------------------------------------+-------
+-----------------------------------------------------
+-----------------------------------------+
|Rule                                    |Outcome|FailureReason
                  |EvaluatedMetrics                          |
```

```
+--------------------------------------+-------
+--------------------------------------------------
+-----------------------------------------+
|ColumnExists "Provider Id"            |Passed |null
                  |{}                                        |
|IsComplete "Provider Id"              |Passed |null
                  |{Column.Provider Id.Completeness -> 1.0}   |
|ColumnValues " Total Discharges " > 15|Failed |Value: 11.0 does not meet the
 constraint requirement!|{Column. Total Discharges .Minimum -> 11.0}|
+--------------------------------------+-------
+--------------------------------------------------
+-----------------------------------------+
```

9. 篩選傳遞的資料列，並從資料品質資料列層級結果檢閱失敗的資料列。

```
owLevelOutcomes = SelectFromCollection.apply(
dfc=EvaluateDataQualityMultiframe,
key="rowLevelOutcomes",
transformation_ctx="rowLevelOutcomes",
)

rowLevelOutcomes_df = rowLevelOutcomes.toDF() # Convert Glue DynamicFrame to
 SparkSQL DataFrame
rowLevelOutcomes_df_passed =
 rowLevelOutcomes_df.filter(rowLevelOutcomes_df.DataQualityEvaluationResult ==
 "Passed") # Filter only the Passed records.
rowLevelOutcomes_df.filter(rowLevelOutcomes_df.DataQualityEvaluationResult ==
 "Failed").show(5, truncate=False) # Review the Failed records
```

輸出：

```
+----------------------------------------+-----------
+-------------------------------------+--------------------------+-------------
+--------------+-----------------+-----------------------------------
+------------------+-------------------------+-----------------------
+--------------------------+-------------------------
+-----------------------------------------+---------------------------
+---------------------------+
|DRG Definition                            |Provider Id|Provider Name
           |Provider Street Address   |Provider City|Provider State|Provider Zip
 Code|Hospital Referral Region Description| Total Discharges | Average Covered
```

```
 Charges | Average Total Payments |Average Medicare Payments|DataQualityRulesPass
      |DataQualityRulesFail                    |DataQualityRulesSkip        |
DataQualityEvaluationResult|
+----------------------------------------+-----------
+-----------------------------------+-------------------------+------------
+-------------+----------------+-----------------------------------
+-----------------+-------------------------+----------------------
+-------------------------+-------------------------
+----------------------------------------+---------------------------
+---------------------------+
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|10005      |MARSHALL MEDICAL CENTER SOUTH
        |2505 U S HIGHWAY 431 NORTH|BOAZ         |AL             |35957
 |AL - Birmingham                      |14               |$15131.85
  |$5787.57               |$4976.71                 |[IsComplete "Provider Id"]|
[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
              |
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|10046      |RIVERVIEW REGIONAL MEDICAL
 CENTER    |600 SOUTH THIRD STREET    |GADSDEN      |AL             |35901
   |AL - Birmingham                      |14               |$67327.92
   |$5461.57               |$4493.57                 |[IsComplete "Provider Id"]|
[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
              |
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|10083      |SOUTH BALDWIN REGIONAL
 MEDICAL CENTER|1613 NORTH MCKENZIE STREET|FOLEY        |AL             |36535
        |AL - Mobile                          |15               |$25411.33
         |$5282.93               |$4383.73                 |[IsComplete "Provider
 Id"]|[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
                    |
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|30002      |BANNER GOOD SAMARITAN MEDICAL
 CENTER |1111 EAST MCDOWELL ROAD   |PHOENIX      |AZ             |85006
 |AZ - Phoenix                         |11               |$34803.81
  |$7768.90               |$6951.45                 |[IsComplete "Provider Id"]|
[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
              |
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|30010      |CARONDELET ST  MARYS HOSPITAL
        |1601 WEST ST MARY'S ROAD  |TUCSON       |AZ             |85745
 |AZ - Tucson                          |12               |$35968.50
  |$6506.50               |$5379.83                 |[IsComplete "Provider Id"]|
[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
              |
+----------------------------------------+-----------
+-----------------------------------+-------------------------+------------
+-------------+----------------+-----------------------------------
+-----------------+-------------------------+----------------------
```

```
+-------------------------+-------------------------
+-----------------------------------------+--------------------------
+--------------------------+
only showing top 5 rows
```

請注意，「AWS Glue 資料品質」新增了四個新欄 (DataQualityRulesPass DataQualityRulesFail DataQualityRulesSkip、、和 DataQualityEvaluationResult)。這指示通過的記錄、失敗的記錄、為資料列層級評估跳過的規則，以及整體資料列層級結果。

10. 將輸出寫入 Amazon S3 儲存貯體，以分析資料並視覺化結果。

```
#Write the Passed records to the destination.

glueContext.write_dynamic_frame.from_options(
       frame = rowLevelOutcomes_df_passed,
       connection_type = "s3",
       connection_options = {"path": "s3://glue-sample-target/output-dir/
medicare_parquet"},
       format = "parquet")
```

資料品質定義語言 (DQDL) 參考

資料品質定義語言 (DQDL) 是您用來定義「AWS Glue 資料品質」規則的網域特定語言。

本指南介紹了關鍵的 DQDL 概念，可幫助您了解該語言。其中也提供 DQDL 規則類型的參考，內含語法和範例。在使用本指南之前，建議您先熟悉 AWS Glue 資料品質。如需詳細資訊，請參閱 AWS Glue 資料品質。

> **Note**
>
> DynamicRules 僅在 AWS Glue ETL 中支持。

內容

- 總和
- SchemaMatch
- 唯一性
- UniqueValueRatio
- DetectAnomalies

DQDL 語法

DQDL 文件區分大小寫，且包含規則集，可將個別資料品質規則分組。若要建構規則集，您必須建立名為 Rules (大寫) 的清單，並以一對方括號分隔。如下列範例所示，清單應包含一或多個以逗號分隔的 DQDL 規則。

```
Rules = [
   IsComplete "order-id",
   IsUnique "order-id"
]
```

規則結構

DQDL 規則的結構取決於規則類型。不過，DQDL 規則通常適合以下格式。

```
<RuleType> <Parameter> <Parameter> <Expression>
```

RuleType 是您要設定的規則類型名稱 (區分大小寫)。例如，IsComplete、IsUnique 或 CustomSql。每種規則類型的規則參數都不同。如需 DQDL 規則類型及其參數的完整參考資料，請參閱 DQDL 規則類型參考。

複合規則

DQDL 支援下列可用來合併規則的邏輯運算子。這些規則稱為複合規則。

以及

當且僅當邏輯 and 運算子連接的規則為 true 時，其結果為 true。否則，合併規則的結果為 false。您使用 and 運算子連接的每個規則都必須以括號括住。

下列範例會使用 and 運算子來合併兩個 DQDL 規則。

```
(IsComplete "id") and (IsUnique "id")
```

或

當且僅當邏輯 or 運算子連接的一個或多個規則為 true 時，其結果為 true。您使用 or 運算子連接的每個規則都必須以括號括住。

下列範例會使用 or 運算子來合併兩個 DQDL 規則。

```
(RowCount "id" > 100) or (IsPrimaryKey "id")
```

您可以使用相同的運算子來連接多個規則，因此允許使用以下規則組合。

```
(Mean "Star_Rating" > 3) and (Mean "Order_Total" > 500) and (IsComplete "Order_Id")
```

不過，您無法將邏輯運算子合併為單一運算式。例如，不允許下列組合。

```
(Mean "Star_Rating" > 3) and (Mean "Order_Total" > 500) or (IsComplete "Order_Id")
```

複合規則的運作方式

根據預設，複合規則會評估為整個資料集或資料表中的個別規則，然後合併結果。換句話說，它首先評估整個列，然後應用運算符。下面將使用示例解釋此默認行為：

```
# Dataset

+------+------+
|myCol1|myCol2|
+------+------+
|     2|     1|
|     0|     3|
+------+------+

# Overall outcome

+-------------------------------------------------------------+-------+
|Rule                                                         |Outcome|
+-------------------------------------------------------------+-------+
|(ColumnValues "myCol1" > 1) OR (ColumnValues "myCol2" > 2)|Failed |
```

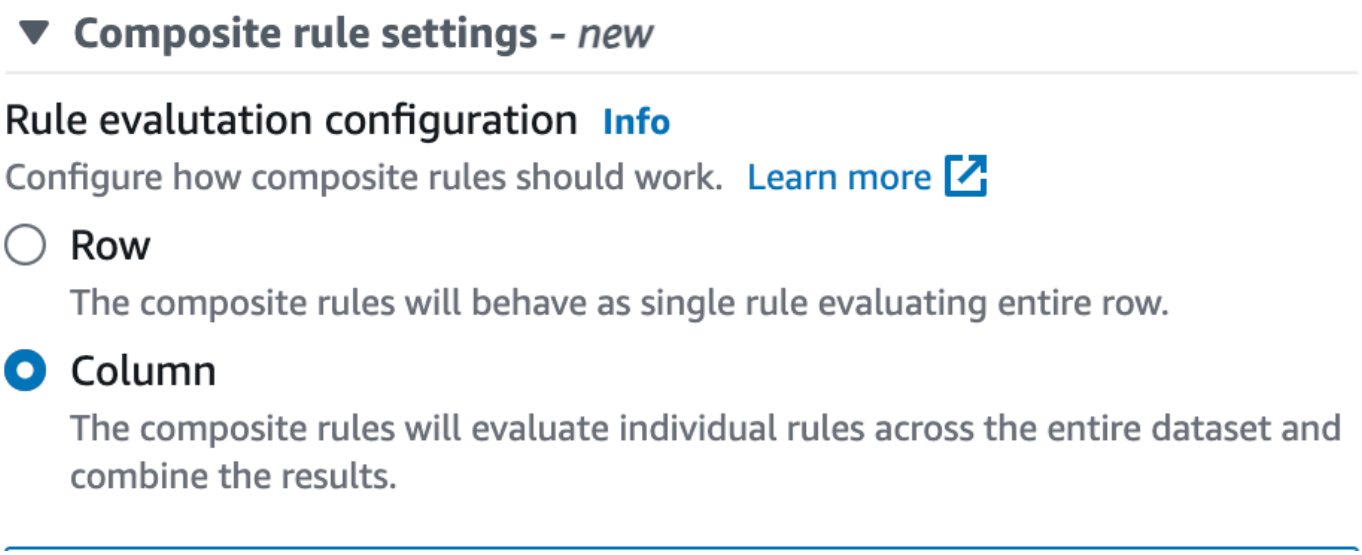
```
+-----------------------------------------------------------+-------+
```

在上面的例子中，AWS Glue Data Quality首先評估(ColumnValues "myCol1" > 1)這將導致失敗。然後它將評估(ColumnValues "myCol2" > 2)哪些也會失敗。兩個結果的組合將被註明為「失敗」。

但是，如果您更喜歡類似 SQL 的行為，則需要評估整行，則必須明確設置ruleEvaluation.scope參數，如下面additionalOptions的代碼片段所示。

```
object GlueApp {
  val datasource = glueContext.getCatalogSource(
    database="<db>",
    tableName="<table>",
    transformationContext="datasource"
  ).getDynamicFrame()

  val ruleset = """
    Rules = [
        (ColumnValues "age" >= 26) OR (ColumnLength "name" >= 4)
    ]
  """

  val dq_results = EvaluateDataQuality.processRows(
    frame=datasource,
    ruleset=ruleset,
    additionalOptions=JsonOptions("""
        {
          "compositeRuleEvaluation.method":"ROW"
        }
      """
    )
  )
}
```

在 AWS Glue Studio 和 AWS Glue 數據目錄中，您可以輕鬆地在用戶界面中設置此選項，如下圖所示。



一旦設定完成，複合規則將表現為評估整個資料列的單一規則。下面的例子說明了這種行為。

```
# Row Level outcome

+------+------+-----------------------------------------------------------------
+---------------------------+
|myCol1|myCol2|DataQualityRulesPass                                              |
DataQualityEvaluationResult|
+------+------+-----------------------------------------------------------------
+---------------------------+
|2     |1     |[(ColumnValues "myCol1" > 1) OR (ColumnValues "myCol2" > 2)]|Passed
                |
|0     |3     |[(ColumnValues "myCol1" > 1) OR (ColumnValues "myCol2" > 2)]|Passed
                |
+------+------+-----------------------------------------------------------------
+---------------------------+
```

此功能無法支援某些規則，因為它們的整體結果取決於臨界值或比率。它們在下面列出。

依賴比率的規則：

- 完整度
- DatasetMatch
- ReferentialIntegrity

- Uniqueness

依賴於閾值的規則：

當下列規則包含臨界值時，就不支援這些規則。但是，不涉及的規則`with threshold`仍然受到支援。

- ColumnDataType
- ColumnValues
- CustomSQL

表達式

如果規則類型不會產生布林值回應，您必須提供運算式作為參數，才能建立布林值回應。例如，下列規則會根據表達式檢查欄中所有值的平均值，以傳回 true 或 false 結果。

```
Mean "colA" between 80 and 100
```

某些規則類型 (例如 `IsUnique` 和 `IsComplete`) 已傳回布林值回應。

下表列出您可以在 DQDL 規則中使用的運算式。

支援的 DQDL 運算式

表達式	描述	範例
= *X*	如果規則類型回應等於 *x*，則解析為 `true`。	`Completeness "colA" = "1.0", ColumnValues "colA" = "2022-06-30"`
! = *X*	x 如果規則類型回應不等於 *x*，則解析為 true。	`ColumnValues "colA" != "a", ColumnValues "colA" != "2022-06-30"`
> *x*	如果規則類型回應大於 *x*，則解析為 `true`。	`ColumnValues "colA" > 10`

表達式	描述	範例
< *x*	如果規則類型回應小於 *x*，則解析為 true。	`ColumnValues "colA" < 1000, ColumnValues "colA" < "2022-06-30"`
>= *x*	如果規則類型回應大於或等於 *x*，則解析為 true。	`ColumnValues "colA" >= 10`
<= *x*	如果規則類型回應小於或等於 *x*，則解析為 true。	`ColumnValues "colA" <= 1000`
between *x* and *y*	如果規則類型回應落在指定範圍內 (不含) 時，則解析為 true。只針對數字和日期類型使用此表達式類型。	`Mean "colA" between 8 and 100, ColumnValues "colA" between "2022-05-31" and "2022-06-30"`
不在 *x* 和 *y* 之間	如果規則類型回應未落在指定範圍內 (含)，則解析為 true。您應該只針對數字和日期類型使用此運算式類型。	`ColumnValues "colA" not between "2022-05-31" and "2022-06-30"`
in [*a, b, c, ...*]	如果規則類型回應在指定集中，則解析為 true。	`ColumnValues "colA" in [ 1, 2, 3 ], ColumnValues "colA" in [ "a", "b", "c" ]`
不在 [*a, b, c,...*]	true如果規則類型回應不在指定集中，則解析為。	`ColumnValues "colA" not in [ 1, 2, 3 ], ColumnValues "colA" not in [ "a", "b", "c" ]`
matches */ab+c/i*	如果規則類型回應符合規則運算式，則解析為 true。	`ColumnValues "colA" matches "[a-zA-Z]*"`

表達式	描述	範例
不匹配 */AB+C/I*	true如果規則類型回應與規則運算式不符，則解析為。	`ColumnValues "colA" not matches "[a-zA-Z]*"`
now()	僅適用於 ColumnValues 規則類型以建立日期運算式。	`ColumnValues "load_date" > (now() - 3 days)`

匹配/在 [...]/不匹配/不在 [...] with threshold	指定符合規則條件的值的百分比。僅適用於ColumnValues ColumnDataType 、和CustomSQL 規則類型。	`ColumnValues "colA" in ["A", "B"] with threshold > 0.8,`
		`ColumnValues "colA" matches "[a-zA-Z]*" with threshold between 0.2 and 0.9`
		`ColumnDataType "colA" = "Timestamp" with threshold > 0.9`

空，空和空白的關鍵字 _ 僅

如果你想驗證一個字符串列是否有一個空，空或只有空格的字符串，你可以使用以下關鍵字：

- NULL /null — 此關鍵字會針對字串資料行中的null值解析為 true。

  `ColumnValues "colA" != NULL with threshold > 0.5`如果超過 50% 的數據沒有空值，則返回 true。

  `(ColumnValues "colA" = NULL) or (ColumnLength "colA" > 5)`對於所有具有空值或長度 >5 的行都將返回 true。請注意，這將需要使用「compositeRuleEvaluation.method」=「ROW」選項。
- EMPTY /空白 — 此關鍵字會針對字串欄中的空字串 (「」) 值解析為 true。某些資料格式會將字串資料行中的空值轉換為空字串。此關鍵字有助於篩選出資料中的空字串。

  `(ColumnValues "colA" = EMPTY) or (ColumnValues "colA" in ["a", "b"])`如果一行為空，「a」或「b」，則返回 true。請注意，這需要使用「compositeRuleEvaluation.method」=「ROW」選項。
- 白色空格 _ ONLY/空白-對於字串欄中只有空格 (「」) 值的字串，此關鍵字會解析為 true。

`ColumnValues "colA" not in ["a", "b", WHITESPACES_ONLY]`如果一行既不是「a」或「b」，也不僅僅是空格，則返回 true。

支援的規則：

- ColumnValues

對於基於數字或日期的表達式，如果要驗證列是否具有 null，則可以使用以下關鍵字。

- NULL /null — 此關鍵字會針對字串資料行中的空值解析為 true。

  `ColumnValues "colA" in [NULL, "2023-01-01"]`如果列中的日期為`2023-01-01`或 null，則返回 true。

  `(ColumnValues "colA" = NULL) or (ColumnValues "colA" between 1 and 9)`對於具有空值或在 1 到 9 之間的值的所有行都將返回 true。請注意，這將需要使用「compositeRuleEvaluation.method」=「ROW」選項。

  支援的規則：

  - ColumnValues

使用 WHERE 子句篩選

您可以在編寫規則時篩選資料。當您要套用條件規則時，這會很有幫助。

```
<DQDL Rule> where "<valid SparkSQL where clause> "
```

必須使用`where`關鍵字指定篩選器，後面接著以引號(`""`)括住的有效 SparkSQL 陳述式。

如果您希望將 where 子句添加到具有閾值的規則中的規則，則應在閾值條件之前指定 where 子句。

```
<DQDL Rule> where "valid SparkSQL statement>" with threshold <threshold condition>
```

使用此語法，您可以編寫如下所示的規則。

```
Completeness "colA" > 0.5 where "colB = 10"
ColumnValues "colB" in ["A", "B"] where "colC is not null" with threshold > 0.9
ColumnLength "colC" > 10 where "colD != Concat(colE, colF)"
```

我們將驗證所提供的 SparkSQL 陳述式是否有效。如果無效，規則評估將失敗，我們將拋出`IllegalArgumentException`具有以下格式的 a：

```
Rule <DQDL Rule> where "<invalid SparkSQL>" has provided an invalid where clause :
<SparkSQL Error>
```

開啟列層級錯誤記錄識別時的 Where 子句行為

使用「 AWS Glue 資料品質」，您可以識別失敗的特定記錄。當應用 where 子句支持行級結果的規則，我們將標記由 where 子句為過濾掉的行`Passed`。

如果您偏好將篩選出的資料列分別標示為`SKIPPED`，您可以`additionalOptions`為 ETL 工作設定下列項目。

```
object GlueApp {
  val datasource = glueContext.getCatalogSource(
    database="<db>",
    tableName="<table>",
    transformationContext="datasource"
  ).getDynamicFrame()

  val ruleset = """
    Rules = [
       IsComplete "att2" where "att1 = 'a'"
    ]
  """

  val dq_results = EvaluateDataQuality.processRows(
    frame=datasource,
    ruleset=ruleset,
    additionalOptions=JsonOptions("""
        {
          "rowLevelConfiguration.filteredRowLabel":"SKIPPED"
        }
      """
    )
  )
}
```

作為一個例子，請參考以下規則和數據框：

```
IsComplete att2 where "att1 = 'a'"
```

id	ATT1	ATT2	列層級結果 (預設值)	資料列層級結果 (略過選項)	說明
1	a	f	通過	通過	
2	b	d	通過	略過	行被過濾掉，因為不 att1是 "a"
3	a	null	失敗	失敗	
4	a	f	通過	通過	
5	b	null	通過	略過	行被過濾掉，因為不 att1是 "a"
6	a	f	通過	通過	

動態規則

您現在可以編寫動態規則，將規則產生的目前指標與其歷史值進行比較。這些歷史比較是透過在表達式中使用 `last()` 運算子來啟用。例如，當目前執行中的資料列數目大於相同資料集的最近先前一個資料列計數時，規則 `RowCount > last()` 便會成功。`last()` 採用可選的自然數引數，描述要考慮的先前指標；`last(k)` 中 `k >= 1` 將參考最後 k 個指標。

- 如果沒有可用的資料點，`last(k)` 將傳回預設值 0.0。
- 如果可用的指標少於 k，`last(k)` 將傳回所有先前的指標。

為了形成使用 `last(k)` 的有效表達式，`k > 1` 需要彙總函數將多個歷史結果簡化為一個數字。例如，`RowCount > avg(last(5))` 將檢查目前資料集的資料列計數是否嚴格大於相同資料集最後五個資料列計數的平均值。`RowCount > last(5)` 將產生錯誤，因為當前資料集的資料列計數不能與清單進行有意義的比較。

支援的彙總函數：

- `avg`
- `median`

- `max`
- `min`
- `sum`
- `std` (標準偏差)
- `abs` (絕對值)
- `index(last(k), i)` 將允許從最後 `k` 個值中選取第 `i` 個最近的值。`i` 從零開始索引，所以 `index(last(3), 0)` 將傳回最新的資料點；而 `index(last(3), 3)` 會導致錯誤，因為只有三個資料點，但我們嘗試對第 4 個最新的資料點編製索引。

範例運算式

ColumnCorrelation

- `ColumnCorrelation "colA" "colB" < avg(last(10))`

DistinctValuesCount

- `DistinctValuesCount "colA" between min(last(10))-1 and max(last(10))+1`

大多數具有數值條件或閾值的規則類型都支援動態規則；請參閱提供的資料表分析器和規則，判斷規則類型是否支援動態規則。

分析器

> **Note**
>
> AWS Glue 資料型錄不支援分析器。

DQDL 規則使用名為 Analyzers 的函數來收集與資料有關的資訊。規則的布林表達式會使用此資訊來判斷規則是成功或是失敗。例如， RowCount 規則`RowCount > 5` 會使用資料列計數分析器來探索資料集中的資料列數目，並將該計數與運算式進行比較，`> 5`以檢查目前資料集中是否有五個以上的資料列。

有時候，我們建議您建立分析器而不是撰寫規則，然後讓這些分析器產生可用來偵測異常的統計資料。對於這種情況，您可以建立分析器。分析器與規則有下列不同之處。

特性	分析器	規則
規則集的一部分	是	是
產生統計資料	是	是
產生觀察	是	是
可以評估和斷言條件	否	是
您可以設定動作，例如在失敗時停止作業、繼續處理作業	否	是

分析器可以在沒有規則的情況下獨立存在，因此您可以快速設定這些分析器並逐步建置資料品質規則。

您可以在規則集的 Analyzers 區塊中輸入某些規則類型，以執行分析器所需的規則並收集資訊，而無需對任何條件套用檢查。某些分析器不會與規則相關聯，在 Analyzers 區塊中只能作為輸入。下表指出每個項目是否受到規則或獨立分析器的支援，以及每個規則類型的其他詳細資訊。

使用分析器的示例規則集

以下規則集使用：

- 動態規則，檢查資料集的成長速度是否超過在過去三次作業執行的結尾平均值
- DistinctValuesCount 分析器，記錄資料集 Name 資料欄中相異值的數目
- ColumnLength 分析器，追蹤隨時間變化的最小和最大 Name 尺寸

您可以在作業執行的「資料品質」索引標籤中檢視分析器指標結果。

```
Rules = [
   RowCount > avg(last(3))
]
Analyzers = [
   DistinctValuesCount "Name",
   ColumnLength "Name"
]
```

說明

您可以使用 '#' 字元在 DQDL 文件中新增註解。DQDL 忽略 '#字符之後的任何內容，直到行結尾之前的任何內容。

```
Rules = [
    # More items should generally mean a higher price, so correlation should be
 positive
    ColumnCorrelation "price" "num_items" > 0
]
```

DQDL 規則類型參考

本節提供「 AWS Glue 資料品質」支援之每種規則類型的參考資料。

> **Note**
>
> - DQDL 目前不支援巢狀或清單類型的資料欄資料。
> - 下表中括號內的值將被替換為規則引數中提供的資訊。
> - 規則通常需要運算式的額外引數。

Ruletype	描述	引數	報告的指標	支援作為規則？	支援作為分析器？	傳回資料列層級結果？	動態規則支援？	產生觀察	支持何處子句語法？
Aggregat Match	透過比較諸如總銷售額等摘要指標，檢查兩個資料集是否相符。這	一或多個彙總	當第一個和第二個彙總資料欄名稱相符時： Column. [C	是	否	否	否	否	否

Ruletype	描述	引數	報告的指標	支援作為規則？	支援作為分析器？	傳回資料列層級結果？	動態規則支援？	產生觀察	支持何處子句語法？
	可讓金融機構比較是否從來源系統中擷取所有資料，因此非常實用。		olumn].gregatetch 當第一個和第二個彙總資料欄名稱不相符時： Column.[Column1,lumn2].gregatetch						

Ruletype	描述	引數	報告的指標	支援作為規則？	支援作為分析器？	傳回資料列層級結果？	動態規則支援？	產生觀察	支持何處子句語法？
AllStatistics	獨立分析器，可為提供的資料欄或資料集中的所有資料欄收集多個指標。	單一資料欄名稱，或 "AllColumns」	所有類型的資料欄： Dataset.RowCou Column.[Column].mpleters Column.[Column].iquenes 字串值資料欄的其他指標： ColumnLgth metrics 數值資料欄的其他指標：	否	是	否	否	否	否

Ruletype	描述	引數	報告的指標	支援作為規則？	支援作為分析器？	傳回資料列層級結果？	動態規則支援？	產生觀察	支持何處子句語法？
			Column\ues metrics						
ColumnCorrelation	檢查兩個資料欄如何相互關聯。	剛好兩個資料欄名稱	Multicolumn.[Column1],[Column2].ColumnCorrelation	是	是	否	是	否	是
ColumnCount	檢查是否有任何資料欄遭到捨棄。	無	Dataset.ColumnCount	是	否	否	是	是	否
ColumnDataType	檢查資料欄是否與資料類型相容。	剛好一個資料欄名稱	Column.[Column].ColumnDataType.Compliance	是	否	否	是，在資料列層級閾值表達式中	否	是

Ruletype	描述	引數	報告的指標	支援作為規則？	支援作為分析器？	傳回資料列層級結果？	動態規則支援？	產生觀察	支持何處子句語法？
ColumnExists	檢查資料集中是否存在資料欄。這可讓客戶建立自助式資料平台，確保某些資料欄可供使用。	剛好一個資料欄名稱	N/A	是	否	否	否	否	否

Ruletype	描述	引數	報告的指標	支援作為規則？	支援作為分析器？	傳回資料列層級結果？	動態規則支援？	產生觀察	支持何處子句語法？
ColumnLength	檢查資料的長度是否一致。	剛好一個資料欄名稱	Column.[Column].MaximumLength Column.[Column].MinimumLength 提供資料列層級閾值時的其他指標： Column.[Column].ColumnValues.Compliance	是	是	是，當提供資料列層級臨界值時	否	是。僅透過分析最小和最大長度來產生觀察	是

Ruletype	描述	引數	報告的指標	支援作為規則？	支援作為分析器？	傳回資料列層級結果？	動態規則支援？	產生觀察	支持何處子句語法？
ColumnN esMatchP ttern	檢查資料欄名稱是否與定義的模式相符。對於控管團隊強制執行資料欄名稱一致性非常實用。	資料欄名稱的 regex	Dataset .Columr mesPatt nMatchR io	是	否	否	否	否	否

Ruletype	描述	引數	報告的指標	支援作為規則？	支援作為分析器？	傳回資料列層級結果？	動態規則支援？	產生觀察	支持何處子句語法？
ColumnValues	檢查每個定義值的資料是否一致。此規則支援規則運算式。	剛好一個資料欄名稱	Column.[Column].Maximum Column.[Column].Minimum 提供資料列層級閾值時的其他指標： Column.[Column].ColumnValues.Compliance	是	是	是，當提供資料列層級臨界值時	否	是。僅透過分析最小和最大值來產生觀察	是
完整度	檢查資料中是否有任何空白或 NULL 值。	剛好一個資料欄名稱	Column.[Column].Completeness	是	是	是	是	是	是

Ruletype	描述	引數	報告的指標	支援作為規則？	支援作為分析器？	傳回資料列層級結果？	動態規則支援？	產生觀察	支持何處子句語法？
CustomS	客戶可以在 SQL 中實作幾乎所有類型的資料品質檢查。	SQL 陳述式 (選用) 資料列層級閾值	Dataset.CustomL 提供資料列層級閾值時的其他指標： Dataset.CustomL.Compliance	是	否	是，當提供資料列層級臨界值時	是	否	否
DataFres ness	檢查資料是否為最新狀態。	剛好一個資料欄名稱	Column.[Column].DataFreshness.Compliance	是	否	是	否	否	是

Ruletype	描述	引數	報告的指標	支援作為規則？	支援作為分析器？	傳回資料列層級結果？	動態規則支援？	產生觀察	支持何處子句語法？
DatasetMatch	比較兩個資料集並識別其是否同步。	參考資料集的名稱 資料欄映射 (選用)要檢查相符項目的資料欄	Dataset [Referer Dataset ias].DatasetMatc	是	否	是	是	否	否
DistinctValuesCount	檢查重複值。	剛好一個資料欄名稱	Column [Column]. stinctValuesCour	是	是	是	是	是	是
DetectAnomalies	檢查其他規則類型報告指標中的異常。	規則類型	規則類型引數報告的指標	是	否	否	否	否	否
Entropy	檢查資料的熵。	剛好一個資料欄名稱	Column [Column]. tropy	是	是	否	是	否	是

Ruletype	描述	引數	報告的指標	支援作為規則？	支援作為分析器？	傳回資料列層級結果？	動態規則支援？	產生觀察	支持何處子句語法？
IsComple e	檢查是否 100% 的資料皆為已完成。	剛好一個資料欄名稱	Column. [C olumn]. mpleter s	是	否	是	否	否	是
IsPrimar Key	檢查資料欄是否為主索引鍵 (非 NULL 且是唯一的)。	剛好一個資料欄名稱	對於單一資料欄： Column. [C olumn]. iquenes 對於多個資料欄： Multicc mn[Comm elimite olumns] niquene	是	否	是	否	否	是
IsUnique	檢查是否 100% 的資料皆為唯一。	剛好一個資料欄名稱	Column. [C olumn]. iquenes	是	否	是	否	否	是

Ruletype	描述	引數	報告的指標	支援作為規則？	支援作為分析器？	傳回資料列層級結果？	動態規則支援？	產生觀察	支持何處子句語法？
Mean	檢查平均值是否與設定的閾值相符。	剛好一個資料欄名稱	Column.[Column].Mean	是	是	是	是	否	是
ReferentialIntegrity	檢查兩個資料集是否具有參照完整性。	資料集中一或多個資料欄名稱 參考資料集中一或多個資料欄名稱	Column.[Reference Dataset Alias].ReferentialIntegrity	是	否	是	是	否	否
RowCou	檢查記錄計數是否與閾值相符。	無	Dataset.RowCou	是	是	否	是	是	是
RowCouatch	檢查兩個資料集之間的記錄計數是否相符。	參考資料集別名	Dataset[Referer Dataset ias].RowCountMat	是	否	否	是	否	否

Ruletype	描述	引數	報告的指標	支援作為規則？	支援作為分析器？	傳回資料列層級結果？	動態規則支援？	產生觀察	支持何處子句語法？
Standarc eviation	檢查標準差是否與閥值相符。	剛好一個資料欄名稱	Column. [C olumn]. andardD iation	是	是	是	是	否	是
Schemal ch	檢查兩個資料集之間的結構描述是否相符。	參考資料集別名	Datase [Refere Datase ias].Sc maMatcl	是	否	否	是	否	否
Sum	檢查總和是否與設定的閾值相符。	剛好一個資料欄名稱	Column. [C olumn]. m	是	是	否	是	否	是
Uniquen s	檢查資料集的唯一性是否與閾值相符。	剛好一個資料欄名稱	Column. [C olumn]. iquenes	是	是	是	是	否	是
UniqueV ueRatio	檢查唯一值定量是否與閥值相符。	剛好一個資料欄名稱	Column. [C olumn]. iqueVal Ratio	是	是	是	是	否	是

主題

- AggregateMatch
- ColumnCorrelation
- ColumnCount
- ColumnDataType
- ColumnExists
- ColumnLength
- ColumnNamesMatchPattern
- ColumnValues
- 完整度
- CustomSQL
- DataFreshness
- DatasetMatch
- DistinctValuesCount
- Entropy
- IsComplete
- IsPrimaryKey
- IsUnique
- Mean
- ReferentialIntegrity
- RowCount
- RowCountMatch
- StandardDeviation
- 總和
- SchemaMatch
- 唯一性
- UniqueValueRatio
- DetectAnomalies

AggregateMatch

根據指定運算式檢查兩個資料欄彙總的比率。此規則類型適用於多個資料集。系統會評估兩個資料欄彙總，並將第一個資料欄彙總的結果除以第二個資料欄彙總的結果來產生比率。系統會根據提供的運算式檢查比率，以產生布林值回應。

語法

資料欄彙總

```
ColumnExists <AGG_OPERATION> (<OPTIONAL_REFERENCE_ALIAS>.<COL_NAME>)
```

- AGG_OPERATION：用於彙總的操作。目前支援 sum 和 avg。

  支援的資料欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- OPTIONAL_REFERENCE_ALIAS：若資料欄來自參考資料集且不是主資料集，則需提供此參數。如果您在 AWS Glue 資料型錄中使用此規則，您的參考別名必須遵循格式 "<database_name>。<table_name>。 <column_name>

  支援的資料欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- COL_NAME：要彙總的資料欄名稱。

  支援的資料欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數

範例：平均值

```
"avg(rating)"
```

範例：總和

```
"sum(amount)"
```

範例：參考資料集中資料欄的平均值

```
"avg(reference.rating)"
```

規則

```
AggregateMatch <AGG_EXP_1> <AGG_EXP_2> <EXPRESSION>
```

- AGG_EXP_1：第一個資料欄彙總。

  支援的資料欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數

  支援的資料欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- AGG_EXP_2：第二個資料欄彙總。

  支援的資料欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數

  支援的欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：使用總和執行彙總比對

下列範例規則會檢查 amount 資料欄中的值總和是否與 total_amount 資料欄中的值總和完全相等。

```
AggregateMatch "sum(amount)" "sum(total_amount)" = 1.0
```

範例：使用平均值執行彙總比對

下列範例規則會檢查 ratings 資料欄中值的平均值是否至少等於 reference 資料集之 ratings 資料欄中值的平均值的 90%。參考資料集會提供為 ETL 或資料型錄體驗中的其他資料來源。

在 AWS Glue ETL 中，您可以使用：

```
AggregateMatch "avg(ratings)" "avg(reference.ratings)" >= 0.9
```

在 AWS Glue 資料型錄中，您可以使用：

```
AggregateMatch "avg(ratings)" "avg(database_name.tablename.ratings)" >= 0.9
```

空行為

AggregateMatch規則會忽略彙總方法 (總和/平均值) 計算中含有 NULL 值的資料列。例如：

```
+---+-----------+
|id |units      |
```

```
+---+-----------+
|100|0          |
|101|null       |
|102|20         |
|103|null       |
|104|40         |
+---+-----------+
```

列的平均值units將是（0 + 20 + 40）/3 = 20。此計算中不會考慮列 101 和 103 列。

ColumnCorrelation

檢查對一個給定的表達式兩列之間的相關性。 AWS Glue 資料品質使用 Pearson 相關係數來測量兩欄之間的線性相關性。結果為 -1 到 1 之間的數字，用於測量關係的強度和方向。

語法

```
ColumnCorrelation <COL_1_NAME> <COL_2_NAME> <EXPRESSION>
```

- COL_1_NAME – 您要評估資料品質規則的第一欄名稱。

  支援的欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- COL_2_NAME – 您要評估資料品質規則的第二個資料欄名稱。

  支援的欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：欄關聯性

下列範例規則會檢查欄 height 和 weight 之間的關聯性係數是否具有較強的正關聯性 (係數值大於 0.8)。

```
ColumnCorrelation "height" "weight" > 0.8
```

```
ColumnCorrelation "weightinkgs" "Salary" > 0.8 where "weightinkgs > 40"
```

動態規則範例

- `ColumnCorrelation "colA" "colB" between min(last(10)) and max(last(10))`

- `ColumnCorrelation "colA" "colB" < avg(last(5)) + std(last(5))`

空行為

ColumnCorrelation規則會在計算相互關聯時忽略含有NULL值的列。例如：

```
+---+-----------+
|id |units      |
+---+-----------+
|100|0          |
|101|null       |
|102|20         |
|103|null       |
|104|40         |
+---+-----------+
```

第 101 和 103 列將會被忽略，而且ColumnCorrelation將會是 1.0。

ColumnCount

根據指定的運算式檢查主資料集的資料欄計數。在該運算式中，您可以使用 > 和 < 之類的運算子指定資料欄的數量或範圍。

語法

```
ColumnCount <EXPRESSION>
```

- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：資料欄計數數字檢查

下列範例規則會檢查資料欄計數是否位於指定範圍內。

```
ColumnCount between 10 and 20
```

動態規則範例

- `ColumnCount >= avg(last(10))`
- `ColumnCount between min(last(10))-1 and max(last(10))+1`

ColumnDataType

根據提供的預期類型，檢查指定資料欄中值的固有資料類型。接受 `with threshold` 運算式以檢查資料欄中值的子集。

語法

```
ColumnDataType <COL_NAME> = <EXPECTED_TYPE>
   ColumnDataType <COL_NAME> = <EXPECTED_TYPE> with threshold <EXPRESSION>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的資料欄類型：字串類型

  支援的資料欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- EXPECTED_TYPE：資料欄中值的預期類型。

  支援的值：布林值、日期、時間戳記、整數、雙精度浮點數、浮點數、長整數

  支援的資料欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- EXPRESSION：選用的運算式，用於指定應屬於預期類型之值的百分比。

  支援的資料欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數

範例：使用資料欄資料類型整數作為字串

下列範例規則會檢查指定資料欄中的值 (類型為字串) 是否實際上為整數。

```
ColumnDataType "colA" = "INTEGER"
```

範例：使用資料欄類型整數作為字串以檢查值的子集

下列範例規則會檢查指定資料欄中是否有超過 90% 的值 (類型為字串) 實際上為整數。

```
ColumnDataType "colA" = "INTEGER" with threshold > 0.9
```

ColumnExists

檢查欄是否存在。

語法

```
ColumnExists <COL_NAME>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：任何欄類型

範例：欄存在

下列範例規則會檢查名為 Middle_Name 的欄是否存在。

```
ColumnExists "Middle_Name"
```

ColumnLength

檢查欄中的每一列長度是否符合指定的運算式。

語法

```
ColumnLength <COL_NAME><EXPRESSION>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：字串
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：欄列長度

下列範例規則會檢查名為 Postal_Code 的欄中每一列的值是否為 5 個字元長。

```
ColumnLength "Postal_Code" = 5
ColumnLength "weightinkgs" = 2 where "weightinkgs > 10"
```

空行為

ColumnLength規則會將 NULL s 視為 0 長度字串。對於一NULL行：

```
ColumnLength "Postal_Code" > 4 # this will fail
```

```
ColumnLength "Postal_Code" < 6 # this will succeed
```

下列範例複合規則提供明確失敗NULL值的方法：

```
(ColumnLength "Postal_Code" > 4) AND (ColumnValues != NULL)
```

ColumnNamesMatchPattern

檢查主資料集中所有資料欄的名稱是否與指定的規則運算式相符。

語法

```
ColumnNamesMatchPattern <PATTERN>
```

- PATTERN：您要根據其評估資料品質規則的模式。

  支援的資料欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數

範例：資料欄名稱與模式相符

下列範例規則會檢查是否所有資料欄皆以 "aws_" 字首開頭

```
ColumnNamesMatchPattern "aws_.*"
ColumnNamesMatchPattern "aws_.*" where "weightinkgs > 10"
```

ColumnValues

您可以針對欄中的值執行運算式。

語法

```
ColumnValues <COL_NAME> <EXPRESSION>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：任何欄類型

- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：允許的值

下列範例規則會檢查指定資料行中的每個值是否位於一組允許的值中 (包括 null、空白和只有空格的字串)。

```
ColumnValues "Country" in [ "US", "CA", "UK", NULL, EMPTY, WHITESPACES_ONLY ]
ColumnValues "gender" in ["F", "M"] where "weightinkgs < 10"
```

範例：規則運算式

下列範例規則會根據規則運算式檢查欄中的值。

```
ColumnValues "First_Name" matches "[a-zA-Z]*"
```

範例：日期值

下列範例規則會根據日期運算式檢查日期欄中的值。

```
ColumnValues "Load_Date" > (now() - 3 days)
```

範例：數值

下列範例規則會檢查欄值是否符合特定數值限制條件。

```
ColumnValues "Customer_ID" between 1 and 2000
```

空行為

對於所有ColumnValues規則 (!=和以外NOT IN)，NULL資料列會使規則失敗。如果規則因為 Null 值而失敗，失敗原因會顯示下列內容：

```
Value: NULL does not meet the constraint requirement!
```

下列範例複合規則提供明確允許NULL值的方法：

```
(ColumnValues "Age" > 21) OR (ColumnValues "Age" = NULL)
```

使用!=和not in語法的否 ColumnValues 定規則會傳遞NULL資料列。例如：

```
ColumnValues "Age" != 21
```

```
ColumnValues "Age" not in [21, 22, 23]
```

下列範例提供了一種明確失敗NULL值的方法

```
(ColumnValues "Age" != 21) AND (ColumnValues "Age" != NULL)
```

```
ColumnValues "Age" not in [21, 22, 23, NULL]
```

完整度

根據指定運算式檢查欄中完整 (非空) 值的百分比。

語法

```
Completeness <COL_NAME> <EXPRESSION>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：任何欄類型
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：空值百分比

下列範例規則會檢查欄中 95% 以上的值是否已完成。

```
Completeness "First_Name" > 0.95
Completeness "First_Name" > 0.95 where "weightinkgs > 10"
```

動態規則範例

- `Completeness "colA" between min(last(5)) - 1 and max(last(5)) + 1`
- `Completeness "colA" <= avg(last(10))`

空行為

CSV 資料格式的注意事項：CSV 欄上的空白列可以顯示多個行為。

- 如果資料行屬於類String型，則空白列會被識別為空字串，且不會使Completeness規則失敗。
- 如果資料行屬於另一種資料類型Int，則空白列會被識別為Completeness規則，NULL且會失敗。

CustomSQL

此規則類型已經過延伸，可支援兩種使用案例：

- 針對資料集執行自訂 SQL 陳述式，並根據指定的運算式檢查傳回值。
- 執行自訂 SQL 陳述式，其中您可以在 SELECT 陳述式中指定資料欄名稱，以便與某些條件進行比較來取得資料列層級結果。

語法

```
CustomSql <SQL_STATEMENT> <EXPRESSION>
```

- SQL_STATEMENT – 傳回單一數值的 SQL 陳述式，並以雙引號括住。
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：用於擷取整體規則結果的自訂 SQL

此範例規則使用 SQL 陳述式來擷取資料集的記錄計數。然後規則會檢查記錄計數是否在 10 到 20 之間。

```
CustomSql "select count(*) from primary" between 10 and 20
```

範例：用於擷取資料列層級結果的自訂 SQL

此範例規則使用 SQL 陳述式，其中您可以在 SELECT 陳述式中指定資料欄名稱，以便與某些條件進行比較來取得資料列層級結果。閾值條件運算式定義了導致整個規則失敗的記錄失敗數量閾值。請注意，規則不能同時包含條件和關鍵字。

```
CustomSql "select Name from primary where Age  > 18"
```

或

```
CustomSql "select Name from primary where Age > 18" with threshold  > 3
```

> **Important**
>
> `primary` 別名代表您要評估的資料集名稱。當您在主控台上使用視覺化 ETL 任務時，`primary` 一律代表正在傳遞至 `EvaluateDataQuality.apply()` 轉換的 `DynamicFrame`。當您使用 AWS Glue 資料型錄對表格執行資料品質工作時，`primary`表示該表格。

如果您是使用 AWS Glue Data Catalog，則也可以使用實際的資料表名稱：

```
CustomSql "select count(*) from database.table" between 10 and 20
```

您還可以聯結多個資料表來比較不同的資料元素：

```
CustomSql "select count(*) from database.table inner join database.table2 on id1 = id2"
 between 10 and 20
```

在 AWS Glue ETL 中，CustomSQL 可以識別資料品質檢查失敗的記錄。您需要傳回屬於目前正在評估資料品質之主資料表的記錄，才能使用此功能。作為查詢的一部分傳回的記錄會視為成功，未傳回的記錄則會視為失敗。

下列規則會確保將期限小於 100 的記錄識別為成功，並將超過該數值的記錄標記為失敗。

```
CustomSql "select id from primary where age < 100"
```

若有 50% 的記錄的期限大於 10，此 CustomSQL 規則將通過，且會一併識別失敗的記錄。此 CustomSQL 傳回的記錄將會視為通過，而未傳回的記錄將會視為失敗。

```
CustomSQL "select ID, CustomerID from primary where age > 10" with threshold > 0.5
```

注意：如果您傳回無法在資料集中使用的記錄，CustomSQL 規則將會失敗。

DataFreshness

評估目前時間與日期欄值之間的差異，以檢查欄中資料的更新狀態。您可以為此規則類型指定以時間為基礎的運算式，確保欄值處於最新狀態。

語法

```
DataFreshness <COL_NAME> <EXPRESSION>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：日期
- EXPRESSION - 以小時或天為單位的數值運算式。您必須在運算式中指定時間單位。

範例：資料更新狀態

下列範例規則會檢查資料更新狀態。

```
DataFreshness "Order_Date" <= 24 hours
DataFreshness "Order_Date" between 2 days and 5 days
```

空行為

對於具有NULL值的行，DataFreshness規則將失敗。如果規則因為 Null 值而失敗，失敗原因會顯示下列內容：

```
80.00 % of rows passed the threshold
```

其中，失敗的資料列中有 20% 包含具有的資料列NULL。

下列範例複合規則提供明確允許NULL值的方法：

```
(DataFreshness "Order_Date" <= 24 hours) OR (ColumnValues "Order_Date" = NULL)
```

Amazon S3 物件的資料新鮮度

有時您需要根據 Amazon S3 檔案建立時間驗證資料的新鮮度。為此，您可以使用以下代碼獲取時間戳並將其添加到數據框中，然後應用數據新鮮度檢查。

```
df = glueContext.create_data_frame.from_catalog(database = "default", table_name =
 "mytable")
df = df.withColumn("file_ts", df["_metadata.file_modification_time"])

Rules = [
 DataFreshness "file_ts" < 24 hours
]
```

DatasetMatch

檢查主資料集中的資料是否與參考資料集中的資料相符。系統會使用提供的索引鍵資料欄映射來聯結這兩個資料集。若您只想檢查這些資料欄中的資料是否相等，則可提供其他資料欄映射。請注意，為DataSetMatch了工作，您的連接鍵應該是唯一的，不應為 NULL（必須是主鍵）。如果不滿足這些條件，則會顯示以下錯誤訊息：「提供的索引鍵映射不適合指定的資料框架」。如果您無法擁有唯一的聯結索引鍵，請考慮使用其他規則類型，例如AggregateMatch比對摘要資料。

語法

```
DatasetMatch <REFERENCE_DATASET_ALIAS> <JOIN CONDITION WITH
 MAPPING> <OPTIONAL_MATCH_COLUMN_MAPPINGS> <EXPRESSION>
```

- REFERENCE_DATASET_ALIAS：參考資料集的別名，用於比較來自主資料集的資料。
- KEY_COLUMN_MATION：以逗號分隔的資料欄名稱清單，構成資料集中的索引鍵。如果兩個資料集中的資料欄名稱不相同，您必須使用 -> 加以分隔
- OPTIONAL_MATCH_COLUMN_MAPPINGS：如果您只想檢查某些資料欄中的資料是否相符，則可提供此參數。它使用與索引鍵資料欄映射相同的語法。如果未提供此參數，則會比對所有剩餘資料欄中的資料。其餘的非索引鍵資料欄在兩個資料集中必須具有相同的名稱。
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：使用 ID 資料欄比對集合資料集

下列範例規則會使用 "ID" 資料欄聯結兩個資料集，檢查是否有 90% 以上的主資料集與參考資料集相符。該規則會比較這種情況下的所有資料欄。

```
DatasetMatch "reference" "ID" >= 0.9
```

範例：使用多個索引鍵資料欄比對集合資料集

在下列範例中，主資料集與參考資料集的索引鍵資料欄名稱各不相同。ID_1 和 ID_2 會在主資料集中共同形成一個複合索引鍵。ID_ref1 和 ID_ref2 會在參考資料集中共同形成一個複合索引鍵。在此情況下，您可使用特殊語法來提供資料欄名稱。

```
DatasetMatch "reference" "ID_1->ID_ref1,ID_ref2->ID_ref2" >= 0.9
```

範例：使用多個索引鍵資料欄比對集合資料集，並檢查特定資料欄是否相符

此範例建立在前面的範例之上。我們僅會檢查包含相符數量的資料欄。此資料欄在主資料集中名為 Amount1，在參考資料集中名為 Amount2。您想要取得完全相符的結果。

```
DatasetMatch "reference" "ID_1->ID_ref1,ID_ref2->ID_ref2" "Amount1->Amount2" >= 0.9
```

DistinctValuesCount

根據指定運算式檢查欄中相異值的數目。

語法

```
DistinctValuesCount <COL_NAME> <EXPRESSION>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：任何欄類型
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：相異欄值計數

下列範例規則會檢查名為 State 的欄是否包含 3 個以上的相異值。

```
DistinctValuesCount "State" > 3
DistinctValuesCount "Customer_ID" < 6  where "Customer_ID < 10"
```

動態規則範例

- `DistinctValuesCount "colA" between avg(last(10))-1 and avg(last(10))+1`
- `DistinctValuesCount "colA" <= index(last(10),2) + std(last(5))`

Entropy

檢查欄的熵值是否符合指定的運算式。熵會測量訊息中包含的資訊層級。鑑於欄中值的概率分佈，熵描述了識別值所需的位元數。

語法

```
Entropy <COL_NAME> <EXPRESSION>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：任何欄類型

- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：欄熵

下列範例規則會檢查名為 Feedback 的欄熵值是否大於一。

```
Entropy "Star_Rating" > 1
Entropy "First_Name" > 1  where "Customer_ID < 10"
```

動態規則範例

- `Entropy "colA" < max(last(10))`
- `Entropy "colA" between min(last(10)) and max(last(10))`

IsComplete

檢查欄中的所有值是否完整 (非空)。

語法

```
IsComplete <COL_NAME>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：任何欄類型

範例：空值

下列範例會檢查名為 email 的欄中所有值是否為非空值。

```
IsComplete "email"
IsComplete "Email" where "Customer_ID between 1 and 50"
IsComplete "Customer_ID"  where "Customer_ID < 16 and Customer_ID != 12"
IsComplete "passenger_count" where "payment_type<>0"
```

空行為

CSV 資料格式的注意事項：CSV 欄上的空白列可以顯示多個行為。

- 如果資料行屬於類String型，則空白列會被識別為空字串，且不會使Completeness規則失敗。
- 如果資料行屬於另一種資料類型Int，則空白列會被識別為Completeness規則，NULL且會失敗。

IsPrimaryKey

檢查欄是否包含主索引鍵。如果欄中的所有值都是唯一且完整的 (非空)，則欄包含主索引鍵。

語法

```
IsPrimaryKey <COL_NAME>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：任何欄類型

範例：主索引鍵

下列範例規則會檢查名為 Customer_ID 的欄是否包含主索引鍵。

```
IsPrimaryKey "Customer_ID"
IsPrimaryKey "Customer_ID" where "Customer_ID < 10"
```

範例：具有多個資料欄的主索引鍵。以下任意範例都有效。

```
IsPrimaryKey "colA" "colB"
IsPrimaryKey "colA" "colB" "colC"
IsPrimaryKey colA "colB" "colC"
```

IsUnique

檢查欄中的所有值是否是唯一的，並傳回布林值。

語法

```
IsUnique <COL_NAME>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：任何欄類型

範例：唯一欄值

下列範例規則會檢查名為 email 的欄中所有值是否是唯一的。

```
IsUnique "email"
IsUnique "Customer_ID" where "Customer_ID < 10"]
```

Mean

檢查欄中所有值的平均值是否符合指定的運算式。

語法

```
Mean <COL_NAME> <EXPRESSION>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：平均值

下列範例規則會檢查欄中所有值的平均值是否超過閾值。

```
Mean "Star_Rating" > 3
Mean "Salary" < 6200 where "Customer_ID < 10"
```

動態規則範例

- `Mean "colA" > avg(last(10)) + std(last(2))`
- `Mean "colA" between min(last(5)) - 1 and max(last(5)) + 1`

空行為

該Mean規則將忽略平均NULL值計算中具有值的行。例如：

```
+---+-----------+
|id |units      |
+---+-----------+
|100|0          |
|101|null       |
|102|20         |
|103|null       |
|104|40         |
+---+-----------+
```

列的平均值units將是（0 + 20 + 40）/3 = 20。此計算中不會考慮列 101 和 103 列。

ReferentialIntegrity

檢查主資料集中資料欄集值在多大程度上為參考資料集中資料欄集的值子集。

語法

```
ReferentialIntegrity <PRIMARY_COLS> <REFERENCE_DATASET_COLS> <EXPRESSION>
```

- PRIMARY_COLS：主資料集中以逗號分隔的資料欄名稱清單。

  支援的資料欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- REFERENCE_DATASET_COLS：此參數包含以句號分隔的兩個部分。第一部分是參考資料集的別名。第二部分是在以大括號括住的參考資料集中以逗號分隔的資料欄名稱清單。

  支援的欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：檢查郵遞區號資料欄的參照完整性

下列範例規則會檢查在主資料集的 zipcode 資料欄中，是否有超過 90% 的值存在於 reference 資料集的 zipcode 資料欄中。

```
ReferentialIntegrity "zipcode" "reference.zipcode" >= 0.9
```

範例：檢查城市和州/省資料欄的參照完整性

在下列範例中，包含城市和州/省資訊的資料欄存在於主資料集和參考資料集中。這兩個資料集中的資料欄名稱各不相同。該規則會檢查主資料集中的資料欄值集是否與參考資料集中的資料欄值集完全相等。

```
ReferentialIntegrity "city,state" "reference.{ref_city,ref_state}" = 1.0
```

動態規則範例

- `ReferentialIntegrity "city,state" "reference.{ref_city,ref_state}" > avg(last(10))`
- `ReferentialIntegrity "city,state" "reference.{ref_city,ref_state}" between min(last(10)) - 1 and max(last(10)) + 1`

RowCount

根據指定的運算式檢查資料集的列計數。在運算式中，您可以使用 > 和 < 之類的運算子指定列數或列的範圍。

語法

```
RowCount <EXPRESSION>
```

- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：列計數數字檢查

下列範例規則會檢查列計數是否在指定範圍內。

```
RowCount between 10 and 100
RowCount between 1 and 50 where "Customer_ID < 10"
```

動態規則範例

```
RowCount > avg(lats(10)) *0.8
```

RowCountMatch

根據指定的運算式，檢查主資料集資料欄計數與參考資料集資料列計數的比率。

語法

```
RowCountMatch <REFERENCE_DATASET_ALIAS> <EXPRESSION>
```

- REFERENCE_DATASET_ALIAS：比較資料列計數的參考資料集別名。

  支援的欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：針對參考資料集檢查資料列計數

下列範例規則會檢查主資料集的資料列計數是否至少為參考資料集資料列計數的 90%。

```
RowCountMatch "reference" >= 0.9
```

StandardDeviation

根據指定運算式檢查欄中所有值的標準差。

語法

```
StandardDeviation <COL_NAME> <EXPRESSION>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：標準差

下列範例規則會檢查名為 colA 的欄中的值標準差是否小於指定值。

```
StandardDeviation "Star_Rating" < 1.5
StandardDeviation "Salary" < 3500 where "Customer_ID < 10"
```

動態規則範例

- `StandardDeviation "colA" > avg(last(10) + 0.1`
- `StandardDeviation "colA" between min(last(10)) - 1 and max(last(10)) + 1`

空行為

StandardDeviation規則將忽略標準差計算中具有NULL值的列。例如：

```
+---+-----------+-----------+
|id |units1     |units2     |
+---+-----------+-----------+
|100|0          |0          |
|101|null       |0          |
|102|20         |20         |
|103|null       |0          |
|104|40         |40         |
+---+-----------+-----------+
```

列的標準差不units1會考慮第 101 和 103 行，並導致到 16.33。列的標準偏差units2將導致 16。

總和

根據指定運算式檢查欄中所有值的總和。

語法

```
Sum <COL_NAME> <EXPRESSION>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：總和

下列範例規則會檢查欄中所有值的總和是否超過指定的閾值。

```
Sum "transaction_total" > 500000
Sum "Salary" < 55600 where "Customer_ID < 10"
```

動態規則範例

- `Sum "ColA" > avg(last(10))`
- `Sum "colA" between min(last(10)) - 1 and max(last(10)) + 1`

空行為

該Sum規則將忽略總和計算中具有NULL值的行。例如：

```
+---+-----------+
|id |units      |
+---+-----------+
|100|0          |
|101|null       |
|102|20         |
|103|null       |
|104|40         |
+---+-----------+
```

列的總和units將不會考慮第 101 和 103 行，結果為（0 + 20 + 40）= 60。

SchemaMatch

檢查主資料集的結構描述是否與參考資料集的結構描述相符。結構描述檢查會以逐個資料欄的形式完成。如果名稱與類型皆完全相同，表示兩個資料欄的結構描述相符。資料欄的順序無關緊要。

語法

```
SchemaMatch <REFERENCE_DATASET_ALIAS> <EXPRESSION>
```

- REFERENCE_DATASET_ALIAS：比較結構描述的參考資料集別名。

  支援的欄類型：位元組、小數、雙精度浮點數、浮點數、整數、長整數、短整數
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例： SchemaMatch

下列範例規則會檢查主資料集的結構描述是否與參考資料集的結構描述完全相符。

```
SchemaMatch "reference" = 1.0
```

唯一性

根據指定運算式檢查欄中唯一值的百分比。唯一值正好出現一次。

語法

```
Uniqueness <COL_NAME> <EXPRESSION>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：任何欄類型
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：唯一性百分比

下列範例規則會檢查欄中唯一值的百分比是否符合特定數值條件。

```
Uniqueness "email" = 1.0
Uniqueness "Customer_ID" != 1.0 where "Customer_ID < 10"
```

動態規則範例

- `Uniqueness "colA" between min(last(10)) and max(last(10))`
- `Uniqueness "colA" >= avg(last(10))`

UniqueValueRatio

根據指定運算式檢查欄的唯一值比率。唯一值比率是唯一值數目除以欄中所有相異值數目的分數。唯一值正好出現一次，而相異值至少出現一次。

例如，集合 `[a, a, b]` 包含一個唯一值 (b) 和兩個相異值 (a 和 b)。所以集合的唯一值比率是 ½ = 0.5。

語法

```
UniqueValueRatio <COL_NAME> <EXPRESSION>
```

- COL_NAME – 您要評估資料品質規則的欄名稱。

  支援的欄類型：任何欄類型
- EXPRESSION – 針對規則類型回應執行的運算式，以產生布林值。如需詳細資訊，請參閱 表達式。

範例：唯一值比率

此範例會根據值範圍來檢查資料欄的唯一值比率。

```
UniqueValueRatio "test_score" between 0 and 0.5
UniqueValueRatio "Customer_ID" between 0 and 0.9 where "Customer_ID < 10"
```

動態規則範例

- `UniqueValueRatio "colA" > avg(last(10))`
- `UniqueValueRatio "colA" <= index(last(10),2) + std(last(5))`

DetectAnomalies

偵測指定資料品質規則的異常情況。每次執行 DetectAnomalies 規則都會為指定規則儲存評估值。當收集到足夠的資料時，異常偵測演算法會擷取該指定規則的所有歷史資料，並執行異常偵測。DetectAnomalies 偵測到異常時，規則會失敗。可以從觀察中獲得偵測到的異常的更多資訊。

語法

```
    DetectAnomalies <RULE_NAME> <RULE_PARAMETERS>
```

RULE_NAME：您希望評估和偵測異常的規則名稱。支援的規則：

- "RowCount"
- "Completeness"
- "Uniqueness"
- "Mean"
- "Sum"
- "StandardDeviation"
- "Entropy"
- "DistinctValuesCount"
- "UniqueValueRatio"
- "ColumnLength"

- "ColumnValues"
- "ColumnCorrelation"

RULE_PARAMETERS：某些規則需要其他參數才能執行。請參閱指定的規則文件查看所需的參數。

範例：的異常 RowCount

例如，如果我們想要偵測 RowCount 異常，我們會以規則名稱的 RowCount 形式提供。

```
DetectAnomalies "RowCount"
```

範例：的異常 ColumnLength

例如，如果我們想要偵測 ColumnLength 異常，我們會以規則名稱和欄名稱的 ColumnLength 形式提供。

```
DetectAnomalies "ColumnLength" "id"
```

使用 API 測量和管理資料品質

本主題介紹如何使用 API 測量和管理資料品質。

內容

先決條件

- 確保 boto3 為最新版本，這樣它才會包含最新的 AWS Glue Data Quality API。
- 確保 AWS CLI 為最新版本，以便包含最新的 CLI。

如果您使用 AWS Glue 任務執行這些 API，您可以使用下列選項，將 boto3 程式庫更新為最新版本：

```
–additional-python-modules boto3==<version>
```

使用 AWS Glue Data Quality 建議

若要啟動 AWS Glue Data Quality 建議執行：

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

    def start_data_quality_rule_recommendation_run(self, database_name, table_name,
 role_arn):
        """
        Starts a recommendation run that is used to generate rules when you don't
 know what rules to write. AWS Glue Data Quality analyzes the data and comes up with
 recommendations for a potential ruleset. You can then triage the ruleset and modify
 the generated ruleset to your liking.

        :param database_name: The name of the AWS Glue database which contains the
 dataset.
        :param table_name: The name of the AWS Glue table against which we want a
 recommendation
        :param role_arn: The Amazon Resource Name (ARN) of an AWS Identity and Access
 Management (IAM) role that grants permission to let AWS Glue access the resources it
 needs.

        """
        try:
            response = self.client.start_data_quality_rule_recommendation_run(
                DataSource={
                    'GlueTable': {
                        'DatabaseName': database_name,
                        'TableName': table_name
                    }
                },
                Role=role_arn
            )
        except ClientError as err:
            logger.error(
```

```
                "Couldn't start data quality recommendation run %s. Here's why: %s:
 %s", name,
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response['RunId']
```

對於建議執行，您可以使用 pushDownPredicates 或 catalogPartitionPredicates 來改善效能，並僅在型錄來源的特定分割區上執行建議。

```
client.start_data_quality_rule_recommendation_run(
            DataSource={
                'GlueTable': {
                    'DatabaseName': database_name,
                    'TableName': table_name,
                    'AdditionalOptions': {
                        'pushDownPredicate': "year=2022"
                    }
                }
            },
            Role=role_arn,
            NumberOfWorkers=2,
            CreatedRulesetName='<rule_set_name>'
  )
```

若要取得 AWS Glue Data Quality 建議執行的結果：

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def get_data_quality_rule_recommendation_run(self, run_id):
        """
        Gets the specified recommendation run that was used to generate rules.

        :param run_id: The id of the data quality recommendation run

        """
        try:
```

```
            response =
 self.client.get_data_quality_rule_recommendation_run(RunId=run_id)
        except ClientError as err:
            logger.error(
                "Couldn't get data quality recommendation run %. Here's why: %s: %s",
 run_id,
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

從上面的回應物件中，您可以擷取執行建議的 RuleSet，以在後續步驟中使用：

```
print(response['RecommendedRuleset'])

Rules = [
    RowCount between 2000 and 8000,
    IsComplete "col1",
    IsComplete "col2",
    StandardDeviation "col3" between 58138330.8 and 64258155.09,
    ColumnValues "col4" between 1000042965 and 1214474826,
    IsComplete "col5"
]
```

若要取得可篩選並列出之所有建議執行的清單：

```
response = client.list_data_quality_rule_recommendation_runs(
    Filter={
        'DataSource': {
            'GlueTable': {
                'DatabaseName': '<database_name>',
                'TableName': '<table_name>'
            }
        }
)
```

若要取消現有的 AWS Glue Data Quality 建議任務：

```
response = client.cancel_data_quality_rule_recommendation_run(
    RunId='dqrun-d4b6b01957fdd79e59866365bf9cb0e40fxxxxxxx'
)
```

使用 AWS Glue Data Quality 規則集

若要建立 AWS Glue Data Quality 規則集：

```
response = client.create_data_quality_ruleset(
    Name='<ruleset_name>',
    Ruleset='Rules = [IsComplete "col1", IsPrimaryKey "col2", RowCount between 2000 and
 8000]',
    TargetTable={
        'TableName': '<table_name>',
        'DatabaseName': '<database_name>'
    }
)
```

若要取得資料品質規則集：

```
response = client.get_data_quality_ruleset(
    Name='<ruleset_name>'
)
print(response)
```

您可以使用此 API 擷取規則集：

```
print(response['Ruleset'])
```

若要列出資料表的所有資料品質規則集：

```
response = client.list_data_quality_rulesets()
```

您可以使用 API 中的篩選條件，篩選在特定資料庫或資料表中附加的所有規則集：

```
response = client.list_data_quality_rulesets(
    Filter={
        'TargetTable': {
            'TableName': '<table_name>',
            'DatabaseName': '<database_name>'
        }
    },
)
```

若要更新資料品質規則集：

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def update_data_quality_ruleset(self, ruleset_name, ruleset_string):
        """
        Update an AWS Glue Data Quality Ruleset

        :param ruleset_name: The name of the AWS Glue Data Quality ruleset to update
        :param ruleset_string: The DQDL ruleset string to update the ruleset with

        """
        try:
            response = self.client.update_data_quality_ruleset(
                Name=ruleset_name,
                Ruleset=ruleset_string
            )
        except ClientError as err:
            logger.error(
                "Couldn't update the AWS Glue Data Quality ruleset. Here's why: %s:
 %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

若要刪除資料品質規則集：

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def delete_data_quality_ruleset(self, ruleset_name):
        """
        Delete a AWS Glue Data Quality Ruleset
```

```
        :param ruleset_name: The name of the AWS Glue Data Quality ruleset to delete

        """
        try:
            response = self.client.delete_data_quality_ruleset(
                Name=ruleset_name
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete the AWS Glue Data Quality ruleset. Here's why: %s:
%s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

使用 AWS Glue Data Quality 執行

若要啟動 AWS Glue Data Quality 執行：

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def start_data_quality_ruleset_evaluation_run(self, database_name, table_name,
 role_name, ruleset_list):
        """
        Start an AWS Glue Data Quality evaluation run

        :param database_name: The name of the AWS Glue database which contains the
 dataset.
        :param table_name: The name of the AWS Glue table against which we want to
 evaluate.
        :param role_arn: The Amazon Resource Name (ARN) of an AWS Identity and Access
 Management (IAM) role that grants permission to let AWS Glue access the resources it
 needs.
        :param ruleset_list: The list of AWS Glue Data Quality ruleset names to
 evaluate.
```

```
        """
        try:
            response = client.start_data_quality_ruleset_evaluation_run(
                DataSource={
                    'GlueTable': {
                        'DatabaseName': database_name,
                        'TableName': table_name
                    }
                },
                Role=role_name,
                RulesetNames=ruleset_list
            )
        except ClientError as err:
            logger.error(
                "Couldn't start the AWS Glue Data Quality Run. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response['RunId']
```

請記住，您可以傳遞 pushDownPredicate 或 catalogPartitionPredicate 參數，以確保資料品質執行只鎖定型錄資料表中的一組特定分割區。例如：

```
response = client.start_data_quality_ruleset_evaluation_run(
    DataSource={
        'GlueTable': {
            'DatabaseName': '<database_name>',
            'TableName': '<table_name>',
            'AdditionalOptions': {
                'pushDownPredicate': 'year=2023'
            }
        }
    },
    Role='<role_name>',
    NumberOfWorkers=5,
    Timeout=123,
    AdditionalRunOptions={
        'CloudWatchMetricsEnabled': False
    },
    RulesetNames=[
        '<ruleset_name>',
    ]
```

```
)
```

若要取得有關 AWS Glue Data Quality 執行的資訊：

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def get_data_quality_ruleset_evaluation_run(self, run_id):
        """
        Get details about an AWS Glue Data Quality Run

        :param run_id: The AWS Glue Data Quality run ID to look up

        """
        try:
            response = self.client.get_data_quality_ruleset_evaluation_run(
                RunId=run_id
            )
        except ClientError as err:
            logger.error(
                "Couldn't look up the AWS Glue Data Quality run ID. Here's why: %s:
 %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

若要從 AWS Glue Data Quality 執行中取得結果：

對於指定的 AWS Glue Data Quality 執行，您可以使用下列方法擷取執行評估的結果：

```
response = client.get_data_quality_ruleset_evaluation_run(
    RunId='d4b6b01957fdd79e59866365bf9cb0e40fxxxxxxx'
)

resultID = response['ResultIds'][0]

response = client.get_data_quality_result(
```

```
    ResultId=resultID
)

print(response['RuleResults'])
```

若要列出所有 AWS Glue Data Quality 執行：

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def list_data_quality_ruleset_evaluation_runs(self, database_name, table_name):
        """
        Lists all the AWS Glue Data Quality runs against a given table

        :param database_name: The name of the database where the data quality runs
        :param table_name: The name of the table against which the data quality runs
 were created

        """
        try:
            response = self.client.list_data_quality_ruleset_evaluation_runs(
                Filter={
                    'DataSource': {
                        'GlueTable': {
                            'DatabaseName': database_name,
                            'TableName': table_name
                        }
                    }
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't list the AWS Glue Quality runs. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

您可以修改篩選子句，僅顯示特定時間之間或針對特定資料表執行的結果。

若要停止進行中的 AWS Glue Data Quality 執行：

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def cancel_data_quality_ruleset_evaluation_run(self, result_id):
        """
        Cancels a given AWS Glue Data Quality run

        :param result_id: The result id of a AWS Glue Data Quality run to cancel

        """
        try:
            response = self.client.cancel_data_quality_ruleset_evaluation_run(
                ResultId=result_id
            )
        except ClientError as err:
            logger.error(
                "Couldn't cancel the AWS Glue Data Quality run. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

使用 AWS Glue Data Quality 結果

若要取得 AWS Glue Data Quality 執行結果：

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client
```

```
    def get_data_quality_result(self, result_id):
        """
        Outputs the result of an AWS Glue Data Quality Result

        :param result_id: The result id of an AWS Glue Data Quality run

        """
        try:
            response = self.client.get_data_quality_result(
                ResultId=result_id
            )
        except ClientError as err:
            logger.error(
                "Couldn't get the AWS Glue Data Quality result. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

若要取消現有的 AWS Glue Data Quality 建議任務：

假設有 AWS Glue Data Quality 執行 ID，您就可以擷取結果 ID，然後取得實際結果，如下所示：

```
response = client.get_data_quality_ruleset_evaluation_run(
    RunId='dqrun-abca77ee126abe1378c1da1ae0750xxxxxxxx'
)

resultID = response['ResultIds'][0]

response = client.get_data_quality_result(
    ResultId=resultID
)

print(resp['RuleResults'])
```

設定提醒、部署和排程

本主題說明如何設定 AWS Glue 資料品質的警示、部署和排程。

內容

  - 將通知格式化為電子郵件
- 在 CloudWatch 整合中設定警示和通知
- 查詢資料品質結果以建置儀表板
- 使用部署資料品質規則 AWS CloudFormation
- 排程資料品質規則

在 Amazon EventBridge 整合中設定警示和通知

AWS 「Glue 資料品質」支援發佈 EventBridge 事件，這些事件會在資料品質規則集評估執行完成時發出。如此您就可以輕鬆設定資料品質規則失敗時的提醒。

以下是在資料型錄中評估資料品質規則集時的範例事件。有了這些資訊，您可以檢閱 Amazon 提供的資料 EventBridge。您可以發出其他 API 呼叫以取得更多詳細資訊。例如，使用結果 ID 呼叫 `get_data_quality_result` API，以取得特定執行的詳細資訊。

```
{
    "version":"0",
    "id":"abcdef00-1234-5678-9abc-def012345678",
    "detail-type":"Data Quality Evaluation Results Available",
    "source":"aws.glue-dataquality",
    "account":"123456789012",
    "time":"2017-09-07T18:57:21Z",
    "region":"us-west-2",
    "resources":[],
    "detail":{
        "context": {
                    "contextType": "GLUE_DATA_CATALOG",
                    "runId":"dqrun-12334567890",
                    "databaseName": "db-123",
                    "tableName": "table-123",
                    "catalogId": "123456789012"
                    },
        "resultID": "dqresult-12334567890",
        "rulesetNames": ["rulset1"],
        "state":"SUCCEEDED",
        "score": 1.00,
        "rulesSucceeded": 100,
        "rulesFailed": 0,
        "rulesSkipped": 0
    }
```

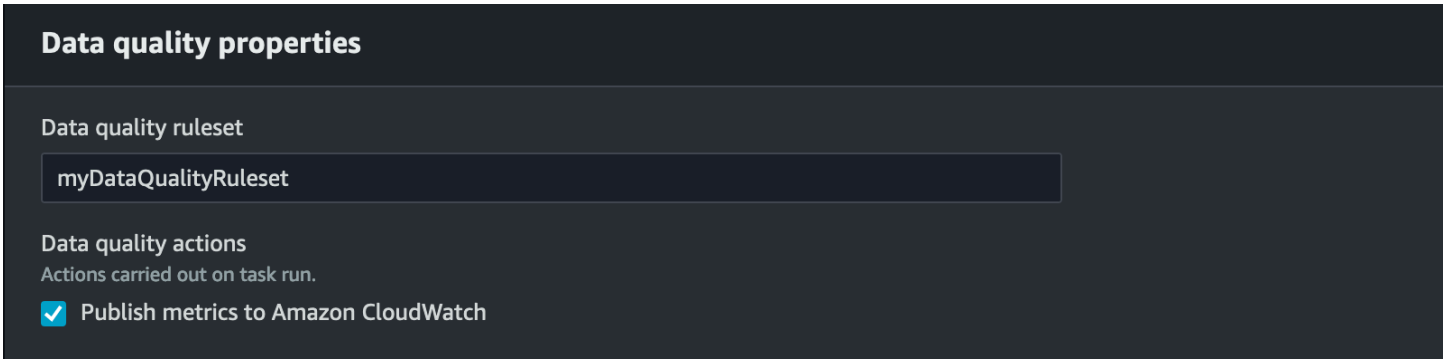
```
}
```

以下是當您在 AWS Glue ETL 或 AWS Glue Studio 筆記本中評估資料品質規則集時發佈的範例事件。

```
{
    "version":"0",
    "id":"abcdef00-1234-5678-9abc-def012345678",
    "detail-type":"Data Quality Evaluation Results Available",
    "source":"aws.glue-dataquality",
    "account":"123456789012",
    "time":"2017-09-07T18:57:21Z",
    "region":"us-west-2",
    "resources":[],
    "detail":{
        "context": {
                    "contextType": "GLUE_JOB",
                    "jobId": "jr-12334567890",
                    "jobName": "dq-eval-job-1234",
                    "evaluationContext": "",
                    }
        "resultID": "dqresult-12334567890",
        "rulesetNames": ["rulset1"],
        "state":"SUCCEEDED",
        "score": 1.00
        "rulesSucceeded": 100,
        "rulesFailed": 0,
        "rulesSkipped": 0
    }
}
```

對於「資料品質」評估同時在「資料目錄」和 ETL 工作中執行，「將量度發佈至 Amazon CloudWatch」選項 (預設為選取) 必須保持選取狀態，才能發 EventBridge 佈才能正常工作。

設定 EventBridge 通知



若要接收發出的事件並定義目標，您必須設定 Amazon EventBridge 規則。若要建立規則：

1. 打開 Amazon EventBridge 控制台。
2. 在導覽列的匯流排區段下選擇規則。
3. 選擇 Create Rule (建立規則)。
4. 在定義規則詳細資訊上：
   a. 對於名稱，輸入 `myDQRule`。
   b. 輸入描述 (選用)。
   c. 對於事件匯流排，請選取您的事件匯流排。如果沒有事件匯流排，請保留其預設值。
   d. 對於規則類型，選取具有事件模式的規則，然後選擇下一步。
5. 在建置事件模式上：
   a. 對於事件來源，請選取AWS 事件或 EventBridge 夥伴事件。
   b. 略過示範事件區段。
   c. 對於建立方法，選取使用模式表單。
   d. 對於事件模式：
      i. 選取事件來源的 AWS 服務。
      ii. 選取「Glue 合資料品質」進行 AWS 維修。
      iii. 對於事件類型，選取可用的資料品質評估結果。
      iv. 對於特定狀態，選取失敗。然後您會看到類似以下內容的事件模式：

```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
  "detail": {
    "state": ["FAILED"]
  }
```

```
}
```

v. 如需更多設定選項，請參閱 事件模式的其他組態選項。

6. 在選取目標上：

a. 對於目標類型，選取 AWS 服務。

b. 使用 [選取目標] 下拉式清單選擇您想要連線的 AWS 服務 (SNS、Lambda、SQS 等)，然後選擇 [下一步]。

7. 在設定標籤上按一下新增標籤以新增選用標籤，然後選擇下一步。

8. 您會看到所有選取項目的摘要頁面。選擇底部的建立規則。

事件模式的其他組態選項

除了根據成功或失敗篩選事件之外，您可能還想要根據不同參數進一步篩選事件。

若要這麼做，請前往「事件模式」區段，然後選取編輯模式以指定其他參數。請注意，事件模式中的欄位需區分大小寫。以下是設定事件模式的範例。

若要從評估特定規則集的特定資料表擷取事件，請使用此類型的模式：

```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
  "detail": {
    "context": {
      "contextType": ["GLUE_DATA_CATALOG"],
      "databaseName": "db-123",
       "tableName": "table-123",
    },
    "rulesetNames": ["ruleset1", "ruleset2"]
    "state": ["FAILED"]
  }
}
```

若要從 ETL 體驗中的特定任務擷取事件，請使用此類型的模式：

```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
  "detail": {
```

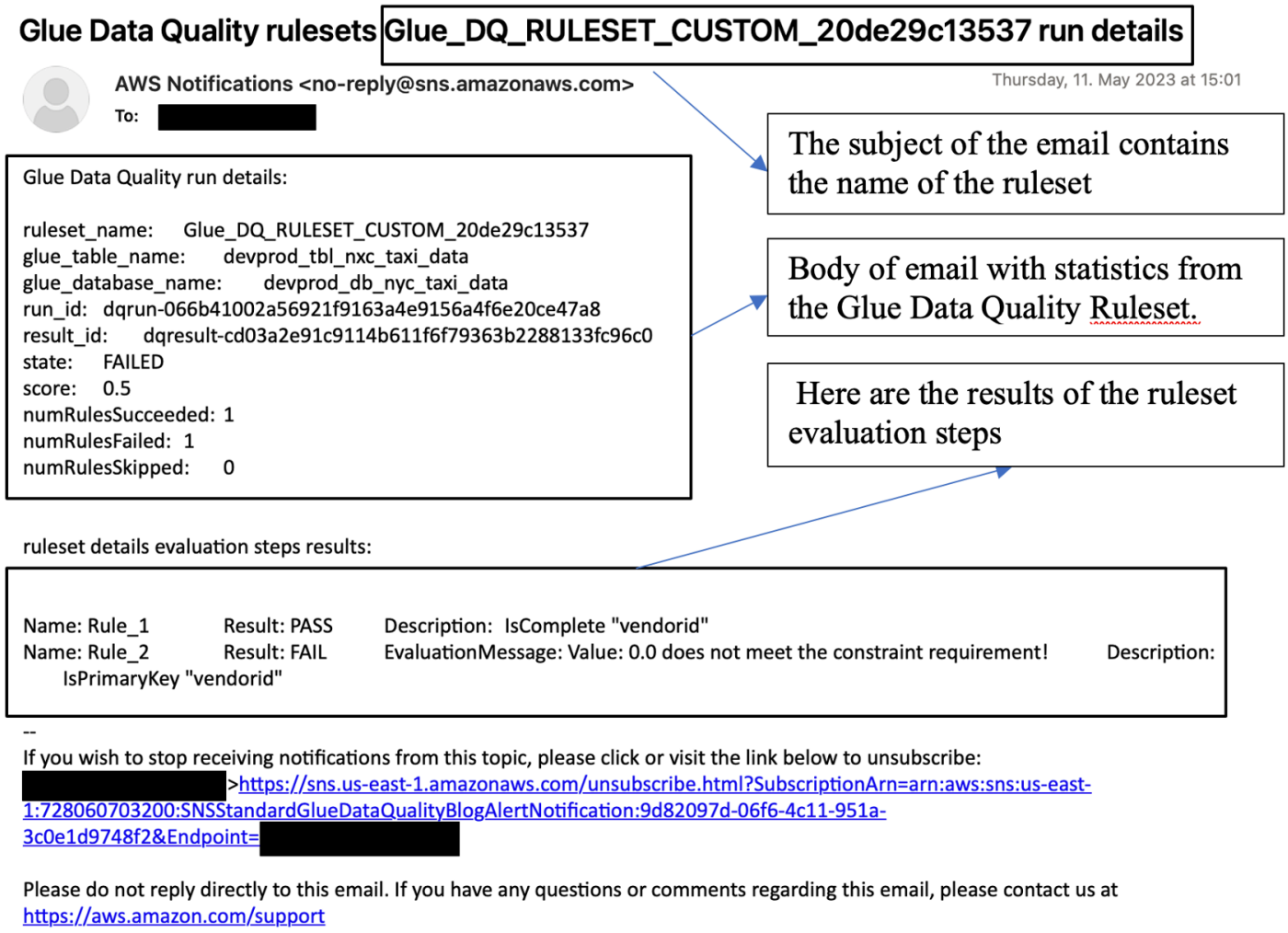
```
    "context": {
      "contextType": ["GLUE_JOB"],
      "jobName": ["dq_evaluation_job1", "dq_evaluation_job2"]
    },
    "state": ["FAILED"]
  }
}
```

若要擷取分數低於特定閾值 (例如 70%) 的事件：

```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
  "detail": {
    "score": [{
      "numeric": ["<=", 0.7]
    }]
  }
}
```

將通知格式化為電子郵件

您有時需要向業務團隊傳送格式良好的電子郵件通知。您可以使用 Amazon EventBridge 和 AWS Lambda 來實現這一目標。



下列範例程式碼可用來格式化資料品質通知以產生電子郵件。

```
import boto3
import json
from datetime import datetime

sns_client = boto3.client('sns')
glue_client = boto3.client('glue')

sns_topic_arn = 'arn:aws:sns:<region-code>:<account-id>:<sns-topic-name>'



def lambda_handler(event, context):
    log_metadata = {}
    message_text = ""
```

```
    subject_text = ""

    if event['detail']['context']['contextType'] == 'GLUE_DATA_CATALOG':
        log_metadata['ruleset_name'] = str(event['detail']['rulesetNames'][0])
        log_metadata['tableName'] = str(event['detail']['context']['tableName'])
        log_metadata['databaseName'] = str(event['detail']['context']['databaseName'])
        log_metadata['runId'] = str(event['detail']['context']['runId'])
        log_metadata['resultId'] = str(event['detail']['resultId'])
        log_metadata['state'] = str(event['detail']['state'])
        log_metadata['score'] = str(event['detail']['score'])
        log_metadata['numRulesSucceeded'] = str(event['detail']['numRulesSucceeded'])
        log_metadata['numRulesFailed'] = str(event['detail']['numRulesFailed'])
        log_metadata['numRulesSkipped'] = str(event['detail']['numRulesSkipped'])

        message_text += "Glue Data Quality run details:\n"
        message_text += "ruleset_name: {}\n".format(log_metadata['ruleset_name'])
        message_text += "glue_table_name: {}\n".format(log_metadata['tableName'])
        message_text += "glue_database_name: {}\n".format(log_metadata['databaseName'])
        message_text += "run_id: {}\n".format(log_metadata['runId'])
        message_text += "result_id: {}\n".format(log_metadata['resultId'])
        message_text += "state: {}\n".format(log_metadata['state'])
        message_text += "score: {}\n".format(log_metadata['score'])
        message_text += "numRulesSucceeded:
{}\n".format(log_metadata['numRulesSucceeded'])
        message_text += "numRulesFailed: {}\n".format(log_metadata['numRulesFailed'])
        message_text += "numRulesSkipped: {}\n".format(log_metadata['numRulesSkipped'])

        subject_text = "Glue Data Quality ruleset {} run
details".format(log_metadata['ruleset_name'])

    else:
        log_metadata['ruleset_name'] = str(event['detail']['rulesetNames'][0])
        log_metadata['jobName'] = str(event['detail']['context']['jobName'])
        log_metadata['jobId'] = str(event['detail']['context']['jobId'])
        log_metadata['resultId'] = str(event['detail']['resultId'])
        log_metadata['state'] = str(event['detail']['state'])
        log_metadata['score'] = str(event['detail']['score'])

        log_metadata['numRulesSucceeded'] = str(event['detail']['numRulesSucceeded'])
        log_metadata['numRulesFailed'] = str(event['detail']['numRulesFailed'])
        log_metadata['numRulesSkipped'] = str(event['detail']['numRulesSkipped'])

        message_text += "Glue Data Quality run details:\n"
        message_text += "ruleset_name: {}\n".format(log_metadata['ruleset_name'])
```

```
        message_text += "glue_job_name: {}\n".format(log_metadata['jobName'])
        message_text += "job_id: {}\n".format(log_metadata['jobId'])
        message_text += "result_id: {}\n".format(log_metadata['resultId'])
        message_text += "state: {}\n".format(log_metadata['state'])
        message_text += "score: {}\n".format(log_metadata['score'])
        message_text += "numRulesSucceeded:
{}\n".format(log_metadata['numRulesSucceeded'])
        message_text += "numRulesFailed: {}\n".format(log_metadata['numRulesFailed'])
        message_text += "numRulesSkipped: {}\n".format(log_metadata['numRulesSkipped'])

        subject_text = "Glue Data Quality ruleset {} run
details".format(log_metadata['ruleset_name'])

   resultID = str(event['detail']['resultId'])
   response = glue_client.get_data_quality_result(ResultId=resultID)
   RuleResults = response['RuleResults']
   message_text += "\n\nruleset details evaluation steps results:\n\n"
   subresult_info = []

   for dic in RuleResults:
        subresult = "Name: {}\t\tResult: {}\t\tDescription: \t{}".format(dic['Name'],
dic['Result'], dic['Description'])
        if 'EvaluationMessage' in dic:
            subresult += "\t\tEvaluationMessage: {}".format(dic['EvaluationMessage'])
        subresult_info.append({
            'Name': dic['Name'],
            'Result': dic['Result'],
            'Description': dic['Description'],
            'EvaluationMessage': dic.get('EvaluationMessage', '')
        })
        message_text += "\n" + subresult

   log_metadata['resultrun'] = subresult_info



   sns_client.publish(
        TopicArn=sns_topic_arn,
        Message=message_text,
        Subject=subject_text
   )

   return {
        'statusCode': 200,
```

```
        'body': json.dumps('Message published to SNS topic')
    }
```

在 CloudWatch 整合中設定警示和通知

我們建議的方法是使用 Amazon 設定資料品質提醒 EventBridge，因為 Amazon EventBridge 需要一次性設定來提醒客戶。但是， CloudWatch 由於熟悉，一些客戶更喜歡 Amazon。對於此類客戶，我們提供與 Amazon 的集成 CloudWatch。

每個「 AWS Glue 資料品質」評估都會在每次資料品質執行時發出一對名為 `glue.data.quality.rules.passed` (表示通過的規則數目) 和 `glue.data.quality.rules.failed` (指出失敗規則的數目) 的測量結果。您可以使用發出的此指標來建立警示，以在指定的資料品質執行低於閾值時提醒使用者。若要開始設定將透過 Amazon SNS 通知傳送電子郵件的警示，請依照以下步驟操作：

若要開始設定將透過 Amazon SNS 通知傳送電子郵件的警示，請依照以下步驟操作：

1. 打開 Amazon CloudWatch 控制台。
2. 選擇指標下的所有指標。您將在標題為 "Glue Data Quality" 的自訂命名空間下看到額外的命名空間。

   > **Note**
   >
   > 開始執行 AWS Glue 資料品質時，請確定已啟用「將指標發佈到 Amazon」 CloudWatch 核取方塊。否則，該特定運行的指標將不會發佈到 Amazon CloudWatch。

   在 `Glue Data Quality` 命名空間下，您可以看到每個資料表中依規則集發出的指標。本主題的目的在於，如果此值超過 1 (表示如果我們看到失敗規則評估的數量大於 1，我們希望收到通知)，我們將使用 `glue.data.quality.rules.failed` 規則和警示。
3. 若要建立警示，請選擇警示下的所有警示。
4. 選擇 Create alarm (建立警示)。
5. 選擇 Select metric (選取指標)。
6. 選取與您建立之資料表對應的 `glue.data.quality.rules.failed` 指標，然後選擇選取指標。
7. 在指標區段下的指定指標和條件索引標籤下：

   a. 在 Statistic (統計資料) 中選擇 Sum (總和)。

b. 對於期間，選擇 1 分鐘。

8. 在條件區段下：

a. 對於閾值類型，選擇靜態。

b. 對於每當 glue.data.quality.rules.failed 為…，選取大於/等於。

c. 對於比…，輸入 1 作為閾值。

這些選擇意味著，如果 `glue.data.quality.rules.failed` 指標發出的值大於或等於 1，我們將觸發警示。但是，如果沒有資料，我們會將其視為可接受。

9. 選擇下一步。

10.在設定動作上：

a. 對於警示狀態觸發區段，選擇警示中。

b. 對於將通知傳送至下列 SNS 主題區段，選擇建立新主題以透過新的 SNS 主題傳送通知。

c. 對於將接收通知的電子郵件端點，輸入電子郵件地址。然後按一下建立主題。

d. 選擇下一步。

11對於警示名稱，輸入 `myFirstDQAlarm`，然後選擇下一步。

12您會看到所有選取項目的摘要頁面。選擇底部的建立警示。

現在，您可以從 Amazon CloudWatch 警報儀表板看到正在創建的警報。

查詢資料品質結果以建置儀表板

您可能想要建置儀表板以顯示資料品質結果。有兩種方式可以進行：

EventBridge 使用以下代碼設置 Amazon 以將數據寫入 Amazon S3：

```
import boto3
import json
from datetime import datetime


s3_client = boto3.client('s3')
glue_client = boto3.client('glue')


s3_bucket = 's3-bucket-name'

def write_logs(log_metadata):
```

```
    try:
        filename = datetime.now().strftime("%m%d%Y%H%M%S") + ".json"
        key_opts = {
            'year': datetime.now().year,
            'month': "{:02d}".format(datetime.now().month),
            'day': "{:02d}".format(datetime.now().day),
            'filename': filename
        }
        s3key = "gluedataqualitylogs/year={year}/month={month}/day={day}/
{filename}".format(**key_opts)
        s3_client.put_object(Bucket=s3_bucket, Key=s3key,
 Body=json.dumps(log_metadata))
    except Exception as e:
        print(f'Error writing logs to S3: {e}')


def lambda_handler(event, context):
    log_metadata = {}
    message_text = ""
    subject_text = ""

    if event['detail']['context']['contextType'] == 'GLUE_DATA_CATALOG':
        log_metadata['ruleset_name'] = str(event['detail']['rulesetNames'][0])
        log_metadata['tableName'] = str(event['detail']['context']['tableName'])
        log_metadata['databaseName'] = str(event['detail']['context']['databaseName'])
        log_metadata['runId'] = str(event['detail']['context']['runId'])
        log_metadata['resultId'] = str(event['detail']['resultId'])
        log_metadata['state'] = str(event['detail']['state'])
        log_metadata['score'] = str(event['detail']['score'])
        log_metadata['numRulesSucceeded'] = str(event['detail']['numRulesSucceeded'])
        log_metadata['numRulesFailed'] = str(event['detail']['numRulesFailed'])
        log_metadata['numRulesSkipped'] = str(event['detail']['numRulesSkipped'])

        message_text += "Glue Data Quality run details:\n"
        message_text += "ruleset_name: {}\n".format(log_metadata['ruleset_name'])
        message_text += "glue_table_name: {}\n".format(log_metadata['tableName'])
        message_text += "glue_database_name: {}\n".format(log_metadata['databaseName'])
        message_text += "run_id: {}\n".format(log_metadata['runId'])
        message_text += "result_id: {}\n".format(log_metadata['resultId'])
        message_text += "state: {}\n".format(log_metadata['state'])
        message_text += "score: {}\n".format(log_metadata['score'])
        message_text += "numRulesSucceeded:
 {}\n".format(log_metadata['numRulesSucceeded'])
        message_text += "numRulesFailed: {}\n".format(log_metadata['numRulesFailed'])
```

```
        message_text += "numRulesSkipped: {}\n".format(log_metadata['numRulesSkipped'])

        subject_text = "Glue Data Quality ruleset {} run
details".format(log_metadata['ruleset_name'])

    else:
        log_metadata['ruleset_name'] = str(event['detail']['rulesetNames'][0])
        log_metadata['jobName'] = str(event['detail']['context']['jobName'])
        log_metadata['jobId'] = str(event['detail']['context']['jobId'])
        log_metadata['resultId'] = str(event['detail']['resultId'])
        log_metadata['state'] = str(event['detail']['state'])
        log_metadata['score'] = str(event['detail']['score'])

        log_metadata['numRulesSucceeded'] = str(event['detail']['numRulesSucceeded'])
        log_metadata['numRulesFailed'] = str(event['detail']['numRulesFailed'])
        log_metadata['numRulesSkipped'] = str(event['detail']['numRulesSkipped'])

        message_text += "Glue Data Quality run details:\n"
        message_text += "ruleset_name: {}\n".format(log_metadata['ruleset_name'])
        message_text += "glue_job_name: {}\n".format(log_metadata['jobName'])
        message_text += "job_id: {}\n".format(log_metadata['jobId'])
        message_text += "result_id: {}\n".format(log_metadata['resultId'])
        message_text += "state: {}\n".format(log_metadata['state'])
        message_text += "score: {}\n".format(log_metadata['score'])
        message_text += "numRulesSucceeded:
{}\n".format(log_metadata['numRulesSucceeded'])
        message_text += "numRulesFailed: {}\n".format(log_metadata['numRulesFailed'])
        message_text += "numRulesSkipped: {}\n".format(log_metadata['numRulesSkipped'])

        subject_text = "Glue Data Quality ruleset {} run
details".format(log_metadata['ruleset_name'])

    resultID = str(event['detail']['resultId'])
    response = glue_client.get_data_quality_result(ResultId=resultID)
    RuleResults = response['RuleResults']
    message_text += "\n\nruleset details evaluation steps results:\n\n"
    subresult_info = []

    for dic in RuleResults:
        subresult = "Name: {}\t\tResult: {}\t\tDescription: \t{}".format(dic['Name'],
dic['Result'], dic['Description'])
        if 'EvaluationMessage' in dic:
            subresult += "\t\tEvaluationMessage: {}".format(dic['EvaluationMessage'])
        subresult_info.append({
```

```
            'Name': dic['Name'],
            'Result': dic['Result'],
            'Description': dic['Description'],
            'EvaluationMessage': dic.get('EvaluationMessage', '')
        })
        message_text += "\n" + subresult

    log_metadata['resultrun'] = subresult_info

    write_logs(log_metadata)


    return {
        'statusCode': 200,
        'body': json.dumps('Message published to SNS topic')
    }
```

寫入 Amazon S3 之後，您可以使用 AWS Glue 爬蟲程式向 Athena 註冊並查詢資料表。

在資料品質評估期間設定 Amazon S3 位置：

在 Glue 資料型錄或 AWS AWS Glue ETL 中執行資料品質任務時，您可以提供 Amazon S3 位置，將資料品質結果寫入 Amazon S3。您可以使用以下語法，透過參考目標以讀取資料品質結果來建立資料表。

請注意，您必須分別執行 `CREATE EXTERNAL TABLE` 和 `MSCK REPAIR TABLE` 查詢。

```
CREATE EXTERNAL TABLE <my_table_name>(
    catalogid string,
    databasename string,
    tablename string,
    dqrunid string,
    evaluationstartedon timestamp,
    evaluationcompletedon timestamp,
    rule string,
    outcome string,
    failurereason string,
    evaluatedmetrics string)
PARTITIONED BY (
    `year` string,
    `month` string,
    `day` string)
```

```
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (

 'paths'='catalogId,databaseName,dqRunId,evaluatedMetrics,evaluationCompletedOn,evaluationStart
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://glue-s3-dq-bucket-us-east-2-results/'
TBLPROPERTIES (
    'classification'='json',
    'compressionType'='none',
    'typeOfData'='file');
```

```
MSCK REPAIR TABLE <my_table_name>;
```

建立上述資料表後，您便可以使用 Amazon Athena 執行分析查詢。

使用部署資料品質規則 AWS CloudFormation

您可以使用 AWS CloudFormation 來建立資料品質規則。如需詳細資訊，請參閱〈AWS CloudFormation AWS Glue〉。

排程資料品質規則

您可以使用下列方法排程資料品質規則：

- 從「資料目錄」排程資料品質規則：沒有程式碼使用者可以使用此選項輕鬆排定其資料品質掃描的時間。 AWS Glue 數據質量將在 Amazon 創建時間表 EventBridge。若要排程資料品質規則：
  - 導覽至規則集，然後按一下執行。
  - 在執行頻率中，選取所需排程並提供任務名稱。此任務名稱是您在中的排程名稱 EventBridge。
- 使用 Amazon EventBridge 和 AWS Step Functions 數來協調資料品質規則的評估和建議。

解決 AWS Glue 資料品質錯誤

如果您在 AWS Glue 資料品質中遇到錯誤，請使用下列解決方案來協助您找出問題的根源並加以修正。

內容

錯誤：缺少 AWS Glue 資料品質模組

錯誤訊息：沒有名為 'awsgluedq' 的模組。

解決方法：在不支援的版本中執行 AWS Glue 資料品質時，就會發生此錯誤。 AWS 只有 Glue 3.0 版及更新版本才支援「Glue 資料品質」。

錯誤： AWS Lake Formation 權限不足

錯誤訊息：使用者類別中的例外狀況:`com.amazonaws.services.glue.model.AccessDeniedException`: 影響 _sdg_參與時的 Lake Formation 權限不足 (服務：AWS Glue；狀態碼：400；錯誤代碼： ；請求識別碼：465ae693-b7ba-4df0-a4e4-6b17xxxxxx AccessDeniedException；代理伺服器：空值)。

解決方案：您必須在 AWS Lake Formation 中提供足夠的權限。

錯誤：規則集的命名不是唯一的

錯誤消息：用戶類中的異常：... 服務 .glue.model。 AlreadyExistsException：另一個具有相同名稱的規則集已存在。

解決方案：規則集是全域的並且必須是唯一的。

錯誤：資料表具有特殊字元

錯誤訊息:使用者類別中的例外狀況：或 .apache.spark.sql。 AnalysisException: 無法解析「C」給定的輸入列：[主要的 .data_end_time，主要的 .data_start_time，主要的 .end_time，主要的. 最後更新，主要的. 處理日期，主要的 .rowhash，主要的 .run_id，主要的 .start_time，主要的 .status]; 第 1 行。

解決方案：目前有一項限制，即無法在具有特殊字元 (例如「.」) 的表格上執行「 AWS Glue 資料品質」。

錯誤：大型規則集的溢位錯誤

錯誤消息：用戶類中的異常：java.lang。 StackOverflowError。

解決方案：如果是大於 2K 規則的大型規則集，則可能會遇到此問題。將規則分解為多個規則集。

錯誤：整體規則狀態為失敗

錯誤條件：我的規則集成功，但我的整體規則狀態失敗。

解決方案：最有可能發生此錯誤，是因為您選擇了在發佈 CloudWatch 時將指標發佈到 Amazon 的選項。如果您的資料集位於 VPC 中，則您的 VPC 可能不允許 AWS Glue 將指標發佈到 Amazon。CloudWatch在這種情況下，您必須為您的 VPC 設置一個端點才能訪問 Amazon。 CloudWatch

AnalysisException: 無法驗證預設資料庫是否存在

錯誤條件: AnalysisException: 無法驗證預設資料庫是否存在:. AccessDeniedException: 預設情況下的 Lake Formation 權限不足 (服務:AWS Glue；狀態碼:400；錯誤代碼:；請求識別碼:XXXXXX-XXXX-XXXX-XXXXXX AccessDeniedException；代理伺服器：空值)

解決方案：在 AWS Glue 任務的型錄整合中，AWS Glue 一律會嘗試檢查預設資料庫是否存在或不使用 AWS Glue `GetDatabase API`。如果未授予 `DESCRIBE` Lake Formation 許可或已授予 `GetDatabase IAM` 許可，則在驗證預設資料庫的存在時，該任務會失敗。

若要解決問題：

1. 在 Lake Formation 中新增預設資料庫的 DESCRIBE 許可。
2. 將附加至 AWS Glue 任務的 IAM 角色設定為 Lake Formation 中的資料庫建立者。這會自動建立預設資料庫，並授予角色所需的 Lake Formation 許可。
3. 停用 `--enable-data-catalog` 選項。(在 AWS Glue Studio 中顯示為使用 Data Catalog 作為 Hive 中繼存放區)。

   如果不需要任務中的 Spark SQL Data Catalog 整合，則可以停用此功能。

錯誤訊息：提供的索引鍵映射不適合指定的資料框架

錯誤條件：提供的索引鍵映射不適合指定的資料框架。

解決方案：您正在使用DataSetMatch規則類型，並且連接鍵具有重複項。聯結索引鍵必須是唯一的，且不得為 NULL。如果您無法擁有唯一的聯結索引鍵，請考慮使用其他規則類型，例如AggregateMatch比對摘要資料。

使用者類別中的例外狀況：java.lang。 RuntimeException ：無法擷取資料。檢查登錄 CloudWatch 以獲取更多詳細信息

錯誤條件：用戶類中的異常：java.lang。 RuntimeException ：無法擷取資料。檢查登錄 CloudWatch 以獲取更多詳細信息。

解決方案：當您在與 Amazon RDS 或比較的 Amazon S3 表格上建立 DQ 規則時，就會發生這種情況。 Amazon Redshift在這些情況下，AWS Glue 無法載入連線。而是嘗試在 Amazon Redshift 或 Amazon RDS 數據集上設置 DQ 規則。這是已知的錯誤。

啟動錯誤：從 S3 下載儲存貯體時發生錯誤

錯誤條件：啟動錯誤：從 S3 下載儲存貯體時發生錯誤：`aws-glue-ml-data-quality-assets-us-east-1, key: jars/aws-glue-ml-data-quality-etl.jar.Access Denied (Service: Amazon S3; Status Code: 403; Please refer logs for details)` 。

解決方案：傳遞給 AWS Glue 資料品質之角色的許可必須允許從之前的 Amazon S3 位置讀取。此 IAM 政策應附加至該角色：

```
{
  "Sid": "allowS3",
  "Effect": "Allow",
```

```
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::aws-glue-ml-data-quality-assets-<region>/*"
}
```

如需詳細許可，請參閱 Data Quality authorization。需要這些程式庫來評估資料集的資料品質。

InvalidInputException （狀態：400）：無法剖析 DataQuality 規則

錯誤條件： InvalidInputException （狀態：400）：無法剖析 DataQuality 規則。

解決方案：出現這種錯誤的可能性有很多。一種可能性是規則中可能有單引號。確認規則都由雙引號括住。例如：

```
Rules = [
ColumnValues "tipo_vinculo" in ["CODO", "DOCO", "COCO", "DODO"] AND "categoria" = 'ES'
    AND "cod_bandera" = 'CEP'
```

將此規則變更為：

```
Rules = [
(ColumnValues "tipovinculo" in [ "CODO", "DOCO", "COCO", "DODO"]) AND (ColumnValues
 "categoria" = "ES")
    AND (ColumnValues "codbandera" = "CEP")
]
```

錯誤：Eventbridge 不會根據我設定的排程觸發 Glue DQ 任務

錯誤條件：Eventbridge 不會根據我設定的排程觸發 AWS Glue Data Quality 任務。

解決方案：觸發任務的角色可能沒有正確的許可。請確定您用來開始作業的角色具有排程評估執行所需的 IAM 設定中提到的許可。

CustomSQL 錯誤

錯誤條件： `The output from CustomSQL must contain at least one column that matches the input dataset for AWS Glue Data Quality to provide row level results. The SQL query is a valid query but no columns from the SQL result`

`are present in the Input Dataset. Ensure that matching columns are returned from the SQL`。

解決方案：SQL 查詢是有效的，但請確認您從主資料表中選取的只有資料欄。選取像 sum 這樣的彙總函數，對主資料表中的資料欄進行計數可能會導致此錯誤。

錯誤條件： `There was a problem when executing your SQL statement: cannot resolve "Col"`。

解決方案：主資料表中不存在此資料欄。

錯誤條件： `The columns that are returned from the SQL statement should only belong to the primary table. "In this case, some columns ( Col ) belong to reference table"`。

解決方案：在 SQL 查詢中，當您將主資料表與其他參考資料表結合時，請確認 select 陳述式只有主資料表中的資料欄名稱，以產生主資料表的資料列層級結果。

動態規則

錯誤條件: `Dynamic rules require job context, and cannot be evaluated in interactive session or data preview.`。

原因：當規則集中存在動態 DQ 規則時，此錯誤訊息可能會出現在資料預覽結果或其他互動式工作階段中。動態規則會參考與特定作業名稱和評估內容相關聯的歷史指標，因此無法在互動式工作階段中評估這些指標。

解決方案：執行 AWS Glue 作業會產生歷史指標，可在相同作業的後續作業執行中參考這些指標。

錯誤條件：

- `[RuleType] rule only supports simple atomic operands in thresholds.`.
- `Function last not yet implemented for [RuleType] rule.`

解決方案：數值表達式中所有 DQDL 規則類型通常都支援動態規則 (請參閱 DQDL 參考)。但是，尚未支援產生多個量度 ColumnValues 和 ColumnLength的某些規則。

錯誤條件： `Binary expression operands must resolve to a single number.`。

原因：動態規則支援 RowCount > avg(last(5)) * 0.9 之類的二進制表達式。此處的二進制表達式是 avg(last(5)) * 0.9。這個規則是有效的，因為 avg(last(5)) 和 0.9 這兩種運算元都會解析為單一數字。RowCount > last(5) * 0.9 是錯誤範例，因為 last(5) 產生的清單無法與目前資料列計數進行有意義的比較。

解決方案：使用彙總函數，將清單值運算元簡化為單一數字。

錯誤條件：

- Rule threshold results in list, and a single value is expected. Use aggregation functions to produce a single value. Valid example: sum(last(10)), avg(last(10)).
- Rule threshold results in empty list, and a single value is expected.

原因：動態規則可用於比較資料集的某些特性與其歷史值。如果提供正整數引數，則最後一個函數允許擷取多個歷史值。例如，last(5) 會擷取在規則作業執行中觀察到的最近五個值。

解決方案：必須使用彙總函數將這些值簡化為單一數字，以便與目前作業執行中觀察到的值進行有意義的比較。

有效的範例：

- RowCount >= avg(last(5))
- RowCount > last(1)
- RowCount < last()

無效的範例：RowCount > last(5)。

錯誤條件：

- Function index used in threshold requires positive integer argument.
- Index argument must be an integer. Valid syntax example: RowCount > index(last(10, 2)), which means RowCount must be greater than third most recent execution from last 10 job runs.

解決方案：編寫動態規則時，您可以使用 index 彙總函數從清單中選取一個歷史值。例如 RowCount > index(last(5) 1) 會檢查目前作業中觀察到的資料列計數是否嚴格大於作業中觀察到的第二個最近的資料列計數。index 從零開始索引。

錯誤條件： `IllegalArgumentException: Parsing Error: Rule Type: DetectAnomalies is not valid`。

解決方案：異常偵測功能僅適用於 AWS Glue 4.0。

錯誤條件： `IllegalArgumentException: Parsing Error: Unexpected condition for rule of type ... no viable alternative at input ...`。

注意：`...` 是動態的。範例：`IllegalArgumentException: Parsing Error: Unexpected condition for rule of type RowCount with number return type, line 4:19 no viable alternative at input '>last'`。

解決方案：異常偵測功能僅適用於 AWS Glue 4.0。

使用者類別中的例外狀況：或。 AnalysisException：組織 .hadoop.hive.ql. 元數據。 HiveException

錯誤條件： `Exception in User Class: org.apache.spark.sql.AnalysisException: org.apache.hadoop.hive.ql.metadata.HiveException: Unable to fetch table mailpiece_submitted. StorageDescriptor#InputFormat cannot be null for table: mailpiece_submitted (Service: null; Status Code: 0; Error Code: null; Request ID: null; Proxy: null)`

原因:您在 Glue 資料目錄中使用 Apache Iceberg，而 AWS Glue 資料型錄中 AWS 的輸入格式屬性為空。

解決方案：當您在 DQ 規則中使用 CustomSQL 規則類型時，就會發生此問題。解決此問題的一種方法是使用「主要」或將目錄名稱添加`glue_catalog.`到`<database>.<table> in Custom ruletype.`

未分類 _ERROR; IllegalArgumentException：解析錯誤：沒有提供規則或分析器。，輸入時沒有可行的替代方法

錯誤條件： `UNCLASSIFIED_ERROR; IllegalArgumentException: Parsing Error: No rules or analyzers provided., no viable alternative at input`

解析度：DQDL 無法剖析。在少數情況下，可能會發生這種情況。如果您使用複合規則，請確定它們有右括號。

```
(RowCount >= avg(last(10)) * 0.6) and (RowCount <= avg(last(10)) * 1.4) instead of
```

```
    RowCount >= avg(last(10)) * 0.6 and RowCount <= avg(last(10)) * 1.4
```

Amazon Q 數據集成 AWS Glue

中的 Amazon Q 資料整合 AWS Glue 是一項新的生成 AI 功能，可 AWS Glue 讓資料工程師和 ETL 開發人員使用自然語言建立資料整合任務。工程師和開發人員可以要求 Amazon Q 編寫任務、疑難排解問題，以及回答有關 AWS Glue 資料整合的問題。

什麼是 Amazon Q？

> **Note**
>
> 由 Amazon 基岩提供支援： AWS 實作自動濫用偵測。由於 Amazon Q 資料整合建立在 Amazon 基岩上，因此使用者可以充分利用 Amazon 基岩中實作的控制項，強制執行人工智慧 (AI) 的安全性、安全性和負責任的使用。

Amazon Q 是採用生成式人工智慧 (AI) 的交談助理，可協助您了解、建置、擴充和操作 AWS 應用程式。支援 Amazon Q 的模型已增強高品質 AWS 內容，讓您獲得更完整、可操作和參考的答案，以加速您的建置 AWS。如需詳細資訊，請參閱什麼是 Amazon Q？

AWS Glue中的 Amazon Q 資料整合是什麼？

中的 Amazon Q 資料整合 AWS Glue 包括下列功能：

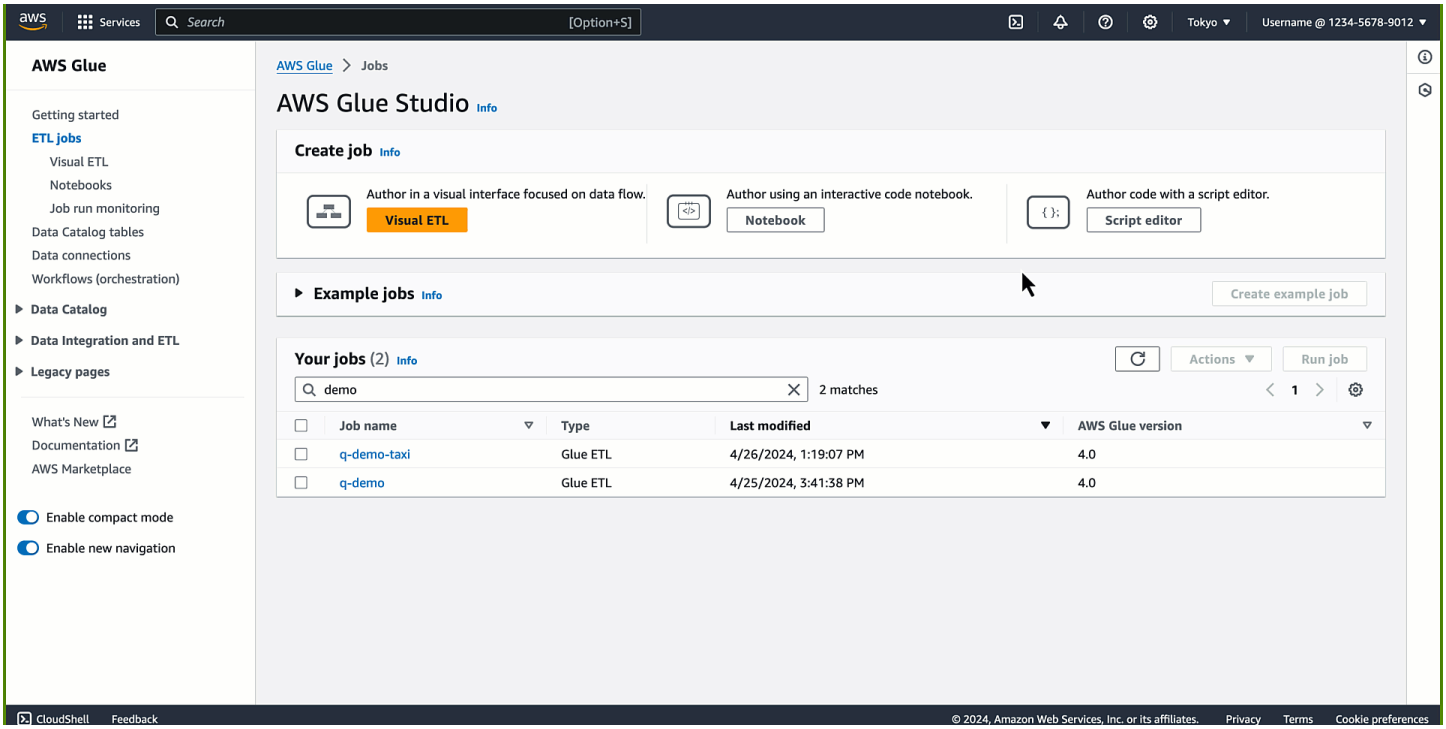
- 聊天 — 中的 Amazon Q 資料整合 AWS Glue 可以用英文回答 AWS Glue 與資料整合網域相關的自然語言問題，例如 AWS Glue 來源和目的地連接器、 AWS Glue ETL 任務、資料目錄、爬蟲和 AWS Lake Formation其他功能文件以及最佳實務。Amazon Q 資料整合中 AWS Glue 會回應 step-by-step 指示，並包含對其資訊來源的參考。
- 資料整合程式碼產生 — 中的 Amazon Q 資料整合 AWS Glue 可以回答有關 AWS Glue ETL 指令碼的問題，並在以英文自然語言問題的情況下產生新程式碼。
- 疑難排解 — 中的 Amazon Q 資料整合 AWS Glue 旨在協助您瞭解任 AWS Glue 務中的錯誤，並提供 step-by-step 指示、根本原因和解決問題。

> **Note**
>
> 中的 Amazon Q 資料整合 AWS Glue 不會使用交談內容來通知 future 對話期間的回應。與 Amazon Q 資料整合的每次對話 AWS Glue 都獨立於您之前或 future 的對話。

是否使用 AWS Glue中的 Amazon Q 資料整合？

在 Amazon Q 面板中，您可以請求 Amazon Q 產生 AWS Glue ETL 指令碼的程式碼，或回答 AWS Glue 功能相關問題或疑難排解錯誤。響應是一個 ETL 腳本，其中 PySpark 包含自定義腳本，審查和執行它的 step-by-step 說明。系統會根據資料整合知識庫產生對問題的回應，其中包含摘要和來源 URL 以供參考。

例如，您可以要求 Amazon Q「請提供從雪花讀取、重新命名欄位並寫入 Redshift」的 Glue 指令碼，並回應中的 Amazon Q 資料整合 AWS Glue 將傳回可執行請求動 AWS Glue 作的工作指令碼。您可以檢閱產生的程式碼，確保其符合請求的意圖。如果滿意，您可以將其部署為生產中的 AWS Glue 工作。您可以要求整合解釋錯誤和失敗，並提出解決方案，藉此疑難排解作業。Amazon Q 可以回答有關資料整合最佳實務的問題 AWS Glue 或問題。



以下是示範中 Amazon Q 資料整合如何協助 AWS Glue 您進行建置的範例問題 AWS Glue：

AWS Glue ETL 代碼生成：

- 撰寫可從 S3 讀取 JSON 的 AWS Glue 指令碼、使用套用對應轉換欄位，然後寫入 Amazon Redshift
- 如何編寫用於從 DynamoDB 讀取的 AWS Glue 腳本， DropNullFields 將轉換和寫入作為實木複合地板應用到 S3？
- 給我一個從 MySQL 讀取的 AWS Glue 腳本，根據我的業務邏輯刪除一些字段，並寫入雪花
- 寫入要從動態 B 讀取的 AWS Glue 任務，然後以 JSON 格式寫入 S3
- 協助我開發 S3 AWS Glue 資料型錄的 AWS Glue 指令碼
- 編寫一個 AWS Glue 工作以從 S3 讀取 JSON，刪除空值並寫入 Redshift

AWS Glue 功能說明：

- 如何使用 AWS Glue 資料品質？
- 如何使用 AWS Glue 工作書籤？
- 如何啟用 AWS Glue 自動調度資源？
- AWS Glue 動態幀和 Spark 數據幀有什麼區別？
- 支援哪些不同類型的連線 AWS Glue？

AWS Glue 疑難排解：

- 如何解決 AWS Glue 工作上的內存不足（OOM）錯誤？
- 設定 AWS Glue 資料品質時，您可能會看到哪些錯誤訊息，以及如何修正這些錯誤訊息？
- 如何解決 Amazon S3 訪問被拒絕的錯誤的任 AWS Glue 務？
- 如何解決 AWS Glue 工作資料隨機播放的問題？

與 Amazon Q 資料整合互動的最佳實務

以下是與 Amazon Q 資料整合互動的最佳實務：

- 與 Amazon Q 資料整合互動時，請提出特定問題、在有複雜請求時進行迭代，並驗證答案的準確性。
- 以自然語言提供資料整合提示時，請盡量具體協助助理確切瞭解您的需求。而不是詢問「從 S3 擷取資料」，而是提供更多詳細資訊，例如「撰寫從 S3 擷取 JSON 檔案的 AWS Glue 指令碼」。
- 在運行之前檢查生成的腳本以確保準確性。如果產生的指令碼有錯誤或與您的意圖不符，請向助理提供如何更正指示。

- 生成式 AI 技術是一種新穎的技術，在反應中可能會出現錯誤，有時也稱為幻覺。在您的環境或工作負載中使用之前，請先測試並檢閱所有程式碼是否有錯誤和漏洞。

Amazon Q 數據集成在 AWS Glue 服務改進

為了協助 Amazon Q 資料整合 AWS Glue 提供最相關的 AWS 服務資訊，我們可能會使用 Amazon Q 的特定內容，例如您詢問 Amazon Q 的問題及其回應，以改善服務。

如需有關我們使用哪些內容以及如何選擇退出的資訊，請參閱 Amazon Q 開發人員使用者指南中的 Amazon Q 開發人員服務改進。

考量事項

使用 AWS Glue中的 Amazon Q 資料整合之前，請考慮下列項目：

- 目前，代碼生成僅適用於 PySpark 內核。生成的代碼是基於 Python 星火的 AWS Glue 作業。
- 如需 Amazon Q 資料整合所支援的程式碼產生功能組合的相關資訊 AWS Glue，請參閱支持的代碼生成功能。

在中設定 Amazon Q 資料整合 AWS Glue

以下各節提供設定 AWS Glue中的 Amazon Q 資料整合的資訊。

主題

- 設定 IAM 許可權限

設定 IAM 許可權限

本主題說明您為 Amazon Q 聊天體驗設定的 IAM 許可，以及 AWS Glue Studio 筆記本體驗。

主題

- 為 Amazon Q 聊天設定 IAM 許可
- 設定 AWS Glue 工作室筆記本的 IAM 許可

為 Amazon Q 聊天設定 IAM 許可

授與 Amazon Q 資料整合所使用的 API 的許可 AWS Glue 需要適當的 AWS Identity and Access Management (IAM) 許可。您可以透過將下列自訂 AWS 政策附加到 IAM 身分 (例如使用者、角色或群組) 來取得許可：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:StartCompletion",
                "glue:GetCompletion"
            ],
            "Resource": [
                "arn:aws:glue:*:*:completion/*"
            ]
        }
    ]
}
```

設定 AWS Glue 工作室筆記本的 IAM 許可

若要在 AWS Glue Studio 筆記本中啟用 Amazon Q 資料整合，請確保將下列權限附加至筆記本 IAM 角色：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:StartCompletion",
                "glue:GetCompletion"
            ],
            "Resource": [
                "arn:aws:glue:*:*:completion/*"
            ]
        },
        {
            "Sid": "CodeWhispererPermissions",
```

```
            "Effect": "Allow",
            "Action": [
                "codewhisperer:GenerateRecommendations"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> 中的 Amazon Q 資料整合 AWS Glue 沒有可透過 AWS 開發套件提供的 API，您可以透過程式設計方式使用這些 API。IAM 政策中使用以下兩個 API，透過 Amazon Q 聊天面板或 AWS Glue Studio 筆記本啟用此體驗：`StartCompletion`和`GetCompletion`。

指派權限

若要提供存取權，請新增權限至您的使用者、群組或角色：

- AWS IAM 身分中心中的使用者和群組：建立權限集。依照 AWS IAM 身分中心使用者指南中的建立權限集中的指示進行。
- 透過身分識別提供者在 IAM 中管理的使用者：為聯合身分建立角色。請按照 IAM 使用者指南 的 為第三方身分提供者 (聯合) 建立角色 中的指示進行操作。
- IAM 使用者：
  - 建立您的使用者可擔任的角色。請按照 IAM 使用者指南 的 為 IAM 使用者建立角色 中的指示進行操作。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循 IAM 使用者指南的新增許可到使用者 (主控台)中的指示。

支持的代碼生成功能

以下是 AWS Glue中的 Amazon Q 資料整合的程式碼產生功能的組合。

來源	轉換	目標
S3 具有以下格式類型：json、csv、parquet、hudi、delta	ApplyMapping	S3 具有以下格式類型：json、csv、avro、orc、parquet、hudi、delta
Glue Data Catalog	RenameField	Glue Data Catalog
Amazon Redshift	DropFields	Amazon Redshift
MySQL	SelectFields	MySQL
Postgres	DropNullFields	Postgres
Oracle	篩選條件	Oracle
SQL Server	Spigot	SQL Server
DynamoDB	自訂 SQL 程式碼	DynamoDB
Snowflake	Aggregate	Snowflake
MongoDB	DropDuplicates	MongoDB
自訂 JDBC 連接器	Join	自訂 JDBC 連接器
定制火花連接器	UNION	定制火花連接器
谷歌 BigQuery		谷歌 BigQuery
Teradata		Teradata
Amazon OpenSearch 服務		Amazon OpenSearch 服務
Vertica		Vertica
蔚藍		蔚藍色 DQL
SAP HANA		SAP HANA
蔚藍宇宙		蔚藍宇宙

互動示例

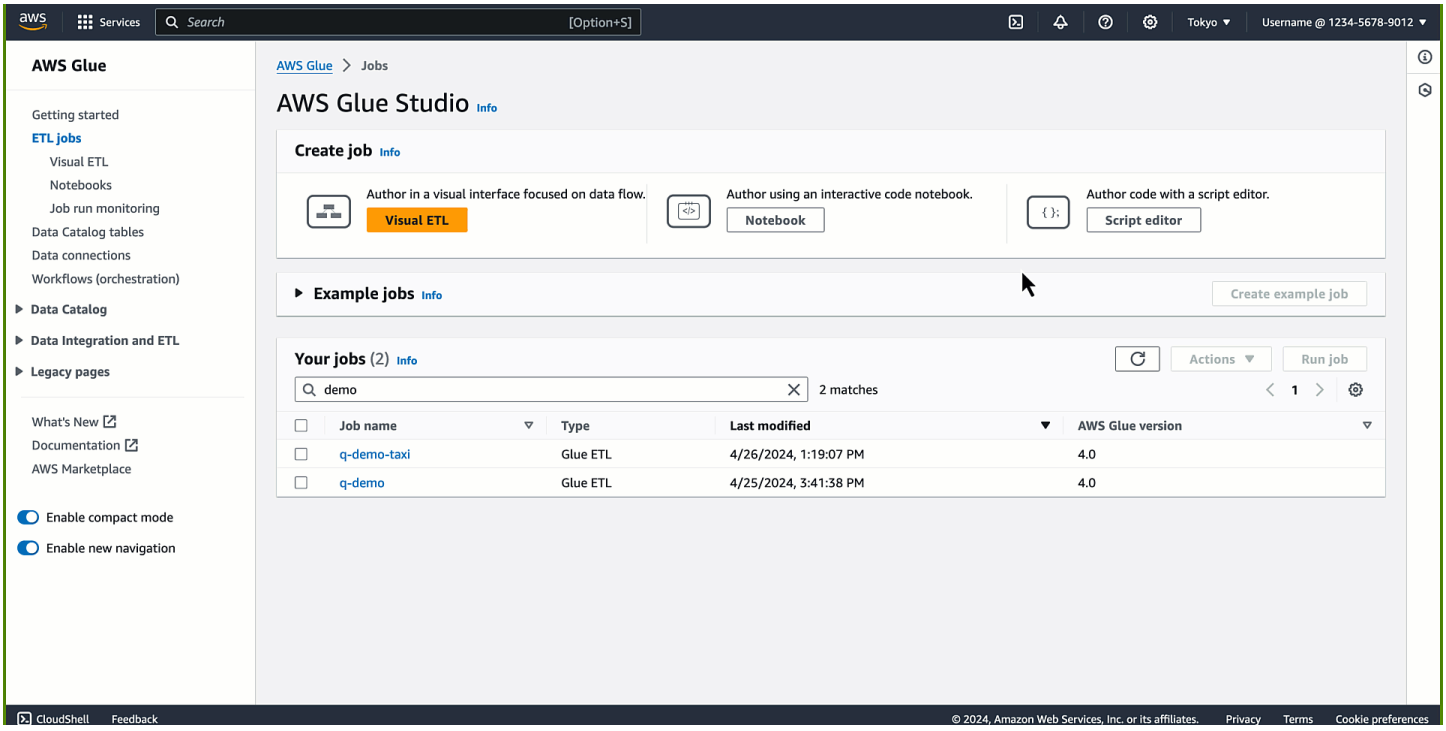
中的 Amazon Q 資料整合 AWS Glue 可讓您在 Amazon Q 面板中輸入您的問題。您可以輸入有關提供的資料整合功能的問題 AWS Glue。一個詳細的答案，連同參考文件，將被退回。

另一個使用案例是產生 AWS Glue ETL 工作指令碼。您可以詢問有關如何執行資料擷取、轉換、載入工作的問題。將傳回產生的 PySpark 指令碼。

主題

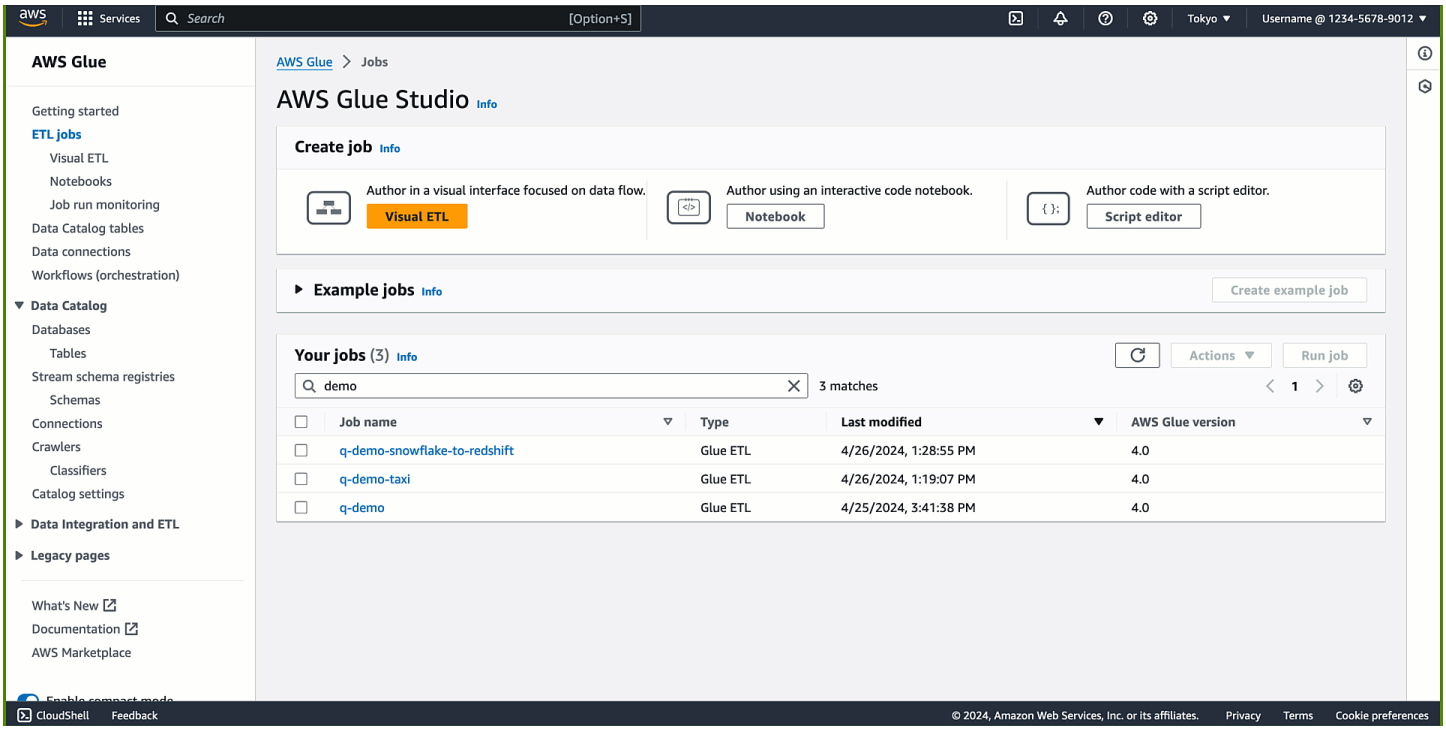
Amazon Q 聊天互動

在 AWS Glue 主控台上，開始撰寫新任務，然後詢問 Amazon 問：「請提供從雪花讀取、重新命名欄位，然後寫入 Redshift 的 Glue 指令碼。」



您會注意到代碼已生成。透過此回應，您可以學習並瞭解如何根據自己的目的撰寫 AWS Glue 程式碼。您可以將生成的代碼複製/粘貼到腳本編輯器並配置佔位符。在工作上設定 AWS Identity and Access Management (IAM) 角色和 AWS Glue 連線後，請儲存並執行工作。任務完成後，您可以開始查詢 Amazon Redshift 中從雪花匯出的資料表。

以下提示會讀取來自兩個不同來源的資料，篩選並個別投影它們，在一個共同金鑰上聯結，然後將輸出寫入第三個目標。詢問 Amazon 問：「我想以鑲木地板格式從 S3 讀取數據，然後選擇一些字段。我也想從 DynamoDB 讀取資料、選取一些欄位，並篩選一些資料列。我想聯合這兩個數據集並將結果寫入 OpenSearch。

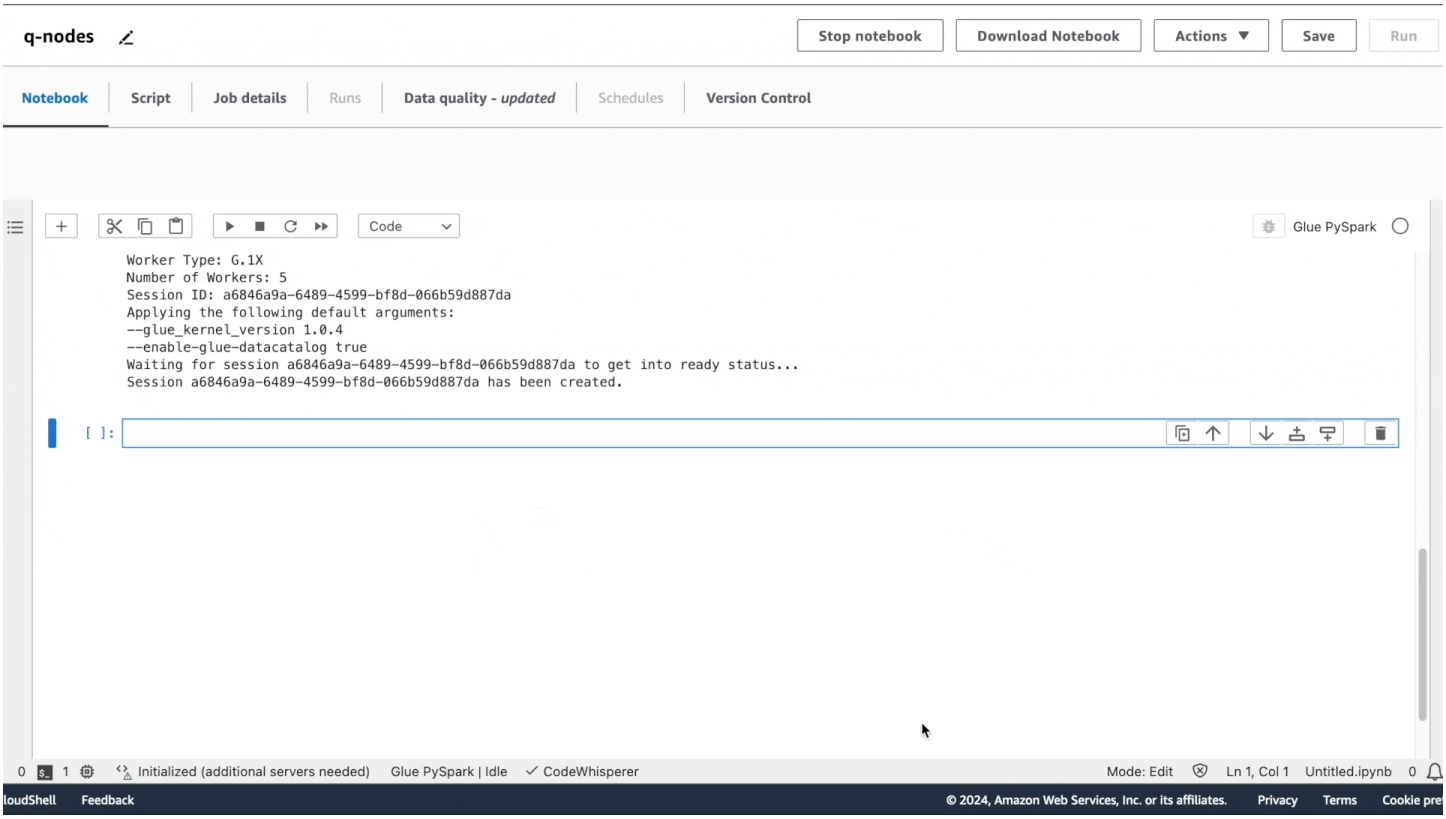


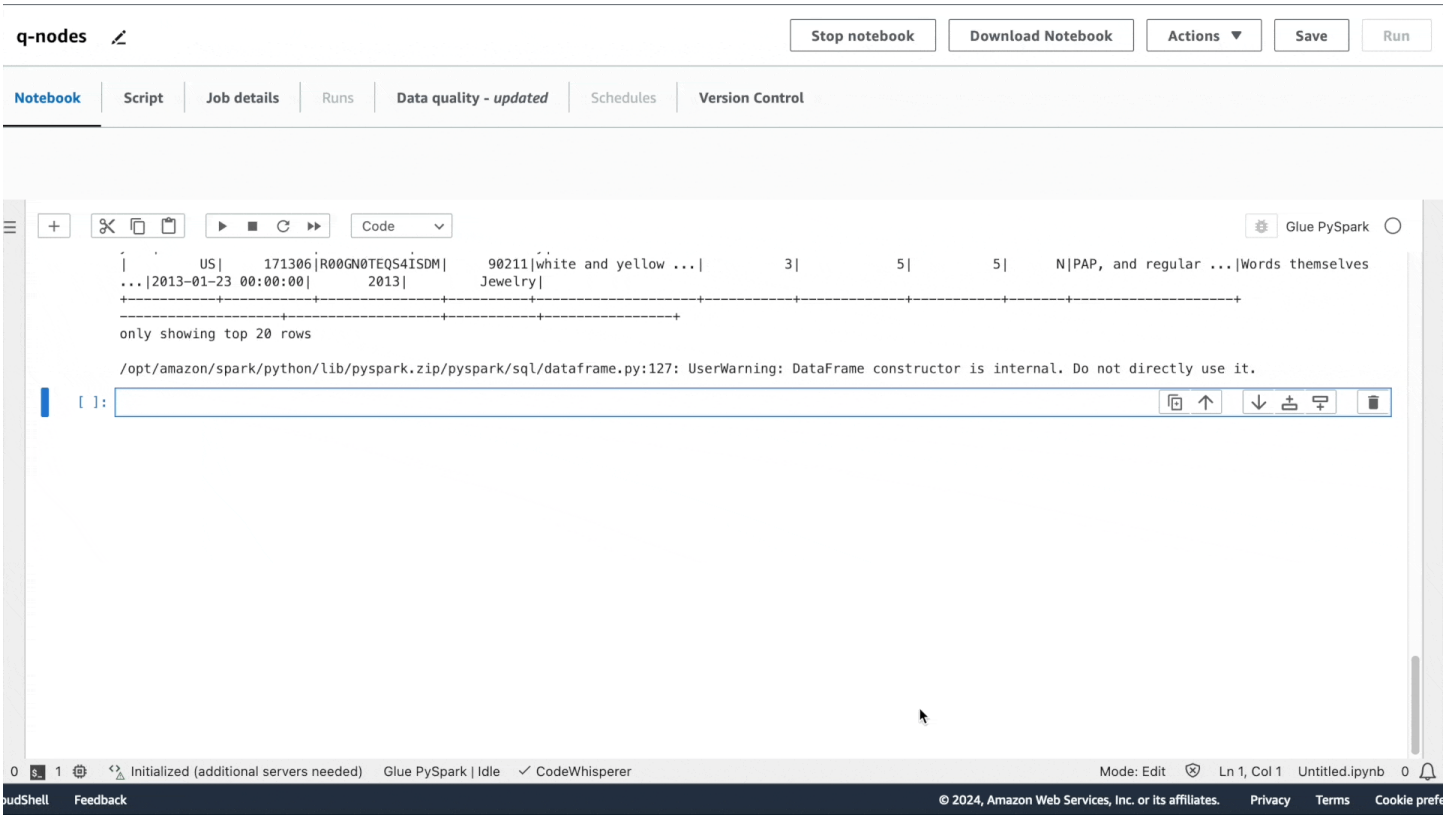
即會產生程式碼。工作完成後，您的索引即可在中使用， OpenSearch 並可供下游工作負載使用。

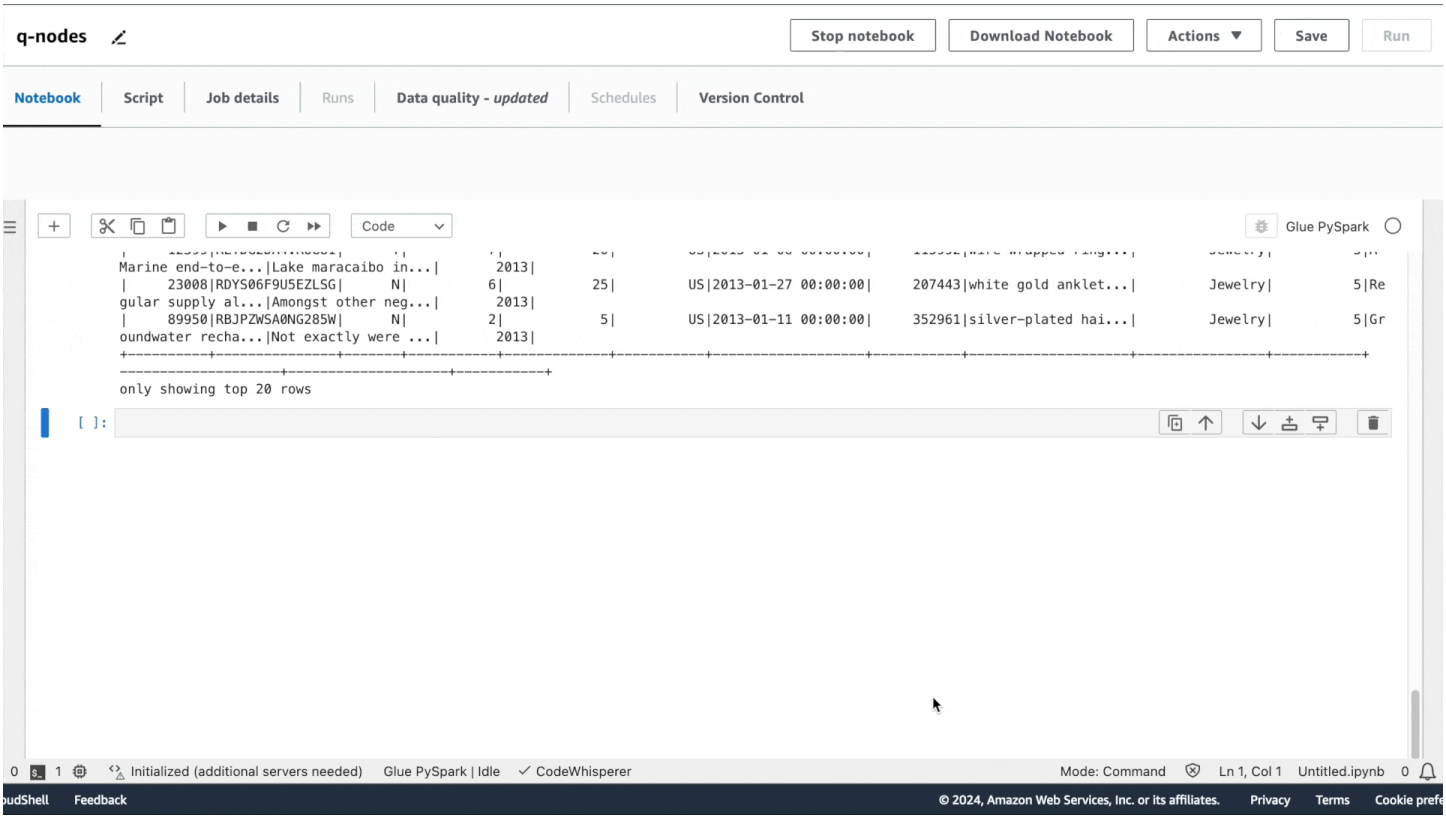
AWS Glue 工作室筆記本互

添加一個新的單元格並輸入您的評論以描述您想要實現的目標。按 Tab 鍵並輸入後，便會顯示建議的代碼。

第一個目的是提取數據：「給我讀取 Glue 數據目錄表的代碼」，然後是 「給我代碼以使用 star_rating>3 應用過濾器轉換」和 「給我將框架寫入 S3 的代碼作為鑲木地板」。


q-nodes
Stop notebook
Download Notebook
Actions
Save
Run
Notebook
Script
Job details
Runs
Data quality - updated
Schedules
Version Control
Code
Glue PySpark
Worker Type: G.1X
Number of Workers: 5
Session ID: a6846a9a-6489-4599-bf8d-066b59d887da
Applying the following default arguments:
--glue_kernel_version 1.0.4
--enable-glue-datacatalog true
Waiting for session a6846a9a-6489-4599-bf8d-066b59d887da to get into ready status...
Session a6846a9a-6489-4599-bf8d-066b59d887da has been created.
[ ]:
Initialized (additional servers needed)
Glue PySpark | Idle
CodeWhisperer
Mode: Edit
Ln 1, Col 1
Untitled.ipynb
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms


q-nodes
Stop notebook
Download Notebook
Actions
Save
Run
Notebook
Script
Job details
Runs
Data quality - updated
Schedules
Version Control
Code
Glue PySpark
only showing top 20 rows
/opt/amazon/spark/python/lib/pyspark.zip/pyspark/sql/dataframe.py:127: UserWarning: DataFrame constructor is internal. Do not directly use it.
[ ]:
Initialized (additional servers needed)
Glue PySpark | Idle
CodeWhisperer
Mode: Edit
Ln 1, Col 1
Untitled.ipynb
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms

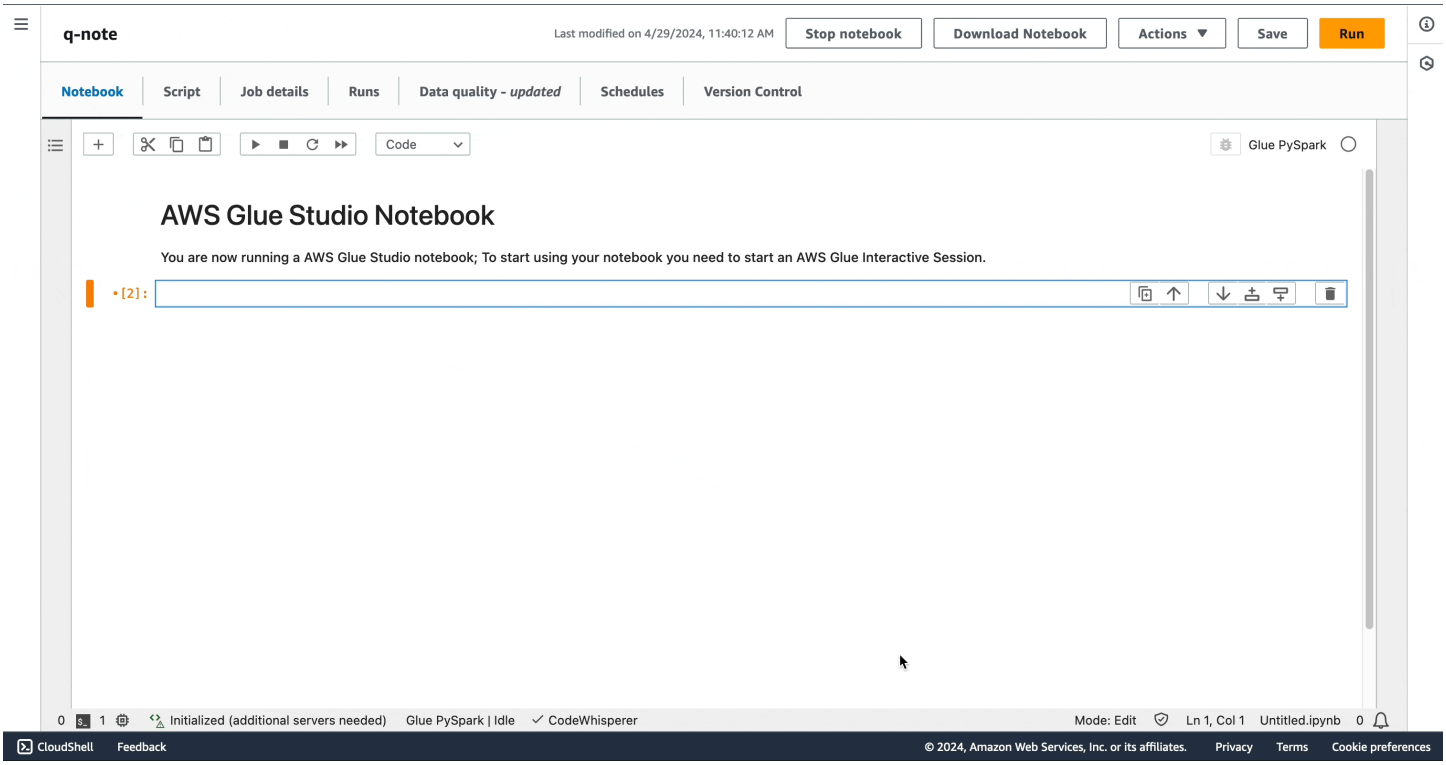


類似於 Amazon Q 聊天體驗，建議使用代碼。如果您按 Tab 鍵，則會選擇建議的程式碼。

您可以通過在生成的代碼中為源填寫適當的選項來運行每個單元格。在執行中的任何時候，您也可以使用show( )方法預覽資料集的範例。

複雜提示

您可以使用單一複雜提示來產生完整腳本。「我在 S3 中有 JSON 數據和甲骨文中的數據需要合併。請提供從兩個來源讀取、執行聯結，然後將結果寫入 Redshift 的 Glue 指令碼。」



您可能會注意到，在筆記型電腦上，中的 Amazon Q 資料整合 AWS Glue 產生的程式碼片段與 Amazon Q 聊天中產生的相同程式碼片段。

您可以選擇 [執行] 或以程式設計方式執行筆記本做為工作。

在 AWS Glue 中的協同運作

以下章節將提供在 AWS Glue 中的任務協同運作資訊。

主題

使用觸發啟動任務和爬蟲程式

在 AWS Glue 中，您可以建立稱為觸發的資料目錄物件，用來手動或自動啟動一或多個爬蟲程式，或是擷取、轉換和載入 (ETL) 任務。使用觸發，您可以設計相依任務和爬蟲程式的鏈結。

> **Note**
>
> 您可以透過定義「工作流程」來完成相同的事情。工作流程較適用於建立複雜得多任務 ETL 操作。如需更多詳細資訊，請參閱 the section called “使用藍圖和工作流程來執行複雜的 ETL 活動”。

主題

AWS Glue 觸發條件

當「引發」時，觸發可以啟動指定的任務和爬蟲程式。觸發可以視需要、根據排程，或是根據事件組合來引發。

> **Note**
>
> 單一觸發只能啟動兩個爬蟲程式。如果您要編目多個資料存放區，請為每個爬蟲程式使用多個來源，而不要同時執行多個爬蟲程式。

觸發可以處於數種狀態中的其中一種。觸發可以是 CREATED、ACTIVATED 或 DEACTIVATED。其中也有轉換狀態，例如 ACTIVATING。如要停止引發觸發，您可以停用觸發。您稍後可以重新啟用。

觸發有三種類型：

Scheduled (已排程)

以 cron 為基礎的時間類型觸發。

您可以根據排程建立一組任務或是爬蟲程式的觸發。您可以指定限制條件，例如任務或爬蟲程式的執行頻率、在一週中的哪一天執行，以及執行的時間。這些限制條件是以 cron 為基礎。當您正在設定觸發的排程時，請考慮 cron 的功能和限制。例如，如果您選擇在每個月的 31 日執行您的爬蟲程式，請注意有些月份不到 31 天。如需 Cron 的詳細資訊，請參閱 任務和爬蟲程式以時間為基礎的排程。

有條件

在上一個任務或爬蟲程式，或是多個任務或多個爬蟲程式滿足條件清單後引發的觸發。

當您建立條件式觸發時，您可以指定要監看的任務清單和爬蟲程式清單。針對每個監看的任務或爬蟲程式，您可以指定要監看的狀態，例如成功、失敗、逾時等。觸發會在監看的任務或爬蟲程式以指定狀態結束時引發。您可以設定觸發，使其在發生任何監看事件，或是所有監看事件都發生時引發。

例如，您可以設定觸發 T1 在任務 J1 和任務 J2 成功完成時啟動任務 J3，以及另外一個觸發 T2 在任務 J1 或任務 J2 失敗時啟動任務 J4。

下表列出觸發監看的任務和爬蟲程式完成狀態 (事件)。

任務完成狀態	爬蟲程式完成狀態
• SUCCEEDED • STOPPED • FAILED • TIMEOUT	• SUCCEEDED • FAILED • CANCELLED

隨需

在您啟用時引發的觸發。隨需觸發永遠不會進入 ACTIVATED 或 DEACTIVATED 狀態。這些觸發一律會停留在 CREATED 狀態。

如此一來，這些觸發便會在存在時準備就緒以供引發；您可以在建立這些觸發時設定標記來啟用排程和條件式觸發。

> **Important**
>
> 因其他任務或爬蟲程式完成而執行的任務或爬蟲程式稱為「相依」。相依任務或爬蟲程式只有在完成的任務或爬蟲程式是由觸發啟動時，才會啟動。相依鏈中所有的任務或爬蟲程式都必須是單一 scheduled (排程) 或 on-demand (隨需) 觸發的子代。

使用觸發傳遞任務參數

觸發可以將參數傳遞至啟動的任務。參數包括任務引數、逾時值、安全組態等。如果觸發啟動多個任務，參數會傳遞至每個任務。

以下是由觸發所傳遞任務引數的規則：

- 如果鍵/值對中的鍵與預設任務引數相符，則傳遞的引數會覆寫預設引數。如果鍵與預設引數不符，則引數會做為額外引數傳遞給任務。
- 如果鍵/值對中的鍵與不可覆寫的引數相符，則會忽略傳遞的引數。

如需詳細資訊，請參閱 AWS Glue API 中的 the section called “觸發”。

新增觸發條件

您可以使用 AWS Glue 主控台、AWS Command Line Interface (AWS CLI) 或 AWS Glue API 新增觸發。

> **Note**
>
> 目前，AWS Glue 主控台在使用觸發時僅支援任務，不支援爬蟲程式。您可以使用 AWS CLI 或 AWS Glue API 來使用任務和爬蟲程式設定觸發。

新增觸發 (主控台)

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。

2. 在導覽窗格的 ETL 下，選擇 Triggers (觸發)。然後請選擇 Add trigger (新增觸發)。
3. 提供下列屬性：

   名稱

   為您的觸發條件設定唯一的獨特名稱。

   觸發條件類型

   請指定下列其中一項：

   - Schedule: (排程：) 觸發會以特定頻率和時間引發。
   - Job events: (任務事件：) 條件式觸發。觸發會在清單中的任何或所有任務與其指定的狀態相符時引發。若要引發觸發，監看的任務必須是由觸發所啟動。針對任何您選擇的任務，您只能監看一個任務事件 (完成狀態)。
   - On-demand: (隨需：) 觸發會在啟動時引發。
4. 完成觸發精靈。在 Review (檢閱) 頁面上，您可以啟用 Schedule (排程) 而 Job events (任務事件) (條件式) 會立即觸發，方法是選取 Enable trigger on creation (在建立時啟用觸發)。

新增觸發 (AWS CLI)

- 輸入與以下相似的命令。

```
aws glue create-trigger --name MyTrigger --type SCHEDULED --schedule  "cron(0 12 *
 * ? *)" --actions CrawlerName=MyCrawler --start-on-creation
```

這個命令會建立名為 `MyTrigger` 的排程觸發，在 UTC 時間每天下午 12:00 執行，並啟動名為 `MyCrawler` 的爬蟲程式。觸發會在已啟用的狀態下建立。

如需更多詳細資訊，請參閱 the section called “AWS Glue 觸發條件”。

任務和爬蟲程式以時間為基礎的排程

您可以在 AWS Glue 中為爬蟲程式和工作定義以時間為基礎的排程。這些排程的定義使用類似 Unix 的 cron 語法。以世界協調時間 (UTC) 指定時間，且排程的最小精度為 5 分鐘。

若要進一步了解設定任務和爬蟲程式以使用排程進行執行的相關資訊，請參閱使用觸發啟動任務和爬蟲程式。

Cron 表達式

Cron 表達式有六個必要欄位，以空格隔開。

Syntax (語法)

```
cron(Minutes Hours Day-of-month Month Day-of-week Year)
```

欄位	Values (數值)	Wildcards (萬用字元)
分鐘	0–59	, - * /
小時	0–23	, - * /
月中的日	1–31	, - * ? / L W
月	1-12 或 JAN-DEC	, - * /
週中的日	1-7 或 SUN-SAT	, - * ? / L
年	1970-2199	, - * /

萬用字元

- , (逗號) 萬用字元包含額外的值。在 `Month` 欄位，`JAN,FEB,MAR` 包括 January (一月)、February (二月)、March (三月)。
- - (破折號) 萬用字元用於指定範圍。在 `Day` 欄位中，1–15 包含指定月份的 1 至 15 號。
- * (星號) 包含欄位中所有的值。在 `Hours` 欄位，* 包含每個小時。
- / (斜線) 萬用字元用於指定增量。在 `Minutes` 欄位，您可以輸入 **`1/10`** 指定每 10 分鐘的間隔，從小時的第一分鐘開始 (例如第 11、第 21、第 31 分鐘)。
- ? (問號) 萬用字元用於表示不限定任何一個。在 `Day-of-month` 欄位，您可以輸入 7，如果您不在意這個月的 7 號是星期幾，就可以在 Day-of-week (週中的日) 欄位中輸入 ?。
- L 萬用字元在 `Day-of-month` 或 `Day-of-week` 欄位可指定月份或週的最後一天。
- W 萬用字元在 `Day-of-month` 欄位可指定任務日。在 `Day-of-month` 欄位，`3W` 指定的是月份中最接近第三個任務日的日子。

限制

- 您無法在同一個 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指定了數值，就必須在另一個欄位中使用 ? (問號)。
- 不支援頻率多於 5 分鐘的 Cron 表達式。

範例

建立排程時，您可以使用下列 cron 字串範例。

分鐘	小時	月中的日	月	週中的日	年	意義
0	10	*	*	?	*	在每天上午 10:00 (UTC) 執行
15	12	*	*	?	*	在每天下午 12:15 (UTC) 執行
0	18	?	*	MON-FRI	*	在每週一至週五下午 6:00 (UTC) 執行
0	8	1	*	?	*	在每個月第一天上午 8:00 (UTC) 執行
0/15	*	*	*	?	*	每 15 分鐘執行
0/10	*	?	*	MON-FRI	*	在週一至週五每 10 分鐘執行
0/5	8-17	?	*	MON-FRI	*	在週一至週五上午

分鐘	小時	月中的日	月	週中的日	年	意義
						8:00 至下午 5:55 (UTC) 之間每 5 分鐘執行

舉例來說，如果要在每天的 12:15 (UTC) 執行某項動作，則可指定如下：

```
cron(15 12 * * ? *)
```

啟用和停用觸發

您可以使用AWS Glue控制台、 AWS Command Line Interface (AWS CLI) 或 AWS Glue API 啟用或停用觸發器。

啟用或停用觸發器 (主控台)

1. 請登入 AWS Management Console 並開啟AWS Glue主控台，網址為 https://console.aws.amazon.com/glue/。
2. 在導覽窗格的 ETL 下，選擇 Triggers (觸發)。
3. 選取所需觸發旁邊的核取方塊，並在 Action (動作) 選單上，選擇 Enable trigger (啟用觸發) 來啟用觸發，或是選擇 Disable trigger (停用觸發) 來停用觸發。

啟用或停用觸發 (AWS CLI)

- 輸入以下其中一個命令。

  ```
  aws glue start-trigger --name MyTrigger

  aws glue stop-trigger --name MyTrigger
  ```

  啟動觸發會啟用觸發，停止觸發會停用觸發。當您啟用隨需觸發時，會立即引發觸發。

如需更多詳細資訊，請參閱 the section called “AWS Glue 觸發條件”。

在 AWS Glue 中使用藍圖和工作流程來執行複雜的 ETL 活動

一些組織的複雜擷取、轉換和載入 (ETL) 程序最好使用多個相依 AWS Glue 任務和爬蟲程式來實作。使用 AWS Glue 工作流程，您可以設計一個複雜的多任務、多爬蟲程式 ETL 程序，讓 AWS Glue 可以作為單一實體來執行和追蹤。建立工作流程並在工作流程中指定任務、爬蟲程式和觸發程序之後，您可以隨需或依排程執行工作流程。

主題

AWS Glue 中的工作流程概觀

在 AWS Glue 中，您可以使用工作流程來建立和視覺化包含多項爬蟲程式、任務和觸發的複雜性擷取、轉換和載入 (ETL) 活動。每項工作流程都管理其所有任務和爬蟲程式的執行和監控。因為工作流程執行每個元件，所以會記錄執行進度和狀態。這為您提供較大型任務和各步驟詳細資訊的概觀。AWS Glue 主控台以圖形視覺化呈現工作流程。

您可以從 AWS Glue 藍圖建立工作流程，或者您可以使用 AWS Management Console 或 AWS Glue API 一次建置一個元件以手動建立工作流程。如需有關藍圖的詳細資訊，請參閱 the section called “藍圖概觀”。

工作流程中的觸發可啟動任務或爬蟲程式，並可在任務和爬蟲程式完成時引發。使用觸發，您可以建立相互依存之任務和爬蟲程式的大型鏈結。除了定義任務和爬蟲程式相依性的工作流程中的觸發之外，每個工作流程都有啟動觸發。啟動觸發有三種類型：

- 排程 — 工作流程會根據您定義的排程啟動。排程可以是每日、每週、每月等，也可以是根據 cron 表達式的自訂排程。
- 隨需 - 工作流程是從 AWS Glue 主控台、API 或 AWS CLI 手動啟動。
- EventBridge - 工作流程會在單一 Amazon EventBridge 事件或一批 Amazon EventBridge 事件發生時啟動。使用此觸發類型，AWS Glue 可以是事件驅動架構中的事件消費者。任何 EventBridge 事件類型都可以啟動工作流程。一個常見的使用案例是 Amazon S3 儲存貯體中的新物件到達 (S3 PutObject 操作)。

  使用批次事件啟動工作流程表示等到收到指定數量的事件，或直到指定的時間過去。當您建立 EventBridge 事件觸發時，您可以選擇性地指定批次條件。如果您指定批次條件，則必須指定批次大小 (事件數)，並且可以選擇性地指定批次時段 (秒數)。預設和最大批次時段為 900 秒 (15 分鐘)。符合的批次條件會先啟動工作流程。當第一個事件到達時，批次時段會啟動。如果您在建立觸發時未指定批次條件，則批次大小預設為 1。

  當工作流程啟動時，批次條件會重設，事件觸發會開始監控下一個批次條件，以便再次啟動工作流程。

  下表顯示批次大小和批次時段如何一起運作以觸發工作流程。

批次大小	批次視窗	產生的觸發條件
10		工作流程會在 10 個 EventBridge 事件抵達時觸發，或在第一個事件抵達後 15 分鐘觸發 (以先發生者為準)。(如未指定時段大小，預設值為 15 分鐘。)
10	2 分鐘	工作流程會在 10 個 EventBridge 事件抵達時觸發，或在第一個事件抵達後 2 分鐘觸發 (以先發生者為準)。
1		工作流程會在第一個事件到達時觸發。窗口大小是無關緊要的。當您建立 EventBridge 事件觸發時如果未指定批次條件，則批次大小預設為 1。

  GetWorkflowRun API 操作會傳回觸發工作流程的批次條件。

無論工作流程的啟動方式為何，您都可以在建立工作流程時指定並行工作流程執行的最大數目。

如果事件或事件批次啟動最終失敗的工作流程執行，則不再考慮該事件或事件批次來啟動工作流程執行。只有在下一個事件或事件批次到達時，才會啟動新的工作流程執行。

> **Important**
>
> 將工作流程中的任務、爬蟲程式和觸發程序總數限制在 100 或更少。如果包含超過 100 個，則嘗試繼續或停止工作流程執行時可能會出現錯誤。

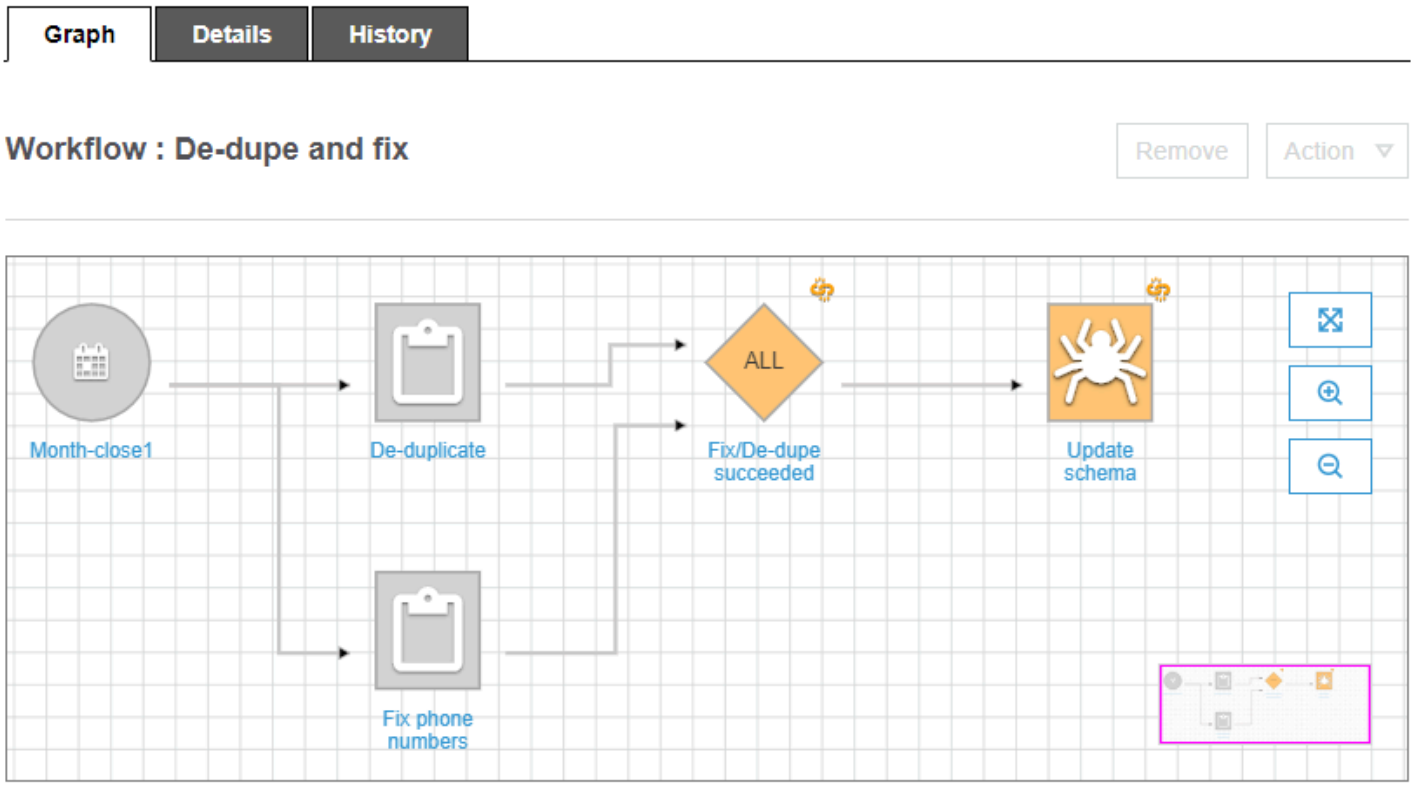
如果工作流程執行會超過為工作流程設定的並行限制，即使符合事件條件，也不會啟動工作流程執行。建議您根據預期的事件量調整工作流程並行限制。AWS Glue 不會重試因超出並行限制而失敗的工作流程執行。同樣地，建議您根據預期的事件數量，在工作流程中調整任務和爬蟲程式的並行限制。

工作流程執行屬性

若要共享和管理整個工作流程回合的狀態，您可以定義預設工作流程回合屬性。這些屬性都是名稱/值對，可供工作流程中的所有任務使用。使用 AWS Glue API，任務可以擷取工作流程回合屬性並修改它們，供工作流程的後續任務使用。

工作流程圖

下圖顯示 AWS Glue 主控台的基本工作流程圖形。您的工作流程可能擁有數十個元件。



此工作流程是由排程觸發 Month-close1 啟動，這會啟動兩項任務 De-duplicate 和 Fix phone numbers。成功完成這兩項任務時，事件觸發 Fix/De-dupe succeeded 會啟動爬蟲程式 Update schema。

靜態和動態工作流程檢視

每項工作流程都有「靜態檢視」和「動態檢視」的概念。靜態檢視指出工作流程的設計。動態檢視是包含每項任務和爬蟲程式最新執行資訊的執行時間檢視。執行資訊包含成功狀態和錯誤的詳細資訊。

當工作流程執行時，主控台會顯示動態檢視，以圖形指示已經完成和還在執行的任務。您也可以使用 AWS Glue API 擷取執行中工作流程的動態檢視。如需更多詳細資訊，請參閱 使用 AWS Glue API 查詢工作流程。

另請參閱

- the section called “從藍圖建立工作流程”
- the section called “手動建立和建構工作流程”
- 工作流程 (適用於工作流程 API)

在 AWS Glue 中手動建立和建構工作流程

您可以使用 AWS Glue 主控台，手動一次一個節點地建立和建置工作流程。

工作流程包含任務、爬蟲程式和觸發。手動建立工作流程前，請先建立工作流程要包含的任務和爬蟲程式。最好指定工作流程的隨需執行爬蟲程式。您可在建立工作流程時建立新的觸發，或者將現有的觸發「複製」到工作流程。當您複製觸發時，所有與觸發相關聯的目錄物件 (觸發它的任務或爬蟲程式以及啟動的任務或爬蟲程式) 都會新增至工作流程。

> **Important**
>
> 將工作流程中的任務、爬蟲程式和觸發程序總數限制在 100 或更少。如果包含超過 100 個，則嘗試繼續或停止工作流程執行時可能會出現錯誤。

您可以將觸發新增到工作流程圖，並定義每項觸發的監看事件和動作，藉以建構您的工作流程。您從「啟動觸發」開始，它可以是隨需觸發或排程觸發，然後透過新增事件 (條件式) 觸發完成圖形。

步驟 1：建立工作流程

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格的 ETL 下，選擇 Workflows (工作流程)。
3. 選擇 Add workflow (新增工作流程)，然後完成 Add a new ETL workflow (新增新的 ETL 工作流程) 表單。

   您新增的任何選用預設回合屬性，都會以引數形式提供給工作流程中的所有任務使用。如需更多詳細資訊，請參閱 在 AWS Glue 中取得及設定工作流程執行屬性。
4. 選擇 Add workflow (新增工作流程)。

   新的工作流程會出現在 Workflows (工作流程) 頁面的清單中。

步驟 2：新增啟動觸發

1. 在 Workflows (工作流程) 頁面中，選擇您的新工作流程。然後，在該頁面底部，確定選取 Graph (圖形) 索引標籤。
2. 選擇 Add trigger (新增觸發)，然後在 Add trigger (新增觸發) 對話方塊中執行以下其中一項操作：

- 選擇 Clone existing (複製現有的項目)，然後選擇要複製的觸發。接著選擇 Add (新增)。

  此觸發會和它監看的任務及爬蟲程式以及它啟動的任務及爬蟲程式一起出現在圖形中。

  如果不小心選了錯誤的觸發，請選取圖形中的觸發，然後選擇 Remove (移除)。

- 選擇 Add new (新增)，然後完成 Add trigger (新增觸發) 表單。

  1. 對於 Trigger type (觸發類型)，選取 Schedule (排程)、On demand (隨需) 或 EventBridge。

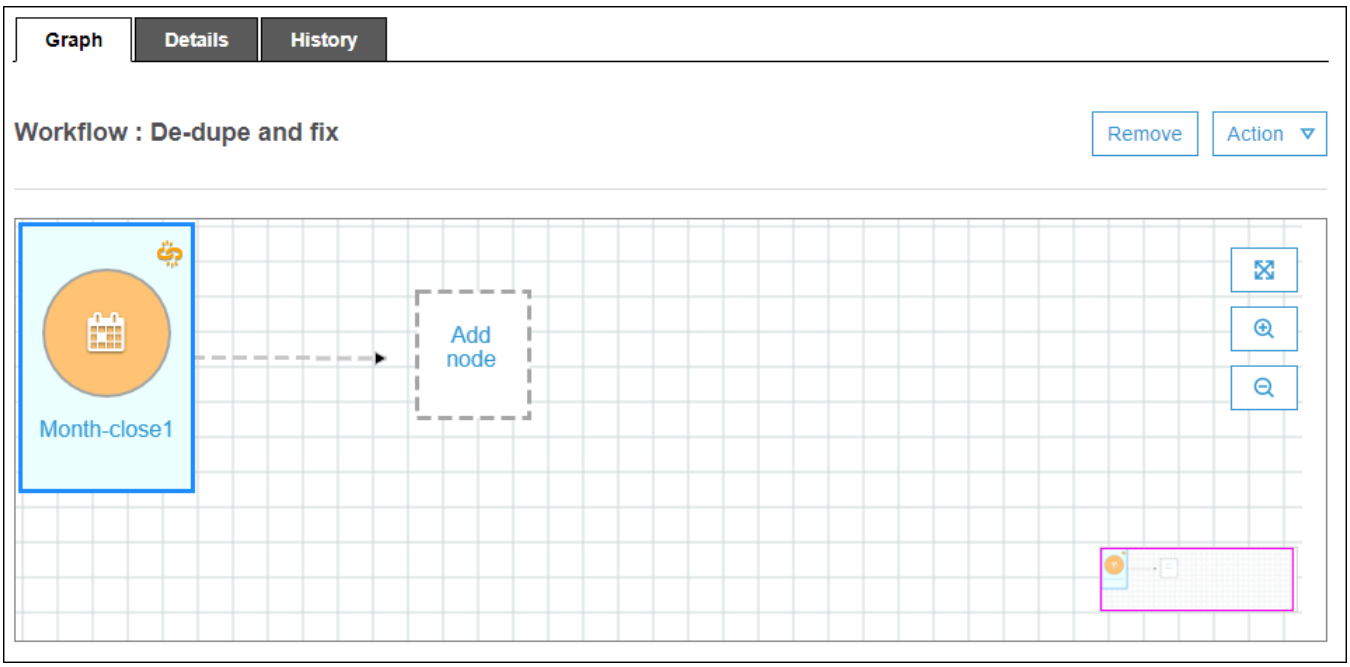
     對於觸發類型 Schedule (排程)，選擇其中一個 Frequency (頻率) 選項。選擇 Custom (自訂) 輸入 `cron` 表達式。

     對於觸發類型 EventBridge event (EventBridge 事件)，輸入 Number of events (事件數) (批次大小)，並選擇性地輸入 Time delay (時間延遲) (批次時段)。如果您省略 Time delay (時間延遲)，批次時段預設為 15 分鐘。如需更多詳細資訊，請參閱 AWS Glue 中的工作流程概觀。

  2. 選擇 Add (新增)。

  此觸發會和預留位置節點 (標記為 Add node (新增節點)) 一起出現在圖形中。在下列範例中，啟動觸發是名為 `Month-close1` 的排程觸發。

  此時，尚未儲存觸發。



3. 如已新增新的觸發，請完成以下步驟：

a. 執行下列任意一項：

- 選擇預留位置節點 (Add node (新增節點))。
- 確認選取啟動觸發，然後在圖形上方的 Action (動作) 選單中，選擇 Add jobs/crawlers to trigger (新增要觸發的任務/爬蟲程式)。

b. 在 Add jobs(s) and crawler(s) to trigger (新增要觸發的任務/爬蟲程式) 對話方塊中，選取一或多項任務或爬蟲程式，然後選擇 Add (新增)。

觸發已儲存，選取的任務或爬蟲程式會出現在圖形中，有來自觸發的連接器。

如果您不小心新增了錯誤的任務或爬蟲程式，您可以選取觸發或連接器，然後選擇 Remove (移除)。

步驟 3：新增更多觸發

新增更多類型為 Event (事件) 的觸發以繼續建構您的工作流程。若要縮放或放大圖形畫布，請使用圖形右側的圖示。針對每項要新增的觸發，請完成以下步驟：

> **Note**
>
> 沒有任何動作可儲存工作流程。新增最後一個觸發並將動作指派給觸發後，工作流程即完成並儲存。您可以隨時返回並新增更多節點。

1. 執行下列任意一項：

- 若要複製現有的觸發，請確保未選取圖形中的任何節點，然後在 Action (動作) 選單中選擇 Add trigger (新增觸發)。
- 若要新增可監看圖形中特定任務或爬蟲程式的新觸發，請選取任務或爬蟲程式節點，然後選擇 Add trigger (新增觸發) 預留位置節點。

您可以新增更多任務或爬蟲程式，以在後續步驟中監看此觸發。

2. 在 Add trigger (新增觸發) 對話方塊中，執行下列其中一項操作：

- 選擇 Add new (新增)，然後完成 Add trigger (新增觸發) 表單。接著選擇 Add (新增)。

此觸發會出現在圖形中。您會在後續步驟中完成此觸發。

- 選擇 Clone existing (複製現有的項目)，然後選擇要複製的觸發。接著選擇 Add (新增)。

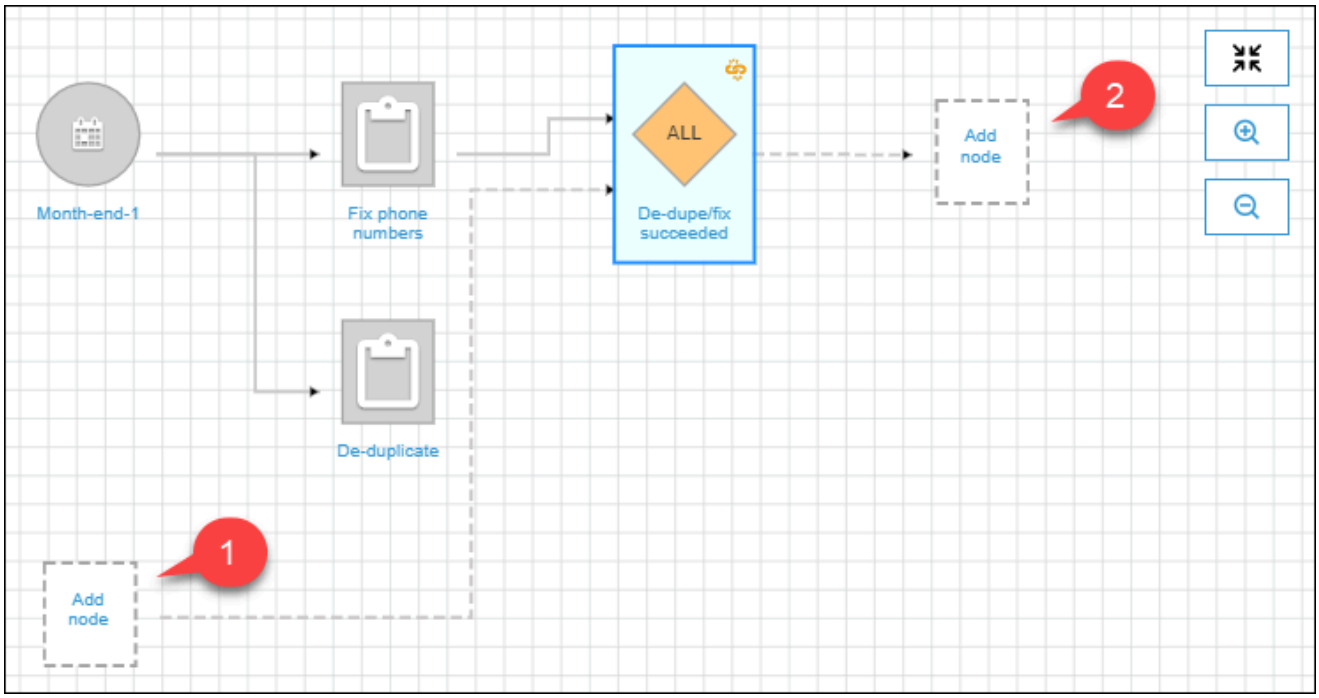
此觸發會和它監看的任務及爬蟲程式以及它啟動的任務及爬蟲程式一起出現在圖形中。

如果您不小心選擇了錯誤的觸發，請在圖形中選取該觸發，然後選擇 Remove (移除)。

3. 如已新增新的觸發，請完成以下步驟：

   a. 選取新的觸發。

      當以下圖形出現時，會選取觸發 `De-dupe/fix succeeded`，且預留位置節點顯示為要監看的 (1) 事件和 (2) 動作。



   b. (如果觸發已監看某個事件，而您想要新增更多監看的任務或爬蟲程式，則為選用。) 選擇要監看事件的預留位置節點，然後在 Add job(s) and crawler(s) to watch (新增要監看的任務和爬蟲程式) 對話方塊中，選取一或多項任務或爬蟲程式。選擇要監看的事件 (SUCCEEDED、FAILED 等等)，然後選擇 Add (新增)。

   c. 確認已選取觸發，然後選擇動作預留位置節點。

   d. 在 Add job(s) and crawler(s) to watch (新增要監看的任務和爬蟲程式) 對話方塊中，選取一或多項任務或爬蟲程式，然後選擇 Add (新增)。

      已選取的任務和爬蟲程式會出現在圖形中，有來自觸發的連接器。

如需工作流程和藍圖的詳細資訊，請參閱下列主題。

- AWS Glue 中的工作流程概觀
- 在 AWS Glue 中執行和監控工作流程
- 在 AWS Glue 中從藍圖建立工作流程

使用 Amazon EventBridge 事件啟動 AWS Glue 工作流程

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

透過 EventBridge 支援，AWS Glue 可以在事件驅動的架構中作為事件生產者和消費者。對於工作流程，AWS Glue 支援任何類型的 EventBridge 事件作為消費者。最常見的使用案例是 Amazon S3 儲存貯體中的新物件。如果您的資料以不規則或未定義的間隔送達，您可以儘可能接近其到達處理此資料。

> **Note**
>
> AWS Glue 不提供 EventBridge 訊息的保證傳遞。如果 EventBridge 傳送重複的訊息，AWS Glue 不會執行重複資料刪除。您必須根據使用案例來管理冪等性。
> 請務必正確設定 EventBridge 規則，以避免傳送不想要的事件。

開始之前

如果您想要使用 Amazon S3 資料事件啟動工作流程，則必須確保將感興趣的 S3 儲存貯體的事件記錄到 AWS CloudTrail 和 EventBridge。若要這麼做，您必須建立 CloudTrail 線索。如需詳細資訊，請參閱建立 AWS 帳戶的追蹤。

使用 EventBridge 事件啟動工作流程

> **Note**
>
> 在下列命令中：
>
> - 將 *<workflow-name>* 取代為要指派給工作流程的名稱。
> - 將 *<trigger-name>* 取代為要指派給觸發的名稱。
> - 將 *<bucket-name>* 取代為 Amazon S3 儲存貯體的名稱。

- 將 *<account-id>* 取代為有效的 AWS 帳戶 ID。
- 將 *<region>* 取代為區域的名稱 (如 us-east-1)。
- 將 *<rule-name>* 取代為要指派給 EventBridge 規則的名稱。

1. 確定您擁有建立和檢視 EventBridge 規則和目標的 AWS Identity and Access Management (IAM) 許可。下列是您可以連接的範例政策。您可能希望將其範圍限制為對操作和資源進行限制。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "events:PutRule",
                "events:DisableRule",
                "events:DeleteRule",
                "events:PutTargets",
                "events:RemoveTargets",
                "events:EnableRule",
                "events:List*",
                "events:Describe*"
            ],
            "Resource": "*"
        }
    ]
}
```

2. 建立 EventBridge 服務在將事件傳遞至 AWS Glue 時可擔任的 IAM 角色。

   a. 在 IAM 主控台的 Create role (建立角色)頁面上，選擇AWS Service (AWS 服務)。然後選擇服務 CloudWatch Events。

   b. 完成 Create role (建立角色) 精靈。精靈會自動連接 `CloudWatchEventsBuiltInTargetExecutionAccess` 和 `CloudWatchEventsInvocationAccess` 政策。

   c. 將下列內嵌策略連接到角色。此政策允許 EventBridge 服務將事件導向至 AWS Glue。

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": [
        "glue:notifyEvent"
      ],
      "Resource": [
        "arn:aws:glue:<region>:<account-id>:workflow/<workflow-name>"
      ]
    }
  ]
}
```

3. 輸入下列命令以建立工作流程。

   如需其他選用命令列參數的相關資訊，請參閱 AWS CLI 命令參考中的 create-workflow。

   ```
   aws glue create-workflow --name <workflow-name>
   ```

4. 輸入下列命令為工作流程建立 EventBridge 事件觸發。這將是工作流程的啟動觸發。將 *<actions>* 替換成要執行的動作 (要啟動的任務和爬蟲程式)。

   請參閱 AWS CLI 命令參考中的 create-trigger 以取得如何編寫 `actions` 引數程式碼的詳細資訊。

   ```
   aws glue create-trigger --workflow-name <workflow-name> --type EVENT --
   name <trigger-name> --actions <actions>
   ```

   如果您希望工作流程由一批事件而非單一 EventBridge 事件觸發，請改為輸入下列命令。

   ```
   aws glue create-trigger --workflow-name <workflow-name> --type EVENT
    --name <trigger-name> --event-batching-condition BatchSize=<number-of-
   events>,BatchWindow=<seconds> --actions <actions>
   ```

   對於 `event-batching-condition` 引數，`BatchSize` 是必要的，而 `BatchWindow` 是選用的。如果忽略 `BatchWindow`，時段會預設為 900 秒，也就是時段大小上限。

   Example

   下列範例會建立觸發，以在三個 EventBridge 事件到達後或第一個 EventBridge 事件到達後五分鐘 (以先到達者為準) 啟動 `eventtest` 工作流程。

```
aws glue create-trigger --workflow-name eventtest --type EVENT --name objectArrival
 --event-batching-condition BatchSize=3,BatchWindow=300 --actions JobName=test1
```

5. 在 Amazon EventBridge 中建立規則。

   a. 在您偏好的文字編輯器中建立規則詳細資訊的 JSON 物件。

   下列範例將 Amazon S3 指定為事件來源、PutObject 作為事件名稱，並將儲存貯體名稱作為請求參數。當新物件到達儲存貯體時，此規則會啟動工作流程。

```
{
  "source": [
    "aws.s3"
  ],
  "detail-type": [
    "AWS API Call via CloudTrail"
  ],
  "detail": {
    "eventSource": [
      "s3.amazonaws.com"
    ],
    "eventName": [
      "PutObject"
    ],
    "requestParameters": {
      "bucketName": [
        "<bucket-name>"
      ]
    }
  }
}
```

   若要在新物件到達儲存貯體內的資料夾時啟動工作流程，您可以將下列程式碼替換為 requestParameters。

```
    "requestParameters": {
      "bucketName": [
        "<bucket-name>"
      ]
      "key" : [{ "prefix" : "<folder1>/<folder2>/*"}}]
  }
```

b. 使用您偏好的工具將規則 JSON 物件轉換為逸出的字串。

```
{\n  \"source\": [\n    \"aws.s3\"\n  ],\n  \"detail-type\": [\n    \"AWS API
 Call via CloudTrail\"\n  ],\n  \"detail\": {\n    \"eventSource\": [\n
 \"s3.amazonaws.com\"\n    ],\n    \"eventName\": [\n      \"PutObject\"\n
  ],\n    \"requestParameters\": {\n      \"bucketName\": [\n        \"<bucket-
name>\"\n      ]\n    }\n  }\n}
```

c. 執行下列命令以建立 JSON 參數範本，您可以編輯該範本，以指定後續 `put-rule` 命令。將輸出儲存在檔案中。在此範例中，檔案稱為 `ruleCommand`。

```
aws events put-rule --name <rule-name> --generate-cli-skeleton >ruleCommand
```

如需關於 `--generate-cli-skeleton` 參數的詳細資訊，請參閱《AWS 命令行介面使用者指南》中的從 JSON 或 YAML 輸入檔案產生 AWS CLI Skeleton 及輸入參數。

輸出檔案現在應該與下列類似。

```
{
    "Name": "",
    "ScheduleExpression": "",
    "EventPattern": "",
    "State": "ENABLED",
    "Description": "",
    "RoleArn": "",
    "Tags": [
        {
            "Key": "",
            "Value": ""
        }
    ],
    "EventBusName": ""
}
```

d. 編輯檔案以選擇性地移除參數，並指定至少 `Name`、`EventPattern` 以及 `State` 參數。對於 `EventPattern` 參數，提供您在上一個步驟中建立的規則詳細資訊提供逸出字串。

```
{
    "Name": "<rule-name>",
    "EventPattern": "{\n  \"source\": [\n    \"aws.s3\"\n  ],\n  \"detail-
type\": [\n    \"AWS API Call via CloudTrail\"\n  ],\n  \"detail\": {\n
```

```
 \"eventSource\": [\n      \"s3.amazonaws.com\"\n    ],\n    \"eventName\": [\n
      \"PutObject\"\n    ],\n    \"requestParameters\": {\n      \"bucketName
\": [\n        \"<bucket-name>\"\n      ]\n    }\n  }\n}",
    "State": "DISABLED",
    "Description": "Start an AWS Glue workflow upon new file arrival in an
 Amazon S3 bucket"
}
```

> **Note**
>
> 最好在您完成建立工作流程之前，將規則保持停用狀態。

e. 輸入下列 put-rule 命令，該命令會從檔案 ruleCommand 中讀取輸入參數。

```
aws events put-rule --name <rule-name> --cli-input-json file://ruleCommand
```

以下輸出表示成功。

```
{
    "RuleArn": "<rule-arn>"
}
```

6. 輸入下列命令將規則連接至目標。目標是 AWS Glue 中的工作流程。將 *<role-name>* 取代為您在此程序開始時建立的角色。

```
aws events put-targets --rule <rule-name> --targets
 "Id"="1","Arn"="arn:aws:glue:<region>:<account-id>:workflow/<workflow-
name>","RoleArn"="arn:aws:iam::<account-id>:role/<role-name>" --region <region>
```

以下輸出表示成功。

```
{
    "FailedEntryCount": 0,
    "FailedEntries": []
}
```

7. 輸入下列命令，確認規則與目標的連線成功。

```
aws events list-rule-names-by-target --target-arn arn:aws:glue:<region>:<account-
id>:workflow/<workflow-name>
```

以下輸出表示成功，其中 *<rule-name>* 是您建立的規則名稱。

```
{
    "RuleNames": [
        "<rule-name>"
    ]
}
```

8. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
9. 選取工作流程，並確認啟動觸發及其動作 (其啟動的任務或爬蟲程式) 顯示在工作流程圖形上。然後繼續 步驟 3：新增更多觸發 中的程序。或者使用 AWS Glue API 或 AWS Command Line Interface 新增其他元件至工作流程。
10. 完全指定工作流程後，請啟用規則。

```
aws events enable-rule --name <rule-name>
```

工作流程現已就緒，可由 EventBridge 事件或事件批次啟動。

> **另請參閱**
>
> - Amazon EventBridge 使用者指南
> - AWS Glue 中的工作流程概觀
> - 在 AWS Glue 中手動建立和建構工作流程

檢視啟動工作流程的 EventBridge 事件

您可以檢視啟動工作流程的 Amazon EventBridge 事件的事件 ID。如果您的工作流程是由一批事件啟動，您可以檢視批次中所有事件的事件 ID。

對於批次大小大於 1 的工作流程，您也可以查看哪個批次條件啟動工作流程：批次大小中的事件數目到達，或批次時段的到期。

檢視啟動工作流程的 EventBridge 事件 (主控台)

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格中，選擇 Workflows (工作流程)。
3. 選取工作流程。然後在底部，選擇 History (歷史記錄) 索引標籤。
4. 選擇工作流程執行，然後選擇 View run details (檢視執行詳細資訊)。
5. 在執行詳細資訊頁面上，找到 Run properties (執行屬性) 欄位，然後尋找 aws:eventIds 金鑰。

   該金鑰的值是 EventBridge 事件 ID 的清單。

檢視啟動工作流程的 EventBridge 事件 (AWS API)

- 在您的 Python 指令碼中包含以下程式碼。

```
workflow_params =
 glue_client.get_workflow_run_properties(Name=workflow_name,RunId=workflow_run_id)
batched_events = workflow_params['aws:eventIds']
```

  `batched_events` 將是字串清單，其中每個字串都是事件 ID。

> **另請參閱**
>
> - Amazon EventBridge 使用者指南
> - the section called “工作流程概觀”

在 AWS Glue 中執行和監控工作流程

如果工作流程的啟動觸發是隨需觸發，您可以從 AWS Glue 主控台啟動工作流程。完成下列步驟來執行和監控工作流程。如果工作流程失敗，您可以檢視執行圖形，判斷失敗的節點。若要協助疑難排解，如果工作流程是從藍圖建立的，您可以檢視藍圖執行以查看用於建立工作流程的藍圖參數值。如需詳細資訊，請參閱 the section called “檢視藍圖執行”。

您可以用 AWS Glue 主控台、API 或 AWS Command Line Interface (AWS CLI) 執行和監控工作流程。

執行和監控工作流程 (主控台)

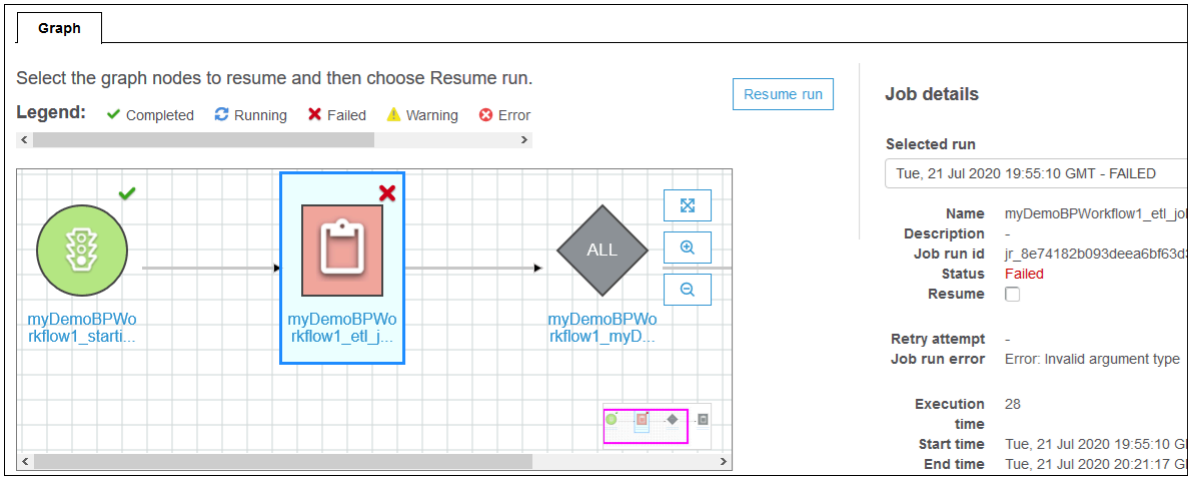
1. 開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格的 ETL 下，選擇 Workflows (工作流程)。
3. 選取工作流程。在 Actions (動作) 選單中，選擇 Run (執行)。
4. 檢查工作流程清單中的 Last run status (上一個執行狀態) 欄。選擇重新整理按鈕以檢視進行中的工作流程狀態。
5. 當工作流程執行時或工作流程完成 (或失敗) 之後，請完成下列步驟來檢視執行詳細資訊。

   a. 確認已選取工作流程，然後選擇 History (歷史記錄) 標籤。

   b. 選擇目前或最近執行的工作流程，然後選擇 View run details (檢視執行詳細資訊)。

      工作流程執行時期圖形會顯示目前的執行狀態。

   c. 選擇圖表中的任何節點，以檢視節點的詳細資訊和狀態。



執行和監控工作流程 (AWS CLI)

1. 輸入以下命令。使用要執行的工作流程取代 *<workflow-name>*。

```
aws glue start-workflow-run --name <workflow-name>
```

   如果成功啟動工作流程，命令會傳回執行 ID。

2. 藉由使用 `get-workflow-run` 命令可檢視工作流程執行狀態。提供工作流程名稱和執行 ID。

```
aws glue get-workflow-run --name myWorkflow --run-id
 wr_d2af14217e8eae775ba7b1fc6fc7a42c795aed3cbcd8763f9415452e2dbc8705
```

下列為範例命令輸出。

```
{
    "Run": {
        "Name": "myWorkflow",
        "WorkflowRunId":
 "wr_d2af14217e8eae775ba7b1fc6fc7a42c795aed3cbcd8763f9415452e2dbc8705",
        "WorkflowRunProperties": {
            "run_state": "COMPLETED",
            "unique_id": "fee63f30-c512-4742-a9b1-7c8183bdaae2"
        },
        "StartedOn": 1578556843.049,
        "CompletedOn": 1578558649.928,
        "Status": "COMPLETED",
        "Statistics": {
            "TotalActions": 11,
            "TimeoutActions": 0,
            "FailedActions": 0,
            "StoppedActions": 0,
            "SucceededActions": 9,
            "RunningActions": 0,
            "ErroredActions": 0
        }
    }
}
```

另請參閱:

- the section called “工作流程概觀”
- the section called “藍圖概觀”

停止工作流程執行

您可以使用 AWS Glue 主控台、AWS Command Line Interface (AWS CLI) 或 AWS Glue API 來停止工作流程執行。當您停止工作流程執行時，所有執行中任務和爬蟲程式都會立即終止，尚未啟動的任務和爬蟲程式則永遠不會啟動。所有執行中任務和爬蟲程式可能需要一分鐘的時間才會停止。工作流程

執行狀態會從 Running (執行中) 變成 Stopping (停止中)，而當工作流程執行完全停止時，狀態會變成 Stopped (已停止)。

在工作流程執行停止之後，您可以檢視執行圖形，查看哪些任務和爬蟲程式已完成，哪些從未啟動。然後，您可以判斷是否必須執行任何步驟以確保資料完整性。停止工作流程執行會導致系統不執行任何自動復原操作。

停止工作流程執行 (主控台)

1. 開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格的 ETL 下，選擇 Workflows (工作流程)。
3. 選擇執行中的工作流程，然後選擇 History (歷程記錄) 標籤。
4. 選擇工作流程執行，然後選擇 Stop run (停止執行)。

   執行狀態會變更為 Stopping (停止中)。
5. (選用) 選擇工作流程執行，選擇 View run details (檢視執行詳細資訊)，然後檢視執行圖形。

停止工作流程執行 (AWS CLI)

- 輸入以下命令。使用工作流程名稱取代 *<workflow-name>*，並用要停止之工作流程執行的執行 ID 取代 *<run-id>*。

  ```
  aws glue stop-workflow-run --name <workflow-name> --run-id <run-id>
  ```

  以下是 stop-workflow-run 命令的範例。

  ```
  aws glue stop-workflow-run --name my-workflow --run-id
   wr_137b88917411d128081069901e4a80595d97f719282094b7f271d09576770354
  ```

修復和繼續工作流程執行

如果工作流程中的一或多個節點 (任務或爬蟲程式) 未成功完成，這表示工作流程僅部分執行。找到根本原因並進行更正後，您可以選取一或多個節點以從其繼續工作流程執行，然後繼續工作流程執行。接著會執行選取的節點以及從這些節點下游的所有節點。

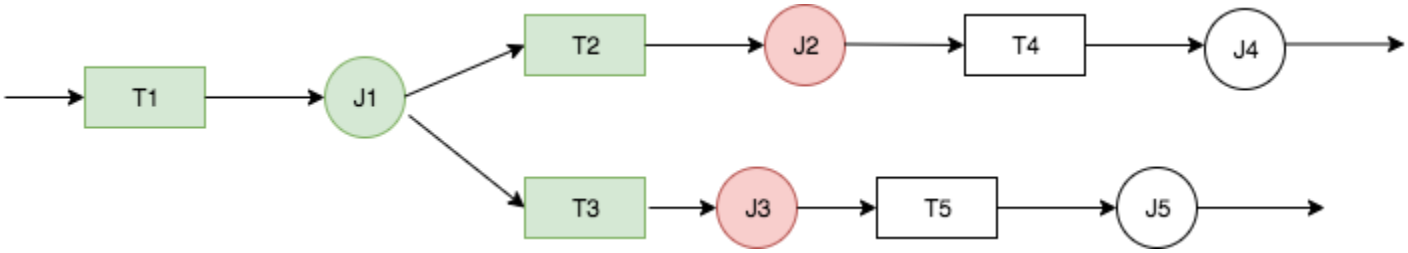
主題

- 繼續工作流程執行：運作方式

- 繼續工作流程執行
- 繼續工作流程執行的注意事項與限制

繼續工作流程執行：運作方式

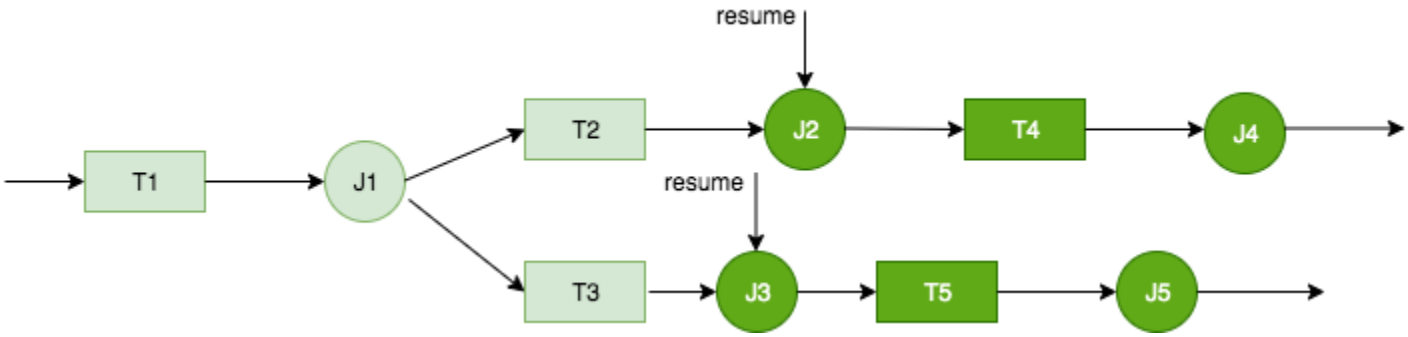
考慮下圖所示的工作流程 W1。



工作流程執行會以下列方式繼續：

1. 觸發器 T1 啟動任務 J1。
2. 成功完成 J1 會引發觸發器 T2 和 T3，它們分別執行任務 J2 和 J3。
3. 任務 J2 和 J3 失敗。
4. 觸發器 T4 和 T5 取決於 J2 和 J3 的成功完成，因此它們不會觸發，並且任務 J4 和 J5 不會執行。工作流程 W1 只是部分執行。

現在假設導致 J2 和 J3 失敗的問題得到更正。會選取 J2 和 J3 作為繼續工作流程執行的起點。



工作流程執行會以下列方式繼續：

1. 任務 J2 和 J3 成功執行。
2. 觸發器 T4 和 T5 引發。
3. 任務 J4 和 J5 成功執行。

已繼續的工作流程執行會以具有新執行 ID 的個別工作流程執行來進行追蹤。當您檢視工作流程歷史記錄時，您可以檢視任何工作流程執行的先前執行 ID。在下列螢幕擷取畫面範例中，具有執行 ID `wr_c7a22...` 的工作流程執行 (第二列) 有一個沒有完成的節點。使用者修正問題並繼續工作流程執行，產生執行 ID `wr_a07e55...` (第一列)。

> **Note**
>
> 在本討論的其餘部分，「已繼續的工作流程執行」一詞指的是先前工作流程執行繼續時所建立的工作流程執行。「原始工作流程執行」是指只有部分執行且需要繼續的工作流程執行。

已繼續的工作流程執行圖

在已繼續的工作流程執行中，雖然只執行一部分節點，但執行圖是完整的圖形。也就是說，未在已繼續的工作流程中執行的節點會從原始工作流程執行的執行圖中複製。在原始工作流程執行中執行的複製任務和爬蟲程式節點包括執行詳細資訊。

再次考慮上圖中的工作流程 W1。從 J2 和 J3 開始繼續工作流程執行時，已繼續的工作流程執行的執行圖會顯示所有任務 (J1 至 J5)，以及所有觸發器 (T1 至 T5)。J1 的執行詳細資訊會從原始工作流程執行複製。

工作流程執行快照

當工作流程執行啟動時，AWS Glue 會在該時間點拍攝工作流程設計圖的快照。此快照用於工作流程執行期間。如果您在執行開始後對任何觸發器進行變更，這些變更不會影響目前的工作流程執行。快照可確保工作流程以一致的方式繼續執行。

快照只會讓觸發器變成不可變。您在工作流程執行期間對下游任務和爬蟲程式所做的變更，會對目前的執行生效。

繼續工作流程執行

請依照下列步驟繼續工作流程執行。您可以用 AWS Glue 主控台、API 或 AWS Command Line Interface (AWS CLI) 繼續工作流程執行。

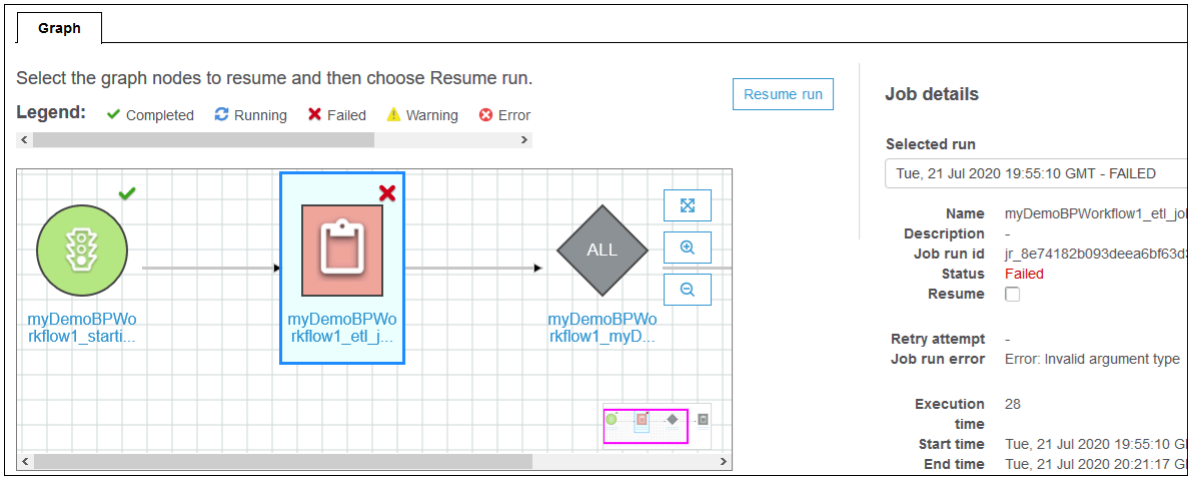
繼續工作流程執行 (主控台)

1. 開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。

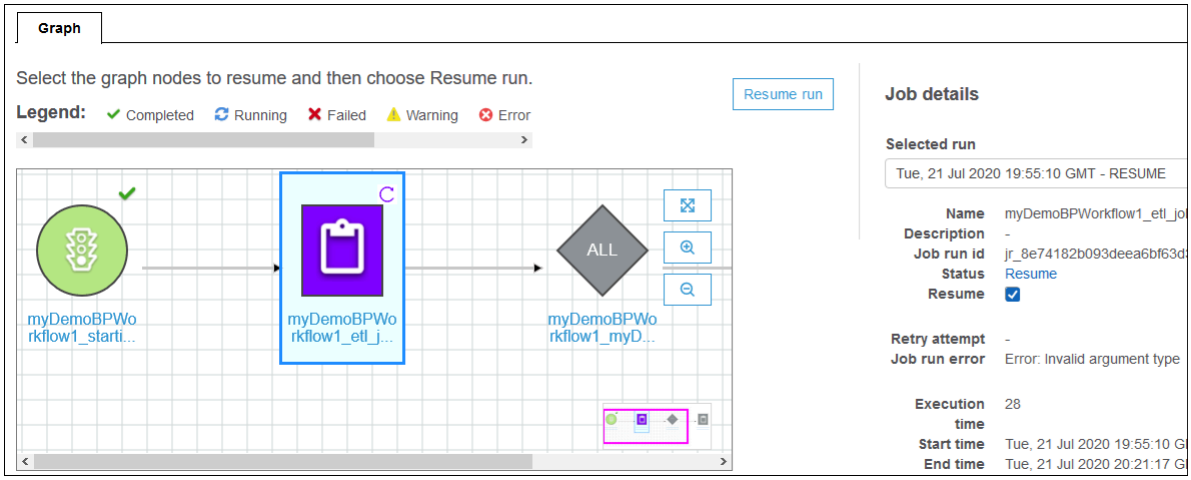
   以具有檢視工作流程和繼續工作流程執行之許可的使用者身分登入。

   > **Note**
   >
   > 若要繼續工作流程執行，您需要 `glue:ResumeWorkflowRun` AWS Identity and Access Management (IAM) 許可。

2. 在導覽窗格中，選擇 Workflows (工作流程)。
3. 選擇工作流程，然後選擇 History (歷程記錄) 索引標籤。
4. 選擇只有部分執行的工作流程執行，然後選擇 View run details (檢視執行詳細資訊)。
5. 在執行圖中，選取您想要重新啟動以及您想要從其繼續工作流程執行的第一個 (或唯一) 節點。
6. 在圖形右側的詳細資訊窗格中，選取 Resume (繼續) 核取方塊。



節點會變更顏色，並在右上角顯示一個小的繼續圖示。



7. 對於要重新啟動的任何其他節點，完成先前兩個步驟。
8. 選擇 Resume run (繼續執行)。

繼續工作流程執行 (AWS CLI)

1. 請確定您具備 `glue:ResumeWorkflowRun` IAM 許可。
2. 擷取您要重新啟動之節點的節點 ID。

   a. 執行原始工作流程執行的 `get-workflow-run` 命令。提供工作流程名稱和執行 ID，然後新增 `--include-graph` 選項，如下列範例所示。從主控台的 History (歷史記錄) 索引標籤，或執行 `get-workflow` 命令來取得執行 ID。

   ```
   aws glue get-workflow-run --name cloudtrailtest1 --run-id
    wr_a07e55f2087afdd415a404403f644a4265278f68b13ba3da08c71924ebe3c3a8 --include-
   graph
   ```

   此命令會將圖形的節點和邊緣當作大型 JSON 物件傳回。

   b. 由節點物件的 `Type` 和 `Name` 屬性找到感興趣的節點。

   以下是輸出中的範例節點物件。

   ```
   {
       "Type": "JOB",
       "Name": "test1_post_failure_4592978",
       "UniqueId":
    "wnode_d1b2563c503078b153142ee76ce545fe5ceef66e053628a786ddd74a05da86fd",
       "JobDetails": {
   ```

```
        "JobRuns": [
            {
                "Id":
 "jr_690b9f7fc5cb399204bc542c6c956f39934496a5d665a42de891e5b01f59e613",
                "Attempt": 0,
                "TriggerName": "test1_aggregate_failure_649b2432",
                "JobName": "test1_post_failure_4592978",
                "StartedOn": 1595358275.375,
                "LastModifiedOn": 1595358298.785,
                "CompletedOn": 1595358298.785,
                "JobRunState": "FAILED",
                "PredecessorRuns": [],
                "AllocatedCapacity": 0,
                "ExecutionTime": 16,
                "Timeout": 2880,
                "MaxCapacity": 0.0625,
                "LogGroupName": "/aws-glue/python-jobs"
            }
        ]
    }
}
```

c. 從節點物件的 UniqueId 屬性取得節點 ID。

3. 執行 resume-workflow-run 命令。提供以空格分隔的工作流程名稱、執行 ID 以及節點 ID 清單，如下列範例所示。

```
aws glue resume-workflow-run --name cloudtrailtest1 --run-id
 wr_a07e55f2087afdd415a404403f644a4265278f68b13ba3da08c71924ebe3c3a8 --node-
ids wnode_ca1f63e918fb855e063aed2f42ec5762ccf71b80082ae2eb5daeb8052442f2f3
 wnode_d1b2563c503078b153142ee76ce545fe5ceef66e053628a786ddd74a05da86fd
```

命令會輸出已繼續 (新) 工作流程執行的執行 ID，以及將要啟動的節點清單。

```
{
    "RunId": "wr_2ada0d3209a262fc1156e4291134b3bd643491bcfb0ceead30bd3e4efac24de9",
    "NodeIds": [
        "wnode_ca1f63e918fb855e063aed2f42ec5762ccf71b80082ae2eb5daeb8052442f2f3"
    ]
}
```

請注意，雖然範例 `resume-workflow-run` 命令列出會重新啟動兩個節點，但範例輸出指出只有一個節點會重新啟動。這是因為一個節點位於另一個節點的下游，而下游節點仍會由工作流程的正常流程重新啟動。

繼續工作流程執行的注意事項與限制

繼續工作流程執行時，請記住下列注意事項和限制。

- 只有當工作流程處於 `COMPLETED` 狀態時，您才能繼續工作流程執行。

> **Note**
>
> 即使工作流程執行中的一個或多個節點未完成，工作流程執行狀態也會顯示為 `COMPLETED`。請務必檢查執行圖以發現任何未成功完成的節點。

- 您可以從原始工作流程執行嘗試執行的任何任務或爬蟲程式節點，繼續工作流程執行。您無法從觸發器節點繼續工作流程執行。
- 重新啟動節點並不會重設其狀態。部分處理的任何資料都不會復原。
- 您可以多次繼續相同的工作流程執行。如果已繼續的工作流程只有部分執行，您可以解決問題然後繼續已繼續的執行。
- 如果您選取兩個要重新啟動的節點，而且它們彼此相依，則上游節點會在下游節點之前執行。事實上，選取下游節點是多餘的，因為它會根據工作流程的正常流程來執行。

在 AWS Glue 中取得及設定工作流程執行屬性

在您的 AWS Glue 工作流程中，使用工作流程回合屬性在任務中共享和管理狀態。您可以在建立工作流程時設定預設回合屬性。然後，當任務執行時，它們可以擷取回合屬性值並選擇性修改它們，供後續工作流程中的任務輸入使用。當任務修改回合屬性時，只有工作流程回合有新的值。預設的執行屬性不會受到影響。

如果您的 AWS Glue 任務並非工作流程的一部分，則系統將不會設定這些屬性。

以下來自擷取、轉換和載入 (ETL) 任務的範本 Python 程式碼，會示範如何取得工作流程回合屬性。

```
import sys
import boto3
from awsglue.transforms import *
```

```
from awsglue.utils import getResolvedOptions
from awsglue.context import GlueContext
from pyspark.context import SparkContext

glue_client = boto3.client("glue")
args = getResolvedOptions(sys.argv, ['JOB_NAME','WORKFLOW_NAME', 'WORKFLOW_RUN_ID'])
workflow_name = args['WORKFLOW_NAME']
workflow_run_id = args['WORKFLOW_RUN_ID']
workflow_params = glue_client.get_workflow_run_properties(Name=workflow_name,
                                          RunId=workflow_run_id)["RunProperties"]

target_database = workflow_params['target_database']
target_s3_location = workflow_params['target_s3_location']
```

將 `target_format` 回合屬性設為 `'csv'` 即可繼續以下程式碼。

```
workflow_params['target_format'] = 'csv'
glue_client.put_workflow_run_properties(Name=workflow_name, RunId=workflow_run_id,
 RunProperties=workflow_params)
```

如需詳細資訊，請參閱下列內容：

- GetWorkflowRunProperties 行動（Python：獲取工作流程 _ 運行屬性）
- PutWorkflowRunProperties 行動（Python：把工作流程 _ 運行屬性）

使用 AWS Glue API 查詢工作流程

AWS Glue 提供豐富的 API 來管理工作流程。您可以使用 AWS Glue API 擷取工作流程的靜態檢視或執行中工作流程的動態檢視。如需更多詳細資訊，請參閱 工作流程。

主題

- 查詢靜態檢視
- 查詢動態檢視

查詢靜態檢視

使用 `GetWorkflow` API 操作來取得靜態檢視，指出工作流程的設計。這個操作會傳回一個由節點和邊緣組成的導向圖，其中的節點代表觸發、任務或爬蟲程式。邊緣定義節點之間的關係。在 AWS Glue 主控台的圖形中，它們會以連接器 (箭頭) 表示。

您也可以使用此操作配合熱門的圖形處理程式庫，例如 NetworkX、igraph、JGraphT 和 Java Universal Network/Graph (JUNG) Framework。因為所有這些程式庫都呈現類以的圖形，所以只需要基本的轉換。

根據與工作流程相關聯的觸發最新定義，此 API 傳回的靜態檢視是最新的檢視。

圖形定義

工作流程圖形 G 是排序對 (N, E)，其中 N 是一組節點而 E是 一組邊緣。「節點」是由唯一數字識別的圖形頂點。節點類型可以是觸發、任務或爬蟲程式。例如：`{name:T1, type:Trigger, uniqueId:1}, {name:J1, type:Job, uniqueId:2}`。

「邊緣」是 2 元組形式 (`src, dest`)，其中 `src` 和 `dest` 是節點，且有從 `src` 到 `dest` 的導向邊緣。

查詢靜態檢視的範例

考慮條件式觸發 T，它會在任務 J1 完成時觸發任務 J2。

```
J1 ---> T ---> J2
```

節點：J1、T、J2

邊緣：(J1, T)、(T, J2)

查詢動態檢視

使用 `GetWorkflowRun` API 操作來取得執行中工作流程的動態檢視。這個操作會傳回相同的圖形靜態檢視和有關工作流程回合的中繼資料。

針對執行，`GetWorkflowRun` 呼叫中代表任務的節點，有當成最新工作流程回合一部分啟動的任務回合清單。您可以使用此清單在圖形本身中顯示每項任務的回合狀態。針對尚未執行的下游相依性，此欄位設定為 `null`。圖形化的資訊可讓您了解任何工作流程目前在任何時間點的狀態。

此 API 傳回的動態檢視會隨工作流程回合啟動時出現的靜態檢視而變。

執行時間節點範例：`{name:T1, type: Trigger, uniqueId:1}`、`{name:J1, type:Job, uniqueId:2, jobDetails:{jobRuns}}`、`{name:C1, type:Crawler, uniqueId:3, crawlerDetails:{crawls}}`

範例 1：動態檢視

以下範例示範簡單的雙觸發工作流程。

- **節點**：t1、j1、t2、j2
- **邊緣**：(t1, j1)、(j1, t2)、(t2, j2)

GetWorkflow 回應包含下列項目。

```
{
    Nodes : [
        {
            "type" : Trigger,
            "name" : "t1",
            "uniqueId" : 1
        },
        {
            "type" : Job,
            "name" : "j1",
            "uniqueId" : 2
        },
        {
            "type" : Trigger,
            "name" : "t2",
            "uniqueId" : 3
        },
        {
            "type" : Job,
            "name" : "j2",
            "uniqueId" : 4
        }
    ],
    Edges : [
        {
            "sourceId" : 1,
            "destinationId" : 2
        },
        {
            "sourceId" : 2,
            "destinationId" : 3
        },
        {
            "sourceId" : 3,
            "destinationId" : 4
        }
}
```

GetWorkflowRun 回應包含下列項目。

```
{
    Nodes : [
        {
            "type" : Trigger,
            "name" : "t1",
            "uniqueId" : 1,
            "jobDetails" : null,
            "crawlerDetails" : null
        },
        {
            "type" : Job,
            "name" : "j1",
            "uniqueId" : 2,
            "jobDetails" : [
                {
                    "id" : "jr_12334",
                    "jobRunState" : "SUCCEEDED",
                    "errorMessage" : "error string"
                }
            ],
            "crawlerDetails" : null
        },
        {
            "type" : Trigger,
            "name" : "t2",
            "uniqueId" : 3,
            "jobDetails" : null,
            "crawlerDetails" : null
        },
        {
            "type" : Job,
            "name" : "j2",
            "uniqueId" : 4,
            "jobDetails" : [
                {
                    "id" : "jr_1233sdf4",
                    "jobRunState" : "SUCCEEDED",
                    "errorMessage" : "error string"
                }
            ],
            "crawlerDetails" : null
        }
```

```
    ],
    Edges : [
        {
            "sourceId" : 1,
            "destinationId" : 2
        },
        {
            "sourceId" : 2,
            "destinationId" : 3
        },
        {
            "sourceId" : 3,
            "destinationId" : 4
        }
}
```

範例 2：具有條件式觸發的多項任務

以下範例顯示具有多項任務和條件式觸發 (t3) 的工作流程。

```
Consider Flow:
T(t1) ---> J(j1) ---> T(t2) ---> J(j2)
             |                    |
             |                    |
             >+------> T(t3) <-----+
                        |
                        |
                     J(j3)

Graph generated:
Nodes: t1, t2, t3, j1, j2, j3
Edges: (t1, j1), (j1, t2), (t2, j2), (j1, t3), (j2, t3), (t3, j3)
```

AWS Glue 中的藍圖和工作流程限制

以下是藍圖和工作流程的限制。

藍圖限制

請牢記下列藍圖限制：

- 藍圖必須在 Amazon S3 儲存貯體所在的相同 AWS 區域中註冊。

- 若要跨 AWS 帳戶分享藍圖，您必須授與 Amazon S3 中藍圖 ZIP 封存的讀取許可。對藍圖 ZIP 封存具有讀取許可的客戶，可以在其 AWS帳戶中註冊並使用它。
- 藍圖參數集存放為單一 JSON 物件。此物件的長度上限為 128 KB。
- 藍圖 ZIP 封存的未壓縮大小上限為 5 MB。壓縮大小上限為 1 MB。
- 將工作流程中的任務、爬蟲程式和觸發程序總數限制在 100 或更少。如果包含超過 100 個，則嘗試繼續或停止工作流程執行時可能會出現錯誤。

工作流程限制

請牢記下列工作流程限制：其中一些註解較適用於手動建立工作流程的使用者。

- Amazon EventBridge 事件觸發的批次大小上限為 100。時段大小上限為 900 秒 (15 分鐘)。
- 一個觸發只能與一個工作流程相關聯。
- 只允許一個啟動觸發 (隨需或排程)。
- 如果工作流程中的任務或爬蟲程式是由工作流程外的觸發程式啟動，則工作流程內相依於任務或爬蟲程式完成 (成功或其他方式) 的任何觸發程式都不會觸發。
- 同樣地，如果工作流程中的任務或爬蟲程式，具有的觸發程式相依於工作流程內外的任務或爬蟲程式完成 (成功或其他方式)，如果任務或爬蟲程式是從工作流程內部啟動，則只有在任務或爬蟲程式完成時才會觸發工作流程中的觸發程式。

對 AWS Glue 中的藍圖錯誤進行故障診斷

如果在使用 AWS Glue 藍圖時遇到錯誤，請使用下列解決方案，以協助您找到問題來源並修復問題。

主題

錯誤：缺少 PySpark 模組

AWS Glue 傳回錯誤「Unknown error executing layout generator function ModuleNotFoundError: No module named 'pyspark'」。

當您解壓縮藍圖封存時，它可能如下所示：

```
$ unzip compaction.zip
Archive:  compaction.zip
   creating: compaction/
  inflating: compaction/blueprint.cfg
  inflating: compaction/layout.py
  inflating: compaction/README.md
  inflating: compaction/compaction.py

$ unzip compaction.zip
Archive:  compaction.zip
  inflating: blueprint.cfg
  inflating: compaction.py
  inflating: layout.py
  inflating: README.md
```

在第一種情況下，與藍圖相關的所有檔案都放置在名為 compaction 的資料夾下，然後轉換成名為 compaction.zip。

在第二種情況下，藍圖所需的所有檔案都未包含在資料夾中，並且做為根檔案新增到 zip 檔案 compaction.zip 中。

允許使用上述任一格式建立檔案。但是，請確認 `blueprint.cfg` 具有產生配置的指令碼中函數名稱的正確路徑。

範例

在案例 1：`blueprint.cfg` 應該具備 `layoutGenerator`，如下所示：

```
layoutGenerator": "compaction.layout.generate_layout"
```

在案例 2：`blueprint.cfg` 應該具備 `layoutGenerator`，如下所示

```
layoutGenerator": "layout.generate_layout"
```

如果未正確包含此路徑，您可能會看到指示的錯誤。例如，如果您具備案例 2 中所述的資料夾結構，而您的 `layoutGenerator` 表示為案例 1，則會看到上述錯誤。

錯誤：缺少藍圖組態檔

AWS Glue 傳回錯誤「Unknown error executing layout generator function FileNotFoundError: [Errno 2] No such file or directory: '/tmp/compaction/blueprint.cfg'」。

blueprint.cfg 應該放置在 ZIP 封存的根層級，或是位於與 ZIP 封存檔具有相同名稱的資料夾中。

當我們提取藍圖 ZIP 封存時，blueprint.cfg 預計可在以下路徑之一找到。如果在下列其中一個路徑中找不到它，您可以看到上述錯誤。

```
$ unzip compaction.zip
Archive:  compaction.zip
   creating: compaction/
  inflating: compaction/blueprint.cfg

$ unzip compaction.zip
Archive:  compaction.zip
  inflating: blueprint.cfg
```

錯誤：缺少匯入的檔案

AWS Glue 傳回錯誤「Unknown error executing layout generator function FileNotFoundError: [Errno 2] No such file or directory:* *'demo-project/foo.py'」。

如果您的配置產生指令碼具有讀取其他檔案的功能，請確保您為要導入的檔案提供了一個完整的路徑。例如，Conversion.py 指令碼可能會在 Layout.py 中參考。如需詳細資訊，請參閱範例藍圖專案。

錯誤：未獲授權，不得在資源上執行 iamPassRole

AWS Glue 傳回錯誤「User: arn:aws:sts::123456789012:assumed-role/AWSGlueServiceRole/GlueSession is not authorized to perform: iam:PassRole on resource: arn:aws:iam::123456789012:role/AWSGlueServiceRole」

如果工作流程中的任務和爬蟲程式具有與從藍圖建立工作流程時傳遞的角色相同的角色，則藍圖角色需要包含 `iam:PassRole` 許可。

如果工作流程中的任務和爬蟲程式扮演了從藍圖建立工作流程實體的角色以外的角色，則藍圖角色需要包含藍圖角色以外其他角色的 `iam:PassRole` 許可。

如需詳細資訊，請參閱藍圖角色的許可。

錯誤：無效的 cron 排程

AWS Glue 傳回錯誤「The schedule cron(0 0 * * * *) is invalid.」

請提供有效的 Cron 表達式。如需詳細資訊，請參閱任務和爬蟲程式的時間排程。

錯誤：具有相同名稱的觸發已經存在

AWS Glue 傳回錯誤「Trigger with name 'foo_starting_trigger' already submitted with different configuration」。

藍圖不需要您在配置指令碼中定義觸發以建立工作流程。藍圖程式庫會根據兩個動作之間定義的相依性來管理觸發建立。

觸發的命名如下：

- 對於工作流程中的啟動觸發，命名為 <workflow_name>_starting_trigger。
- 對於工作流程中依賴於一個或多個上游節點完成的節點 (任務/爬蟲程式)；AWS Glue 使用名稱 <workflow_name>_<node_name>_trigger 定義觸發

此錯誤表示具有相同名稱的觸發已經存在。您可以刪除現有的觸發，然後重新執行工作流程建立。

> **Note**
>
> 刪除工作流程並不會刪除工作流程中的節點。雖然工作流程已刪除，但觸發仍會留下。因此，您可能不會收到「工作流程已經存在」錯誤，但在建立工作流程、刪除工作流程，然後嘗試從相同藍圖重新建立相同名稱的情況下，您可能會收到「觸發已經存在」錯誤。

錯誤：名稱為 foo 的工作流程已存在。

工作流程名稱應該是唯一的。請嘗試使用其他名稱。

錯誤：在指定的 layoutGenerator 路徑中找不到模組

AWS Glue 傳回錯誤「Unknown error executing layout generator function ModuleNotFoundError: No module named 'crawl_s3_locations」。

```
layoutGenerator": "crawl_s3_locations.layout.generate_layout"
```

例如，如果您有上述的 layoutGenerator 路徑，那麼當您解壓縮藍圖封存時，它需要如下所示：

```
$ unzip crawl_s3_locations.zip
Archive:  crawl_s3_locations.zip
   creating: crawl_s3_locations/
  inflating: crawl_s3_locations/blueprint.cfg
  inflating: crawl_s3_locations/layout.py
  inflating: crawl_s3_locations/README.md
```

當您解壓縮歸檔時，如果藍圖歸檔如下所示，那麼您可能會得到上述錯誤。

```
$ unzip crawl_s3_locations.zip
Archive:  crawl_s3_locations.zip
  inflating: blueprint.cfg
  inflating: layout.py
  inflating: README.md
```

您可以看到沒有名為 `crawl_s3_locations` 的資料夾，而當 `layoutGenerator` 路徑透過模組 `crawl_s3_locations` 引用配置檔案，您就會得到上述錯誤。

錯誤：連線欄位中的驗證錯誤

AWS Glue 傳回錯誤「Unknown error executing layout generator function TypeError: Value ['foo'] for key Connections should be of type <class 'dict'>!」

這是驗證錯誤。`Job` 類別的 `Connections` 欄位應是字典，但卻提供了導致錯誤的值清單。

```
User input was list of values
Connections= ['string']

Should be a dict like the following
Connections*=*{'Connections': ['string']}
```

若要避免在從藍圖建立工作流程時發生這些執行時間錯誤，您可以驗證工作流程、任務和爬蟲程式定義，如測試藍圖所述。

請參閱 AWS Glue 藍圖類別參考中的語法，以在配置指令碼中定義 AWS Glue 任務、爬蟲程式和工作流程。

AWS Glue 藍圖的人物和角色許可

以下是 AWS Glue 藍圖人物和角色的典型人物和建議的 AWS Identity and Access Management (IAM) 角色。

主題

藍圖人物

以下是通常參與 AWS Glue 藍圖生命週期的人物。

人物	描述
AWS Glue 開發人員	開發、測試和發佈藍圖。
AWS Glue 管理員	註冊、維護和授與藍圖的許可。
資料分析	執行藍圖以建立工作流程。

如需更多詳細資訊，請參閱 the section called “藍圖概觀”。

藍圖人物的許可

以下是每個藍圖人物的建議許可。

藍圖的 AWS Glue 開發人員許可

AWS Glue 開發人員必須對用於發佈藍圖的 Amazon S3 儲存貯體具有寫入許可。通常，開發人員會在上傳藍圖後註冊藍圖。在這種情況下，開發人員需要 the section called “藍圖的 AWS Glue 管理員許可” 中列出的許可。此外，如果開發人員想要在註冊藍圖後測試藍圖，他或她也需要 the section called “藍圖的資料分析師許可” 中列出的許可。

藍圖的 AWS Glue 管理員許可

下列政策會授予註冊、檢視及維護 AWS Glue 藍圖的許可。

> **Important**
>
> 在下列政策中，將 *<s3-bucket-name>* 和 *<prefix>* 替換為 Amazon S3 路徑，以便上傳藍圖 ZIP 封存進行註冊。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateBlueprint",
                "glue:UpdateBlueprint",
                "glue:DeleteBlueprint",
                "glue:GetBlueprint",
                "glue:ListBlueprints",
                "glue:BatchGetBlueprints"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::<s3-bucket-name>/<prefix>/*"
        }
    ]
}
```

藍圖的資料分析師許可

下列政策會授與執行藍圖和檢視產生的工作流程和工作流程元件的許可。它還授予 AWS Glue 擔任角色的 PassRole，以便建立工作流程和工作流程元件。

政策會授與任何資源的許可。如果您想要設定個別藍圖的細微存取，請針對藍圖 ARN 使用下列格式：

```
arn:aws:glue:<region>:<account-id>:blueprint/<blueprint-name>
```

> **Important**
>
> 在下列政策中，將 *<account-id>* 替換為有效的 AWS 帳戶以及將 *<role-name>* 替換為用於執行藍圖的角色名稱。請參閱the section called “藍圖角色的許可”以取得此角色所需的許可。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListBlueprints",
                "glue:GetBlueprint",
                "glue:StartBlueprintRun",
                "glue:GetBlueprintRun",
                "glue:GetBlueprintRuns",
                "glue:GetCrawler",
                "glue:ListTriggers",
                "glue:ListJobs",
                "glue:BatchGetCrawlers",
                "glue:GetTrigger",
                "glue:BatchGetWorkflows",
                "glue:BatchGetTriggers",
                "glue:BatchGetJobs",
                "glue:BatchGetBlueprints",
                "glue:GetWorkflowRun",
                "glue:GetWorkflowRuns",
                "glue:ListCrawlers",
                "glue:ListWorkflows",
                "glue:GetJob",
                "glue:GetWorkflow",
                "glue:StartWorkflowRun"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::<account-id>:role/<role-name>"
        }
    ]
}
```

藍圖角色的許可

以下是用於從藍圖建立工作流程之 IAM 角色的建議許可。該角色必須與 glue.amazonaws.com 有信任關係。

> **Important**
>
> 在下列政策中，將 *<account-id>* 替換為有效的 AWS 帳戶，以及將 *<role-name>* 替換為角色的名稱。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateJob",
                "glue:GetCrawler",
                "glue:GetTrigger",
                "glue:DeleteCrawler",
                "glue:CreateTrigger",
                "glue:DeleteTrigger",
                "glue:DeleteJob",
                "glue:CreateWorkflow",
                "glue:DeleteWorkflow",
                "glue:GetJob",
                "glue:GetWorkflow",
                "glue:CreateCrawler"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::<account-id>:role/<role-name>"
        }
    ]
}
```

> **Note**
>
> 如果工作流程中的任務和爬蟲程式擔任此角色以外的角色，則此政策必須包含其他角色上的 `iam:PassRole` 許可，而不是藍圖角色上的許可。

AWS Glue 中的開發藍圖

您的組織可能有一組類似的 ETL 使用案例，這些案例可能會受益於能夠參數化單一工作流程來處理所有工作流程。為了解決此需求，AWS Glue 可讓您定義藍圖，可讓您用來產生工作流程。藍圖接受參數，因此資料分析師可以從單一藍圖建立不同的工作流程來處理類似的 ETL 使用案例。在您建立藍圖後，您可以將藍圖重複使用於不同部門、團隊和專案。

主題

AWS Glue 中的藍圖概觀

> **Note**
>
> AWS Glue 主控台的下列區域目前無法使用藍圖功能：亞太區域 (雅加達) 和中東 (阿拉伯聯合大公國)。

AWS Glue 藍圖提供建立和共用 AWS Glue 工作流程的方法。當存在可用於類似使用案例的複雜 ETL 程序時，您可以建立單一藍圖，而不是為每個使用案例各建立一個 AWS Glue 工作流程。

藍圖會指定工作流程中要包含的任務和爬蟲程式，並指定工作流程使用者在執行藍圖以建立工作流程時提供的參數。使用參數可讓單一藍圖產生各種類似使用案例的工作流程。如需工作流程的相關詳細資訊，請參閱 AWS Glue 中的工作流程概觀。

以下是藍圖的使用案例範例：

- 您想要分割現有的資料集。藍圖的輸入參數是 Amazon Simple Storage Service (Amazon S3) 來源和目標路徑，以及分割區欄的清單。

- 您想要將 Amazon DynamoDB 資料表快照到像 Amazon Redshift 這樣的 SQL 資料存放區。藍圖的輸入參數是 DynamoDB 資料表名稱和 AWS Glue 連線，用於指定 Amazon Redshift 叢集和目標資料庫。
- 您想要將多個 Amazon S3 路徑中的 CSV 資料轉換為 Parquet。您想要 AWS Glue 工作流程，以針對每個路徑包含個別的爬蟲程式和任務。輸入參數是 AWS Glue Data Catalog 中的目的地資料庫，以及 Amazon S3 路徑的逗號分隔清單。請注意，在此情況下，工作流程建立的爬蟲程式和任務數目會變動。

藍圖元件

藍圖是含有下列元件的 ZIP 封存：

- Python 配置產生器指令碼

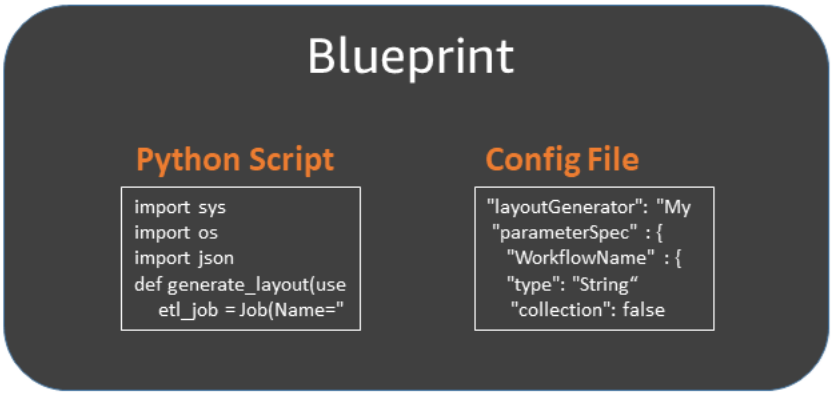
  包含指定工作流程配置的函數—為工作流程建立的爬蟲程式和任務、任務和爬蟲程式屬性，以及任務和爬蟲程式之間的相依性。此函數接受藍圖參數，並傳回 AWS Glue 用於產生工作流程的工作流程結構 (JSON 物件)。因為您使用 Python 指令碼來產生工作流程，所以您可以新增適合您使用案例的自己邏輯。

- 組態檔案

  指定產生工作流程配置的 Python 函數完整名稱。同時指定指令碼所使用之所有藍圖參數的名稱、資料類型和其他屬性。

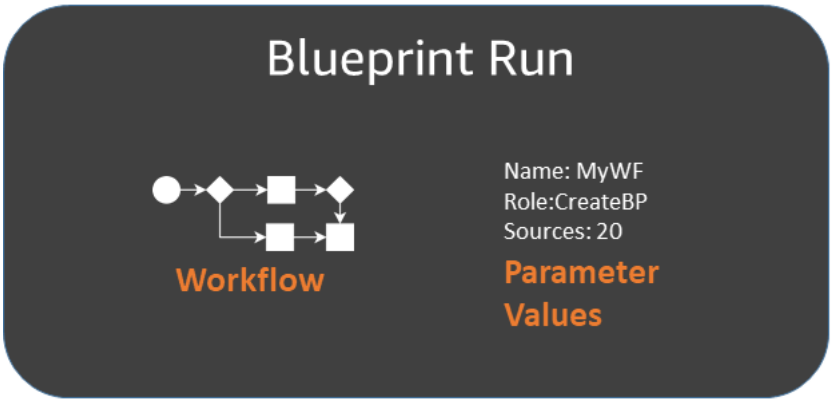
- (選用) ETL 指令碼和支援檔案

  做為進階使用案例，您可以參數化任務所使用之 ETL 指令碼的位置。您可以在 ZIP 封存中包含任務指令碼檔案，並為要將指令碼複製到的 Amazon S3 位置指定藍圖參數。配置產生器指令碼可以將 ETL 指令碼複製到指定的位置，並將該位置指定為任務指令碼位置屬性。您也可以包含任何程式庫或其他支援檔案，前提是您的指令碼處理它們。



藍圖執行

當您從藍圖建立工作流程時，AWS Glue 會執行藍圖，此藍圖會啟動非同步處理程序以建立工作流程，以及工作流程封裝的任務、爬蟲程式和觸發。AWS Glue 會使用藍圖執行來協調工作流程及其元件的建立。您可以透過檢視藍圖執行狀態來檢視建立程序的狀態。藍圖執行也會儲存您為藍圖參數提供的值。



您可以使用 AWS Glue 主控台或 AWS Command Line Interface (AWS CLI) 來檢視藍圖。檢視或疑難排解工作流程時，您一律可以傳回藍圖執行以檢視用於建立工作流程的藍圖參數值。

藍圖的生命週期

藍圖使用 AWS Glue 來開發、測試、註冊，然後執行以建立工作流程。藍圖生命週期通常涉及三個人物。

人物	任務
AWS Glue 開發人員	• 寫入工作流程配置指令碼並建立組態檔案。 • 使用 AWS Glue 服務提供的程式庫在本機測試藍圖。 • 建立指令碼、組態檔和支援檔案的 ZIP 封存，並將存檔發佈到 Amazon S3 中的某個位置。

人物	任務
	• 將儲存貯體政策新增至 Amazon S3 儲存貯體，該政策授予儲存貯體物件的讀取許可給 AWS Glue 管理員的 AWS 帳戶。 • 將 Amazon S3 中 ZIP 封存的 IAM 讀取許可授予 AWS Glue 管理員。
AWS Glue 管理員	• 向 AWS Glue 註冊藍圖。AWS Glue 會將 ZIP 封存的複本複製到保留的 Amazon S3 位置。 • 將藍圖上的 IAM 許可授與資料分析師。
資料分析	• 執行藍圖以建立工作流程，並提供藍圖參數值。檢查藍圖執行狀態，以確保已順利產生工作流程和工作流程元件。 • 執行並疑難排解工作流程。在執行工作流程之前，可以檢視 AWS Glue 主控台中的工作流程設計圖來驗證工作流程。

另請參閱

- AWS Glue 中的開發藍圖
- 在 AWS Glue 中從藍圖建立工作流程
- AWS Glue 藍圖的人物和角色許可

AWS Glue 中的開發藍圖

作為 AWS Glue 開發人員，您可以建立和發佈藍圖，供資料分析師用來產生工作流程。

主題

- 開發藍圖概觀
- 開發藍圖的先決條件
- 撰寫藍圖程式碼
- 範例藍圖專案

- 測試藍圖
- 發佈藍圖
- AWS Glue 藍圖類別參考
- 藍圖範例

另請參閱

- AWS Glue 中的藍圖概觀

開發藍圖概觀

開發程序的第一個步驟是找出可從藍圖中受益的常見使用案例。一個典型的使用案例涉及重複出現的 ETL 問題，您認為應該以一般方式解決。接下來，設計實作一般化使用案例的藍圖，並定義藍圖輸入參數，這些參數搭配使用可以從一般化使用案例定義特定的使用案例。

藍圖包含內含藍圖參數組態檔的專案，以及定義工作流程要產生之配置的指令碼。配置定義要建立的任務和爬蟲程式 (在藍圖指令碼術語中稱為實體)。

請勿直接在配置指令碼中指定任何觸發程序。而是改為撰寫程式碼來指定指令碼建立之任務與爬蟲程式之間的相依性。AWS Glue 會根據您的相依性規範來產生觸發程序。配置指令碼的輸出是工作流程物件，其中包含所有工作流程實體的規格。

您可以使用以下 AWS Glue 藍圖程式庫來建置工作流程物件：

- `awsglue.blueprint.base_resource` – 程式庫所使用的基本資源程式庫。
- `awsglue.blueprint.workflow` – 用於定義 `Workflow` 類別的程式庫。
- `awsglue.blueprint.job` – 用於定義 `Job` 類別的程式庫。
- `awsglue.blueprint.crawler` – 用於定義 `Crawler` 類別的程式庫。

唯一支援配置產生的其他程式庫是可用於 Python Shell 的程式庫。

發佈藍圖之前，您可以使用藍圖程式庫中定義的方法在本機測試藍圖。

當您準備好將藍圖提供給資料分析師使用時，您可以將指令碼、參數組態檔以及任何支援的檔案 (例如其他指令碼和程式庫) 封裝成單一可部署的資產。然後，您將資產上傳到 Amazon S3，並要求管理員向 AWS Glue 註冊。

如需藍圖範例專案的相關資訊，請參閱範例藍圖專案和藍圖範例。

開發藍圖的先決條件

若想開發藍圖，您應先熟悉如何使用 AWS Glue 以及為 Apache Spark ETL 任務或 Python Shell 任務編寫指令碼。此外，您也必須完成下列設定任務。

- 下載四個 AWS Python 程式庫以用在藍圖配置指令碼中。
- 設定 AWS 開發套件。
- 設定 AWS CLI。

下載 Python 程式庫

從 GitHub 下載以下程式庫，並將它們安裝到您的專案中：

- https://github.com/awslabs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/base_resource.py
- https://github.com/awslabs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/workflow.py
- https://github.com/awslabs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/crawler.py
- https://github.com/awslabs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/job.py

設定 AWS Java SDK

對於 AWS Java 開發套件，您必須新增 `jar` 檔案，其中包含藍圖的 API。

1. 如果您尚未這麼做，請先設定適用於 Java 的 AWS 開發套件。
   - 對於 Java 1.x，請按照AWS SDK for Java 開發人員指南中設定 AWS SDK for Java的指示進行。
   - 對於 Java 2.x，請按照AWS SDK for Java 2.x 開發人員指南中設定 AWS SDK for Java 2.x的指示進行。
2. 下載用戶端 `jar` 檔案，該檔案可以存取藍圖的 API。
   - 對於 Java 1.x：s3://awsglue-custom-blueprints-preview-artifacts/awsglue-java-sdk-preview/AWSGlueJavaClient-1.11.x.jar
   - 對於 Java 2.x：s3://awsglue-custom-blueprints-preview-artifacts/awsglue-java-sdk-v2-preview/AwsJavaSdk-Glue-2.0.jar
3. 新增用戶端 `jar` 到 Java classpath 的前面以覆寫 AWS Java 開發套件提供的 AWS Glue 用戶端。

```
export CLASSPATH=<path-to-preview-client-jar>:$CLASSPATH
```

4. (選用) 使用下列 Java 應用程式測試開發套件。應用程式應輸出空的清單。

   使用您的憑證取代 accessKey 和 secretKey，並用您的區域取代 us-east-1。

```
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.glue.AWSGlue;
import com.amazonaws.services.glue.AWSGlueClientBuilder;
import com.amazonaws.services.glue.model.ListBlueprintsRequest;

public class App{
    public static void main(String[] args) {
        AWSCredentials credentials = new BasicAWSCredentials("accessKey",
 "secretKey");
        AWSCredentialsProvider provider = new
 AWSStaticCredentialsProvider(credentials);
        AWSGlue glue = AWSGlueClientBuilder.standard().withCredentials(provider)
                .withRegion("us-east-1").build();
        ListBlueprintsRequest request = new
 ListBlueprintsRequest().withMaxResults(2);
        System.out.println(glue.listBlueprints(request));
    }
}
```

設定 AWS Python SDK

下列步驟假設您的電腦已安裝 Python 2.7 版或更新版本，或 3.6 版或更新版本。

1. 下載以下 boto3 wheel 檔案。如果提示開啟或儲存，請儲存檔案。s3://awsglue-custom-blueprints-preview-artifacts/aws-python-sdk-preview/boto3-1.17.31-py2.py3-none-any.whl
2. 下載以下 botocore wheel 檔案：s3://awsglue-custom-blueprints-preview-artifacts/aws-python-sdk-preview/botocore-1.20.31-py2.py3-none-any.whl
3. 檢查 Python 版本。

```
python --version
```

4. 根據您的 Python 版本，輸入下列指令 (適用於 Linux)：

   - 適用於 Python 2.7 或更新版本。

     ```
     python3 -m pip install --user virtualenv
     source env/bin/activate
     ```

   - 適用於 Python 3.6 或更新版本。

     ```
     python3 -m venv python-sdk-test
     source python-sdk-test/bin/activate
     ```

5. 安裝 botocore wheel 檔案。

   ```
   python3 -m pip install <download-directory>/botocore-1.20.31-py2.py3-none-any.whl
   ```

6. 安裝 boto3 wheel 檔案。

   ```
   python3 -m pip install <download-directory>/boto3-1.17.31-py2.py3-none-any.whl
   ```

7. 在 `~/.aws/credentials` 和 `~/.aws/config` 檔案中設定您的憑證和預設區域。如需詳細資訊，請參閱 AWS Command Line Interface 使用者指南中的設定 AWS CLI。

8. (選用) 測試您的設定。以下命令應會傳回空的清單。

   將 `us-east-1` 取代為您的區域。

   ```
   $ python
   >>> import boto3
   >>> glue = boto3.client('glue', 'us-east-1')
   >>> glue.list_blueprints()
   ```

設定預覽版 AWS CLI

1. 若您尚未這樣做，請先在電腦上安裝和/或更新 AWS Command Line Interface (AWS CLI)。執行此動作最簡單的方法是使用 `pip` (Python 安裝程式公用程式)：

   ```
   pip install awscli --upgrade --user
   ```

   您可以在這裡找到 AWS CLI 的完整安裝指示：安裝 AWS Command Line Interface。

2. 從以下位置下載 AWS CLI wheel 檔案：s3://awsglue-custom-blueprints-preview-artifacts/awscli-preview-build/awscli-1.19.31-py2.py3-none-any.whl
3. 安裝 AWS CLI wheel 檔案。

```
python3 -m pip install awscli-1.19.31-py2.py3-none-any.whl
```

4. 執行 `aws configure` 命令。設定 AWS 憑證 (包括存取金鑰和私密金鑰) 和 AWS 區域。您可以在這裡找到關於設定 AWS CLI 的資訊：設定 AWS CLI。
5. 測試 AWS CLI。以下命令應會傳回空的清單。

   將 `us-east-1` 取代為您的區域。

```
aws glue list-blueprints --region us-east-1
```

撰寫藍圖程式碼

您建立的每個藍圖專案至少必須包含下列檔案：

- 定義工作流程的 Python 配置指令碼。指令碼包含一個函數，用於定義工作流程中的實體 (任務和爬蟲程式)，以及它們之間的相依性。
- 組態檔案 `blueprint.cfg`，它定義了：
  - 工作流程配置定義函數的完整路徑。
  - 藍圖接受的參數。

主題

- 建立藍圖配置指令碼
- 建立組態檔案
- 指定藍圖參數

建立藍圖配置指令碼

藍圖配置指令碼必須包含在工作流程中產生實體的函數。您可以根據喜好命名此功能。AWS Glue 會使用組態檔來判斷函數的完整名稱。

您的配置函數會執行下列操作：

- (選用) 執行個體化 Job 類別以建立 Job 物件，並傳遞 Command 和 Role 等引數。這些是您使用 AWS Glue 主控台或 API 建立任務時需要指定的任務屬性。
- (選用) 執行個體化 Crawler 類別以建立 Crawler 物件，並傳遞名稱、角色和目標引數。
- 若要指出物件 (工作流程實體) 之間的相依性，請傳遞 DependsOn 和 WaitForDependencies 附加引數至 Job() 和 Crawler()。本節稍後會說明這些引數。
- 執行個體化 Workflow 類別來建立傳回至 AWS Glue 的工作流程物件，傳遞 Name 引數、Entities 引數，以及選用的 OnSchedule 引數。Entities 引數會指定工作流程中要包含的所有任務和爬蟲程式。若要了解如何建構 Entities 物件的詳細資訊，請參閱本節稍後提供的範例專案。
- 傳回 Workflow 物件。

如需 Job、Crawler 及 Workflow 類別的定義，請參閱AWS Glue 藍圖類別參考。

配置函數必須接受以下輸入引數。

引數	描述
user_params	藍圖參數名稱與值的 Python 字典。如需更多詳細資訊，請參閱 指定藍圖參數。
system_params	Python 字典包含兩個屬性：region 和 accountId 。

以下是名為 Layout.py 的檔案中的範例配置產生器指令碼：

```
import argparse
import sys
import os
import json
from awsglue.blueprint.workflow import *
from awsglue.blueprint.job import *
from awsglue.blueprint.crawler import *


def generate_layout(user_params, system_params):

    etl_job = Job(Name="{}_etl_job".format(user_params['WorkflowName']),
                  Command={
                      "Name": "glueetl",
```

```
                    "ScriptLocation": user_params['ScriptLocation'],
                    "PythonVersion": "2"
                },
                Role=user_params['PassRole'])
    post_process_job = Job(Name="{}_post_process".format(user_params['WorkflowName']),
                           Command={
                               "Name": "pythonshell",
                               "ScriptLocation": user_params['ScriptLocation'],
                               "PythonVersion": "2"
                           },
                           Role=user_params['PassRole'],
                           DependsOn={
                               etl_job: "SUCCEEDED"
                           },
                           WaitForDependencies="AND")
    sample_workflow = Workflow(Name=user_params['WorkflowName'],
                           Entities=Entities(Jobs=[etl_job, post_process_job]))
    return sample_workflow
```

範例指令碼會匯入所需的藍圖程式庫，並包含會產生具有兩個任務之工作流程的 `generate_layout` 函數。這是一個非常簡單的指令碼。較複雜的指令碼可以使用額外的邏輯和參數來產生包含許多任務和爬蟲程式 (甚至包含可變數量的任務和爬蟲程式) 的工作流程。

使用 DependsOn 引數

DependsOn 引數是此實體對工作流程中其他實體之相依性的字典表示。其具有如下格式。

```
DependsOn = {dependency1 : state, dependency2 : state, ...}
```

此字典中的索引鍵代表實體的物件參考，而不是名稱，而值是對應於要監視之狀態的字串。AWS Glue 會推斷適當的觸發程序。如需有效狀態，請參閱條件結構。

例如，任務可能取決於爬蟲程式是否成功完成。如果您定義名為 `crawler2` 的爬蟲程式物件，如下所示：

```
crawler2 = Crawler(Name="my_crawler", ...)
```

那麼依賴 `crawler2` 的物件將包含一個建構函數引數，例如：

```
DependsOn = {crawler2 : "SUCCEEDED"}
```

例如：

```
job1 = Job(Name="Job1", ..., DependsOn = {crawler2 : "SUCCEEDED", ...})
```

如果實體省略 DependsOn，則該實體取決於工作流程啟動觸發。

使用 WaitForDependencies 引數

WaitForDependencies 引數定義任務或爬蟲程式實體應該等到它依賴的所有實體都完成，還是等到任何實體完成。

允許的值是「AND」或「ANY」。

使用 OnSchedule 引數

Workflow 類別建構函數的 OnSchedule 引數是 cron 表達式，定義工作流程的啟動觸發定義。

如果指定此引數，AWS Glue 會建立具有對應排程的排程觸發程序。若未指定，工作流程的啟動觸發是隨需觸發。

建立組態檔案

藍圖組態檔是必要檔案，定義用於產生工作流程的指令碼進入點，以及藍圖接受的參數。這個檔案必須命名為 blueprint.cfg。

以下是範例組態檔。

```
{
    "layoutGenerator": "DemoBlueprintProject.Layout.generate_layout",
    "parameterSpec" : {
           "WorkflowName" : {
                "type": "String",
                "collection": false
           },
           "WorkerType" : {
                "type": "String",
                "collection": false,
                "allowedValues": ["G1.X", "G2.X"],
                "defaultValue": "G1.X"
           },
           "Dpu" : {
                "type" : "Integer",
                "allowedValues" : [2, 4, 6],
                "defaultValue" : 2
           },
```

```
            "DynamoDBTableName": {
                 "type": "String",
                 "collection" : false
            },
            "ScriptLocation" : {
                 "type": "String",
                 "collection": false
     }
    }
}
```

layoutGenerator 屬性會在產生配置的指令碼中指定函數的完整名稱。

parameterSpec 屬性指定此藍圖接受的參數。如需更多詳細資訊，請參閱 指定藍圖參數。

> **Important**
>
> 您的組態檔案必須包含工作流程名稱做為藍圖參數，或者您必須在配置指令碼中產生唯一的工作流程名稱。

指定藍圖參數

組態檔案在 parameterSpec JSON 物件中包含藍圖參數規格。parameterSpec 包含一或多個參數物件。

```
"parameterSpec": {
    "<parameter_name>": {
      "type": "<parameter-type>",
      "collection": true|false,
      "description": "<parameter-description>",
      "defaultValue": "<default value for the parameter if value not specified>"
      "allowedValues": "<list of allowed values>"
    },
    "<parameter_name>": {
       ...
    }
  }
```

以下是為每個參數物件編寫程式碼的規則：

- 參數名稱和 type 是必要的。所有其他屬性是選用的。

- 如果您指定 defaultValue 屬性，則參數是選用的。否則，參數是必要的，且從藍圖建立工作流程的資料分析師必須為其提供值。
- 如果您將 collection 屬性設定為 true，則參數可以採取值集合。集合可以是任何資料類型。
- 如果您指定 allowedValues，則 AWS Glue 主控台會顯示值的下拉式清單，供資料分析師從藍圖建立工作流程時選擇。

以下是 type 的允許值：

參數資料類型	備註
String	-
Integer	-
Double	-
Boolean	可能值為 true 和 false。在 AWS Glue 主控台的 Create a workflow from <blueprint> (從 <藍圖> 建立工作流程) 頁面上產生核取方塊。
S3Uri	完整的 Amazon S3 路徑，開頭為 s3://。在 Create a workflow from <blueprint> (從 <藍圖> 建立工作流程) 頁面上產生文字欄位和 Browse (瀏覽) 按鈕。
S3Bucket	僅限 Amazon S3 儲存貯體名稱。在 Create a workflow from <blueprint> (從 <藍圖> 建立工作流程) 頁面上產生儲存貯體選擇器。
IAMRoleArn	AWS Identity and Access Management (IAM) 角色的 Amazon Resource Name (ARN)。在 Create a workflow from <blueprint> (從 <藍圖> 建立工作流程) 頁面上產生角色選擇器。
IAMRoleName	IAM 角色的名稱。在 Create a workflow from <blueprint> (從 <藍圖> 建立工作流程) 頁面上產生角色選擇器。

範例藍圖專案

資料格式轉換是頻繁的擷取、轉換和載入 (ETL) 使用案例。在一般的分析工作負載中，以欄為基礎的檔案格式 (例如 Parquet 或 ORC) 優先於 CSV 或 JSON 等文字格式。此範例藍圖可讓您將資料從 CSV/JSON 等格式轉換為 Parquet，以供 Amazon S3 上的檔案使用。

此藍圖會取得藍圖參數定義的 S3 路徑清單，將資料轉換為 Prquet 格式，並將其寫入另一個藍圖參數指定的 S3 位置。配置指令碼會為每個路徑建立爬蟲程式和任務。配置指令碼還會將 Conversion.py 中的 ETL 指令碼上傳至另一個藍圖參數指定的 S3 儲存貯體。然後，配置指令碼會將上傳的指令碼指定為每個任務的 ETL 指令碼。專案的 ZIP 封存包含配置指令碼、ETL 指令碼和藍圖組態檔。

如需藍圖範例專案的相關資訊，請參閱藍圖範例。

以下是檔案 Layout.py 中的配置指令碼。

```
from awsglue.blueprint.workflow import *
from awsglue.blueprint.job import *
from awsglue.blueprint.crawler import *
import boto3

s3_client = boto3.client('s3')

# Ingesting all the S3 paths as Glue table in parquet format
def generate_layout(user_params, system_params):
    #Always give the full path for the file
    with open("ConversionBlueprint/Conversion.py", "rb") as f:
        s3_client.upload_fileobj(f, user_params['ScriptsBucket'], "Conversion.py")
    etlScriptLocation = "s3://{}/Conversion.py".format(user_params['ScriptsBucket'])

    crawlers = []
    jobs = []
    workflowName = user_params['WorkflowName']
    for path in user_params['S3Paths']:
      tablePrefix = "source_"
      crawler = Crawler(Name="{}_crawler".format(workflowName),
                        Role=user_params['PassRole'],
                        DatabaseName=user_params['TargetDatabase'],
                        TablePrefix=tablePrefix,
                        Targets= {"S3Targets": [{"Path": path}]})
      crawlers.append(crawler)
      transform_job = Job(Name="{}_transform_job".format(workflowName),
                          Command={"Name": "glueetl",
                                   "ScriptLocation": etlScriptLocation,
                                   "PythonVersion": "3"},
                          Role=user_params['PassRole'],
                          DefaultArguments={"--database_name":
 user_params['TargetDatabase'],
                                            "--table_prefix": tablePrefix,
                                            "--region_name": system_params['region'],
```

```
                                            "--output_path":
user_params['TargetS3Location']},
                        DependsOn={crawler: "SUCCEEDED"},
                        WaitForDependencies="AND")
      jobs.append(transform_job)
    conversion_workflow = Workflow(Name=workflowName, Entities=Entities(Jobs=jobs,
Crawlers=crawlers))
    return conversion_workflow
```

以下是對應的藍圖組態檔 blueprint.cfg。

```
{
    "layoutGenerator": "ConversionBlueprint.Layout.generate_layout",
    "parameterSpec" : {
        "WorkflowName" : {
            "type": "String",
            "collection": false,
            "description": "Name for the workflow."
        },
        "S3Paths" : {
            "type": "S3Uri",
            "collection": true,
            "description": "List of Amazon S3 paths for data ingestion."
        },
        "PassRole" : {
            "type": "IAMRoleName",
            "collection": false,
            "description": "Choose an IAM role to be used in running the job/crawler"
        },
        "TargetDatabase": {
            "type": "String",
            "collection" : false,
            "description": "Choose a database in the Data Catalog."
        },
        "TargetS3Location": {
            "type": "S3Uri",
            "collection" : false,
            "description": "Choose an Amazon S3 output path: ex:s3://<target_path>/."
        },
        "ScriptsBucket": {
            "type": "S3Bucket",
            "collection": false,
```

```
            "description": "Provide an S3 bucket name(in the same AWS Region) to store
 the scripts."
        }
    }
}
```

檔案 Conversion.py 中的以下指令碼是上傳的 ETL 指令碼。請注意，它在轉換過程中會保留分割結構。

```
import sys
from pyspark.sql.functions import *
from pyspark.context import SparkContext
from awsglue.transforms import *
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions
import boto3

args = getResolvedOptions(sys.argv, [
    'JOB_NAME',
    'region_name',
    'database_name',
    'table_prefix',
    'output_path'])
databaseName = args['database_name']
tablePrefix = args['table_prefix']
outputPath = args['output_path']

glue = boto3.client('glue', region_name=args['region_name'])

glue_context = GlueContext(SparkContext.getOrCreate())
spark = glue_context.spark_session
job = Job(glue_context)
job.init(args['JOB_NAME'], args)

def get_tables(database_name, table_prefix):
    tables = []
    paginator = glue.get_paginator('get_tables')
    for page in paginator.paginate(DatabaseName=database_name, Expression=table_prefix
+"*"):
        tables.extend(page['TableList'])
    return tables
```

```
for table in get_tables(databaseName, tablePrefix):
    tableName = table['Name']
    partitionList = table['PartitionKeys']
    partitionKeys = []
    for partition in partitionList:
        partitionKeys.append(partition['Name'])

    # Create DynamicFrame from Catalog
    dyf = glue_context.create_dynamic_frame.from_catalog(
        name_space=databaseName,
        table_name=tableName,
        additional_options={
            'useS3ListImplementation': True
        },
        transformation_ctx='dyf'
    )

    # Resolve choice type with make_struct
    dyf = ResolveChoice.apply(
        frame=dyf,
        choice='make_struct',
        transformation_ctx='resolvechoice_' + tableName
    )

    # Drop null fields
    dyf = DropNullFields.apply(
        frame=dyf,
        transformation_ctx="dropnullfields_" + tableName
    )

    # Write DynamicFrame to S3 in glueparquet
    sink = glue_context.getSink(
        connection_type="s3",
        path=outputPath,
        enableUpdateCatalog=True,
        partitionKeys=partitionKeys
    )
    sink.setFormat("glueparquet")

    sink.setCatalogInfo(
        catalogDatabase=databaseName,
        catalogTableName=tableName[len(tablePrefix):]
    )
    sink.writeFrame(dyf)
```

```
job.commit()
```

> **Note**
>
> 只能提供兩個 Amazon S3 路徑做為範例藍圖的輸入。這是因為 AWS Glue 觸發程序僅限於叫用兩個爬蟲程式動作。

測試藍圖

當您開發程式碼時，您應該執行本機測試，以確認工作流程配置是否正確。

本機測試不會產生 AWS Glue 任務、爬蟲程式或觸發程序。相反地，您可以在本機執行配置指令碼，並使用 `to_json()` 和 `validate()` 方法來列印物件並尋找錯誤。這些方法可用於程式庫中定義的全部三個類別。

有兩種方式可以處理 AWS Glue 傳遞給您配置函數的 `user_params` 和 `system_params` 引數。您的測試工作台程式碼可以建立範例藍圖參數值的字典，並將其作為 `user_params` 引數傳遞給配置函數。或者，您也可以移除對 `user_params` 的參考並用硬式編碼字串替換它們。

如果您的程式碼在 `system_params` 引數中使用 `region` 和 `accountId` 屬性，您可以傳入自己的 `system_params` 字典。

測試藍圖

1. 在具有程式庫的目錄中啟動 Python 解釋器，或將藍圖檔案和提供的程式庫載入到您偏好的整合開發環境 (IDE)。
2. 確保您的程式碼匯入提供的程式庫。
3. 將程式碼新增到配置函數以在任何實體或 `Workflow` 物件上呼叫 `validate()` 或 `to_json()`。例如，如果您的程式碼建立名為 `mycrawler` 的 `Crawler` 物件，您可以如下所示呼叫 `validate()`。

   ```
   mycrawler.validate()
   ```

   您可以如下所示列印 `mycrawler`：

   ```
   print(mycrawler.to_json())
   ```

如果您在物件上呼叫 to_json，則不需要同時呼叫 validate()，因為 to_json() 會呼叫 validate()。

在工作流程物件上呼叫這些方法非常有用。假設您的指令碼將工作流程物件命名為 my_workflow，請如下所示驗證和列印工作流程物件。

```
print(my_workflow.to_json())
```

如需 to_json() 和 validate() 的詳細資訊，請參閱「Class 方法」。

您也可以匯入 pprint，然後美化顯示工作流程物件，如本節稍後的範例所示。

4. 執行程式碼、修正錯誤，最後移除對 validate() 或 to_json() 的呼叫。

Example

下列範例顯示如何建構範例藍圖參數的字典，並將其作為 user_params 引數傳遞給配置函數 generate_compaction_workflow。它也會顯示如何美化顯示產生的工作流程物件。

```
from pprint import pprint
from awsglue.blueprint.workflow import *
from awsglue.blueprint.job import *
from awsglue.blueprint.crawler import *

USER_PARAMS = {"WorkflowName": "compaction_workflow",
               "ScriptLocation": "s3://awsexamplebucket1/scripts/threaded-
compaction.py",
               "PassRole": "arn:aws:iam::111122223333:role/GlueRole-ETL",
               "DatabaseName": "cloudtrial",
               "TableName": "ct_cloudtrail",
               "CoalesceFactor": 4,
               "MaxThreadWorkers": 200}


def generate_compaction_workflow(user_params: dict, system_params: dict) -> Workflow:
    compaction_job = Job(Name=f"{user_params['WorkflowName']}_etl_job",
                         Command={"Name": "glueetl",
                                  "ScriptLocation": user_params['ScriptLocation'],
                                  "PythonVersion": "3"},
                         Role="arn:aws:iam::111122223333:role/
AWSGlueServiceRoleDefault",
```

```
                        DefaultArguments={"DatabaseName": user_params['DatabaseName'],
                                          "TableName": user_params['TableName'],
                                          "CoalesceFactor":
 user_params['CoalesceFactor'],
                                          "max_thread_workers":
 user_params['MaxThreadWorkers']})

    catalog_target = {"CatalogTargets": [{"DatabaseName": user_params['DatabaseName'],
 "Tables": [user_params['TableName']]}]}

    compacted_files_crawler = Crawler(Name=f"{user_params['WorkflowName']}_post_crawl",
                                      Targets = catalog_target,
                                      Role=user_params['PassRole'],
                                      DependsOn={compaction_job: "SUCCEEDED"},
                                      WaitForDependencies="AND",
                                      SchemaChangePolicy={"DeleteBehavior": "LOG"})

    compaction_workflow = Workflow(Name=user_params['WorkflowName'],
                                   Entities=Entities(Jobs=[compaction_job],

 Crawlers=[compacted_files_crawler]))
    return compaction_workflow

generated = generate_compaction_workflow(user_params=USER_PARAMS, system_params={})
gen_dict = generated.to_json()

pprint(gen_dict)
```

發佈藍圖

開發藍圖後，您必須將它上傳至 Amazon S3。您必須擁有用於發佈藍圖的 Amazon S3 儲存貯體的寫入許可。您也必須確定負責註冊藍圖的 AWS Glue 管理員具有 Amazon S3 儲存貯體的讀取存取。如需 AWS Glue 藍圖之人物和角色的建議 AWS Identity and Access Management (IAM) 許可政策，請參閱 AWS Glue 藍圖的人物和角色許可。

發佈藍圖

1. 建立必要的指令碼、資源和藍圖組態檔。
2. 將所有檔案新增到 ZIP 封存，然後將 ZIP 檔案上傳到 Amazon S3。使用與使用者註冊和執行藍圖之區域相同區域的 S3 儲存貯體。

   您可以使用以下命令，從命令列建立 ZIP 檔案。

```
zip -r folder.zip folder
```

3. 新增儲存貯體政策，將讀取許可授與 AWS 所需的帳戶。以下是政策範例。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-blueprints/*"
        }
    ]
}
```

4. 授予 Amazon S3 儲存貯體上的 IAM `s3:GetObject` 許可給 AWS Glue 管理員或負責註冊藍圖的人員。如需授予管理員的範例政策，請參閱藍圖的 AWS Glue 管理員許可。

完成藍圖的本機測試後，您可能也想要在 AWS Glue 上測試藍圖。若要在 AWS Glue 上測試藍圖，必須對其進行註冊。您可以使用 IAM 授權或使用個別測試帳戶來限制誰可以看到註冊的藍圖。

另請參閱:

- 在 AWS Glue 中註冊藍圖

AWS Glue 藍圖類別參考

AWS Glue 藍圖的程式庫會定義您在工作流程配置指令碼中使用的三個類別：`Job`、`Crawler` 及 `Workflow`。

主題

- Job 類別
- Crawler 類別
- Workflow 類別

- Class 方法

Job 類別

Job 類別代表 AWS Glue ETL 任務。

必要建構函數引數

以下為 Job 類別的必要建構函數引數。

引數名稱	類型	描述
Name	str	要指派給任務的名稱。AWS Glue 會在名稱中新增隨機產生的尾碼，以區分任務與其他藍圖執行所建立的任務。
Role	str	任務在執行時應擔任角色的 Amazon Resource Name (ARN)。
Command	dict	Job 命令，如 API 文件中的 JobCommand 結構所指定。

選用建構函數引數

以下為 Job 類別的選用建構函數引數。

引數名稱	類型	描述
DependsOn	dict	任務所依賴的工作流程實體清單。如需更多詳細資訊，請參閱 使用 DependsOn 引數。
WaitForDependencies	str	指出任務應該等到它依賴的所有實體都完成，還是等到任何實體完成，然後再執行。如需更多詳細資訊，請參閱 使用 WaitForDependencies 引數。如果任務僅依賴於一個實體，則省略。
(Job 屬性)	-	AWS Glue API 文件 Job 結構 中列出的任何任務屬性 (CreatedOn 和 LastModifiedOn 除外)。

Crawler 類別

Crawler 類別代表 AWS Glue 爬蟲程式。

必要建構函數引數

以下為 Crawler 類別的必要建構函數引數。

引數名稱	類型	描述
Name	str	要指派給任務的名稱。AWS Glue 會在名稱中新增隨機產生的尾碼，以區分爬蟲程式與其他藍圖執行所建立的爬蟲程式。
Role	str	爬蟲程式執行時應擔任角色的 ARN。
Targets	dict	待編目的目標集合。Targets 類別建構函數引數定義於 API 文件中的 CrawlerTargets 結構。所有 Targets 建構函數引數都是選用的，但您必須至少傳遞一個。

選用建構函數引數

以下為 Crawler 類別的選用建構函數引數。

引數名稱	類型	描述
DependsOn	dict	爬蟲程式所依賴的工作流程實體清單。如需更多詳細資訊，請參閱 使用 DependsOn 引數。
WaitForDependencies	str	指出爬蟲程式應該等到它依賴的所有實體都完成，還是等到任何實體完成，然後再執行。如需更多詳細資訊，請參閱 使用 WaitForDependencies 引數。如果爬蟲程式只依賴一個實體，則省略。
(Crawler 屬性)	-	AWS Glue API 文件 Crawler 結構 中列出的任何爬蟲程式屬性，但以下項目除外：

引數名稱	類型	描述
		• State • CrawlElapsedTime • CreationTime • LastUpdated • LastCrawl • Version

Workflow 類別

Workflow 類別代表 AWS Glue 工作流程。工作流程配置指令碼會傳回 Workflow 物件。AWS Glue 會根據該物件建立工作流程。

必要建構函數引數

以下為 Workflow 類別的必要建構函數引數。

引數名稱	類型	描述
Name	str	要指派給工作流程的名稱。
Entities	Entities	要包含在工作流程中的實體 (任務和爬蟲程式) 集合。Entities 類別建構函數接受 Jobs 引數 (Job 物件的清單)，以及 Crawlers 引數 (Crawler 物件的清單)。

選用建構函數引數

以下為 Workflow 類別的選用建構函數引數。

引數名稱	類型	描述
Description	str	請參閱 Workflow 結構。
DefaultRunProperties	dict	請參閱 Workflow 結構。

引數名稱	類型	描述
OnSchedule	str	cron 表達式。

Class 方法

所有這三個類別包括以下方法。

validate()

驗證物件的屬性，如果發現錯誤，則輸出訊息並結束。如果沒有錯誤，則不會產生輸出。對於 Workflow 類別，在工作流程中的每個實體上呼叫自己。

to_json()

將物件序列化為 JSON。也會呼叫 validate()。對於 Workflow 類別，JSON 物件包含任務和爬蟲程式清單，以及任務和爬蟲程式相依性規格所產生的觸發清單。

藍圖範例

AWS Glue 藍圖 Github 儲存庫提供許多範例藍圖專案。這些範例僅供參考，並非作為生產用途。

範例專案的標題如下：

- 壓縮：此藍圖會建立任務，根據所需的檔案大小，將輸入檔案壓縮成較大的區塊。
- 轉換：此藍圖會將各種標準檔案格式的輸入檔案轉換為 Apache Prquet 格式，並針對分析工作負載進行最佳化。
- 網路爬取 Amazon S3 位置：此藍圖會網路爬取多個 Amazon S3 位置，以將中繼資料表新增至 Data Catalog 。
- 與 Data Catalog 的自訂連線：此藍圖使用 AWS Glue 自訂連接器存取資料存放區、讀取記錄，並根據記錄結構描述在 AWS Glue Data Catalog 中填入資料表定義。
- 編碼：此藍圖會將非 UTF 檔案轉換為 UTF 編碼檔案。
- 分割：此藍圖會建立分割任務，根據特定分割索引鍵將輸出檔案放置到分割區。
- 將 Amazon S3 資料匯入到 DynamoDB 資料表：此藍圖會將資料從 Amazon S3 匯入到 DynamoDB 資料表。
- 要管控的標準資料表：此藍圖會將 AWS Glue Data Catalog 資料表匯入 Lake Formation 資料表。

在 AWS Glue 中註冊藍圖

AWS Glue 開發人員已編寫藍圖並將 ZIP 封存上傳到 Amazon Simple Storage Service (Amazon S3) 之後，AWS Glue 管理員必須註冊藍圖。註冊藍圖以讓其可供使用。

當您註冊藍圖時，AWS Glue 會將藍圖封存複製到保留的 Amazon S3 位置。然後，您可以從上傳位置刪除封存。

若要註冊藍圖，您需要具備包含上傳封存的 Amazon S3 位置的讀取許可。您也需要 AWS Identity and Access Management (IAM) 許可 `glue:CreateBlueprint`。如需必須註冊、檢視和維護藍圖的 AWS Glue 管理員的建議許可，請參閱 藍圖的 AWS Glue 管理員許可。

您可以使用 AWS Glue 主控台、AWS Glue API，或 AWS Command Line Interface (AWS CLI) 註冊藍圖。

註冊藍圖 (主控台)

1. 確保您對 Amazon S3 中的藍圖 ZIP 封存具有讀取許可 (`s3:GetObject`)。
2. 開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。

   以具有註冊藍圖的許可的使用者身分登入。切換到與 Amazon S3 儲存貯體相同的 AWS 區域，其中包含藍圖 ZIP 封存。
3. 在導覽窗格中，選擇 blueprints (藍圖)。然後在 blueprints (藍圖) 頁面，選擇 Add blueprint (新增藍圖)。
4. 輸入藍圖名稱，以及選用描述。
5. 對於 ZIP archive location (S3) (ZIP 封存位置 (S3))，輸入已上傳藍圖 ZIP 封存的 Amazon S3 路徑。在路徑中包含封存檔案名稱，並以 `s3://` 開頭。
6. (選用) 新增標記一或多個標籤。
7. 選擇 Add blueprint (新增藍圖)。

   blueprints (藍圖) 頁面會傳回，並顯示藍圖狀態為 `CREATING`。選擇重新整理按鈕，直到狀態變更為 `ACTIVE` 或 `FAILED`。
8. 如果狀態為 `FAILED`，請選取藍圖，然後在 Actions (動作) 選單，選擇 View (檢視)。

   詳細資訊頁面會顯示失敗的原因。如果錯誤訊息為「無法存取以下位置的物件...」或「以下位置的物件存取被拒絕...」，請檢閱下列需求：

   - 您登入的使用者身分必須具有 Amazon S3 中藍圖 ZIP 封存的讀取許可。

- 包含 ZIP 封存的 Amazon S3 儲存貯體必須有儲存貯體政策，以授權物件的讀取許可給您的 AWS 帳戶 ID。如需詳細資訊，請參閱 AWS Glue 中的開發藍圖。
- 您使用的 Amazon S3 儲存貯體必須與您在主控台上登入的區域位在同一區域。

9. 確保資料分析師具有藍圖的許可。

   藍圖的資料分析師許可 中顯示適用於資料分析師的建議 IAM 政策。此政策授予任何資源上的 `glue:GetBlueprint`。如果您的政策在資源層級更精細，請授與資料分析師對此新建立資源的許可。

註冊藍圖 (AWS CLI)

1. 輸入以下命令。

```
aws glue create-blueprint --name <blueprint-name> [--description <description>] --
blueprint-location s3://<s3-path>/<archive-filename>
```

2. 如要檢查藍圖狀態，請輸入以下命令。重複此指令，直到狀態變為 ACTIVE 或 FAILED。

```
aws glue get-blueprint --name <blueprint-name>
```

   如果狀態為 FAILED 並且錯誤訊息為「無法存取以下位置的物件...」或「以下位置的物件存取被拒絕...」中，請檢閱下列需求：

   - 您登入的使用者身分必須具有 Amazon S3 中藍圖 ZIP 封存的讀取許可。
   - 包含 ZIP 封存的 Amazon S3 儲存貯體必須有儲存貯體政策，以授權物件的讀取許可給您的 AWS 帳戶 ID。如需詳細資訊，請參閱 發佈藍圖。
   - 您使用的 Amazon S3 儲存貯體必須與您在主控台上登入的區域位在同一區域。

> 另請參閱:
>
> - AWS Glue 中的藍圖概觀

檢視 AWS Glue 中的藍圖

檢視藍圖可以檢閱藍圖描述、狀態和參數規格，以及下載藍圖 ZIP 封存。

您可以使用 AWS Glue 主控台、AWS Glue API 或 AWS Command Line Interface (AWS CLI) 來檢視藍圖。

檢視藍圖 (主控台)

1. 開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格中，選擇 blueprints (藍圖)。
3. 在 blueprints (藍圖) 頁面上，選取藍圖。然後在 Actions (動作) 選單，選擇 View (檢視)。

檢視藍圖 (AWS CLI)

- 輸入下列命令以僅檢視藍圖名稱、描述和狀態。將 *<blueprint-name>* 替換為要檢視的藍圖名稱。

```
aws glue get-blueprint --name <blueprint-name>
```

命令輸出類似如下所示。

```
{
    "Blueprint": {
        "Name": "myDemoBP",
        "CreatedOn": 1587414516.92,
        "LastModifiedOn": 1587428838.671,
        "BlueprintLocation": "s3://awsexamplebucket1/demo/
DemoBlueprintProject.zip",
        "Status": "ACTIVE"
    }
}
```

輸入以下命令，也可以檢視參數規格。

```
aws glue get-blueprint --name <blueprint-name>  --include-parameter-spec
```

命令輸出類似如下所示。

```
{
    "Blueprint": {
        "Name": "myDemoBP",
        "CreatedOn": 1587414516.92,
```

```
        "LastModifiedOn": 1587428838.671,
        "ParameterSpec": "{\"WorkflowName\":{\"type\":\"String\",\"collection
\":false,\"description\":null,\"defaultValue\":null,\"allowedValues\":null},
\"PassRole\":{\"type\":\"String\",\"collection\":false,\"description\":null,
\"defaultValue\":null,\"allowedValues\":null},\"DynamoDBTableName\":{\"type
\":\"String\",\"collection\":false,\"description\":null,\"defaultValue\":null,
\"allowedValues\":null},\"ScriptLocation\":{\"type\":\"String\",\"collection
\":false,\"description\":null,\"defaultValue\":null,\"allowedValues\":null}}",
        "BlueprintLocation": "s3://awsexamplebucket1/demo/
DemoBlueprintProject.zip",
        "Status": "ACTIVE"
    }
}
```

新增 `--include-blueprint` 引數以在輸出中包含 URL，您可以貼到瀏覽器中，來下載 AWS Glue 儲存的藍圖 ZIP 封存。

> 另請參閱:
>
> - AWS Glue 中的藍圖概觀

更新 AWS Glue 中的藍圖

如果您有修訂配置指令碼、已修訂的藍圖參數集或已修訂的支援檔案，則可以更新藍圖。更新藍圖會建立新的版本。

更新藍圖不會影響從藍圖建立的現有工作流程。

您可以使用 AWS Glue 主控台、AWS Glue API 或 AWS Command Line Interface (AWS CLI) 更新藍圖。

下列處理程序假設 AWS Glue 開發人員已建立並上傳更新的藍圖 ZIP 封存到 Amazon S3。

更新藍圖 (主控台)

1. 確保您對 Amazon S3 中的藍圖 ZIP 封存具有讀取許可 (`s3:GetObject`)。
2. 開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。

以具有更新藍圖的使用者身分登入。切換到與 Amazon S3 儲存貯體相同的 AWS 區域，其中包含藍圖 ZIP 封存。

3. 在導覽窗格中，選擇 blueprints (藍圖)。
4. 在 Blueprints (藍圖) 頁面上，選取藍圖，然後在 Actions (動作) 選單，選擇 Edit (編輯)。
5. 在 Edit a blueprint (編輯藍圖) 頁面上，更新藍圖 Description (描述) 或 ZIP archive location (S3) (ZIP 封存位置 (S3))。務必在路徑中包含封存名稱。
6. 選擇 Save (儲存)。

   blueprints (藍圖) 頁面會傳回，並顯示藍圖狀態為 UPDATING。選擇重新整理按鈕，直到狀態變更為 ACTIVE 或 FAILED。
7. 如果狀態為 FAILED，請選取藍圖，然後在 Actions (動作) 選單，選擇 View (檢視)。

   詳細資訊頁面會顯示失敗的原因。如果錯誤訊息為「無法存取以下位置的物件...」或「以下位置的物件存取被拒絕...」，請檢閱下列需求：

   - 您登入的使用者身分必須具有 Amazon S3 中藍圖 ZIP 封存的讀取許可。
   - 包含 ZIP 封存的 Amazon S3 儲存貯體必須有儲存貯體政策，以授權物件的讀取許可給您的 AWS 帳戶 ID。如需詳細資訊，請參閱 發佈藍圖。
   - 您使用的 Amazon S3 儲存貯體必須與您在主控台上登入的區域位在同一區域。

   > **Note**
   >
   > 如果更新失敗，則下一次執行藍圖會使用已成功註冊或更新的最新版本藍圖。

更新藍圖 (AWS CLI)

1. 輸入以下命令。

```
aws glue update-blueprint --name <blueprint-name> [--description <description>] --
blueprint-location s3://<s3-path>/<archive-filename>
```

2. 如要檢查藍圖狀態，請輸入以下命令。重複此指令，直到狀態變為 ACTIVE 或 FAILED。

```
aws glue get-blueprint --name <blueprint-name>
```

如果狀態為 FAILED 並且錯誤訊息為「無法存取以下位置的物件...」或「以下位置的物件存取被拒絕...」中，請檢閱下列需求：

- 您登入的使用者身分必須具有 Amazon S3 中藍圖 ZIP 封存的讀取許可。
- 包含 ZIP 封存的 Amazon S3 儲存貯體必須有儲存貯體政策，以授權物件的讀取許可給您的 AWS 帳戶 ID。如需詳細資訊，請參閱 發佈藍圖。
- 您使用的 Amazon S3 儲存貯體必須與您在主控台上登入的區域位在同一區域。

另請參閱

- AWS Glue 中的藍圖概觀

在 AWS Glue 中從藍圖建立工作流程

您可以手動建立 AWS Glue 工作流程，一次新增一個元件，或者您可以從 AWS Glue 藍圖建立工作流程。AWS Glue 包含常見使用案例的藍圖。您的 AWS Glue 開發人員可以建立額外的藍圖。

Important

將工作流程中的任務、爬蟲程式和觸發程序總數限制在 100 或更少。如果包含超過 100 個，則嘗試繼續或停止工作流程執行時可能會出現錯誤。

使用藍圖時，您可以根據藍圖定義的一般使用案例，快速產生特定使用案例的工作流程。您可以透過為藍圖參數提供值來定義特定的使用案例。例如，對資料集進行分割的藍圖可以將 Amazon S3 來源和目標路徑做為參數。

AWS Glue 透過執行藍圖，從藍圖建立工作流程。藍圖執行會儲存您提供的參數值，並用於追蹤建立工作流程及其元件的進度和結果。疑難排解工作流程時，您可以檢視藍圖執行以決定用於建立工作流程的藍圖參數值。

若要建立和檢視工作流程，您需要特定 IAM 許可。如需建議的 IAM 政策，請參閱 藍圖的資料分析師許可。

您可以使用 AWS Glue 主控台、AWS Glue API 或 AWS Command Line Interface (AWS CLI) 來從藍圖建立工作流程。

從藍圖建立工作流程 (主控台)

1. 開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。

   以具有建立工作流程許可的使用者身分登入。

2. 在導覽窗格中，選擇 blueprints (藍圖)。

3. 選取藍圖，然後在 Actions (動作) 選單，選擇 Create workflow (建立工作流程)。

4. 在 Create a workflow from <blueprint-name> (從 <藍圖名稱> 建立工作流程) 頁面上，輸入下列資訊：

   藍圖參數

   這些因藍圖設計而異。如需這些參數的相關問題資訊，請聯絡開發人員。藍圖通常包含工作流程名稱的參數。

   IAM 角色

   AWS Glue 擔任之用來建立工作流程及其元件的角色。此角色必須擁有建立和刪除工作流程、任務、爬蟲程式和觸發的許可。如需角色的建議政策，請參閱藍圖角色的許可。

5. 選擇 Submit (提交)。

   Blueprint Details (藍圖詳細資訊) 頁面出現，並在底部顯示藍圖執行清單。

6. 在藍圖執行清單中，檢查最上層的藍圖執行以取得工作流程建立狀態。

   起始狀態為 RUNNING。選擇重新整理按鈕，直到狀態變為 SUCCEEDED 或 FAILED。

7. 執行下列任意一項：

   - 如果完成狀態為 SUCCEEDED，您可以移至 Workflows (工作流程) 頁面，選取新建立的工作流程，然後執行它。在執行工作流程之前，您可以先檢閱設計圖形。
   - 如果狀態為 FAILED，請選取藍圖執行，然後在 Actions (動作) 功能表，選擇 View (檢視) 以查看錯誤訊息。

如需工作流程和藍圖的詳細資訊，請參閱下列主題。

- AWS Glue 中的工作流程概觀
- 更新 AWS Glue 中的藍圖
- 在 AWS Glue 中手動建立和建構工作流程

檢視 AWS Glue 中的藍圖執行

檢視藍圖執行以查看下列資訊：

- 已建立的工作流程名稱。
- 用於建立工作流程的藍圖參數值。
- 工作流程建立操作的狀態。

您可以使用 AWS Glue 主控台、AWS Glue API 或 AWS Command Line Interface (AWS CLI) 檢視藍圖執行。

檢視藍圖執行 (主控台)

1. 開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格中，選擇 blueprints (藍圖)。
3. 在 blueprints (藍圖) 頁面上，選取藍圖。然後在 Actions (動作) 選單，選擇 View (檢視)。
4. 在 Blueprint Details (藍圖詳細資訊) 頁面底部，選取藍圖執行，然後在 Actions (動作) 選單，選擇 View (檢視)。

檢視藍圖 (AWS CLI)

- 輸入以下命令。將 *<blueprint-name>* 替換為藍圖的名稱。將 *<blueprint-run-id>* 替換為藍圖執行 ID。

```
aws glue get-blueprint-run --blueprint-name <blueprint-name> --run-id <blueprint-run-id>
```

另請參閱:

- AWS Glue 中的藍圖概觀

適用於 AWS Glue 的 AWS CloudFormation

AWS CloudFormation 是一項可建立多個 AWS 資源的服務。AWS Glue 提供 API 操作，以在 AWS Glue Data Catalog 中建立物件。不過，若是以 AWS CloudFormation 範本檔案來定義和建立 AWS Glue 物件及其他相關的 AWS 資源物件，應會更加簡單輕鬆。接下來即可將建立物件的程序自動化。

AWS CloudFormation 提供了簡化的語法，JSON (JavaScript Object Notation) 或 YAML (YAML Ain't Markup Language) 均有，可加快 AWS 資源的建立速度。可使用 AWS CloudFormation 範本來定義資料目錄物件，例如資料庫、資料表、分割區、爬蟲程式、分類器及連線。也可定義 ETL 物件，如任務、觸發條件、開發端點。您建立一個範本來描述您需要的所有 AWS 資源，而 AWS CloudFormation 負責為您佈建與設定這些資源。

如需詳細資訊，請參閱 AWS CloudFormation 使用者指南中的AWS CloudFormation 是什麼?和使用 AWS CloudFormation 範本。

若打算使用相容於 AWS Glue 的 AWS CloudFormation 範本，身為管理員的您必須將存取權限授予其所相依的 AWS CloudFormation 和 AWS 服務和動作。若要授予建立 AWS CloudFormation 資源的許可，請將下列政策交給使用 AWS CloudFormation 的使用者：

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

下表包含 AWS CloudFormation 範本可代替您執行的動作。包含 AWS 資源類型相關資訊的連結，以及可已新增至 AWS CloudFormation 範本的屬性類型。

AWS Glue 資源	AWS CloudFormation 範本	AWS Glue 範例
分類器	AWS::Glue::Classifier	Grok 分類器、JSON 分類器、XML 分類器

AWS Glue 資源	AWS CloudFormation 範本	AWS Glue 範例
Connection	AWS::Glue::Connection	MySQL 連線
爬蟲程式	AWS::Glue::Crawler	Amazon S3 爬蟲程式、MySQL 爬蟲程式
資料庫	AWS::Glue::Database	空資料庫、含資料表的資料庫
開發端點	AWS::Glue::DevEndpoint	開發端點
任務	AWS::Glue::Job	Amazon S3 任務、JDBC 任務
機器學習轉換	AWS::Glue::MLTransform	機器學習轉換
資料品質規則集	AWS::Glue::DataQualityRuleset	資料品質規則集，使用 EventBridge 排程器的資料品質規則集
分區	AWS::Glue::Partition	表格分割區
資料表	AWS::Glue::Table	資料庫表格
觸發條件	AWS::Glue::Trigger	隨需觸發、排程觸發、條件式觸發

若要開始，請使用下方的範本，使用您自己的中繼資料加以自訂。然後使用 AWS CloudFormation 主控台建立 AWS CloudFormation 堆疊，以新增物件到 AWS Glue 和任何相關的服務。AWS Glue 物件有許多欄位為選填。這些範本會說明 AWS Glue 物件若要正常有效運作需要填寫哪些欄位，或哪些欄位為必填。

AWS CloudFormation 範本格式可以是 JSON 或 YAML。這些範例會使用 YAML 以方便閱讀。範例內有評論 (#) 會說明範本中定義的值。

AWS CloudFormation 範本可包含 `Parameters` 區段。可以變更範本文字中的該區段，或是在 YAML 檔案提交至 AWS CloudFormation 主控台時也可變更，以建立堆疊。此範本的 `Resources` 區段內含 AWS Glue 的定義和相關物件。AWS CloudFormation 範本語法定義可能會包含屬性，內有更多詳細的屬性語法。建立 AWS Glue 物件時，並不需要使用到所有屬性。以下範本為用於建立 AWS Glue 物件的常見屬性的範例值。

適用於 AWS Glue 資料庫的範例 AWS CloudFormation 範本

資料目錄內的 AWS Glue 資料庫含有中繼資料資料表。資料庫是由極少量的屬性所組成，可使用 AWS CloudFormation 範本建立於資料目錄內。以下範本的範例可協助您開始使用，並說明 AWS CloudFormation 堆疊與 AWS Glue 的使用方法。此範本範例唯一建立的資源是名為 `cfn-mysampledatabase` 的資料庫。您可以編輯範例文字，或是在提交 YAML 時更改 AWS CloudFormation 主控台的值，加以變更資料庫。

以下顯示的是用於建立 AWS Glue 資料庫的常見屬性的範例值。如需 AWS CloudFormation 的 AWS Glue 資料庫範本的詳細資訊，請參閱 AWS::Glue::Database。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CloudFormation template in YAML to demonstrate creating a database named
 mysampledatabase
# The metadata created in the Data Catalog points to the flights public S3 bucket
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  CFNDatabaseName:
    Type: String
    Default: cfn-mysampledatabse

# Resources section defines metadata for the Data Catalog
Resources:
# Create an AWS Glue database
  CFNDatabaseFlights:
    Type: AWS::Glue::Database
    Properties:
      # The database is created in the Data Catalog for your account
      CatalogId: !Ref AWS::AccountId
      DatabaseInput:
        # The name of the database is defined in the Parameters section above
        Name: !Ref CFNDatabaseName
        Description: Database to hold tables for flights data
        LocationUri: s3://crawler-public-us-east-1/flight/2016/csv/
        #Parameters: Leave AWS database parameters blank
```

AWS Glue 資料庫、資料表和分割區的範例 AWS CloudFormation 範本

AWS Glue 資料表內含的中繼資料定義了希望以 ETL 指定碼處理的資料之結構和位置。在此資料表中，可定義要用以將資料處理平行化的分區。分區是您以金鑰值定義的資料區塊。舉例而言，使用月份做為金鑰值，則所有一月份的資料都會包含在同一個分區內。在 AWS Glue 中，資料庫可含有資料表，而資料表可包含分區。

以下範例顯示了如何使用 AWS CloudFormation 範本產生資料庫、資料表和分區。基本資料格式為 `csv`，並以逗號 (,) 分隔。由於資料庫必須在含有資料表前先已存在，而資料表必須先存在才可建立分區，因此範本使用 `DependsOn` 陳述式在物件建立時定義其相依性。

此範例中的值定義了某個資料表，表內含有從某個公開的 Amazon S3 儲存貯體取得的航班資料。為了說明之用，僅定義了少許資料欄位和一個分區金鑰。資料目錄中也定義了四個分區。有些用於描述基本資料的儲存的欄位也會顯示於 `StorageDescriptor` 的欄位中。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CloudFormation template in YAML to demonstrate creating a database, a table,
 and partitions
# The metadata created in the Data Catalog points to the flights public S3 bucket
#
# Parameters substituted in the Resources section
# These parameters are names of the resources created in the Data Catalog
Parameters:
  CFNDatabaseName:
    Type: String
    Default: cfn-database-flights-1
  CFNTableName1:
    Type: String
    Default: cfn-manual-table-flights-1
# Resources to create metadata in the Data Catalog
Resources:
###
# Create an AWS Glue database
  CFNDatabaseFlights:
    Type: AWS::Glue::Database
    Properties:
      CatalogId: !Ref AWS::AccountId
      DatabaseInput:
```

```
        Name: !Ref CFNDatabaseName
        Description: Database to hold tables for flights data
###
# Create an AWS Glue table
  CFNTableFlights:
    # Creating the table waits for the database to be created
    DependsOn: CFNDatabaseFlights
    Type: AWS::Glue::Table
    Properties:
      CatalogId: !Ref AWS::AccountId
      DatabaseName: !Ref CFNDatabaseName
      TableInput:
        Name: !Ref CFNTableName1
        Description: Define the first few columns of the flights table
        TableType: EXTERNAL_TABLE
        Parameters: {
    "classification": "csv"
  }
#       ViewExpandedText: String
        PartitionKeys:
        # Data is partitioned by month
        - Name: mon
          Type: bigint
        StorageDescriptor:
          OutputFormat: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
          Columns:
          - Name: year
            Type: bigint
          - Name: quarter
            Type: bigint
          - Name: month
            Type: bigint
          - Name: day_of_month
            Type: bigint
          InputFormat: org.apache.hadoop.mapred.TextInputFormat
          Location: s3://crawler-public-us-east-1/flight/2016/csv/
          SerdeInfo:
            Parameters:
              field.delim: ","
            SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 1
# Create an AWS Glue partition
  CFNPartitionMon1:
    DependsOn: CFNTableFlights
```

```
    Type: AWS::Glue::Partition
    Properties:
      CatalogId: !Ref AWS::AccountId
      DatabaseName: !Ref CFNDatabaseName
      TableName: !Ref CFNTableName1
      PartitionInput:
        Values:
        - 1
        StorageDescriptor:
          OutputFormat: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
          Columns:
          - Name: mon
            Type: bigint
          InputFormat: org.apache.hadoop.mapred.TextInputFormat
          Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=1/
          SerdeInfo:
            Parameters:
              field.delim: ","
            SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 2
# Create an AWS Glue partition
  CFNPartitionMon2:
    DependsOn: CFNTableFlights
    Type: AWS::Glue::Partition
    Properties:
      CatalogId: !Ref AWS::AccountId
      DatabaseName: !Ref CFNDatabaseName
      TableName: !Ref CFNTableName1
      PartitionInput:
        Values:
        - 2
        StorageDescriptor:
          OutputFormat: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
          Columns:
          - Name: mon
            Type: bigint
          InputFormat: org.apache.hadoop.mapred.TextInputFormat
          Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=2/
          SerdeInfo:
            Parameters:
              field.delim: ","
            SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 3
# Create an AWS Glue partition
```

```
  CFNPartitionMon3:
    DependsOn: CFNTableFlights
    Type: AWS::Glue::Partition
    Properties:
      CatalogId: !Ref AWS::AccountId
      DatabaseName: !Ref CFNDatabaseName
      TableName: !Ref CFNTableName1
      PartitionInput:
        Values:
        - 3
        StorageDescriptor:
          OutputFormat: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
          Columns:
          - Name: mon
            Type: bigint
          InputFormat: org.apache.hadoop.mapred.TextInputFormat
          Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=3/
          SerdeInfo:
            Parameters:
              field.delim: ","
            SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 4
# Create an AWS Glue partition
  CFNPartitionMon4:
    DependsOn: CFNTableFlights
    Type: AWS::Glue::Partition
    Properties:
      CatalogId: !Ref AWS::AccountId
      DatabaseName: !Ref CFNDatabaseName
      TableName: !Ref CFNTableName1
      PartitionInput:
        Values:
        - 4
        StorageDescriptor:
          OutputFormat: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
          Columns:
          - Name: mon
            Type: bigint
          InputFormat: org.apache.hadoop.mapred.TextInputFormat
          Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=4/
          SerdeInfo:
            Parameters:
              field.delim: ","
```

```
            SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
```

適用於 AWS Glue grok 分類器的範例 AWS CloudFormation 範本

AWS Glue 分類器可判斷資料的結構描述。One 類型的自訂分類器會使用 grok 模式配對您的資料。若模式比對符合，則會使用自訂分類器來建立資料表的結構資料，並將 `classification` 設為分類器定義中所設的值。

這個範例所建立的分類器，會建立含有一個名為 `message` 的欄位的資料結構，並將分類設為 `greedy`。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a classifier
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the classifier to be created
  CFNClassifierName:
    Type: String
    Default: cfn-classifier-grok-one-column-1

#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create classifier that uses grok pattern to put all data in one column and classifies
 it as "greedy".
  CFNClassifierFlights:
    Type: AWS::Glue::Classifier
    Properties:
      GrokClassifier:
        #Grok classifier that puts all data in one column
        Name: !Ref CFNClassifierName
        Classification: greedy

        GrokPattern: "%{GREEDYDATA:message}"
        #CustomPatterns: none
```

適用於 AWS Glue JSON 分類器的範例 AWS CloudFormation 範本

AWS Glue 分類器可判斷資料的結構描述。一種自訂分類器使用 JsonPath 字串，定義 JSON 以供分類器分類。AWS Glue 支援JsonPath 的運算子子集，如撰寫 JsonPath 自訂分類器中所述。

如果模式符合，則自訂分類器可用於建立資料表的結構描述。

這個範本所建立的分類器會建立結構描述，每個記錄皆位於物件中的 Records3 陣列。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a JSON classifier
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the classifier to be created
  CFNClassifierName:
    Type: String
    Default: cfn-classifier-json-one-column-1

#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create classifier that uses a JSON pattern.
  CFNClassifierFlights:
    Type: AWS::Glue::Classifier
    Properties:
      JSONClassifier:
        #JSON classifier
        Name: !Ref CFNClassifierName
        JsonPath: $.Records3[*]
```

適用於 AWS Glue XML 分類器的範例 AWS CloudFormation 範本

AWS Glue 分類器可判斷資料的結構描述。一種自訂分類器指定 XML 標籤，以在經剖析的 XML 文件內指定包含各記錄的元素。若模式比對符合，則會使用自訂分類器來建立資料表的結構資料，並將 classification 設為分類器定義中所設的值。

這個範例所建立的分類器，會建立一個每個記錄皆位於 Record 標籤的結構描述，並將分類設為 XML。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating an XML classifier
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the classifier to be created
  CFNClassifierName:
    Type: String
    Default: cfn-classifier-xml-one-column-1

#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create classifier that uses the XML pattern and classifies it as "XML".
  CFNClassifierFlights:
    Type: AWS::Glue::Classifier
    Properties:
      XMLClassifier:
        #XML classifier
        Name: !Ref CFNClassifierName
        Classification: XML
        RowTag: <Records>
```

適用於 Amazon S3 AWS Glue 爬蟲程式的範例 AWS CloudFormation 範本

AWS Glue 爬蟲程式會在資料目錄中建立與資料對應的中繼資料資料表。接下來可使用這些資料表定義做為 ETL 任務的來源和目標。

此範例會在資料目錄中建立一個爬蟲程式、所需的 IAM 角色、AWS Glue 資料庫。在執行此爬蟲程式時，其會擔任 IAM 角色，並為公開的航班資料的資料庫建立一份資料表。資料表建立時會附帶字首「cfn_sample_1_」。此範本所建立的 IAM 角色允許全域許可，您可能會想建立一個自訂角色。此分類器並未定義任何自訂的分類器。預設使用 AWS Glue 內建的分類器。

當您將此範例提交至 AWS CloudFormation 主控台時，務必要確認您想要建立 IAM 角色。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a crawler
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the crawler to be created
  CFNCrawlerName:
    Type: String
    Default: cfn-crawler-flights-1
  CFNDatabaseName:
    Type: String
    Default: cfn-database-flights-1
  CFNTablePrefixName:
    Type: String
    Default: cfn_sample_1_
#
#
# Resources section defines metadata for the Data Catalog
Resources:
#Create IAM Role assumed by the crawler. For demonstration, this role is given all
 permissions.
  CFNRoleFlights:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          -
            Effect: "Allow"
            Principal:
              Service:
                - "glue.amazonaws.com"
            Action:
              - "sts:AssumeRole"
      Path: "/"
      Policies:
        -
          PolicyName: "root"
```

```
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              -
                Effect: "Allow"
                Action: "*"
                Resource: "*"
  # Create a database to contain tables created by the crawler
   CFNDatabaseFlights:
     Type: AWS::Glue::Database
     Properties:
       CatalogId: !Ref AWS::AccountId
       DatabaseInput:
         Name: !Ref CFNDatabaseName
         Description: "AWS Glue container to hold metadata tables for the flights
  crawler"
  #Create a crawler to crawl the flights data on a public S3 bucket
   CFNCrawlerFlights:
     Type: AWS::Glue::Crawler
     Properties:
       Name: !Ref CFNCrawlerName
       Role: !GetAtt CFNRoleFlights.Arn
       #Classifiers: none, use the default classifier
       Description: AWS Glue crawler to crawl flights data
       #Schedule: none, use default run-on-demand
       DatabaseName: !Ref CFNDatabaseName
       Targets:
         S3Targets:
           # Public S3 bucket with the flights data
           - Path: "s3://crawler-public-us-east-1/flight/2016/csv"
       TablePrefix: !Ref CFNTablePrefixName
       SchemaChangePolicy:
         UpdateBehavior: "UPDATE_IN_DATABASE"
         DeleteBehavior: "LOG"
       Configuration: "{\"Version\":1.0,\"CrawlerOutput\":{\"Partitions\":
{\"AddOrUpdateBehavior\":\"InheritFromTable\"},\"Tables\":{\"AddOrUpdateBehavior\":
\"MergeNewColumns\"}}}"
```

適用於 AWS Glue 連線的範例 AWS CloudFormation 範本

資料目錄內的 AWS Glue 連線含有連線到 JDBC 資料庫所需的 JDBC 和網路資訊。在連線到 JDBC 資料庫以探索或執行 ETL 任務時，均會用到此資訊。

此範例會建立一個連至 Amazon RDS MySQL 資料庫的連線，名為 devdb。使用此連線時，也須提供 IAM 角色、資料庫登入資料、網路連線的值。請參閱範本內的必要欄位詳細資訊。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a connection
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the connection to be created
  CFNConnectionName:
    Type: String
    Default: cfn-connection-mysql-flights-1
  CFNJDBCString:
    Type: String
    Default: "jdbc:mysql://xxx-mysql.yyyyyyyyyyyyyy.us-east-1.rds.amazonaws.com:3306/
devdb"
  CFNJDBCUser:
    Type: String
    Default: "master"
  CFNJDBCPassword:
    Type: String
    Default: "12345678"
    NoEcho: true
#
#
# Resources section defines metadata for the Data Catalog
Resources:
  CFNConnectionMySQL:
    Type: AWS::Glue::Connection
    Properties:
      CatalogId: !Ref AWS::AccountId
      ConnectionInput:
        Description: "Connect to MySQL database."
        ConnectionType: "JDBC"
        #MatchCriteria: none
        PhysicalConnectionRequirements:
          AvailabilityZone: "us-east-1d"
          SecurityGroupIdList:
           - "sg-7d52b812"
```

```
          SubnetId: "subnet-84f326ee"
        ConnectionProperties: {
          "JDBC_CONNECTION_URL": !Ref CFNJDBCString,
          "USERNAME": !Ref CFNJDBCUser,
          "PASSWORD": !Ref CFNJDBCPassword
        }
        Name: !Ref CFNConnectionName
```

適用於 JDBC AWS Glue 爬蟲程式的範例 AWS CloudFormation 範本

AWS Glue 爬蟲程式會在資料目錄中建立與資料對應的中繼資料資料表。接下來可使用這些資料表定義做為 ETL 任務的來源和目標。

此範例會在資料目錄中建立一個爬蟲程式、所需的 IAM 角色、AWS Glue 資料庫。在執行此爬蟲程式時，其會擔任 IAM 角色，並為儲存在某個 MySQL 資料庫內的航班資料所建的資料庫建立一份資料表。資料表建立時會附帶字首「cfn_jdbc_1_」。此範本所建立的 IAM 角色允許全域許可，您可能會想建立一個自訂角色。無法為 JDBC 資料定義自訂分類器。預設使用 AWS Glue 內建的分類器。

當您將此範例提交至 AWS CloudFormation 主控台時，務必要確認您想要建立 IAM 角色。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a crawler
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the crawler to be created
  CFNCrawlerName:
    Type: String
    Default: cfn-crawler-jdbc-flights-1
# The name of the database to be created to contain tables
  CFNDatabaseName:
    Type: String
    Default: cfn-database-jdbc-flights-1
# The prefix for all tables crawled and created
  CFNTablePrefixName:
    Type: String
```

```
    Default: cfn_jdbc_1_
# The name of the existing connection to the MySQL database
  CFNConnectionName:
    Type: String
    Default: cfn-connection-mysql-flights-1
# The name of the JDBC path (database/schema/table) with wildcard (%) to crawl
  CFNJDBCPath:
    Type: String
    Default: saldev/%
#
#
# Resources section defines metadata for the Data Catalog
Resources:
#Create IAM Role assumed by the crawler. For demonstration, this role is given all
 permissions.
  CFNRoleFlights:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          -
            Effect: "Allow"
            Principal:
              Service:
                - "glue.amazonaws.com"
            Action:
              - "sts:AssumeRole"
      Path: "/"
      Policies:
        -
          PolicyName: "root"
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              -
                Effect: "Allow"
                Action: "*"
                Resource: "*"
 # Create a database to contain tables created by the crawler
  CFNDatabaseFlights:
    Type: AWS::Glue::Database
    Properties:
      CatalogId: !Ref AWS::AccountId
```

```
      DatabaseInput:
        Name: !Ref CFNDatabaseName
        Description: "AWS Glue container to hold metadata tables for the flights
 crawler"
 #Create a crawler to crawl the flights data in MySQL database
  CFNCrawlerFlights:
    Type: AWS::Glue::Crawler
    Properties:
      Name: !Ref CFNCrawlerName
      Role: !GetAtt CFNRoleFlights.Arn
      #Classifiers: none, use the default classifier
      Description: AWS Glue crawler to crawl flights data
      #Schedule: none, use default run-on-demand
      DatabaseName: !Ref CFNDatabaseName
      Targets:
        JdbcTargets:
          # JDBC MySQL database with the flights data
          - ConnectionName: !Ref CFNConnectionName
            Path: !Ref CFNJDBCPath
          #Exclusions: none
      TablePrefix: !Ref CFNTablePrefixName
      SchemaChangePolicy:
        UpdateBehavior: "UPDATE_IN_DATABASE"
        DeleteBehavior: "LOG"
   Configuration: "{\"Version\":1.0,\"CrawlerOutput\":{\"Partitions\":
{\"AddOrUpdateBehavior\":\"InheritFromTable\"},\"Tables\":{\"AddOrUpdateBehavior\":
\"MergeNewColumns\"}}}"
```

適用於 Amazon S3 至 Amazon S3 的 AWS Glue 任務的範例 AWS CloudFormation 範本

在資料目錄中的 AWS Glue 任務含有在 AWS Glue 中執行指令碼所需的參數值。

此範例會建立一項任務，用以讀取來自 Amazon S3 儲存貯體的航班資料 (格式為 `csv`)，並將之寫入 Amazon S3 Parquet 檔案。此任務所執行的此指令碼必須已先存在。可以使用 AWS Glue 主控台為您的環境產生 ETL 指令碼。執行工作時，也必須提供具有正確許可的 IAM 角色。

常見的參數值會出現在範本中。舉例而言，`AllocatedCapacity` (DPU) 預設值為 5。

```
---
AWSTemplateFormatVersion: '2010-09-09'
```

```
# Sample CFN YAML to demonstrate creating a job using the public flights S3 table in a
 public bucket
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the job to be created
  CFNJobName:
    Type: String
    Default: cfn-job-S3-to-S3-2
# The name of the IAM role that the job assumes. It must have access to data, script,
 temporary directory
  CFNIAMRoleName:
    Type: String
    Default: AWSGlueServiceRoleGA
# The S3 path where the script for this job is located
  CFNScriptLocation:
    Type: String
    Default: s3://aws-glue-scripts-123456789012-us-east-1/myid/sal-job-test2
#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create job to run script which accesses flightscsv table and write to S3 file as
 parquet.
# The script already exists and is called by this job
  CFNJobFlights:
    Type: AWS::Glue::Job
    Properties:
      Role: !Ref CFNIAMRoleName
      #DefaultArguments: JSON object
      # If script written in Scala, then set DefaultArguments={'--job-language';
 'scala', '--class': 'your scala class'}
      #Connections:  No connection needed for S3 to S3 job
      #  ConnectionsList
      #MaxRetries: Double
      Description: Job created with CloudFormation
      #LogUri: String
      Command:
        Name: glueetl
        ScriptLocation: !Ref CFNScriptLocation
             # for access to directories use proper IAM role with permission to buckets
 and folders that begin with "aws-glue-"
```

```
            # script uses temp directory from job definition if required (temp
 directory not used S3 to S3)
            # script defines target for output as s3://aws-glue-target/sal
      AllocatedCapacity: 5
      ExecutionProperty:
        MaxConcurrentRuns: 1
      Name: !Ref CFNJobName
```

適用於 JDBC 至 Amazon S3 的 AWS Glue 任務的範例 AWS CloudFormation 範本

在資料目錄中的 AWS Glue 任務含有在 AWS Glue 中執行指令碼所需的參數值。

此範例會建立一項任務，如名為 `cfn-connection-mysql-flights-1` 的連線所定義，從 MySQL JDBC 資料庫讀取航班資料，並將資料寫入 Amazon S3 Parquet 檔案。此任務所執行的此指令碼必須已先存在。可以使用 AWS Glue 主控台為您的環境產生 ETL 指令碼。執行工作時，也必須提供具有正確許可的 IAM 角色。

常見的參數值會出現在範本中。舉例而言，`AllocatedCapacity` (DPU) 預設值為 5。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a job using a MySQL JDBC DB with the flights
 data to an S3 file
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the job to be created
  CFNJobName:
    Type: String
    Default: cfn-job-JDBC-to-S3-1
# The name of the IAM role that the job assumes. It must have access to data, script,
 temporary directory
  CFNIAMRoleName:
    Type: String
    Default: AWSGlueServiceRoleGA
# The S3 path where the script for this job is located
  CFNScriptLocation:
    Type: String
```

```
    Default: s3://aws-glue-scripts-123456789012-us-east-1/myid/sal-job-dec4a
# The name of the connection used for JDBC data source
  CFNConnectionName:
    Type: String
    Default: cfn-connection-mysql-flights-1
#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create job to run script which accesses JDBC flights table via a connection and write
 to S3 file as parquet.
# The script already exists and is called by this job
  CFNJobFlights:
    Type: AWS::Glue::Job
    Properties:
      Role: !Ref CFNIAMRoleName
      #DefaultArguments: JSON object
      # For example, if required by script, set temporary directory as
 DefaultArguments={'--TempDir'; 's3://aws-glue-temporary-xyc/sal'}
      Connections:
        Connections:
        - !Ref CFNConnectionName
      #MaxRetries: Double
      Description: Job created with CloudFormation using existing script
      #LogUri: String
      Command:
        Name: glueetl
        ScriptLocation: !Ref CFNScriptLocation
             # for access to directories use proper IAM role with permission to buckets
 and folders that begin with "aws-glue-"
             # if required, script defines temp directory as argument TempDir and used
 in script like redshift_tmp_dir = args["TempDir"]
             # script defines target for output as s3://aws-glue-target/sal
      AllocatedCapacity: 5
      ExecutionProperty:
        MaxConcurrentRuns: 1
      Name: !Ref CFNJobName
```

AWS Glue 隨需觸發條件的範例 AWS CloudFormation 範本

資料目錄中的 AWS Glue 觸發條件含有必要的參數值，在觸發條件觸動而開始執行任務時會需要。啟用後，隨需觸發條件即會觸動。

此範例會建立一項隨需觸發條件，會開始進行名為 cfn-job-S3-to-S3-1 的任務。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating an on-demand trigger
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  # The existing job to be started by this trigger
  CFNJobName:
    Type: String
    Default: cfn-job-S3-to-S3-1
  # The name of the trigger to be created
  CFNTriggerName:
    Type: String
    Default: cfn-trigger-ondemand-flights-1
#
# Resources section defines metadata for the Data Catalog
# Sample CFN YAML to demonstrate creating an on-demand trigger for a job
Resources:
# Create trigger to run an existing job (CFNJobName) on an on-demand schedule.
  CFNTriggerSample:
    Type: AWS::Glue::Trigger
    Properties:
      Name:
        Ref: CFNTriggerName
      Description: Trigger created with CloudFormation
      Type: ON_DEMAND
      Actions:
        - JobName: !Ref CFNJobName
        # Arguments: JSON object
      #Schedule:
      #Predicate:
```

AWS Glue 排程觸發條件的範例 AWS CloudFormation 範本

資料目錄中的 AWS Glue 觸發條件含有必要的參數值，在觸發條件觸動而開始執行任務時會需要。排程觸發條件在啟用時即會觸發，並會跳出 cron 計時器。

此範例會建立一項排程觸發條件，會開始進行名為 cfn-job-S3-to-S3-1 的任務。計時器為 cron 表達式，在任務天每 10 分鐘就會執行一次任務。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a scheduled trigger
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  # The existing job to be started by this trigger
  CFNJobName:
    Type: String
    Default: cfn-job-S3-to-S3-1
  # The name of the trigger to be created
  CFNTriggerName:
    Type: String
    Default: cfn-trigger-scheduled-flights-1
#
# Resources section defines metadata for the Data Catalog
# Sample CFN YAML to demonstrate creating a scheduled trigger for a job
#
Resources:
# Create trigger to run an existing job (CFNJobName) on a cron schedule.
  TriggerSample1CFN:
    Type: AWS::Glue::Trigger
    Properties:
      Name:
        Ref: CFNTriggerName
      Description: Trigger created with CloudFormation
      Type: SCHEDULED
      Actions:
        - JobName: !Ref CFNJobName
        # Arguments: JSON object
      # # Run the trigger every 10 minutes on Monday to Friday
      Schedule: cron(0/10 * ? * MON-FRI *)
      #Predicate:
```

AWS Glue 條件式觸發條件的範例 AWS CloudFormation 範本

資料目錄中的 AWS Glue 觸發條件含有必要的參數值，在觸發條件觸動而開始執行任務時會需要。條件式觸發條件會在啟用時觸發，例如任務成功完成。

此範例會建立一項條件式觸發條件，會開始進行名為 cfn-job-S3-to-S3-1 的任務。此任務會在名為 cfn-job-S3-to-S3-2 的任務成功完成後發動。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a conditional trigger for a job, which starts
 when another job completes
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  # The existing job to be started by this trigger
  CFNJobName:
    Type: String
    Default: cfn-job-S3-to-S3-1
  # The existing job that when it finishes causes trigger to fire
  CFNJobName2:
    Type: String
    Default: cfn-job-S3-to-S3-2
  # The name of the trigger to be created
  CFNTriggerName:
    Type: String
    Default: cfn-trigger-conditional-1
#
Resources:
# Create trigger to run an existing job (CFNJobName) when another job completes
 (CFNJobName2).
  CFNTriggerSample:
    Type: AWS::Glue::Trigger
    Properties:
      Name:
        Ref: CFNTriggerName
      Description: Trigger created with CloudFormation
      Type: CONDITIONAL
      Actions:
        - JobName: !Ref CFNJobName
        # Arguments: JSON object
```

```
      #Schedule: none
      Predicate:
        #Value for Logical is required if more than 1 job listed in Conditions
        Logical: AND
        Conditions:
          - LogicalOperator: EQUALS
            JobName: !Ref CFNJobName2
            State: SUCCEEDED
```

AWS Glue 開發端點的範例 AWS CloudFormation 範本

AWS Glue 機器學習轉換是一種自訂轉換，可清理您的資料。目前有一個名為 FindMatches 的可用轉換。FindMatches 轉換可讓您識別資料集中重複或相符的記錄，即使記錄沒有通用的唯一識別符，也沒有欄位完全相符。

此範例會建立機器學習轉換。如需有關建立機器學習轉換所需參數的詳細資訊，請參閱 使用 AWS Lake Formation FindMatches 比對記錄。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a machine learning transform
#
# Resources section defines metadata for the machine learning transform
Resources:
  MyMLTransform:
    Type: "AWS::Glue::MLTransform"
    Condition: "isGlueMLGARegion"
    Properties:
      Name: !Sub "MyTransform"
      Description: "The bestest transform ever"
      Role: !ImportValue MyMLTransformUserRole
      GlueVersion: "1.0"
      WorkerType: "Standard"
      NumberOfWorkers: 5
      Timeout: 120
      MaxRetries: 1
      InputRecordTables:
        GlueTables:
          - DatabaseName: !ImportValue MyMLTransformDatabase
            TableName: !ImportValue MyMLTransformTable
      TransformParameters:
```

```
        TransformType: "FIND_MATCHES"
        FindMatchesParameters:
          PrimaryKeyColumnName: "testcolumn"
          PrecisionRecallTradeoff: 0.5
          AccuracyCostTradeoff: 0.5
          EnforceProvidedLabels: True
      Tags:
        key1: "value1"
        key2: "value2"
      TransformEncryption:
        TaskRunSecurityConfigurationName: !ImportValue
 MyMLTransformSecurityConfiguration
        MLUserDataEncryption:
          MLUserDataEncryptionMode: "SSE-KMS"
          KmsKeyId: !ImportValue MyMLTransformEncryptionKey
```

AWS Glue Data Quality 規則集的 AWS CloudFormation 範本範例

AWS Glue Data Quality 規則集包含可在資料型錄中的資料表上評估的規則。將規則集放置在目標資料表上後，您便可以進入資料型錄並執行評估，該評估會根據規則集中的這些規則執行資料。從評估資料列計數到評估資料的參照完整性，這些規則可能有所不同。

下列範例是 CloudFormation 範本，可在指定的目標資料表上建立包含各種規則的規則集。

```
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a DataQualityRuleset
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

  # The name of the ruleset to be created
  RulesetName:
    Type: String
    Default: "CFNRulesetName"
  RulesetDescription:
    Type: String
    Default: "CFN DataQualityRuleset"
  # Rules that will be associated with this ruleset
  Rules:
    Type: String
    Default: 'Rules = [
```

```
        RowCount > 100,
        IsUnique "id",
        IsComplete "nametype"
        ]'
  # Name of database and table within Data Catalog which the ruleset will
  # be applied too
  DatabaseName:
    Type: String
    Default: "ExampleDatabaseName"
  TableName:
    Type: String
    Default: "ExampleTableName"

# Resources section defines metadata for the Data Catalog
Resources:
  # Creates a Data Quality ruleset under specified rules
  DQRuleset:
    Type: AWS::Glue::DataQualityRuleset
    Properties:
      Name: !Ref RulesetName
      Description: !Ref RulesetDescription
      # The String within rules must be formatted in DQDL, a language
      # used specifically to make rules
      Ruleset: !Ref Rules
      # The targeted table must exist within Data Catalog alongside
      # the correct database
      TargetTable:
        DatabaseName: !Ref DatabaseName
        TableName: !Ref TableName
```

使用 EventBridge 排程器的 AWS Glue Data Quality 規則集的 AWS CloudFormation 範本範例

AWS Glue Data Quality 規則集包含可在資料型錄中的資料表上評估的規則。將規則集放置在目標資料表上後，您便可以進入資料型錄並執行評估，該評估會根據規則集中的這些規則執行資料。您也可以在 CloudFormation 範本中新增 EventBridge 排程器，以便在定時間隔為您排程這些規則集評估，而不必手動進入資料型錄來評估規則集。

下列範例是 CloudFormation 範本，可建立資料品質規則集和 EventBridge 排程器，每五分鐘即評估上述規則集一次。

```
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a DataQualityRuleset
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

  # The name of the ruleset to be created
  RulesetName:
    Type: String
    Default: "CFNRulesetName"
  # Rules that will be associated with this Ruleset
  Rules:
    Type: String
    Default: 'Rules = [
        RowCount > 100,
        IsUnique "id",
        IsComplete "nametype"
        ]'
  # The name of the Schedule to be created
  ScheduleName:
    Type: String
    Default: "ScheduleDQRulsetEvaluation"
  # This expression determines the rate at which the Schedule will evaluate
  # your data using the above ruleset
  ScheduleRate:
    Type: String
    Default: "rate(5 minutes)"
  # The Request that being sent must match the details of the Data Quality Ruleset
  ScheduleRequest:
    Type: String
    Default: '
        { "DataSource": { "GlueTable": { "DatabaseName": "ExampleDatabaseName",
         "TableName": "ExampleTableName" } },
         "Role": "role/AWSGlueServiceRoleDefault",
          "RulesetNames": [ ""CFNRulesetName"" ] }
        '

# Resources section defines metadata for the Data Catalog
Resources:
  # Creates a Data Quality ruleset under specified rules
  DQRuleset:
    Type: AWS::Glue::DataQualityRuleset
```

```
    Properties:
      Name: !Ref RulesetName
      Description: "CFN DataQualityRuleset"
      # The String within rules must be formatted in DQDL, a language
      # used specifically to make rules
      Ruleset: !Ref Rules
      # The targeted table must exist within Data Catalog alongside
      # the correct database
      TargetTable:
        DatabaseName: "ExampleDatabaseName"
        TableName: "ExampleTableName"
  # Create a Scheduler to schedule evaluation runs on the above ruleset
  ScheduleDQEval:
    Type: AWS::Scheduler::Schedule
    Properties:
      Name: !Ref ScheduleName
      Description: "Schedule DataQualityRuleset Evaluations"
      FlexibleTimeWindow:
        Mode: "OFF"
      ScheduleExpression: !Ref ScheduleRate
      ScheduleExpressionTimezone: "America/New_York"
      State: "ENABLED"
      Target:
        # The ARN is the API that will be run, since we want to evaluate our ruleset
        # we want this specific ARN
        Arn: "arn:aws:scheduler:::aws-sdk:glue:startDataQualityRulesetEvaluationRun"
        # Your RoleArn must have approval to schedule
        RoleArn: "arn:aws:iam::123456789012:role/AWSGlueServiceRoleDefault"
        # This is the Request that is being sent to the Arn
        Input: '
        { "DataSource": { "GlueTable": { "DatabaseName": "sampledb", "TableName":
"meteorite" } },
          "Role": "role/AWSGlueServiceRoleDefault",
           "RulesetNames": [ "TestCFN" ] }
        '
```

AWS Glue 開發端點的範例 AWS CloudFormation 範本

AWS Glue 開發端點是一種環境，可讓您用於開發並測試 AWS Glue 指令碼。

次範例會建立一個開發端點，僅使用可成功建立端點的最低限量參數值。如需開發端點設定所需的參數的詳細資訊，請參閱 專為 AWS Glue 的開發設定聯網。

您要提供現有的 IAM 角色 ARN (Amazon Resource Name) 以建立開發端點。若打算在開發端點上建立筆記型電腦伺服器，請提供有效的 RSA 公有金鑰，並將對應的私有金鑰保持在可用狀態。

> **Note**
>
> 只要是您建立並與開發端點關聯的筆記本伺服器，您就可以管理。因此，如果您刪除開發端點之後要刪除筆記本伺服器，就必須在 AWS CloudFormation 主控台刪除 AWS CloudFormation 堆疊。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a development endpoint
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the crawler to be created
  CFNEndpointName:
    Type: String
    Default: cfn-devendpoint-1
  CFNIAMRoleArn:
    Type: String
    Default: arn:aws:iam::123456789012/role/AWSGlueServiceRoleGA
#
#
# Resources section defines metadata for the Data Catalog
Resources:
  CFNDevEndpoint:
    Type: AWS::Glue::DevEndpoint
    Properties:
      EndpointName: !Ref CFNEndpointName
      #ExtraJarsS3Path: String
      #ExtraPythonLibsS3Path: String
      NumberOfNodes: 5
      PublicKey: ssh-rsa public.....key myuserid-key
      RoleArn: !Ref CFNIAMRoleArn
      SecurityGroupIds:
        - sg-64986c0b
```

```
      SubnetId: subnet-c67cccac
```

AWS Glue 編程指南

腳本包含從源中提取數據，對其進行轉換並將其加載到目標的代碼。 AWS Glue 啟動工作時執行指令碼。

AWS Glue ETL 指令碼可以在 Python 或 Scala 中進行程式碼撰寫。雖然所有任務類型皆可使用 Python 來編寫，您亦可使用 Scala 來撰寫 AWS Glue for Spark 任務。當您在中自動產生工作的原始程式碼邏輯時 AWS Glue Studio，會建立指令碼。您可以編輯這個指令碼，也可以提供自己的指令碼來處理您的 ETL 任務。

提供您的自訂指令碼

指令碼會在 AWS Glue中執行擷取、轉換和載入 (ETL) 工作。當您為任務自動產生原始碼邏輯，指令碼也會隨之建立。您可以編輯產生的指令碼，或是提供自訂的指令碼。

若要在 AWS Glue 中提供自訂的指令碼，請遵循下述一般步驟：

1. 請登入 AWS Management Console 並開啟AWS Glue主控台，網址為 https://console.aws.amazon.com/glue/。
2. 選擇 ETL 任務索引標籤，然後檢視建立任務區段。選擇指令碼編輯器選項。
3. 在 This job runs (此任務執行) 下，選擇以下其中一項：
   - 使用樣板程式碼建立新的指令碼
   - 上傳和編輯現有指令碼
4. 在任務詳細資訊頁面上，選擇執行自訂指令碼所需的 IAM 角色。如需詳細資訊，請參閱 AWS Glue 的身分識別與存取管理。
5. 選擇指令碼參考的任何連線。欲連線至必要的 JDBC 資料存放區，就需要這些物件。

   彈性網路界面是一種虛擬的網路介面，而您可在 Virtual Private Cloud (VPC) 中將其連接至執行個體。選擇連接至指令碼使用之資料存放區所需的彈性網路界面。
6. 提供任務類型特定的其他組態，包括參數。如需有關任務類型組態的詳細資訊，請參閱 使用 AWS Glue Studio 建立視覺化 ETL 任務 一節。
7. 在指令碼索引標籤上，貼上或撰寫自訂指令碼。

請使用本節中的內容，進行自訂指令碼的撰寫程序。

如需在 AWS Glue 新增任務的詳細資訊，請參閱 使用 AWS Glue Studio 建立視覺化 ETL 任務。

如需 step-by-step 指引，請參閱主AWS Glue控台中的新增工作教學課程。

Spark 指令碼程式設計

除了測試和執行之外，AWS Glue 讓寫入或自動產生擷取、轉換與載入 (ETL) 指令碼都變得輕鬆容易。本節說明 AWS Glue 推出的 Apache Spark 擴充功能，並提供範例，來示範以 Python 和 Scala 撰寫程式碼和執行 ETL 指令碼的方法。

> **Important**
>
> 不同版本的 AWS Glue 所支援的 Apache Spark 版本並不相同。您的自訂指令碼必須與支援的 Apache Spark 版本相容。如需 AWS Glue 版本的相關資訊，請參閱 Glue version job property。

主題

教學課程：撰寫 Spark 指令碼的 AWS Glue

本教學將向您介紹撰寫 AWS Glue 指令碼的程序。您可以按照任務的排程執行指令碼，也可以與互動式工作階段互動執行。如需任務的詳細資訊，請參閱使用 AWS Glue Studio 建立視覺化 ETL 任務。如需互動式工作階段的詳細資訊，請參閱the section called “AWS Glue 互動式工作階段概觀”。

AWS Glue Studio 視覺化編輯器提供無程式碼的圖形介面，可用於建置 AWS Glue 工作。 AWS Glue 腳本回視覺作業。您可透過這些指令碼存取一組經過擴增的工具，這些工具能與 Apache Spark 程式搭配運作。您可以存取原生 Spark API，以及 AWS Glue 程式庫，以便從 G AWS lue 指令碼中擷取、轉換和載入 (ETL) 工作流程。

本教學課程中，您會擷取、轉換及載入違停罰單資料集。執行這項工作的指令碼在形式和功能上與 AWS 大數據部落格上使用 AWS Glue Studio 讓 ETL 更容易產生的指令碼相同，後者介紹了 AWS Glue Studio 視覺化編輯器。透過在工作中執行此指令碼，您可以將其與視覺化工作進行比較，並查看 AWS Glue ETL 指令碼的運作方式。如此一來，您就可以做好準備，以便日後使用目前視覺化任務尚未提供的其他功能。

在本教學課程中，您將使用 Python 語言和程式庫。類似的功能在 Scala 中可用。在完成本教程之後，您應該能夠生成並檢查 Scala 示例腳本，以了解如何執行 Scala AWS Glue ETL 腳本編寫過程。

必要條件

本教學課程具備下列先決條件：

- 與 AWS Glue Studio 部落格文章指示您執行 AWS CloudFormation 範本的必要條件相同。

  此範本使用 AWS Glue 資料目錄來管理中可用的停車票資料集`s3://aws-bigdata-blog/artifacts/gluestudio/`。它建立將被引用以下資源：
- AWS Glue StudioRole：要執行 AWS Glue 任務的 IAM 角色
- AWS Glue StudioAmazon S3Bucket：儲存部落格相關檔案的 Amazon S3 儲存貯體名稱
- AWS Glue StudioTicketsYYZDB：AWS Glue Data Catalog 資料庫
- AWS Glue StudioTableTickets— 用作來源的資料目錄表
- AWS Glue StudioTableTrials— 用作來源的資料目錄表
- AWS Glue StudioParkingTicketCount — 要用作目標的資料目錄表
- 在 AWS Glue 工作室部落格文章中產生的指令碼。如果部落格文章變更，指令碼在下列文字中同樣可用。

產生範例指令碼

您可以使用 AWS Glue Studio 視覺化編輯器做為強大的程式碼產生工具，為您要撰寫的指令碼建立支架。您將使用此工具建立範例指令碼。

如果您想跳過這些步驟，系統會提供指令碼。

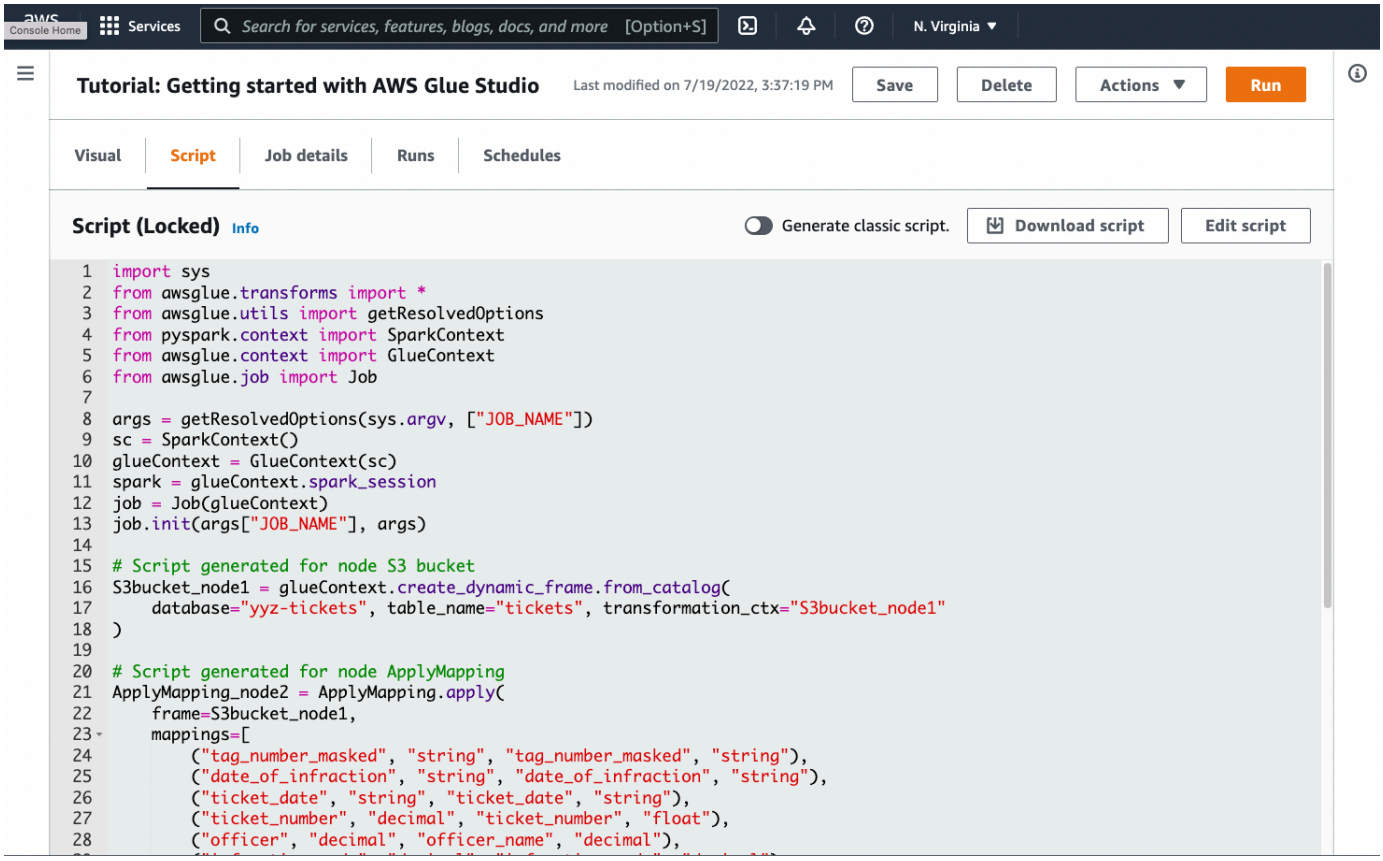
教學課程範例指令碼

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
```

```
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node S3 bucket
S3bucket_node1 = glueContext.create_dynamic_frame.from_catalog(
    database="yyz-tickets", table_name="tickets", transformation_ctx="S3bucket_node1"
)

# Script generated for node ApplyMapping
ApplyMapping_node2 = ApplyMapping.apply(
    frame=S3bucket_node1,
    mappings=[
        ("tag_number_masked", "string", "tag_number_masked", "string"),
        ("date_of_infraction", "string", "date_of_infraction", "string"),
        ("ticket_date", "string", "ticket_date", "string"),
        ("ticket_number", "decimal", "ticket_number", "float"),
        ("officer", "decimal", "officer_name", "decimal"),
        ("infraction_code", "decimal", "infraction_code", "decimal"),
        ("infraction_description", "string", "infraction_description", "string"),
        ("set_fine_amount", "decimal", "set_fine_amount", "float"),
        ("time_of_infraction", "decimal", "time_of_infraction", "decimal"),
    ],
    transformation_ctx="ApplyMapping_node2",
)

# Script generated for node S3 bucket
S3bucket_node3 = glueContext.write_dynamic_frame.from_options(
    frame=ApplyMapping_node2,
    connection_type="s3",
    format="glueparquet",
    connection_options={"path": "s3://DOC-EXAMPLE-BUCKET", "partitionKeys": []},
    format_options={"compression": "gzip"},
    transformation_ctx="S3bucket_node3",
)

job.commit()
```

產生範例指令碼

1. 完成 AWS Glue 工作室教學課程。若要完成此教學課程，請參閱從範例工作在 AWS Glue Studio 中建立工作。
2. 導覽至任務頁面上的 Script (指令碼) 索引標籤，如以下螢幕擷取畫面所示：



3. 複製 Script (指令碼) 索引標籤中的完整內容。透過在 Job details (任務詳細資訊) 中設定指令碼語言，您可以在產生 Python 程式碼或 Scala 程式碼之間來回切換。

步驟 1. 建立任務並貼上指令碼

在此步驟中，您要在中建立 AWS Glue 工作 AWS Management Console。這會設定允許 AWS Glue 執行指令碼的組態。此操作還會為您建立一個存放和編輯指令碼的地方。

建立任務

1. 在中 AWS Management Console，導覽至 AWS Glue 登陸頁面。
2. 在側邊的導覽窗格中，選擇 Jobs (任務)。
3. 在 Create job (建立任務) 中選擇 Spark script editor (Spark 指令碼編輯器)，接著選擇 Create (建立)。
4. 選用 – 將指令碼的完整文字貼入 Script (指令碼) 窗格中。或者，您可以按照教學課程進行操作。

步驟 2. 匯入 AWS Glue 程式庫

您需要設定指令碼，使其與在指令碼外部定義的程式碼和組態互動。這項工作是在 AWS Glue 工作室幕後完成的。

在此步驟中，您會執行下列動作。

- 匯入並初始化 `GlueContext` 物件。從指令碼編寫的角度來看，這是最重要的匯入。這會公開定義來源和目標資料集所使用的標準方法，也就是任何 ETL 指令碼的起點。若要進一步了解 `GlueContext` 類別，請參閱 GlueContext 類。
- 初始化 `SparkContext` 和 `SparkSession`。這些可讓您配置 AWS Glue 工作內可用的 Spark 引擎。您不需要直接在介紹性 AWS Glue 指令碼中使用它們。
- 呼叫 `getResolvedOptions`，準備您的任務參數以在指令碼內使用。如需有關解析任務參數的詳細資訊，請參閱 the section called “getResolvedOptions”。
- 初始化 `Job`。該`Job`對象設置配置並跟踪各種可選 AWS Glue 功能的狀態。您的指令碼可以在沒有 `Job` 物件的情況下執行，但最佳實務是將其初始化，以免之後整合這些功能時造成混淆。

  其中一項功能是任務書籤，您可以在此教學課程中選擇性地設定該功能。您可以在以下章節中了解有關任務書籤的資訊：the section called “選用 – 啟用任務書籤”。

在此程序中，您需編寫下列程式碼。此程式碼是產生的範例指令碼的一部分。

```
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)
```

若要匯入 AWS Glue 程式庫

- 複製此段程式碼並貼入 Script (指令碼) 編輯器中。

> **Note**
>
> 您或許會認為，複製程式碼是一種不好的工程實務。在本教學課程中，我們建議您在所有 AWS Glue ETL 指令碼中持續命名核心變數。

步驟 3。從來源擷取資料

在任何 ETL 程序中，您都需要先定義要變更的來源資料集。在 AWS Glue Studio 視覺化編輯器中，您可以透過建立來源節點來提供此資訊。

在此步驟中，您需為 `create_dynamic_frame.from_catalog` 方法提供 `database` 和 `table_name`，以便從 AWS Glue 資料目錄所設定的來源擷取資料。

在先前的步驟中，您已初始化 `GlueContext` 物件。您可以使用此物件來尋找設定來源所需使用的方法，例如 `create_dynamic_frame.from_catalog`。

在此程序中，您需使用 `create_dynamic_frame.from_catalog` 編寫下列程式碼。此程式碼是產生的範例指令碼的一部分。

```
S3bucket_node1 = glueContext.create_dynamic_frame.from_catalog(
    database="yyz-tickets", table_name="tickets", transformation_ctx="S3bucket_node1"
    )
```

從來源擷取資料

1. 檢查文件以尋找從 AWS Glue 資料型錄中定義的來源擷取資料的方法。`GlueContext`這些方法記錄於 the section called “GlueContext”。選擇 create_dynamic_frame.from_catalog 方法。在 `glueContext` 上呼叫此方法。
2. 檢查 `create_dynamic_frame.from_catalog` 的文件。此方法需要 `database` 和 `table_name` 參數。為 `create_dynamic_frame.from_catalog` 提供必要的參數。

   AWS Glue 資料型錄會儲存來源資料位置和格式的相關資訊，並在必要條件區段中進行設定。您不必直接為指令碼提供該資訊。
3. 選用 – 為該方法提供 `transformation_ctx` 參數，以支援任務書籤。您可以在以下章節中了解有關任務書籤的資訊：the section called “選用 – 啟用任務書籤”。

> **Note**
>
> 擷取資料的常用方法
> the section called “create_dynamic_frame_from_catalog”用於連線至 AWS Glue 資料型錄中的表格。
> 如果您需要直接為任務提供描述來源結構和位置的組態，請參閱 the section called “create_dynamic_frame_from_options” 方法。您需要提供比使用 `create_dynamic_frame.from_catalog` 時更詳細的參數以描述您的資料。
> 請參閱有關 `format_options` 和 `connection_parameters` 的補充文件來識別所需參數。有關如何提供關於來源資料格式的指令碼資訊的說明，請參閱 the section called “資料格式選項”。有關如何提供關於來源資料位置的指令碼資訊的說明，請參閱 the section called “連線參數”。
> 如果您要從串流來源讀取資訊，您可以透過 the section called “create_data_frame_from_catalog” 或 the section called “create_data_frame_from_options” 方法為任務提供來源資訊。請注意，這些方法會傳回 Apache Spark `DataFrames`。
> 儘管參考文件參考的是 `create_dynamic_frame_from_catalog`，但產生的程式碼會呼叫 `create_dynamic_frame.from_catalog`。這些方法最終會呼叫相同的程式碼，並包含在內，以便您可以編寫更簡潔的程式碼。您可以檢視 Python 包裝函式的來源，藉以確認這一點，該包裝函式可在 aws-glue-libs 找到。

步驟 4. 使用 AWS Glue 來轉換資料

在 ETL 程序中擷取來源資料後，您需要說明要如何變更資料。您可以在 AWS Glue Studio 視覺化編輯器中建立轉換節點來提供此資訊。

在此步驟中，您需為 `ApplyMapping` 方法提供目前所需欄位名稱和類型的映射，以轉換您的 `DynamicFrame`。

執行以下轉換。

- 捨棄這四個 `location` 和 `province` 索引鍵。
- 將 `officer` 的名稱變更為 `officer_name`。
- 將 `ticket_number` 和 `set_fine_amount` 的類型變更為 `float`。

`create_dynamic_frame.from_catalog` 將為您提供 `DynamicFrame` 物件。A `DynamicFrame` 代表 AWS Glue 中的資料集。 AWS Glue 轉換是改變的操作`DynamicFrames`。

> **Note**
>
> 什麼是 DynamicFrame？
> DynamicFrame 是一種抽象，允許您連接資料集，該資料集中包含資料中項目的名稱和類型的描述。在阿帕奇星火，類似的抽象存在稱為 DataFrame. 如需的說明 DataFrames，請參閱星火 SQL 指南。
> DynamicFrames 可讓您動態描述資料集結構描述。考慮一個帶有價格列的數據集，其中一些條目將價格存儲為字符串，而其他條目則將價格存儲為雙精度。 AWS Glue 會計算結構描述 on-the-fly — 它會為每一列建立自我描述記錄。
> 不一致的欄位 (如價格) 在框架結構描述中用類型 (ChoiceType) 明確表示。您可以使用 DropFields 捨棄不一致的欄位，或使用 ResolveChoice 來解析欄位，藉以處理欄位不一致的問題。這些轉換可從 DynamicFrame 取得。然後，您可以使用 writeDynamicFrame 將資料寫回資料湖。
> 您可以從 DynamicFrame 類別上的方法呼叫許多相同的轉換，這樣可以得到更具可讀性的指令碼。如需 DynamicFrame 的相關資訊，請參閱 the section called “DynamicFrame”。

在此程序中，您需使用 ApplyMapping 編寫下列程式碼。此程式碼是產生的範例指令碼的一部分。

```
ApplyMapping_node2 = ApplyMapping.apply(
    frame=S3bucket_node1,
    mappings=[
        ("tag_number_masked", "string", "tag_number_masked", "string"),
        ("date_of_infraction", "string", "date_of_infraction", "string"),
        ("ticket_date", "string", "ticket_date", "string"),
        ("ticket_number", "decimal", "ticket_number", "float"),
        ("officer", "decimal", "officer_name", "decimal"),
        ("infraction_code", "decimal", "infraction_code", "decimal"),
        ("infraction_description", "string", "infraction_description", "string"),
        ("set_fine_amount", "decimal", "set_fine_amount", "float"),
        ("time_of_infraction", "decimal", "time_of_infraction", "decimal"),
    ],
    transformation_ctx="ApplyMapping_node2",
)
```

若要使用 AWS Glue 轉換資料

1. 檢查文件以識別要變更和捨棄欄位的轉換。如需詳細資訊，請參閱 the section called “GlueTransform”。選擇 ApplyMapping 轉換。如需 ApplyMapping 的相關資訊，請參閱 the section called “ApplyMapping”。在 ApplyMapping 轉換物件上呼叫 apply。

   > **Note**
   >
   > 什麼是 ApplyMapping？
   > ApplyMapping 採用 DynamicFrame 並對其進行轉換。這需使用代表欄位轉換的元組清單，亦即「映射」。前兩個元組元素 (欄位名稱和類型) 用於識別框架中的欄位。第二對參數同樣是欄位名稱和類型。
   > ApplyMapping 將源字段轉換為目標名稱，然後鍵入一個新的DynamicFrame，它返回。未提供的欄位會在傳回值中遭到捨棄。
   > 您可以呼叫與 DynamicFrame 上的 apply_mapping 方法相同的轉換 (而不必呼叫 apply) 來建立更流暢、更易讀的程式碼。如需詳細資訊，請參閱 the section called “apply_mapping”。

2. 檢查 ApplyMapping 的文件以識別所需參數。請參閱the section called “ApplyMapping”。您會發現此方法需要 frame 和 mappings 參數。為 ApplyMapping 提供必要的參數。

3. 選用 – 為該方法提供 transformation_ctx，以支援任務書籤。您可以在以下章節中了解有關任務書籤的資訊：the section called “選用 – 啟用任務書籤”。

> **Note**
>
> Apache Spark 功能
> 我們提供轉換以簡化您任務中的 ETL 工作流程。您還可以存取任務中 Spark 程式所提供的程式庫，這些程式庫專為更廣泛的用途而建立。如要使用這些程式庫，您需在 DynamicFrame 和 DataFrame 之間轉換。
> 您可以使用 the section called “toDF” 建立 DataFrame。然後，您可以使用上的可用方法 DataFrame 來轉換資料集。如需這些方法的詳細資訊，請參閱DataFrame。然後，您可以使用the section called “fromDF”向後轉換，以使用 AWS Glue 操作將幀加載到目標。

步驟 5. 將資料載入目標

轉換資料之後，通常需將轉換後的資料存放在與來源不同的地方。您可以在 AWS Glue Studio 視覺化編輯器中建立目標節點來執行此作業。

在此步驟中，您需為 `write_dynamic_frame.from_options` 方法提供 `connection_type`、`connection_options`、`format` 和 `format_options`，將資料載入 Amazon S3 中的目標儲存貯體。

在步驟 1 中，您初始化了 `GlueContext` 物件。在 AWS Glue 中，您可以在這裡找到用來設定目標的方法，就像來源一樣。

在此程序中，您需使用 `write_dynamic_frame.from_options` 編寫下列程式碼。此程式碼是產生的範例指令碼的一部分。

```
S3bucket_node3 = glueContext.write_dynamic_frame.from_options(
    frame=ApplyMapping_node2,
    connection_type="s3",
    format="glueparquet",
    connection_options={"path": "s3://DOC-EXAMPLE-BUCKET", "partitionKeys": []},
    format_options={"compression": "gzip"},
    transformation_ctx="S3bucket_node3",
    )
```

將資料載入目標

1. 檢查文件，查找將資料載入目標 Amazon S3 儲存貯體的方法。這些方法記錄於 the section called “GlueContext”。選擇 the section called “write_dynamic_frame_from_options” 方法。在 `glueContext` 上呼叫此方法。

   > **Note**
   >
   > 常用資料載入方法 `write_dynamic_frame.from_options` 是最常用的資料載入方法，它支持 AWS Glue 中可用的所有目標。
   > 如果您要寫入 AWS Glue 連線中定義的 JDBC 目標，請使用此方the section called “write_dynamic_frame_from_jdbc_conf”法。 AWS Glue 連線會儲存如何連線至資料來源的相關資訊。如此一來，您就不需在 `connection_options` 提供該資訊，不過您仍然需要使用 `connection_options` 來提供 `dbtable`。

write_dynamic_frame.from_catalog 不是常用的資料載入方法。此方法會在不更新基礎資料集的情況下更新 AWS Glue 資料型錄，並與變更基礎資料集的其他程序一起使用。如需詳細資訊，請參閱 the section called “更新結構描述並新增分割區”。

2. 檢查 the section called “write_dynamic_frame_from_options” 的文件。此方法需要 frame、connection_type、format、connection_options 和 format_options。在 glueContext 上呼叫此方法。

   a. 請參閱有關 format_options 和 format 的補充文件以識別您需要的參數。如需資料格式的說明，請參閱 the section called “資料格式選項”。

   b. 請參閱有關 connection_type 和 connection_options 的補充文件以識別您需要的參數。如需連線的說明，請參閱 the section called “連線參數”。

   c. 為 write_dynamic_frame.from_options 提供必要的參數。此方法的組態與 create_dynamic_frame.from_options 類似。

3. 選用 – 向 write_dynamic_frame.from_options 提供 transformation_ctx，以支援任務書籤。您可以在以下章節中了解有關任務書籤的資訊：the section called “選用 – 啟用任務書籤”。

步驟 6. 遞交 **Job** 物件

您在步驟 1 中初始化了 Job 物件。您需要在指令碼結尾手動斷定其生命週期，某些選用功能需要此操作才能正常執行。這項工作是在 AWS Glue 工作室幕後完成的。

在此步驟中，在 Job 物件上呼叫 commit 方法。

在此程序中，您需編寫下列程式碼。此程式碼是產生的範例指令碼的一部分。

```
job.commit()
```

遞交 **Job** 物件

1. 如果您尚未執行此操作，請執行先前章節中概述的選用步驟，將 transformation_ctx 納入其中。
2. 呼叫 commit。

選用 – 啟用任務書籤

之前的每個步驟都已經指示您設定 transformation_ctx 參數。這與名為任務書籤的功能有關。

藉由任務書籤，您可針對資料集定期執行任務，達到節省時間和金錢的目的，同時還能輕鬆追蹤先前的工作。Job 書籤會追蹤先前執行資料集中 AWS Glue 轉換的進度。通過跟踪先前運行結束的位置，AWS Glue 可以將其工作限制為以前未處理的行。如需任務書籤的詳細資訊，請參閱 the section called “使用任務書籤追蹤處理的資料”。

若要啟用任務書籤，請先新增 `transformation_ctx` 陳述式至我們所提供的函數，如之前的範例所述。任務書籤狀態會在執行期間維持不變，而 `transformation_ctx` 參數是存取該狀態所需的索引鍵。這些陳述式自身不會執行任何操作。您還需要在任務的組態中啟用該功能。

在此程序中，您會使用 AWS Management Console啟用任務書籤。

啟用任務書籤

1. 導覽至相應任務的 Job details (任務詳細資訊) 一節。
2. 將 Job bookmark (任務書籤) 設定為 Enable (啟用)。

步驟 7. 將程式碼作為任務執行

在此步驟中，您需執行任務以確認您是否已成功完成本教學課程。這是通過點擊一個按鈕來完成的，就像在 AWS Glue Studio 可視化編輯器中一樣。

將程式碼作為任務執行

1. 在標題列上選擇 Untitled job (未命名任務) 以編輯和設定您的任務名稱。
2. 導覽至 Job details (任務詳細資訊) 索引標籤。為您的任務指派 IAM Role (IAM 角色)。您可以在 AWS Glue Studio 教學課程的先決條件中使用 AWS CloudFormation 範本所建立的範本所建立的項目。如果您已完成該教學課程，則該角色應可用，名為 `AWS Glue StudioRole`。
3. 選擇 Save (儲存) 以儲存您的指令碼。
4. 選擇 Run (執行) 以執行您的任務。
5. 導覽至 Runs (執行) 索引標籤，以確認您的任務已完成。
6. 導覽至 *`DOC-EXAMPLE-BUCKET`*，即 `write_dynamic_frame.from_options` 的目標。確認輸出符合您的期望。

如需有關設定和管理任務的詳細資訊，請參閱 the section called “提供您的自訂指令碼”。

其他資訊

阿帕奇星火庫和方法在 AWS Glue 腳本中可用。您可以查看 Spark 文件以了解您可以使用這些包含的程式庫來做什麼。如需詳細資訊，請參閱 Spark 來源儲存庫的範例區段。

AWS Glue 2.0 + 默認情況下包括幾個常見的 Python 庫。在 Scala 或 Python 環境中，還有一些機制可以將自己的依賴項加載到 AWS Glue 作業中。如需 Python 相依性的相關資訊，請參閱 the section called “Python 程式庫”。

有關如何在 Python 中使用 AWS Glue 功能的更多示例，請參閱the section called “Python 範例”。Scala 和 Python 任務具有相同的功能，所以我們的 Python 範例應能為您提供以 Scala 執行類似工作的些許想法。

程式 AWS Glue ETL 指令碼 PySpark

您可以在 GitHub 網站的範例儲存庫AWS Glue中找到的 Python 程式碼範例和公用程式。AWS Glue

在 AWS Glue 使用 Python

AWS Glue 支援 PySpark Python 方言的延伸，以編寫擷取、轉換和載入 (ETL) 工作的指令碼。本節說明如何以 ETL 指令碼和 AWS Glue API 使用 Python。

- 設定以 AWS Glue 使用 Python
- 在 Python 中呼叫 AWS Glue API
- 以 AWS Glue 使用 Python 程式庫
- AWS Glue Python 程式碼範例

AWS Glue PySpark 副檔名

AWS Glue已經創建了以下擴展到 PySpark Python 方言。

- 使用 `getResolvedOptions` 存取參數
- PySpark 延伸模組類型
- DynamicFrame 類
- DynamicFrameCollection 類別
- DynamicFrameWriter 類別

- DynamicFrameReader 類
- GlueContext 類

AWS Glue PySpark 轉換

AWS Glue已經創建了以下轉換類在 PySpark ETL 操作中使用。

- GlueTransform base 類別
- ApplyMapping 類別
- DropFields 類別
- DropNullFields 類別
- ErrorsAsDynamicFrame 類別
- FillMissingValues 類別
- Filter 類別
- FindIncrementalMatches 類別
- FindMatches 類別
- FlatMap 類別
- Join 類別
- Map 類別
- MapToCollection 類別
- mergeDynamicFrame
- Relationalize 類別
- RenameField 類別
- ResolveChoice 類別
- SelectFields 類別
- SelectFromCollection 類別
- Spigot 類別
- SplitFields 類別
- SplitRows 類別
- Unbox 類別

- UnnestFrame 類別

設定以 AWS Glue 使用 Python

使用 Python 開發 Spark 任務的 ETL 指令碼。ETL 任務支援的 Python 版本取決於任務的 AWS Glue 版本。如需有關 AWS Glue 版本的詳細資訊，請參閱 Glue version job property。

設定您的系統以 AWS Glue 使用 Python

依照以下步驟安裝 Python 以及能夠呼叫 AWS Glue API。

1. 如果您尚未安裝 Python，請至 Python.org 下載頁面下載及安裝。
2. 依照 AWS CLI 文件所述的方式，安裝 AWS Command Line Interface (AWS CLI)。

   使用 Python 並不一定需要 AWS CLI。但是，對於使用您的帳戶登入資料來設定並驗證 AWS 是否正常運作而言，安裝並設定它是很便利的方法。
3. 依照 Boto3 快速入門所述的方式，安裝適用於 Python 的 AWS SDK (Boto 3)。

   AWS Glue 尚未提供 Boto 3 資源 API。目前，只能使用 Boto 3 用戶端 API。

   如需有關 Boto 3 的詳細資訊，請參閱AWSSDK for Python (Boto3) 入門。

您可以在 GitHub 網站上的 AWS Glue 範例儲存庫中，找到適用於 AWS Glue 的 Python 程式碼範例與公用程式。

在 Python 中呼叫 AWS Glue API

請注意，AWS Glue 尚未提供 Boto 3 資源 API。目前，只能使用 Boto 3 用戶端 API。

Python 中的 AWS Glue API 名稱

在 Java 及其他程式設計語言中，AWS Glue API 的名稱通常是 CamelCased。但是，從 Python 呼叫時，這些一般名稱會變更為小寫字母，並以底線字元區隔名稱的部分，使其更為「Pythonic」。在 AWS Glue API 參考文件中，這些 Pythonic 名稱會列在一般 CamelCased 名稱後面的刮號中。

但是，雖然 AWS Glue API 名稱本身會轉換為小寫字母，但其參數名稱仍維持大寫字母。請務必記住這一點，因為如下所述，在呼叫 AWS Glue API 時，應以名稱傳遞參數。

在 AWS Glue 中傳遞和存取 Python 參數

在 Python 中呼叫 AWS Glue API，最好以名稱明確傳遞參數。例如：

```
job = glue.create_job(Name='sample', Role='Glue_DefaultRole',
                      Command={'Name': 'glueetl',
                               'ScriptLocation': 's3://my_script_bucket/scripts/
my_etl_script.py'})
```

這可協助您了解 Python 建立字典的名稱/值元組，讓您在 Job 結構 或 JobRun 結構 中將其指定為 ETL 指令碼的引數。Boto 3 會透過 REST API 呼叫，以 JSON 格式將它們傳送到 AWS Glue。這表示您在指令碼中存取這些引數時，無法倚賴引數的排序。

例如，假設您在 Python Lambda 處理常式函式中起始 JobRun，而且要指定幾個參數。您的程式碼看起來類似如下：

```
from datetime import datetime, timedelta

client = boto3.client('glue')

def lambda_handler(event, context):
  last_hour_date_time = datetime.now() - timedelta(hours = 1)
  day_partition_value = last_hour_date_time.strftime("%Y-%m-%d")
  hour_partition_value = last_hour_date_time.strftime("%-H")

  response = client.start_job_run(
               JobName = 'my_test_Job',
               Arguments = {
                 '--day_partition_key':   'partition_0',
                 '--hour_partition_key':  'partition_1',
                 '--day_partition_value':  day_partition_value,
                 '--hour_partition_value': hour_partition_value } )
```

若要在您的 ETL 指令碼中可靠地存取這些參數，請使用 AWS Glue 的 `getResolvedOptions` 指定其名稱，然後從產生的字典存取這些參數：

```
import sys
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv,
                          ['JOB_NAME',
                           'day_partition_key',
                           'hour_partition_key',
                           'day_partition_value',
                           'hour_partition_value'])
```

```
print "The day partition key is: ", args['day_partition_key']
print "and the day partition value is: ", args['day_partition_value']
```

如果想傳遞一個巢狀 JSON 字串的引數，以在將參數值傳遞給 AWS Glue ETL 任務時保留參數值，則您必須在開始任務執行之前對參數字串進行編碼，然後在任務指令碼參考其之前對參數字串進行解碼。例如，請試想有下列引數字串：

```
glue_client.start_job_run(JobName = "gluejobname", Arguments={
"--my_curly_braces_string": '{"a": {"b": {"c": [{"d": {"e": 42}}]}}}'
})
```

若要正確傳遞此參數，您應該將引數編碼為 Base64 編碼的字串。

```
import base64
...
sample_string='{"a": {"b": {"c": [{"d": {"e": 42}}]}}}'
sample_string_bytes = sample_string.encode("ascii")

base64_bytes = base64.b64encode(sample_string_bytes)
base64_string = base64_bytes.decode("ascii")
...
glue_client.start_job_run(JobName = "gluejobname", Arguments={
"--my_curly_braces_string": base64_bytes})
...
sample_string_bytes = base64.b64decode(base64_bytes)
sample_string = sample_string_bytes.decode("ascii")
print(f"Decoded string: {sample_string}")
...
```

範例：建立和執行任務

以下範例顯示如何使用 Python 呼叫 AWS Glue API，以建立和執行 ETL 任務。

建立和執行任務

1. 建立 AWS Glue 用戶端執行個體：

   ```
   import boto3
   glue = boto3.client(service_name='glue', region_name='us-east-1',
                 endpoint_url='https://glue.us-east-1.amazonaws.com')
   ```

2. 建立任務。您必須使用 `glueetl` 做為 ETL 命令的名稱，如以下程式碼所示：

```
myJob = glue.create_job(Name='sample', Role='Glue_DefaultRole',
                          Command={'Name': 'glueetl',
                                   'ScriptLocation': 's3://my_script_bucket/
scripts/my_etl_script.py'})
```

3. 為您在上個步驟建立的任務啟動新的執行：

```
myNewJobRun = glue.start_job_run(JobName=myJob['Name'])
```

4. 取得任務狀態：

```
status = glue.get_job_run(JobName=myJob['Name'], RunId=myNewJobRun['JobRunId'])
```

5. 列印任務執行的目前狀態：

```
print(status['JobRun']['JobRunState'])
```

以 AWS Glue 使用 Python 程式庫

AWS Glue 可讓您安裝額外的 Python 模組和程式庫，以便搭配 AWS Glue ETL 使用。

主題

- 使用 pip 在 AWS Glue 2.0+ 中安裝其他 Python 模組
- 包括具有 PySpark 本地功能的 Python 文件
- 使用視覺轉換的程式設計腳本
- AWS Glue 中已提供 Python 模組
- 壓縮程式庫以加入
- 在 AWS Glue 工作室筆記本中加載 Python 庫
- 在開發端點載入 Python 程式庫
- 在工作中使用 Python 程式庫或 JobRun

使用 pip 在 AWS Glue 2.0+ 中安裝其他 Python 模組

AWS Glue 使用 Python Package Installer (pip3) 來安裝 AWS Glue ETL 使用的其他模組。您可以使用 `--additional-python-modules` 參數與逗號分隔的 Python 模組清單來新增新模組或變更現有

模組的版本。您可以將分佈上傳到 Amazon S3 來安裝程式庫的自訂發行版本，然後在模組清單中包含 Amazon S3 物件的路徑。

您可以使用 `--python-modules-installer-option` 參數將其他選項傳遞給 pip3。例如，您可以傳遞 `"--upgrade"` 來升級由 `"--additional-python-modules"` 指定的套件。如需更多範例，請參閱使用 AWS Glue 2.0 從輪子為 Spark ETL 工作負載建置 Python 模組。

如果您的 Python 相依性以傳遞方式取決於原生、已編譯的程式碼，您可能會執行下列限制： AWS Glue 不支援在工作環境中編譯原生程式碼。不過， AWS Glue 任務會在 Amazon Linux 2 環境中執行。透過 Wheel 分發套件，您也許能夠以編譯形式提供原生相依性。

例如，要更新或新增新的 `scikit-learn` 模組，請使用以下鍵/值：`"--additional-python-modules", "scikit-learn==0.21.3"`。

此外，在 `--additional-python-modules` 選項中，您可以指定 Python wheel 模組的 Amazon S3 路徑。例如：

```
--additional-python-modules s3://aws-glue-native-spark/tests/j4.2/ephem-3.7.7.1-cp37-cp37m-linux_x86_64.whl,s3://aws-glue-native-spark/tests/j4.2/fbprophet-0.6-py3-none-any.whl,scikit-learn==0.21.3
```

您可以`--additional-python-modules`在AWS Glue主控台的 [Job 參數] 欄位中指定，或透過變更 AWS SDK 中的工作引數來指定。如需有關設定任務參數的詳細資訊，請參閱 the section called “任務參數”。

包括具有 PySpark 本地功能的 Python 文件

AWS Glue用 PySpark 來在 AWS Glue ETL 工作中包含 Python 檔案。您需使用可用的 `--additional-python-modules` 管理相依性。您可以使用 `--extra-py-files` 任務參數來包含 Python 檔案。相依性必須在 Amazon S3 中託管，且引數值應為以逗號分隔的 Amazon S3 路徑清單，並不含空格。此功能的行為類似於您搭配 Spark 使用的 Python 相依性管理。如需 Spark 中 Python 相依性管理的詳細資訊，請參閱 Apache Spark 文件中的使用 PySpark 原生功能頁面。 `--extra-py-files`在未封裝其他程式碼的情況下，或者當您移轉具有現有工具鏈的 Spark 程式以管理相依性時非常有用。為了使您的相依性工具可維護，您必須在提交之前綁定相依性。

使用視覺轉換的程式設計腳本

使用 AWS Glue Studio 視覺化介面建立 AWS Glue 工作時，您可以使用受管資料轉換節點和自訂視覺化轉換來轉換資料。如需受管資料轉換節點的詳細資訊，請參閱the section called “編輯 AWS Glue 受

管資料轉換節點”。如需有關自訂視覺轉換的詳細資訊，請參閱the section called “ 自訂視覺化轉換 ”。只有當工作 [語言] 設定為使用 Python 時，才能產生使用視覺轉換的指令碼。

使用視覺化轉換產生 AWS Glue 工作時， AWS Glue Studio 會使用工作組態中的`--extra-py-files`參數，在執行階段環境中包含這些轉換。如需有關任務參數的詳細資訊，請參閱 the section called “任務參數”。變更產生的指令碼或執行階段環境時，您必須保留此工作組態，指令碼才能順利執行。

AWS Glue 中已提供 Python 模組

若要變更這些已提供模組的版本，請提供帶有 `--additional-python-modules` 任務參數的新版本。

AWS Glue version 2.0

AWS Glue 2.0 版包含以下開箱即用的 Python 模組：

- avro-python3==1.10.0
- awscli==1.27.60
- boto3==1.12.4
- botocore==1.15.4
- certifi==2019.11.28
- chardet==3.0.4
- click==8.1.3
- colorama==0.4.4
- cycler==0.10.0
- Cython==0.29.15
- docutils==0.15.2
- enum34==1.1.9
- fsspec==0.6.2
- idna==2.9
- importlib-metadata==6.0.0
- jmespath==0.9.4
- joblib==0.14.1

- kiwisolver==1.1.0
- matplotlib==3.1.3
- mpmath==1.1.0
- nltk==3.5
- numpy==1.18.1
- pandas==1.0.1
- patsy==0.5.1
- pmdarima==1.5.3
- ptvsd==4.3.2
- pyarrow==0.16.0
- pyasn1==0.4.8
- pydevd==1.9.0
- pyhocon==0.3.54
- PyMy平方米
- pyparsing==2.4.6
- python-dateutil==2.8.1
- pytz==2019.3
- PyYAML==5.3.1
- regex==2022.10.31
- requests==2.23.0
- rsa==4.7.2
- s3fs==0.4.0
- s3transfer==0.3.3
- scikit-learn==0.22.1
- scipy==1.4.1
- setuptools==45.2.0
- six==1.14.0
- Spark==1.0
- statsmodels==0.11.1

- subprocess32==3.5.4
- sympy==1.5.1
- tbats==1.0.9
- tqdm==4.64.1
- typing-extensions==4.4.0
- urllib3==1.25.8
- wheel==0.35.1
- zipp==3.12.0

AWS Glue 版本 3.0

AWS Glue 3.0 版包含以下開箱即用的 Python 模組：

- aiobotocore==1.4.2
- aiohttp==3.8.3
- aioitertools==0.11.0
- aiosignal==1.3.1
- async-timeout==4.0.2
- asynctest==0.13.0
- attrs==22.2.0
- avro-python3==1.10.2
- boto3==1.18.50
- botocore==1.21.50
- certifi==2021.5.30
- chardet==3.0.4
- charset-normalizer==2.1.1
- click==8.1.3
- cycler==0.10.0
- Cython==0.29.4
- docutils==0.17.1

- enum34==1.1.10
- frozenlist==1.3.3
- fsspec==2021.8.1
- idna==2.10
- importlib-metadata==6.0.0
- jmespath==0.10.0
- joblib==1.0.1
- kiwisolver==1.3.2
- matplotlib==3.4.3
- mpmath==1.2.1
- multidict==6.0.4
- nltk==3.6.3
- numpy==1.19.5
- packaging==23.0
- pandas==1.3.2
- patsy==0.5.1
- Pillow==9.4.0
- pip==23.0
- pmdarima==1.8.2
- ptvsd==4.3.2
- pyarrow==5.0.0
- pydevd==2.5.0
- pyhocon==0.3.58
- PyMy平方米
- pyparsing==2.4.7
- python-dateutil==2.8.2
- pytz==2021.1
- PyYAML==5.4.1
- regex==2022.10.31

- requests==2.23.0
- s3fs==2021.8.1
- s3transfer==0.5.0
- scikit-learn==0.24.2
- scipy==1.7.1
- six==1.16.0
- Spark==1.0
- statsmodels==0.12.2
- subprocess32==3.5.4
- sympy==1.8
- tbats==1.1.0
- threadpoolctl==3.1.0
- tqdm==4.64.1
- typing_extensions==4.4.0
- urllib3==1.25.11
- wheel==0.37.0
- wrapt==1.14.1
- yarl==1.8.2
- zipp==3.12.0

AWS Glue 版本 4.0

AWS Glue 4.0 版包含以下開箱即用的 Python 模組：

- aiobotocore==2.4.1
- aiohttp==3.8.3
- aioitertools==0.11.0
- aiosignal==1.3.1
- async-timeout==4.0.2
- asynctest==0.13.0
- attrs==22.2.0

- avro-python3==1.10.2
- boto3==1.24.70
- botocore==1.27.59
- certifi==2021.5.30
- chardet==3.0.4
- charset-normalizer==2.1.1
- click==8.1.3
- cycler==0.10.0
- Cython==0.29.32
- docutils==0.17.1
- enum34==1.1.10
- frozenlist==1.3.3
- fsspec==2021.8.1
- idna==2.10
- importlib-metadata==5.0.0
- jmespath==0.10.0
- joblib==1.0.1
- 萬花筒
- kiwisolver==1.4.4
- matplotlib==3.4.3
- mpmath==1.2.1
- multidict==6.0.4
- nltk==3.7
- numpy==1.23.5
- packaging==23.0
- pandas==1.5.1
- patsy==0.5.1
- Pillow==9.4.0
- pip==23.0.1

- 大概地
- pmdarima==2.0.1
- ptvsd==4.3.2
- pyarrow==10.0.0
- pydevd==2.5.0
- pyhocon==0.3.58
- PyMy平方米
- pyparsing==2.4.7
- python-dateutil==2.8.2
- pytz==2021.1
- PyYAML==6.0.1
- regex==2022.10.31
- requests==2.23.0
- s3fs==2022.11.0
- s3transfer==0.6.0
- scikit-learn==1.1.3
- scipy==1.9.3
- setuptools==49.1.3
- six==1.16.0
- statsmodels==0.13.5
- subprocess32==3.5.4
- sympy==1.8
- tbats==1.1.0
- threadpoolctl==3.1.0
- tqdm==4.64.1
- typing_extensions==4.4.0
- urllib3==1.25.11
- wheel==0.37.0
- wrapt==1.14.1

- yarl==1.8.2
- zipp==3.10.0

壓縮程式庫以加入

除非程式庫包含在單一的 `.py` 檔案裡，否則應封裝於 `.zip` 封存中。套件目錄應位於封存的根目錄，且套件必須包含一個 `__init__.py` 檔案。接著 Python 就可以正常匯入套件。

如果程式庫僅由一個 `.py` 檔案裡的單一 Python 模組組成，則不必將其置於 `.zip` 檔案。

在 AWS Glue 工作室筆記本中加載 Python 庫

若要在 AWS Glue Studio 筆記本中指定 Python 程式庫，請參閱安裝其他 Python 模組。

在開發端點載入 Python 程式庫

若要將不同的程式庫集用於不同的 ETL 指令碼，您可以為各程式庫集設定個別的開發端點，或是覆寫每次切換指令碼時開發端點載入的程式庫 `.zip` 檔案。

在建立開發端點時，您可以使用主控台為其指定一或多個程式庫 .zip 檔案。指派名稱和 IAM 角色後，請選擇 Script Libraries and job parameters (optional) [指令碼程式庫與任務參數 (選用)]，並在 `.zip`Python library path (Python 程式庫路徑) 方塊中輸入程式庫 檔案的完整 Amazon S3 路徑。例如：

```
s3://bucket/prefix/site-packages.zip
```

您也可以為檔案指定多個完整路徑，以逗號但不含空格的方式隔開，例如：

```
s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/lib_X.zip
```

如果您在後來更新這些 `.zip` 檔案，您可以使用主控台將檔案重新匯入開發端點。導覽至該開發端點，從 Action (動作) 選單選擇 Update ETL libraries (更新 ETL 程式庫)。

您可以透過類似方式使用 AWS Glue API 指定程式庫檔案。呼叫 CreateDevEndpoint 行動（Python：創建開發端點）以建立開發端點時，您可以在 `ExtraPythonLibsS3Path` 參數中為程式庫指定一個或多個完整路徑，而呼叫的格式如下：

```
dep = glue.create_dev_endpoint(
```

```
            EndpointName="testDevEndpoint",
            RoleArn="arn:aws:iam::123456789012",
            SecurityGroupIds="sg-7f5ad1ff",
            SubnetId="subnet-c12fdba4",
            PublicKey="ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCtp04H/y...",
            NumberOfNodes=3,
            ExtraPythonLibsS3Path="s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/
lib_X.zip")
```

更新開發端點時，您也可以更新其載入的程式庫，方式是使用 DevEndpointCustomLibraries 物件，並在呼叫 UpdateDevEndpoint （更新開發端點） 時將 `UpdateEtlLibraries` 參數設定為 `True`。

在工作中使用 Python 程式庫或 JobRun

在主控台建立新任務時，您可以指定一個或多個程式庫 .zip 檔案，方式是選擇 Script Libraries and job parameters (optional) (指令碼程式庫與任務參數 (選用))，並以與建立開發端點相同的方式輸入完整 Amazon S3 路徑：

```
s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/lib_X.zip
```

若要呼叫 CreateJob (建立工作 (_A)，您可以使用 `--extra-py-files` 預設參數來為預設程式庫指定一個或多個完整路徑，如下所示：

```
job = glue.create_job(Name='sampleJob',
                      Role='Glue_DefaultRole',
                      Command={'Name': 'glueetl',
                               'ScriptLocation': 's3://my_script_bucket/scripts/
my_etl_script.py'},
                      DefaultArguments={'--extra-py-files': 's3://bucket/prefix/
lib_A.zip,s3://bucket_B/prefix/lib_X.zip'})
```

然後，當您啟動時 JobRun，您可以使用不同的庫設置覆蓋默認庫設置：

```
runId = glue.start_job_run(JobName='sampleJob',
                           Arguments={'--extra-py-files': 's3://bucket/prefix/
lib_B.zip'})
```

AWS Glue Python 程式碼範例

- 程式碼範例：加入和關聯化資料

- 程式碼範例：使用 ResolveChoice Lambda 和 ApplyMapping

程式碼範例：加入和關聯化資料

本範例使用從 http://everypolitician.org/ 下載至 Amazon Simple Storage Service (Amazon S3) 的 sample-dataset 儲存貯體的資料集：s3://awsglue-datasets/examples/us-legislators/all。資料集包含美國國會議員和他們在美國眾議院和參議院內座位的 JSON 格式的資料，已針對教學用途稍作修改，並透過公有 Amazon S3 儲存貯體提供。

您可以在網站範例儲存庫中的join_and_relationalize.py檔案中找到此範例的AWS Glue原始程式 GitHub 碼。

本指南將利用這項資料告訴您如何執行下列動作：

- 使用AWS Glue爬蟲程式將存放在公用 Amazon S3 儲存貯體中的物件分類，並將其結構描述儲存到 AWS Glue 資料型錄中。
- 檢查爬蟲程式所產生的資料表中繼資料和結構描述。
- 編寫 Python 擷取、傳輸和載入 (ETL) 指令碼，使用 Data Catalog 中的中繼資料執行下列動作：
  - 將來自不同原始檔案的資料加入到單一資料表 (也就是將資料去正規化)。
  - 篩選加入的資料表，依國會議員類型放入不同的資料表。
  - 將產生的資料寫入到單獨的 Apache Parquet 檔案中，供以後分析之用。

在執行時偵錯 Python 或 PySpark 指令碼的建議方式 AWS 是在 AWS Glue Studio 上使用筆記型電腦。

步驟 1：在 Amazon S3 儲存貯體中網路爬取資料

1. 請登入 AWS Management Console，然後開啟AWS Glue主控台，網址為 https://console.aws.amazon.com/glue/。
2. 遵循中的步驟設定爬行者程式，建立新的爬行者程式，以將資料s3://awsglue-datasets/examples/us-legislators/all集編目到 AWS Glue 資料目錄legislators中名稱的資料庫。範例資料已放在這個公有 Amazon S3 儲存貯體中。
3. 執行新的爬蟲程式，接著查看 legislators 資料庫。

   爬蟲程式建立下列中繼資料資料表：

   - persons_json

- memberships_json
- organizations_json
- events_json
- areas_json
- countries_r_json

這是一個包含國會議員和其歷史的半標準化資料表集合。

步驟 2：新增樣板指令碼至開發端點筆記本

將以下樣板指令碼貼至開發端點以匯入您需要的 AWS Glue 程式庫，並且設定單一的 GlueContext：

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

glueContext = GlueContext(SparkContext.getOrCreate())
```

步驟 3：從資料目錄中的資料檢查結構描述

接下來，您可以輕鬆地 DynamicFrame 從 AWS Glue 資料型錄建立檢查，並檢查資料的結構描述。例如，若要查看 persons_json 資料表的結構描述，請將下列內容新增到筆記本：

```
persons = glueContext.create_dynamic_frame.from_catalog(
             database="legislators",
             table_name="persons_json")
print "Count: ", persons.count()
persons.printSchema()
```

以下為列印呼叫的輸出：

```
Count:  1961
```

```
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- name: string
|    |    |-- lang: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string
```

資料表中的每個人都是美國國會的成員。

若要檢視 memberships_json 資料表的結構描述，請輸入如下命令：

```
memberships = glueContext.create_dynamic_frame.from_catalog(
                 database="legislators",
                 table_name="memberships_json")
print "Count: ", memberships.count()
memberships.printSchema()
```

其輸出如下：

```
Count:  10439
root
|-- area_id: string
|-- on_behalf_of_id: string
|-- organization_id: string
|-- role: string
|-- person_id: string
|-- legislative_period_id: string
|-- start_date: string
|-- end_date: string
```

organizations 為政黨和參議院與眾議院這兩個議會殿堂。若要檢視 organizations_json 資料表的結構描述，請輸入如下命令：

```
orgs = glueContext.create_dynamic_frame.from_catalog(
           database="legislators",
           table_name="organizations_json")
print "Count: ", orgs.count()
orgs.printSchema()
```

其輸出如下：

```
Count:  13
root
|-- classification: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
```

```
|-- id: string
|-- name: string
|-- seats: int
|-- type: string
```

步驟 4：篩選資料

接著，保留需要的欄位，將 id 重新命名為 org_id。資料集很小，可以從整體來檢視。

toDF() 會將 DynamicFrame 轉換為 Apache Spark DataFrame，因此您可套用 Apache Spark SQL 中現有的轉換：

```
orgs = orgs.drop_fields(['other_names',
                         'identifiers']).rename_field(
                             'id', 'org_id').rename_field(
                                'name', 'org_name')
orgs.toDF().show()
```

以下將顯示輸出：

```
+--------------+--------------------+--------------------+--------------------+-----
+-----------+--------------------+
|classification|              org_id|            org_name|               links|seats|
     type|               image|
+--------------+--------------------+--------------------+--------------------+-----
+-----------+--------------------+
|         party|            party/al|                  AL|                null| null|
     null|                null|
|         party|      party/democrat|            Democrat|[[website,http://...| null|
     null|https://upload.wi...|
|         party|party/democrat-li...|    Democrat-Liberal|[[website,http://...| null|
     null|                null|
|   legislature|d56acebe-8fdc-47b...|House of Represen...|                null|  435|
lower house|                null|
|         party|   party/independent|         Independent|                null| null|
     null|                null|
|         party|party/new_progres...|     New Progressive|[[website,http://...| null|
     null|https://upload.wi...|
|         party|party/popular_dem...|    Popular Democrat|[[website,http://...| null|
     null|                null|
```

```
|         party|    party/republican|          Republican|[[website,http://...| null|
     null|https://upload.wi...|
|         party|party/republican-...|Republican-Conser...|[[website,http://...| null|
     null|                null|
|         party|      party/democrat|            Democrat|[[website,http://...| null|
     null|https://upload.wi...|
|         party|   party/independent|         Independent|                null| null|
     null|                null|
|         party|    party/republican|          Republican|[[website,http://...| null|
     null|https://upload.wi...|
|   legislature|8fa6c3d2-71dc-478...|              Senate|                null|  100|
upper house|                null|
+--------------+--------------------+--------------------+--------------------+-----
+-----------+--------------------+
```

輸入以下以檢視出現在 memberships 的 organizations：

```
memberships.select_fields(['organization_id']).toDF().distinct().show()
```

以下將顯示輸出：

```
+--------------------+
|     organization_id|
+--------------------+
|d56acebe-8fdc-47b...|
|8fa6c3d2-71dc-478...|
+--------------------+
```

步驟 5：全部整合為一

現在，使用 AWS Glue 加入這些關聯式表格，並建立一份關於國會議員 memberships 及其對應的 organizations 的完整歷史記錄資料表。

1. 首先，加入 persons 和 memberships 的 id 和 person_id。
2. 接著，將結果加入到 orgs 的 org_id 和 organization_id。
3. 然後，捨棄冗餘欄位 person_id 和 org_id。

您可以在同一 (延伸) 指令碼行執行所有這些操作：

```
l_history = Join.apply(orgs,
                       Join.apply(persons, memberships, 'id', 'person_id'),
                       'org_id', 'organization_id').drop_fields(['person_id',
 'org_id'])
print "Count: ", l_history.count()
l_history.printSchema()
```

其輸出如下：

```
Count:  10439
root
|-- role: string
|-- seats: int
|-- org_name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- type: string
|-- sort_name: string
|-- area_id: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- on_behalf_of_id: string
|-- other_names: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- name: string
|    |    |-- lang: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- name: string
|-- birth_date: string
|-- organization_id: string
|-- gender: string
|-- classification: string
|-- death_date: string
|-- legislative_period_id: string
```

```
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- image: string
|-- given_name: string
|-- family_name: string
|-- id: string
|-- start_date: string
|-- end_date: string
```

您現在取得最終的資料表，可用於分析。您可以用精巧、有效率的格式編寫，以用於分析 (也就是 Parquet)，在 AWS Glue、Amazon Athena 或 Amazon Redshift Spectrum 上執行 SQL。

以下呼叫將資料表編寫到多個檔案，在稍後執行分析時支援快速平行讀取：

```
glueContext.write_dynamic_frame.from_options(frame = l_history,
          connection_type = "s3",
          connection_options = {"path": "s3://glue-sample-target/output-dir/
legislator_history"},
          format = "parquet")
```

若要將所有歷史記錄資料合併成單一檔案，您必須將其轉換為資料框架、分割並編寫：

```
s_history = l_history.toDF().repartition(1)
s_history.write.parquet('s3://glue-sample-target/output-dir/legislator_single')
```

或者，如果您希望將其分為參議院和眾議院：

```
l_history.toDF().write.parquet('s3://glue-sample-target/output-dir/legislator_part',
                               partitionBy=['org_name'])
```

步驟 6：轉換關聯式資料庫的資料

AWS Glue 可讓您輕鬆地將資料寫入關聯式資料庫，例如 Amazon Redshift，即使是半結構化資料。其提供轉換 `relationalize`，可將 DynamicFrames 扁平化，無論框架中的物件多複雜。

使用本範例中的 l_history DynamicFrame，以根資料表 (hist_root) 的名稱和暫時任務路徑傳送至 relationalize。將傳回 DynamicFrameCollection。然後，您可以將 DynamicFrames 的名稱列在該集合中：

```
dfc = l_history.relationalize("hist_root", "s3://glue-sample-target/temp-dir/")
dfc.keys()
```

以下為 keys 呼叫的輸出：

```
[u'hist_root', u'hist_root_contact_details', u'hist_root_links',
 u'hist_root_other_names', u'hist_root_images', u'hist_root_identifiers']
```

Relationalize 將歷史記錄資料表分成六個新資料表：根資料表包含在 DynamicFrame 中的各物件記錄，和陣列的輔助資料表。關聯式資料庫中的陣列處理通常為次最佳化，尤其在這些陣列變得龐大時。將陣列分成不同的資料表，可加快查詢速度。

接著，檢查 contact_details 以查看分隔：

```
l_history.select_fields('contact_details').printSchema()
dfc.select('hist_root_contact_details').toDF().where("id = 10 or id =
 75").orderBy(['id','index']).show()
```

以下為 show 呼叫的輸出：

```
root
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
+---+-----+-----------------------+-------------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-------------------------+
| 10|    0|                    fax|                         |
| 10|    1|                       |             202-225-1314|
| 10|    2|                  phone|                         |
| 10|    3|                       |             202-225-3772|
| 10|    4|                twitter|                         |
```

```
| 10|    5|                          |          MikeRossUpdates|
| 75|    0|                      fax|                          |
| 75|    1|                          |             202-225-7856|
| 75|    2|                    phone|                          |
| 75|    3|                          |             202-225-2711|
| 75|    4|                  twitter|                          |
| 75|    5|                          |                SenCapito|
+---+-----+--------------------------+--------------------------+
```

contact_details 欄位為原始 DynamicFrame 中結構的陣列。這些陣列的每個元素都是輔助資料表中的單獨資料列，以 index 編製索引。此處的 id 是 hist_root 資料表的外部金鑰，金鑰為 contact_details：

```
dfc.select('hist_root').toDF().where(
    "contact_details = 10 or contact_details = 75").select(
       ['id', 'given_name', 'family_name', 'contact_details']).show()
```

以下為其輸出：

```
+--------------------+----------+-----------+---------------+
|                  id|given_name|family_name|contact_details|
+--------------------+----------+-----------+---------------+
|f4fc30ee-7b42-432...|      Mike|       Ross|             10|
|e3c60f34-7d1b-4c0...|   Shelley|     Capito|             75|
+--------------------+----------+-----------+---------------+
```

請注意，這些命令將使用 toDF()，然後是 where 表達式，來篩選您想要查看的資料列。

因此，加入 hist_root 資料表與輔助資料表可執行下列動作：

- 無需陣列支援便能將資料載入到資料庫。
- 使用 SQL 查詢陣列中的每個個別項目。

使用 AWS Glue 連線以安全存放和存取您的 Amazon Redshift 憑證。有關如何建立自己的連線的詳細資訊，請參閱 連線至資料。

您現已準備好透過一次循環一個 DynamicFrames 來將資料寫入連接器：

```
for df_name in dfc.keys():
  m_df = dfc.select(df_name)
  print "Writing to table: ", df_name
  glueContext.write_dynamic_frame.from_jdbc_conf(frame = m_df, connection settings
 here)
```

您的連接器設定將因您的關聯式資料庫類型而異：

- 如需有關寫入 Amazon Redshift 的指示，請參閱the section called “Redshift 連線”。
- 若為其他資料庫，請參閱 AWS Glue for Spark 中 ETL 的連線類型和選項。

結論

整體而言，AWS Glue 極具彈性。它讓您用幾行程式碼便能完成通常要好幾天才能完成撰寫的任務。您可以在上 GitHub的AWS Glue範例中找到 Python 檔案join_and_relationalize.py中的整個 source-to-target ETL 指令碼。

程式碼範例：使用 ResolveChoice Lambda 和 ApplyMapping

此範例使用的資料集，包含從兩個 Data.CMS.gov 資料集下載的美國聯邦醫療保險 (Medicare) 供應商付款資料：「住院患者預期付款系統供應商前 100 大診斷相關群組摘要 - FY2011」和「住院患者費用資料 FY 2011」。下載資料之後，我們修改了資料集，以在檔案結尾處引入幾個錯誤記錄。上述經修改的檔案位於公有 Amazon S3 儲存貯體，位置在 s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv。

您可以在AWS Glue範例 GitHub 儲存庫的data_cleaning_and_lambda.py檔案中找到此範例的原始程式碼。

在執行時偵錯 Python 或 PySpark 指令碼的建議方式 AWS 是在 AWS Glue Studio 上使用筆記型電腦。

步驟 1：在 Amazon S3 儲存貯體中網路爬取資料

1. 請登入 AWS Management Console 並開啟AWS Glue主控台，網址為 https://console.aws.amazon.com/glue/。
2. 遵循中所述的程序設定爬行者程式，建立可編目s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv檔案的新爬行者程式，並將產生的中繼資料放入 AWS Glue Data Catalog payments 中名為的資料庫中。
3. 執行新的爬蟲程式，接著查看 payments 資料庫。在讀取檔案開頭，確定其格式和分隔符號後，爬蟲程式應會於資料庫建立一個命名為 medicare 的中繼資料資料表。

新 medicare 資料表的結構描述如下：

```
Column  name                                Data type
================================================
drg definition                                string
provider id                                   bigint
provider name                                 string
provider street address                       string
provider city                                 string
provider state                                string
provider zip code                             bigint
hospital referral region description          string
total discharges                              bigint
average covered charges                       string
average total payments                        string
average medicare payments                     string
```

步驟 2：新增樣板指令碼至開發端點筆記本

將以下樣板指令碼貼至開發端點以匯入您需要的 AWS Glue 程式庫，並且設定單一的 GlueContext：

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

glueContext = GlueContext(SparkContext.getOrCreate())
```

步驟 3：比較不同的結構描述剖析

接著，您可以查看 Apache Spark DataFrame 辨識出的結構描述是否跟 AWS Glue 爬蟲程式所記錄的相同。執行此程式碼：

```
medicare = spark.read.format(
   "com.databricks.spark.csv").option(
   "header", "true").option(
```

```
   "inferSchema", "true").load(
   's3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv')
medicare.printSchema()
```

以下為 printSchema 呼叫的輸出：

```
root
 |-- DRG Definition: string (nullable = true)
 |-- Provider Id: string (nullable = true)
 |-- Provider Name: string (nullable = true)
 |-- Provider Street Address: string (nullable = true)
 |-- Provider City: string (nullable = true)
 |-- Provider State: string (nullable = true)
 |-- Provider Zip Code: integer (nullable = true)
 |-- Hospital Referral Region Description: string (nullable = true)
 |--  Total Discharges : integer (nullable = true)
 |--  Average Covered Charges : string (nullable = true)
 |--  Average Total Payments : string (nullable = true)
 |-- Average Medicare Payments: string (nullable = true)
```

接著，查看 AWS Glue DynamicFrame 產生的結構描述：

```
medicare_dynamicframe = glueContext.create_dynamic_frame.from_catalog(
       database = "payments",
       table_name = "medicare")
medicare_dynamicframe.printSchema()
```

printSchema 的輸出如下：

```
root
 |-- drg definition: string
 |-- provider id: choice
 |    |-- long
 |    |-- string
 |-- provider name: string
 |-- provider street address: string
 |-- provider city: string
 |-- provider state: string
 |-- provider zip code: long
```

```
 |-- hospital referral region description: string
 |-- total discharges: long
 |-- average covered charges: string
 |-- average total payments: string
 |-- average medicare payments: string
```

在 DynamicFrame 產生的結構描述中，provider id 可以是 long 或 string 類型。DataFrame 結構描述將 Provider Id 列為 string 類型， Data Catalog 則是將 provider id 列為 bigint 類型。

何者正確？ 在檔案的最後有兩筆記錄 (總共 160,000 筆記錄)，且該欄位中有 string 值。這些就是之前引入以示範產生問題的錯誤記錄。

為了解決這種問題，AWS GlueDynamicFrame 採用 choice (選擇) 類型的概念。在此例中，DynamicFrame 展示了 long 和 string 值都會在該欄出現。AWS Glue 爬蟲程式遺漏了 string 值，原因是只考量到資料的前 2 MB。Apache Spark DataFrame 會考量整個資料集，但被強制將最普遍的類型指派給該欄位，亦即 string。事實上，Spark 在遇到複雜類型或不熟悉的變化時，通常都會採取最普遍的作法。

要查詢 provider id 欄，請先解決選擇類型。您可以使用 DynamicFrame 中的 resolveChoice 轉換方法，藉由以下的 cast:long 選項將 string 值轉換為 long 值。

```
medicare_res = medicare_dynamicframe.resolveChoice(specs = [('provider
 id','cast:long')])
medicare_res.printSchema()
```

printSchema 輸出就會是：

```
root
 |-- drg definition: string
 |-- provider id: long
 |-- provider name: string
 |-- provider street address: string
 |-- provider city: string
 |-- provider state: string
 |-- provider zip code: long
 |-- hospital referral region description: string
 |-- total discharges: long
 |-- average covered charges: string
 |-- average total payments: string
```

```
 |-- average medicare payments: string
```

如果有無法轉換的 `string` 值，AWS Glue 會插入 `null`。

另一個選項是將選擇類型轉換為 `struct`，這會保留兩種類型的值。

接著，查看異常的資料列。

```
medicare_res.toDF().where("'provider id' is NULL").show()
```

您會見到以下情況：

```
+--------------------+-----------+--------------+----------------------+-------------
+--------------+-----------------+------------------------------------+---------------
+----------------------+----------------------+------------------------+
|      drg definition|provider id|  provider name|provider street address|provider
 city|provider state|provider zip code|hospital referral region description|total
 discharges|average covered charges|average total payments|average medicare payments|
+--------------------+-----------+--------------+----------------------+-------------
+--------------+-----------------+------------------------------------+---------------
+----------------------+----------------------+------------------------+
|948 - SIGNS & SYM...|       null|           INC|       1050 DIVISION ST|
 MAUSTON|           WI|           53948|                        WI - Madison|
      12|             $11961.41|             $4619.00|                $3775.33|
|948 - SIGNS & SYM...|       null| INC- ST JOSEPH|     5000 W CHAMBERS ST|
 MILWAUKEE|           WI|           53210|                        WI - Milwaukee|
        14|             $10514.28|             $5562.50|                $4522.78|
+--------------------+-----------+--------------+----------------------+-------------
+--------------+-----------------+------------------------------------+---------------
+----------------------+----------------------+------------------------+
```

現在移除兩筆不正確的記錄，如下所示：

```
medicare_dataframe = medicare_res.toDF()
medicare_dataframe = medicare_dataframe.where("'provider id' is NOT NULL")
```

步驟 4：映射資料並使用 Apache Spark Lambda 函數

AWS Glue 尚未直接支援 Lambda 函式，亦即使用者定義的函式。但是您隨時可以從 Apache Spark `DataFrame` 來回轉換 `DynamicFrame`，以利用除了 `DynamicFrames` 特殊功能之外的 Spark 功能。

接著，將付款資訊轉為數字，讓 Amazon Redshift 或 Amazon Athena 等分析引擎可以更快處理。

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

chop_f = udf(lambda x: x[1:], StringType())
medicare_dataframe = medicare_dataframe.withColumn(
        "ACC", chop_f(
            medicare_dataframe["average covered charges"])).withColumn(
                "ATP", chop_f(
                    medicare_dataframe["average total payments"])).withColumn(
                        "AMP", chop_f(
                            medicare_dataframe["average medicare payments"]))
medicare_dataframe.select(['ACC', 'ATP', 'AMP']).show()
```

show 呼叫的輸出如下：

```
+--------+-------+-------+
|     ACC|    ATP|    AMP|
+--------+-------+-------+
|32963.07|5777.24|4763.73|
|15131.85|5787.57|4976.71|
|37560.37|5434.95|4453.79|
|13998.28|5417.56|4129.16|
|31633.27|5658.33|4851.44|
|16920.79|6653.80|5374.14|
|11977.13|5834.74|4761.41|
|35841.09|8031.12|5858.50|
|28523.39|6113.38|5228.40|
|75233.38|5541.05|4386.94|
|67327.92|5461.57|4493.57|
|39607.28|5356.28|4408.20|
|22862.23|5374.65|4186.02|
|31110.85|5366.23|4376.23|
|25411.33|5282.93|4383.73|
| 9234.51|5676.55|4509.11|
|15895.85|5930.11|3972.85|
|19721.16|6192.54|5179.38|
|10710.88|4968.00|3898.88|
|51343.75|5996.00|4962.45|
+--------+-------+-------+
only showing top 20 rows
```

資料仍然全是字串。我們可以使用強大的 apply_mapping 轉換方法來捨棄、重新命名、轉換、巢套資料，讓其他資料程式設計語言和系統能夠輕易存取：

```
from awsglue.dynamicframe import DynamicFrame
medicare_tmp_dyf = DynamicFrame.fromDF(medicare_dataframe, glueContext, "nested")
medicare_nest_dyf = medicare_tmp_dyf.apply_mapping([('drg definition', 'string', 'drg',
 'string'),
                ('provider id', 'long', 'provider.id', 'long'),
                ('provider name', 'string', 'provider.name', 'string'),
                ('provider city', 'string', 'provider.city', 'string'),
                ('provider state', 'string', 'provider.state', 'string'),
                ('provider zip code', 'long', 'provider.zip', 'long'),
                ('hospital referral region description', 'string','rr', 'string'),
                ('ACC', 'string', 'charges.covered', 'double'),
                ('ATP', 'string', 'charges.total_pay', 'double'),
                ('AMP', 'string', 'charges.medicare_pay', 'double')])
medicare_nest_dyf.printSchema()
```

printSchema 輸出如下：

```
root
 |-- drg: string
 |-- provider: struct
 |    |-- id: long
 |    |-- name: string
 |    |-- city: string
 |    |-- state: string
 |    |-- zip: long
 |-- rr: string
 |-- charges: struct
 |    |-- covered: double
 |    |-- total_pay: double
 |    |-- medicare_pay: double
```

將資料轉回 Spark DataFrame 後，您就可以顯示其樣貌：

```
medicare_nest_dyf.toDF().show()
```

其輸出如下：

```
+--------------------+--------------------+--------------+--------------------+
```

```
|                 drg|            provider|              rr|             charges|
+--------------------+--------------------+----------------+--------------------+
|039 - EXTRACRANIA...|[10001,SOUTHEAST ...|     AL - Dothan|[32963.07,5777.24...|
|039 - EXTRACRANIA...|[10005,MARSHALL M...|AL - Birmingham|[15131.85,5787.57...|
|039 - EXTRACRANIA...|[10006,ELIZA COFF...|AL - Birmingham|[37560.37,5434.95...|
|039 - EXTRACRANIA...|[10011,ST VINCENT...|AL - Birmingham|[13998.28,5417.56...|
|039 - EXTRACRANIA...|[10016,SHELBY BAP...|AL - Birmingham|[31633.27,5658.33...|
|039 - EXTRACRANIA...|[10023,BAPTIST ME...|AL - Montgomery|[16920.79,6653.8,...|
|039 - EXTRACRANIA...|[10029,EAST ALABA...|AL - Birmingham|[11977.13,5834.74...|
|039 - EXTRACRANIA...|[10033,UNIVERSITY...|AL - Birmingham|[35841.09,8031.12...|
|039 - EXTRACRANIA...|[10039,HUNTSVILLE...|AL - Huntsville|[28523.39,6113.38...|
|039 - EXTRACRANIA...|[10040,GADSDEN RE...|AL - Birmingham|[75233.38,5541.05...|
|039 - EXTRACRANIA...|[10046,RIVERVIEW ...|AL - Birmingham|[67327.92,5461.57...|
|039 - EXTRACRANIA...|[10055,FLOWERS HO...|     AL - Dothan|[39607.28,5356.28...|
|039 - EXTRACRANIA...|[10056,ST VINCENT...|AL - Birmingham|[22862.23,5374.65...|
|039 - EXTRACRANIA...|[10078,NORTHEAST ...|AL - Birmingham|[31110.85,5366.23...|
|039 - EXTRACRANIA...|[10083,SOUTH BALD...|     AL - Mobile|[25411.33,5282.93...|
|039 - EXTRACRANIA...|[10085,DECATUR GE...|AL - Huntsville|[9234.51,5676.55,...|
|039 - EXTRACRANIA...|[10090,PROVIDENCE...|     AL - Mobile|[15895.85,5930.11...|
|039 - EXTRACRANIA...|[10092,D C H REGI...|AL - Tuscaloosa|[19721.16,6192.54...|
|039 - EXTRACRANIA...|[10100,THOMAS HOS...|     AL - Mobile|[10710.88,4968.0,...|
|039 - EXTRACRANIA...|[10103,BAPTIST ME...|AL - Birmingham|[51343.75,5996.0,...|
+--------------------+--------------------+----------------+--------------------+
only showing top 20 rows
```

步驟 5：寫入資料至 Apache Parquet

AWS Glue 可讓您輕鬆以關聯式資料庫能有效取用的格式 (例如 Apache Parquet) 撰寫資料：

```
glueContext.write_dynamic_frame.from_options(
       frame = medicare_nest_dyf,
       connection_type = "s3",
       connection_options = {"path": "s3://glue-sample-target/output-dir/
medicare_parquet"},
       format = "parquet")
```

AWS Glue PySpark 延伸模組參考

AWS Glue 為 PySpark Python 方言建立了下列延伸模組。

- 使用 getResolvedOptions 存取參數
- PySpark 延伸模組類型

- DynamicFrame 類
- DynamicFrameCollection 類別
- DynamicFrameWriter 類別
- DynamicFrameReader 類
- GlueContext 類

使用 **getResolvedOptions** 存取參數

AWS Glue `getResolvedOptions(args, options)` 公用程式功能可讓您存取執行任務時傳送到指令碼的引數。若要使用此功能，請先從 AWS Glue `utils` 模組與 `sys` 模組一起匯入：

```
import sys
from awsglue.utils import getResolvedOptions
```

`getResolvedOptions(args, options)`

- `args` - `sys.argv` 所含的引數清單。
- `options` - 想要擷取之引數名稱的 Python 陣列。

Example 擷取傳送到 JobRun 的引數

假設您在指令碼內建立 JobRun，或許在 Lambda 函數內：

```
response = client.start_job_run(
             JobName = 'my_test_Job',
             Arguments = {
               '--day_partition_key':   'partition_0',
               '--hour_partition_key':  'partition_1',
               '--day_partition_value':  day_partition_value,
               '--hour_partition_value': hour_partition_value } )
```

若要擷取傳送的引數，您可以使用 `getResolvedOptions` 函數，如下所示：

```
import sys
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv,
                          ['JOB_NAME',
```

```
                            'day_partition_key',
                            'hour_partition_key',
                            'day_partition_value',
                            'hour_partition_value'])
print "The day-partition key is: ", args['day_partition_key']
print "and the day-partition value is: ", args['day_partition_value']
```

請注意，每個引數的定義開頭形式為兩個連字號，在指令碼中參考的引數則不含連字號。引數只使用底線，不使用連字號。您的引數需要遵循此慣例才能被解析。

PySpark 延伸模組類型

AWS Glue PySpark 延伸模組所使用的類型。

DataType

其他 AWS Glue 類型的基本類別。

`__init__(properties={})`

- `properties` – 資料類型的屬性 (選用)。

`typeName(cls)`

傳回 AWS Glue 類型類別的類型 (亦即從尾端移除「Type」的類別名稱)。

- `cls` – 從 AWS Glue 衍生而來的 `DataType` 類別執行個體。

`jsonValue( )`

傳回 JSON 物件，其中包含類別的資料類型和屬性：

```
{
  "dataType": typeName,
  "properties": properties
}
```

AtomicType 和 simple 衍生性產品

繼承自及延伸 DataType 類別，並做為所有 AWS Glue 不可部分完成資料類型的基本類別。

`fromJsonValue(cls, json_value)`

以來自 JSON 物件的值，初始化類別執行個體。

- `cls` – 要初始化的 AWS Glue 類別執行個體。
- `json_value` – 用於載入金鑰/值對的來源 JSON 物件。

以下類型為 AtomicType 類別的簡單衍生產品：

- `BinaryType` – 二進位資料。
- `BooleanType` – 布林值。
- `ByteType` – 位元組值。
- `DateType` – 日期時間值。
- `DoubleType` – 浮點雙精度值。
- `IntegerType` – 整數值。
- `LongType` – 長整數值。
- `NullType` – null 值。
- `ShortType` – 短整數值。
- `StringType` – 文字字串。
- `TimestampType` – 時間戳記值 (單位通常為 1970/1/1 起的秒數)。
- `UnknownType` – 無法識別類型的值。

DecimalType(AtomicType)

沿用自和延伸 AtomicType 類別來代表十進位數字 (以十進位數字表示，而非二進位 base-2 數字)。

`__init__(precision=10, scale=2, properties={})`

- `precision` – 十進位數字的位數 (選用，預設為 10)。
- `scale` – 小數點右邊的位數 (選用，預設為 2)。
- `properties` – 十進位數字段屬性 (選用)。

EnumType(AtomicType)

繼承自和延伸 AtomicType 類別，以表示有效選項的列舉。

__init__(options)

- options – 列舉的選項清單。

集合類型

- ArrayType(DataType)
- ChoiceType(DataType)
- MapType(DataType)
- Field(Object)
- StructType(DataType)
- EntityType(DataType)

ArrayType(DataType)

__init__(elementType=UnknownType(), properties={})

- elementType – 陣列中的元素類型 (選用，預設為 UnknownType)。
- properties – 陣列的屬性 (選用)。

ChoiceType(DataType)

__init__(choices=[], properties={})

- choices – 可能的選項清單 (選用)。
- properties – 這些選項的屬性 (選用)。

add(new_choice)

將選項新增至可能的選項清單。

- new_choice – 要加入至可能選項清單的選項。

merge(new_choices)

合併新選項的清單與現有的選項清單。

- new_choices – 合併新選項與現有選項的清單。

MapType(DataType)

__init__(valueType=UnknownType, properties={})

- valueType – 映射中的值類型 (選用，預設為 UnknownType)。
- properties – 映射的屬性 (選用)。

Field(Object)

從衍生自 DataType 的物件建立欄位物件。

__init__(name, dataType, properties={})

- name – 要指派到欄位的名稱。
- dataType – 要從中建立欄位的物件。
- properties – 欄位的屬性 (選用)。

StructType(DataType)

定義資料結構 (struct)。

__init__(fields=[], properties={})

- fields – 要包含在結構中的欄位 (Field 類型) 清單 (選用)。
- properties – 結構的屬性 (選用)。

add(field)

- field – 要新增到架構的物件類型 Field。

hasField(field)

如果此結構有相同名稱的欄位，將傳回 True，否則將傳回 False。

- field – 欄位名稱，或名稱已使用的物件類型 Field。

getField(field)

- field – 欄位名稱，或使用其名稱的物件類型 Field。如果結構有相同名稱的欄位，將會傳回。

EntityType(DataType)

__init__(entity, base_type, properties)

此類別尚未實作。

其他類型

- DataSource(object)
- DataSink(object)

DataSource(object)

__init__(j_source, sql_ctx, name)

- j_source – 資料來源。
- sql_ctx – SQL 內容。
- name – 資料來源名稱。

setFormat(format, **options)

- format – 要用於設定資料來源的格式。
- options – 要用於設定資料來源的選項集合。如需有關格式選項的詳細資訊，請參閱 the section called “資料格式選項”。

getFrame()

傳回資料來源的 DynamicFrame。

DataSink(object)

__init__(j_sink, sql_ctx)

- j_sink – 要建立的目的地。
- sql_ctx – 資料目的地的 SQL 內容。

setFormat(format, **options)

- format – 要用於設定資料目的地的格式。
- options – 要用於設定資料目的地的選項集合。如需有關格式選項的詳細資訊，請參閱 the section called “資料格式選項”。

setAccumulableSize(size)

- size – 要設定的 accumulable 大小 (以位元組為單位)。

writeFrame(dynamic_frame, info="")

- dynamic_frame – 所要撰寫的 DynamicFrame。
- info – 有關 DynamicFrame 的資訊 (選用)。

write(dynamic_frame_or_dfc, info="")

撰寫 DynamicFrame 或 DynamicFrameCollection。

- dynamic_frame_or_dfc – 要撰寫的 DynamicFrame 物件或 DynamicFrameCollection 物件。
- info – 有關要撰寫的 DynamicFrame 或 DynamicFrames 的資訊 (選用)。

DynamicFrame 類

Apache Spark 其中一個主要抽象為 SparkSQL DataFrame，其與 R 和 pandas 中找到的 DataFrame 結構類似。DataFrame 類似於資料表並支援功能樣式 (對應/減少/篩選/等) 操作和 SQL 操作 (選擇、專案、彙總)。

DataFrames 功能強大，受到廣泛採用，但其在擷取、轉換和載入 (ETL) 操作上受到限制。最重要的是，其需要指定結構描述，才能載入任何資料。SparkSQL 可解決此問題，其進行兩次資料傳送，第一個推斷結構描述，第二個則載入資料。不過，此推斷相當有限，無法滿足龐大資料的實際需求。例如，相同的欄位在不同的記錄內可能為不同的類型。Apache Spark 通常讓出並使用原始欄位文字回報類型為 string。這可能不正確，而且您可能需要更精確控制如何解決結構描述的差異。此外，對於大型資料集，額外傳送來源資料的代價可能使人卻步地高昂。

為了解決這些限制， AWS Glue 引入了DynamicFrame. DynamicFrame 類似 DataFrame，但每筆記錄均為自我描述，且開始時不需結構描述。而是在需要 on-the-fly時AWS Glue計算結構描述，並使用選擇 (或聯集) 類型明確地對結構描述不一致性進行編碼。您可以解決這些不一致，讓您的資料集相容於需要固定結構描述的資料存放區。

同樣地，DynamicRecord 代表 DynamicFrame 內的邏輯記錄。其類似 Spark DataFrame 中的資料列，除了它是自我描述的，以及可用於不符合固定結構描述的資料。搭配使用 AWS Glue 時 PySpark，通常不會獨立操作DynamicRecords。相反地，您可以透過其 DynamicFrame 一起轉換資料集。

您可以在解決任何結構描述不一致後反覆轉換 DynamicFrames 和 DataFrames。

— construction —

- __init__
- fromDF
- toDF

__init__

__init__(jdf, glue_ctx, name)

- jdf – Java 虛擬機器 (JVM) 中資料框架的參考。
- glue_ctx – GlueContext 類 物件。
- name – 選用名稱字串，預設是空的。

fromDF

fromDF(dataframe, glue_ctx, name)

將 DataFrame 欄位轉換為 DynamicFrame 欄位，藉此將 DataFrame 轉換為 DynamicRecord。傳回新的 DynamicFrame。

DynamicRecord 代表 DynamicFrame 中的邏輯記錄。它類似 Spark DataFrame 中的一列，除了它是自我描述的，以及可用於不符合固定結構描述的資料。

此函數會預期 DataFrame 中具有重複名稱的資料欄已受到解析。

- dataframe – 要轉換的 Apache Spark SQL DataFrame (必要)。
- glue_ctx – 指定轉換內容的 GlueContext 類 物件 (必要)。
- name— 結果的名稱DynamicFrame（ 自 AWS Glue 3.0 以來是可選的 ）。

toDF

toDF(options)

將 DynamicRecords 轉換為 DataFrame 欄位，藉此將 DynamicFrame 轉換為 Apache Spark DataFrame。傳回新的 DataFrame。

DynamicRecord 代表 DynamicFrame 中的邏輯記錄。它類似 Spark DataFrame 中的一列，除了它是自我描述的，以及可用於不符合固定結構描述的資料。

- options – 選項清單。如果您選擇 Project 和 Cast 動作類型，請指定目標類型。範例如下。

```
>>>toDF([ResolveOption("a.b.c", "KeepAsStruct")])
>>>toDF([ResolveOption("a.b.c", "Project", DoubleType())])
```

— information —

- count
- 結構描述
- printSchema
- 顯示
- repartition
- coalesce

count

count( ) – 傳回基礎 DataFrame 中的資料列數量。

結構描述

`schema( )` – 傳回此 DynamicFrame 的結構描述，或者，假如不可用，則傳回基礎 DataFrame 的結構描述。

如需有關組成此結構描述的 DynamicFrame 類型的詳細資訊，請參閱 the section called “類型”。

printSchema

`printSchema( )` – 列印基礎 DataFrame 的結構描述。

顯示

`show(num_rows)` – 列印基礎 DataFrame 的指定資料列數量。

repartition

`repartition(numPartitions)` – 傳回包含 numPartitions 個分割區的新 DynamicFrame。

coalesce

`coalesce(numPartitions)` – 傳回包含 numPartitions 個分割區的新 DynamicFrame。

— transforms —

- apply_mapping
- drop_fields
- 篩選條件
- join
- map
- mergeDynamicFrame
- 關聯化
- rename_field
- resolveChoice
- select_fields
- spigot
- split_fields
- split_rows

- unbox
- the section called “聯集”
- 解巢狀
- unnest_ddb_json
- write

apply_mapping

apply_mapping(mappings, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

套用宣告映射至 DynamicFrame，並傳回將這些映射套用至您指定欄位的新 DynamicFrame。未指定的欄位將從新的 DynamicFrame 中省略。

- mappings –映射元組的清單 (必要)。每個清單包括：(來源欄、來源類型、目標欄、目標類型)。

  如果來源資料欄的名稱中有一個小點 "."，則您必須在其前後加上反引號 "``"。例如，若要將 this.old.name (字串) 對應至 thisNewName，會使用以下元組：

  ```
  ("`this.old.name`", "string", "thisNewName", "string")
  ```

- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與此轉換回報錯誤關聯的字串 (選用)。
- stageThreshold – 此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。
- totalThreshold –直到及包含此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。

範例：使用 apply_map 來重新命名欄位並變更欄位類型

以下程式碼顯示使用 apply_mapping 方法重新命名所選欄位和更改欄位類型的方法。

> **Note**
>
> 若要存取此範例中使用的資料集，請參閱 程式碼範例：加入和關聯化資料 並依照 步驟 1：在 Amazon S3 儲存貯體中網路爬取資料 中的說明進行。

```
# Example: Use apply_mapping to reshape source data into
# the desired column names and types as a new DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create a DynamicFrame and view its schema
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
)
print("Schema for the persons DynamicFrame:")
persons.printSchema()

# Select and rename fields, change field type
print("Schema for the persons_mapped DynamicFrame, created with apply_mapping:")
persons_mapped = persons.apply_mapping(
    [
        ("family_name", "String", "last_name", "String"),
        ("name", "String", "first_name", "String"),
        ("birth_date", "String", "date_of_birth", "Date"),
    ]
)
persons_mapped.printSchema()
```

輸出

```
Schema for the persons DynamicFrame:
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
```

```
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string

Schema for the persons_mapped DynamicFrame, created with apply_mapping:
root
|-- last_name: string
|-- first_name: string
|-- date_of_birth: date
```

drop_fields

drop_fields(paths, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

呼叫 FlatMap 類別 轉換，從 DynamicFrame 移除欄位。傳回捨棄了指定欄位的新 DynamicFrame。

- paths – 字串清單。各包含您想捨棄的欄位節點的完整路徑。您可以使用點標記法來指定巢狀欄位。例如，如果欄位 first 是樹狀結構中的子欄位 name，您可以指定 "name.first" 為路徑。

  如果欄位節點的名稱中有常值 .，您必須以反引號將名稱括起 (`)。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與此轉換回報錯誤關聯的字串 (選用)。
- stageThreshold – 此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。

- totalThreshold –直到及包含此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。

範例：使用 drop_fields 從 **DynamicFrame** 中移除欄位

此程式碼範例使用 drop_fields 方法從 DynamicFrame 中移除選取的頂層和巢狀欄位。

範例資料集

此範例使用下列資料集，該資料集由程式碼中的 EXAMPLE-FRIENDS-DATA 表格表示：

```
{"name": "Sally", "age": 23, "location": {"state": "WY", "county": "Fremont"},
 "friends": []}
{"name": "Varun", "age": 34, "location": {"state": "NE", "county": "Douglas"},
 "friends": [{"name": "Arjun", "age": 3}]}
{"name": "George", "age": 52, "location": {"state": "NY"}, "friends": [{"name":
 "Fred"}, {"name": "Amy", "age": 15}]}
{"name": "Haruki", "age": 21, "location": {"state": "AK", "county": "Denali"}}
{"name": "Sheila", "age": 63, "friends": [{"name": "Nancy", "age": 22}]}
```

範例程式碼

```
# Example: Use drop_fields to remove top-level and nested fields from a DynamicFrame.
# Replace MY-EXAMPLE-DATABASE with your Glue Data Catalog database name.
# Replace EXAMPLE-FRIENDS-DATA with your table name.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create a DynamicFrame from Glue Data Catalog
glue_source_database = "MY-EXAMPLE-DATABASE"
glue_source_table = "EXAMPLE-FRIENDS-DATA"

friends = glueContext.create_dynamic_frame.from_catalog(
    database=glue_source_database, table_name=glue_source_table
)
print("Schema for friends DynamicFrame before calling drop_fields:")
friends.printSchema()
```

```
# Remove location.county, remove friends.age, remove age
friends = friends.drop_fields(paths=["age", "location.county", "friends.age"])
print("Schema for friends DynamicFrame after removing age, county, and friend age:")
friends.printSchema()
```

輸出

```
Schema for friends DynamicFrame before calling drop_fields:
root
|-- name: string
|-- age: int
|-- location: struct
|    |-- state: string
|    |-- county: string
|-- friends: array
|    |-- element: struct
|    |    |-- name: string
|    |    |-- age: int

Schema for friends DynamicFrame after removing age, county, and friend age:
root
|-- name: string
|-- location: struct
|    |-- state: string
|-- friends: array
|    |-- element: struct
|    |    |-- name: string
```

篩選條件

filter(f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

傳回新的 DynamicFrame，其中包含所有 DynamicRecords，其滿足輸入 DynamicFrame 且指定的述詞函數 f。

- f – 要套用至 DynamicFrame 的述詞函數。此函數必須以 DynamicRecord 做為引數並傳回 True，如果 DynamicRecord 符合篩選條件要求，否則將傳回 False (必要)。

  DynamicRecord 代表 DynamicFrame 中的邏輯記錄。它類似 Spark DataFrame 中的一列，除了它是自我描述的，以及可用於不符合固定結構描述的資料。

- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與此轉換回報錯誤關聯的字串 (選用)。
- stageThreshold – 此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。
- totalThreshold –直到及包含此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。

範例：使用篩選條件取得已篩選的欄位選取

此範例使用filter方法來建立新的DynamicFrame，其中包括對另一個 DynamicFrame 的欄位的已篩選選取。

跟 map 方法一樣，filter 需要一個函數作為引數，該引數應用於原始 DynamicFrame 中的每個記錄。該函數需要一個記錄作為輸入，並傳回一個布林值。如果傳回值為 true，記錄會包含在所產生的 DynamicFrame 中。如果傳回值為 false，記錄會被排除在外。

> **Note**
>
> 若要存取此範例中使用的資料集，請參閱 程式碼範例：使用 ResolveChoice Lambda 和 ApplyMapping 並依照 步驟 1：在 Amazon S3 儲存貯體中網路爬取資料 中的說明進行。

```
# Example: Use filter to create a new DynamicFrame
# with a filtered selection of records

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create DynamicFrame from Glue Data Catalog
medicare = glueContext.create_dynamic_frame.from_options(
    "s3",
    {
        "paths": [
            "s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv"
        ]
```

```
    },
    "csv",
    {"withHeader": True},
)

# Create filtered DynamicFrame with custom lambda
# to filter records by Provider State and Provider City
sac_or_mon = medicare.filter(
    f=lambda x: x["Provider State"] in ["CA", "AL"]
    and x["Provider City"] in ["SACRAMENTO", "MONTGOMERY"]
)

# Compare record counts
print("Unfiltered record count: ", medicare.count())
print("Filtered record count:  ", sac_or_mon.count())
```

輸出

```
Unfiltered record count:  163065
Filtered record count:   564
```

join

join(paths1, paths2, frame2, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

執行與其他 DynamicFrame 的對等性加入，並傳回產生的 DynamicFrame。

- paths1 – 要加入的此框架中的金鑰清單。
- paths2 – 要加入的其他框架中的金鑰清單。
- frame2 – 要加入的其他 DynamicFrame。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與此轉換回報錯誤關聯的字串 (選用)。
- stageThreshold – 此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。
- totalThreshold –直到及包含此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。

範例：使用聯結合併 **DynamicFrames**

這個例子使用該join方法在三個上執行聯接DynamicFrames。 AWS Glue 會根據您提供的欄位金鑰執行聯結。產生的 DynamicFrame 包含兩個原始影格的列，其中指定之索引鍵相符。

請注意，join 轉換會保持所有欄位不變。這意味著您指定要匹配的字段會顯示在結果中 DynamicFrame，即使它們是多餘的並且包含相同的鍵也是如此。在此範例中，我們使用 drop_fields 在聯結後移除這些多餘的索引鍵。

> **Note**
>
> 若要存取此範例中使用的資料集，請參閱 程式碼範例：加入和關聯化資料 並依照 步驟 1：在 Amazon S3 儲存貯體中網路爬取資料 中的說明進行。

```
# Example: Use join to combine data from three DynamicFrames

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Load DynamicFrames from Glue Data Catalog
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
)
memberships = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="memberships_json"
)
orgs = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="organizations_json"
)
print("Schema for the persons DynamicFrame:")
persons.printSchema()
print("Schema for the memberships DynamicFrame:")
memberships.printSchema()
print("Schema for the orgs DynamicFrame:")
orgs.printSchema()

# Join persons and memberships by ID
```

```
persons_memberships = persons.join(
    paths1=["id"], paths2=["person_id"], frame2=memberships
)

# Rename and drop fields from orgs
# to prevent field name collisions with persons_memberships
orgs = (
    orgs.drop_fields(["other_names", "identifiers"])
    .rename_field("id", "org_id")
    .rename_field("name", "org_name")
)

# Create final join of all three DynamicFrames
legislators_combined = orgs.join(
    paths1=["org_id"], paths2=["organization_id"], frame2=persons_memberships
).drop_fields(["person_id", "org_id"])

# Inspect the schema for the joined data
print("Schema for the new legislators_combined DynamicFrame:")
legislators_combined.printSchema()
```

輸出

```
Schema for the persons DynamicFrame:
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
```

```
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string

Schema for the memberships DynamicFrame:
root
|-- area_id: string
|-- on_behalf_of_id: string
|-- organization_id: string
|-- role: string
|-- person_id: string
|-- legislative_period_id: string
|-- start_date: string
|-- end_date: string

Schema for the orgs DynamicFrame:
root
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- id: string
|-- classification: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- image: string
|-- seats: int
|-- type: string
```

```
Schema for the new legislators_combined DynamicFrame:
root
|-- role: string
|-- seats: int
|-- org_name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- type: string
|-- sort_name: string
|-- area_id: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- on_behalf_of_id: string
|-- other_names: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- name: string
|    |    |-- lang: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- name: string
|-- birth_date: string
|-- organization_id: string
|-- gender: string
|-- classification: string
|-- legislative_period_id: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- image: string
|-- given_name: string
|-- start_date: string
|-- family_name: string
|-- id: string
|-- death_date: string
|-- end_date: string
```

map

`map(f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)`

傳回套用指定映射函數至原始 DynamicFrame 中所有記錄而產生的新 DynamicFrame。

- f – 套用到 DynamicFrame 中所有記錄的映射函數。此函數必須以 DynamicRecord 做為引數，並傳回新的 DynamicRecord (必要)。

  DynamicRecord 代表 DynamicFrame 中的邏輯記錄。它類似 Apache Spark DataFrame 中的一列，除了它是自我描述的，以及可用於不符合固定結構描述的資料。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與轉換中的錯誤相關的字串 (選用)。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用)。預設為零。
- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用)。預設為零。

範例：使用 map 將函數套用至 **DynamicFrame** 中的每個記錄

此範例示範如何使用 map 方法將函數套用至 DynamicFrame 的每個記錄。具體來說，此範例套用名為 MergeAddress 函數至每個記錄，以便將多個地址欄位合併為一個 struct 類型。

> **Note**
>
> 若要存取此範例中使用的資料集，請參閱 程式碼範例：使用 ResolveChoice Lambda 和 ApplyMapping 並依照 步驟 1：在 Amazon S3 儲存貯體中網路爬取資料 中的說明進行。

```
# Example: Use map to combine fields in all records
# of a DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create a DynamicFrame and view its schema
medicare = glueContext.create_dynamic_frame.from_options(
```

```
        "s3",
        {"paths": ["s3://awsglue-datasets/examples/medicare/
Medicare_Hospital_Provider.csv"]},
        "csv",
        {"withHeader": True})
print("Schema for medicare DynamicFrame:")
medicare.printSchema()

# Define a function to supply to the map transform
# that merges address fields into a single field
def MergeAddress(rec):
  rec["Address"] = {}
  rec["Address"]["Street"] = rec["Provider Street Address"]
  rec["Address"]["City"] = rec["Provider City"]
  rec["Address"]["State"] = rec["Provider State"]
  rec["Address"]["Zip.Code"] = rec["Provider Zip Code"]
  rec["Address"]["Array"] = [rec["Provider Street Address"], rec["Provider City"],
 rec["Provider State"], rec["Provider Zip Code"]]
  del rec["Provider Street Address"]
  del rec["Provider City"]
  del rec["Provider State"]
  del rec["Provider Zip Code"]
  return rec


# Use map to apply MergeAddress to every record
mapped_medicare = medicare.map(f = MergeAddress)
print("Schema for mapped_medicare DynamicFrame:")
mapped_medicare.printSchema()
```

輸出

```
Schema for medicare DynamicFrame:
root
|-- DRG Definition: string
|-- Provider Id: string
|-- Provider Name: string
|-- Provider Street Address: string
|-- Provider City: string
|-- Provider State: string
|-- Provider Zip Code: string
|-- Hospital Referral Region Description: string
|-- Total Discharges: string
```

```
|-- Average Covered Charges: string
|-- Average Total Payments: string
|-- Average Medicare Payments: string

Schema for mapped_medicare DynamicFrame:
root
|-- Average Total Payments: string
|-- Average Covered Charges: string
|-- DRG Definition: string
|-- Average Medicare Payments: string
|-- Hospital Referral Region Description: string
|-- Address: struct
|    |-- Zip.Code: string
|    |-- City: string
|    |-- Array: array
|    |    |-- element: string
|    |-- State: string
|    |-- Street: string
|-- Provider Id: string
|-- Total Discharges: string
|-- Provider Name: string
```

mergeDynamicFrame

mergeDynamicFrame(stage_dynamic_frame, primary_keys, transformation_ctx = "", options = {}, info = "", stageThreshold = 0, totalThreshold = 0)

根據指定的主索引鍵來合併此 DynamicFrame 與暫存 DynamicFrame 以識別記錄。重複的記錄 (具有相同主索引鍵的記錄) 不會被刪除重複資料。如果暫存影格中沒有相符的記錄，則會保留來源中的所有記錄 (包括重複項)。如果暫存影格具有相符的記錄，則暫存影格中的記錄會覆寫 AWS Glue 中來源的記錄。

- stage_dynamic_frame – 要合併的暫存 DynamicFrame。
- primary_keys - 要從來源和暫存動態影格比對記錄的主索引鍵欄位清單。
- transformation_ctx - 用來擷取目前轉換之中繼資料的唯一字串 (選用)。
- options - JSON 名稱值組的字串，可提供此轉換的額外資料。目前未使用此引數。
- info – String。與轉換中的錯誤相關的任何字串。
- stageThreshold – Long。在給定轉換中的錯誤數量，其處理需要輸出錯誤。
- totalThreshold – Long。在此轉換之前 (包括在此轉換中) 的錯誤總數，其處理需要輸出錯誤。

此方法會傳回透過將此 DynamicFrame 與暫存 DynamicFrame 合併而取得的新 DynamicFrame。

在下列情況下，傳回的 DynamicFrame 包含記錄 A：

- 如果 A 同時存在於來源影格和暫存影格，則會傳回暫存影格中的 A。
- 如果 A 位於來源資料表中而 A.primaryKeys 不在 stagingDynamicFrame 中，則 A 不會在暫存資料表中更新。

來源影格和暫存影格不需要具有相同的結構描述。

示例：用 mergeDynamicFrame 於**DynamicFrames**根據主鍵合併兩個

下列程式碼範例示範如何使用 mergeDynamicFrame 方法，根據主索引鍵 id 將 DynamicFrame 與「暫存」DynamicFrame 合併。

範例資料集

該範例使用稱為 split_rows_collection 的來自 DynamicFrameCollection 的兩個 DynamicFrames。以下是 split_rows_collection 中的索引鍵清單。

```
dict_keys(['high', 'low'])
```

範例程式碼

```
# Example: Use mergeDynamicFrame to merge DynamicFrames
# based on a set of specified primary keys

from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import SelectFromCollection

# Inspect the original DynamicFrames
frame_low = SelectFromCollection.apply(dfc=split_rows_collection, key="low")
print("Inspect the DynamicFrame that contains rows where ID < 10")
frame_low.toDF().show()

frame_high = SelectFromCollection.apply(dfc=split_rows_collection, key="high")
print("Inspect the DynamicFrame that contains rows where ID > 10")
frame_high.toDF().show()

# Merge the DynamicFrames based on the "id" primary key
```

```
merged_high_low = frame_high.mergeDynamicFrame(
    stage_dynamic_frame=frame_low, primary_keys=["id"]
)

# View the results where the ID is 1 or 20
print("Inspect the merged DynamicFrame that contains the combined rows")
merged_high_low.toDF().where("id = 1 or id= 20").orderBy("id").show()
```

輸出

```
Inspect the DynamicFrame that contains rows where ID < 10
+---+-----+-----------------------+-----------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-----------------------+
|  1|    0|                    fax|           202-225-3307|
|  1|    1|                  phone|           202-225-5731|
|  2|    0|                    fax|           202-225-3307|
|  2|    1|                  phone|           202-225-5731|
|  3|    0|                    fax|           202-225-3307|
|  3|    1|                  phone|           202-225-5731|
|  4|    0|                    fax|           202-225-3307|
|  4|    1|                  phone|           202-225-5731|
|  5|    0|                    fax|           202-225-3307|
|  5|    1|                  phone|           202-225-5731|
|  6|    0|                    fax|           202-225-3307|
|  6|    1|                  phone|           202-225-5731|
|  7|    0|                    fax|           202-225-3307|
|  7|    1|                  phone|           202-225-5731|
|  8|    0|                    fax|           202-225-3307|
|  8|    1|                  phone|           202-225-5731|
|  9|    0|                    fax|           202-225-3307|
|  9|    1|                  phone|           202-225-5731|
| 10|    0|                    fax|           202-225-6328|
| 10|    1|                  phone|           202-225-4576|
+---+-----+-----------------------+-----------------------+
only showing top 20 rows

Inspect the DynamicFrame that contains rows where ID > 10
+---+-----+-----------------------+-----------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-----------------------+
| 11|    0|                    fax|           202-225-6328|
| 11|    1|                  phone|           202-225-4576|
```

```
| 11|    2|                  twitter|           RepTrentFranks|
| 12|    0|                      fax|             202-225-6328|
| 12|    1|                    phone|             202-225-4576|
| 12|    2|                  twitter|           RepTrentFranks|
| 13|    0|                      fax|             202-225-6328|
| 13|    1|                    phone|             202-225-4576|
| 13|    2|                  twitter|           RepTrentFranks|
| 14|    0|                      fax|             202-225-6328|
| 14|    1|                    phone|             202-225-4576|
| 14|    2|                  twitter|           RepTrentFranks|
| 15|    0|                      fax|             202-225-6328|
| 15|    1|                    phone|             202-225-4576|
| 15|    2|                  twitter|           RepTrentFranks|
| 16|    0|                      fax|             202-225-6328|
| 16|    1|                    phone|             202-225-4576|
| 16|    2|                  twitter|           RepTrentFranks|
| 17|    0|                      fax|             202-225-6328|
| 17|    1|                    phone|             202-225-4576|
+---+-----+-----------------------+-------------------------+
only showing top 20 rows

Inspect the merged DynamicFrame that contains the combined rows
+---+-----+-----------------------+-------------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-------------------------+
|  1|    0|                      fax|             202-225-3307|
|  1|    1|                    phone|             202-225-5731|
| 20|    0|                      fax|             202-225-5604|
| 20|    1|                    phone|             202-225-6536|
| 20|    2|                  twitter|                USRepLong|
+---+-----+-----------------------+-------------------------+
```

關聯化

relationalize(root_table_name, staging_path, options, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

將 DynamicFrame 轉換為適合關聯式資料庫的表單。當您想要將資料從 DynamoDB 等 NoSQL 環境移動到 MySQL 等關聯式資料庫時，關聯化 DynamicFrame 特別有用。

透過對巢狀化欄解除巢狀化並將陣列欄直轉橫，可產生框架清單。使用在解除巢狀化階段中所產生的聯結鍵，將直轉橫的陣列欄聯結至根資料表。

- root_table_name – 根資料表的名稱。
- staging_path – 該方法用來以 CSV 格式存放直轉橫資料表分區的路徑 (選用)。直轉橫資料表從這個路徑讀回。
- options – 選用參數的字典。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與此轉換回報錯誤關聯的字串 (選用)。
- stageThreshold – 此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。
- totalThreshold –直到及包含此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。

範例：使用 relationalize 來壓平合併 **DynamicFrame** 中的巢狀化結構描述

此程式碼範例使用 relationalize 方法，將巢狀化結構描述壓平合併為適合關聯式資料庫的表單。

範例資料集

此範例會將稱為 legislators_combined 的 DynamicFrame 與下列結構描述搭配使用。legislators_combined 具有多個巢狀化欄位，例如 links、images 和 contact_details，這些欄位將由 relationalize 轉換進行壓平合併。

```
root
|-- role: string
|-- seats: int
|-- org_name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- type: string
|-- sort_name: string
|-- area_id: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- on_behalf_of_id: string
|-- other_names: array
|    |-- element: struct
|    |    |-- note: string
```

```
|    |    |-- name: string
|    |    |-- lang: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- name: string
|-- birth_date: string
|-- organization_id: string
|-- gender: string
|-- classification: string
|-- legislative_period_id: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- image: string
|-- given_name: string
|-- start_date: string
|-- family_name: string
|-- id: string
|-- death_date: string
|-- end_date: string
```

範例程式碼

```
# Example: Use relationalize to flatten
# a nested schema into a format that fits
# into a relational database.
# Replace DOC-EXAMPLE-S3-BUCKET/tmpDir with your own location.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Apply relationalize and inspect new tables
legislators_relationalized = legislators_combined.relationalize(
    "l_root", "s3://DOC-EXAMPLE-BUCKET/tmpDir"
)
legislators_relationalized.keys()
```

```
# Compare the schema of the contact_details
# nested field to the new relationalized table that
# represents it
legislators_combined.select_fields("contact_details").printSchema()
legislators_relationalized.select("l_root_contact_details").toDF().where(
    "id = 10 or id = 75"
).orderBy(["id", "index"]).show()
```

輸出

下列輸出可讓您將稱為 contact_details 的巢狀化欄位結構描述與 relationalize 轉換所建立的資料表進行比較。請注意，資料表記錄使用稱為 id 的外部索引鍵和代表陣列位置的 index 資料欄連結回主資料表。

```
dict_keys(['l_root', 'l_root_images', 'l_root_links', 'l_root_other_names',
 'l_root_contact_details', 'l_root_identifiers'])

root
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string

+---+-----+-----------------------+-------------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-------------------------+
| 10|    0|                    fax|             202-225-4160|
| 10|    1|                  phone|             202-225-3436|
| 75|    0|                    fax|             202-225-6791|
| 75|    1|                  phone|             202-225-2861|
| 75|    2|                twitter|               RepSamFarr|
+---+-----+-----------------------+-------------------------+
```

rename_field

rename_field(oldName, newName, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

重新命名此 DynamicFrame 中的欄位，並傳回欄位重新命名的新 DynamicFrame。

- oldName – 要重新命名之節點的完整路徑。

如果舊名稱內有小點，RenameField 無法正常運作，除非在前後加上反引號 (`)。例如，若要將 this.old.name 換成 thisNewName，可以用下列方式呼叫 rename_field。

```
newDyF = oldDyF.rename_field("`this.old.name`", "thisNewName")
```

- newName – 新的名稱，做為完整路徑。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與此轉換回報錯誤關聯的字串 (選用)。
- stageThreshold – 此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。
- totalThreshold –直到及包含此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。

範例：使用 rename_field 重新命名 **DynamicFrame** 中的欄位

此程式碼範例會使用 rename_field 方法重新命名 DynamicFrame 中的欄位。請注意，此範例使用方法鏈結同時重新命名多個欄位。

> **Note**
>
> 若要存取此範例中使用的資料集，請參閱 程式碼範例：加入和關聯化資料 並依照 步驟 1：在 Amazon S3 儲存貯體中網路爬取資料 中的說明進行。

範例程式碼

```
# Example: Use rename_field to rename fields
# in a DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Inspect the original orgs schema
orgs = glueContext.create_dynamic_frame.from_catalog(
```

```
    database="legislators", table_name="organizations_json"
)
print("Original orgs schema: ")
orgs.printSchema()

# Rename fields and view the new schema
orgs = orgs.rename_field("id", "org_id").rename_field("name", "org_name")
print("New orgs schema with renamed fields: ")
orgs.printSchema()
```

輸出

```
Original orgs schema:
root
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- id: string
|-- classification: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- image: string
|-- seats: int
|-- type: string

New orgs schema with renamed fields:
root
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
```

```
|    |    |-- note: string
|    |    |-- name: string
|-- classification: string
|-- org_id: string
|-- org_name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- image: string
|-- seats: int
|-- type: string
```

resolveChoice

resolveChoice(specs = None, choice = "" , database = None , table_name = None , transformation_ctx="", info="", stageThreshold=0, totalThreshold=0, catalog_id = None)

在此 DynamicFrame 中解析所選類型，並傳回新的 DynamicFrame。

- specs – 要解析的特定模棱兩可項目的清單，形式皆為 tuple：(field_path, action)。

  有兩種方式可以使用 resolveChoice。第一種是使用 specs 引數指定一系列的特定的欄以及解析它們的方式。resolveChoice 的其他模式是使用 choice 引數為所有 ChoiceTypes 指定單一解析度。

  specs 的值指定為由 (field_path, action) 對組成的元組。field_path 值代表模棱兩可的特定元素，action 值則代表對應的解析動作。可行的動作如下：

  - cast:*type* - 嘗試將所有值轉換至指定類型。例如：cast:int。
  - make_cols - 將每個不同的類型轉換為具有 *columnName_type* 名稱的欄。透過將資料壓平合併來解析可能的模棱兩可項目。例如，如果 columnA 可能是 int 或 string，則在得出的 DynamicFrame 中，解析動作會產生名為 columnA_int 和 columnA_string 的兩個欄。
  - make_struct – 藉由使用 struct 表示資料，來解決可能的模棱兩可項目。舉例來說，如果欄中的資料可能是 int 或 string，則 make_struct 動作會在產生的 DynamicFrame 中產生結構欄。每個結構都包含 int 和 string。
  - project:*type* - 藉由將所有資料預測為一種可能的資料類型，來解決可能的模棱兩可項目。舉例來說，如果欄中的資料可能是 int 或 string，則使用 project:string 動作會在結果的 DynamicFrame 中產生欄，其中所有的 int 值皆轉換為字串。

若 field_path 識別到陣列，在陣列的名稱後放置空白的方括號以避免模棱兩可的狀況。例如，假設您使用如下結構化的資料：

```
"myList": [
  { "price": 100.00 },
  { "price": "$100.00" }
]
```

您可以選取數值而不是價格字串版本，方法是將 field_path 設定為 "myList[].price"，且將 action 設定為 "cast:double"。

> **Note**
>
> 您只能使用 specs 和 choice 參數的其中一項。如果 specs 參數不是 None，則 choice 參數必須為空字串。相反地，如果 choice 不是空字串，則 specs 參數必須為 None。

- choice – 為所有 ChoiceTypes 指定單一解析度。您可以在 ChoiceTypes 的完整清單在執行時間之前是未知的情況下使用此模式。除了以上列出的 specs 動作，此引數也支援下列動作：
  - match_catalog – 嘗試將每個 ChoiceType 投射至指定 Data Catalog 資料表中的對應類型。
- database – 搭配 match_catalog 動作使用的 Data Catalog 資料庫。
- table_name – 搭配 match_catalog 動作使用的 Data Catalog 資料表。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與此轉換回報錯誤關聯的字串 (選用)。
- stageThreshold – 此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。
- totalThreshold – 直到及包含此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設為零，表示流程不會錯誤輸出。
- catalog_id – 要存取之 Data Catalog 的目錄 ID (Data Catalog 的帳戶 ID)。當設定為 None (預設值) 時，它會使用呼叫帳戶的目錄 ID。

範例：使用 resolveChoice 來處理包含多種類型的資料欄

此程式碼範例會使用 resolveChoice 方法來指定如何處理包含多種類型值的 DynamicFrame 資料欄。該範例演示了處理具有不同類型欄的兩種常見方法：

- 將資料欄轉換為單一資料類型。

- 將所有類型保留在單獨的欄中。

範例資料集

> **Note**
>
> 若要存取此範例中使用的資料集，請參閱 程式碼範例：使用 ResolveChoice Lambda 和 ApplyMapping 並依照 步驟 1：在 Amazon S3 儲存貯體中網路爬取資料 中的說明進行。

此範例將稱為 `medicare` 的 `DynamicFrame` 與下列結構描述搭配使用：

```
root
|-- drg definition: string
|-- provider id: choice
|    |-- long
|    |-- string
|-- provider name: string
|-- provider street address: string
|-- provider city: string
|-- provider state: string
|-- provider zip code: long
|-- hospital referral region description: string
|-- total discharges: long
|-- average covered charges: string
|-- average total payments: string
|-- average medicare payments: string
```

範例程式碼

```
# Example: Use resolveChoice to handle
# a column that contains multiple types

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Load the input data and inspect the "provider id" column
```

```
medicare = glueContext.create_dynamic_frame.from_catalog(
    database="payments", table_name="medicare_hospital_provider_csv"
)
print("Inspect the provider id column:")
medicare.toDF().select("provider id").show()

# Cast provider id to type long
medicare_resolved_long = medicare.resolveChoice(specs=[("provider id", "cast:long")])
print("Schema after casting provider id to type long:")
medicare_resolved_long.printSchema()
medicare_resolved_long.toDF().select("provider id").show()

# Create separate columns
# for each provider id type
medicare_resolved_cols = medicare.resolveChoice(choice="make_cols")
print("Schema after creating separate columns for each type:")
medicare_resolved_cols.printSchema()
medicare_resolved_cols.toDF().select("provider id_long", "provider id_string").show()
```

輸出

```
Inspect the 'provider id' column:
+-----------+
|provider id|
+-----------+
|   [10001,]|
|   [10005,]|
|   [10006,]|
|   [10011,]|
|   [10016,]|
|   [10023,]|
|   [10029,]|
|   [10033,]|
|   [10039,]|
|   [10040,]|
|   [10046,]|
|   [10055,]|
|   [10056,]|
|   [10078,]|
|   [10083,]|
|   [10085,]|
|   [10090,]|
|   [10092,]|
```

```
|   [10100,]|
|   [10103,]|
+-----------+
only showing top 20 rows

Schema after casting 'provider id' to type long:
root
|-- drg definition: string
|-- provider id: long
|-- provider name: string
|-- provider street address: string
|-- provider city: string
|-- provider state: string
|-- provider zip code: long
|-- hospital referral region description: string
|-- total discharges: long
|-- average covered charges: string
|-- average total payments: string
|-- average medicare payments: string

+-----------+
|provider id|
+-----------+
|      10001|
|      10005|
|      10006|
|      10011|
|      10016|
|      10023|
|      10029|
|      10033|
|      10039|
|      10040|
|      10046|
|      10055|
|      10056|
|      10078|
|      10083|
|      10085|
|      10090|
|      10092|
|      10100|
|      10103|
+-----------+
```

```
only showing top 20 rows

Schema after creating separate columns for each type:
root
|-- drg definition: string
|-- provider id_string: string
|-- provider id_long: long
|-- provider name: string
|-- provider street address: string
|-- provider city: string
|-- provider state: string
|-- provider zip code: long
|-- hospital referral region description: string
|-- total discharges: long
|-- average covered charges: string
|-- average total payments: string
|-- average medicare payments: string

+----------------+------------------+
|provider id_long|provider id_string|
+----------------+------------------+
|           10001|              null|
|           10005|              null|
|           10006|              null|
|           10011|              null|
|           10016|              null|
|           10023|              null|
|           10029|              null|
|           10033|              null|
|           10039|              null|
|           10040|              null|
|           10046|              null|
|           10055|              null|
|           10056|              null|
|           10078|              null|
|           10083|              null|
|           10085|              null|
|           10090|              null|
|           10092|              null|
|           10100|              null|
|           10103|              null|
+----------------+------------------+
only showing top 20 rows
```

select_fields

`select_fields(paths, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)`

傳回包含所選欄位的新 DynamicFrame。

- `paths` – 字串清單。每個字串清單均為您想要選擇的最上層節點的路徑。
- `transformation_ctx` – 用於識別狀態資訊的唯一字串 (選用)。
- `info` – 與此轉換回報錯誤關聯的字串 (選用)。
- `stageThreshold` – 此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。
- `totalThreshold` –直到及包含此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。

範例：使用 select_fields 來用所選欄位建立新的 **DynamicFrame**

以下程式碼範例顯示如何使用 `select_fields` 方法建立新的 DynamicFrame，其具有從現有 DynamicFrame 中選取的欄位清單。

> **Note**
>
> 若要存取此範例中使用的資料集，請參閱 程式碼範例：加入和關聯化資料 並依照 步驟 1：在 Amazon S3 儲存貯體中網路爬取資料 中的說明進行。

```
# Example: Use select_fields to select specific fields from a DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create a DynamicFrame and view its schema
persons = glueContext.create_dynamic_frame.from_catalog(
```

```
    database="legislators", table_name="persons_json"
)
print("Schema for the persons DynamicFrame:")
persons.printSchema()

# Create a new DynamicFrame with chosen fields
names = persons.select_fields(paths=["family_name", "given_name"])
print("Schema for the names DynamicFrame, created with select_fields:")
names.printSchema()
names.toDF().show()
```

輸出

```
Schema for the persons DynamicFrame:
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
```

```
|-- death_date: string

Schema for the names DynamicFrame:
root
|-- family_name: string
|-- given_name: string

+-----------+----------+
|family_name|given_name|
+-----------+----------+
|    Collins|   Michael|
|   Huizenga|      Bill|
|    Clawson|    Curtis|
|    Solomon|    Gerald|
|     Rigell|    Edward|
|      Crapo|   Michael|
|      Hutto|      Earl|
|      Ertel|     Allen|
|     Minish|    Joseph|
|    Andrews|    Robert|
|     Walden|      Greg|
|      Kazen|   Abraham|
|     Turner|   Michael|
|      Kolbe|     James|
|  Lowenthal|      Alan|
|    Capuano|   Michael|
|   Schrader|      Kurt|
|     Nadler|   Jerrold|
|     Graves|       Tom|
|   McMillan|      John|
+-----------+----------+
only showing top 20 rows
```

簡化日常生活

simplify_ddb_json(): DynamicFrame

簡化中特別位於 DynamoDB JSON 結構中的巢狀資料行，並傳回新的簡化。DynamicFrame DynamicFrame如果 List 類型中有多種類型或 Map 類型，則 List 中的元素將不會簡化。請注意，這是一種特定類型的轉換，其行為與一般unnest轉換不同，且需要資料已經位於 DynamoDB JSON 結構中。如需詳細資訊，請參閱 DynamoDB JSON。

例如，讀取 DynamoDB JSON 結構的匯出結構描述與以下類似：

```
root
|-- Item: struct
|    |-- parentMap: struct
|    |    |-- M: struct
|    |    |    |-- childMap: struct
|    |    |    |    |-- M: struct
|    |    |    |    |    |-- appName: struct
|    |    |    |    |    |    |-- S: string
|    |    |    |    |    |-- packageName: struct
|    |    |    |    |    |    |-- S: string
|    |    |    |    |    |-- updatedAt: struct
|    |    |    |    |    |    |-- N: string
|    |-- strings: struct
|    |    |-- SS: array
|    |    |    |-- element: string
|    |-- numbers: struct
|    |    |-- NS: array
|    |    |    |-- element: string
|    |-- binaries: struct
|    |    |-- BS: array
|    |    |    |-- element: string
|    |-- isDDBJson: struct
|    |    |-- BOOL: boolean
|    |-- nullValue: struct
|    |    |-- NULL: boolean
```

`simplify_ddb_json()` 轉換會將此轉換為：

```
root
|-- parentMap: struct
|    |-- childMap: struct
|    |    |-- appName: string
|    |    |-- packageName: string
|    |    |-- updatedAt: string
|-- strings: array
|    |-- element: string
|-- numbers: array
|    |-- element: string
|-- binaries: array
|    |-- element: string
|-- isDDBJson: boolean
```

```
|-- nullValue: null
```

範例：使用簡化方式來叫 DynamoDB JSON 簡化

此程式碼範例會使用此simplify_ddb_json方法來使用 AWS Glue DynamoDB 匯出連接器、叫用 DynamoDB JSON 簡化作業，以及列印分割區數目。

範例程式碼

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext()
glueContext = GlueContext(sc)

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type = "dynamodb",
    connection_options = {
        'dynamodb.export': 'ddb',
        'dynamodb.tableArn': '<table arn>',
        'dynamodb.s3.bucket': '<bucket name>',
        'dynamodb.s3.prefix': '<bucket prefix>',
        'dynamodb.s3.bucketOwner': '<account_id of bucket>'
    }
)
simplified = dynamicFrame.simplify_ddb_json()
print(simplified.getNumPartitions())
```

spigot

spigot(path, options={})

將範例記錄寫入指定的目的地，以協助您驗證任務執行的轉換。

- path - 要寫入的目的地路徑 (必要)。
- options – 指定選項的索引鍵/值對 (選用)。"topk" 選項指定應寫入第一個 k 記錄。"prob" 選項指定選擇任何給定記錄的概率 (小數)。您可以使用其來選擇要寫入的記錄。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。

範例：使用 spigot 將範例欄位從 **DynamicFrame** 寫入到 Amazon S3

此程式碼範例會在套用 `select_fields` 轉換後，使用 `spigot` 方法將範例記錄寫入 Amazon S3 儲存貯體。

範例資料集

> **Note**
>
> 若要存取此範例中使用的資料集，請參閱 程式碼範例：加入和關聯化資料 並依照 步驟 1：在 Amazon S3 儲存貯體中網路爬取資料 中的說明進行。

此範例將稱為 `persons` 的 DynamicFrame 與下列結構描述搭配使用：

```
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
```

```
|    |    |-- value: string
|-- death_date: string
```

範例程式碼

```
# Example: Use spigot to write sample records
# to a destination during a transformation
# from pyspark.context import SparkContext.
# Replace DOC-EXAMPLE-BUCKET with your own location.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Load table data into a DynamicFrame
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
)

# Perform the select_fields on the DynamicFrame
persons = persons.select_fields(paths=["family_name", "given_name", "birth_date"])

# Use spigot to write a sample of the transformed data
# (the first 10 records)
spigot_output = persons.spigot(
    path="s3://DOC-EXAMPLE-BUCKET", options={"topk": 10}
)
```

輸出

以下是 spigot 寫入 Amazon S3 的資料範例。由於範例程式碼指定了 options={"topk": 10}，範例資料會包含前 10 筆記錄。

```
{"family_name":"Collins","given_name":"Michael","birth_date":"1944-10-15"}
{"family_name":"Huizenga","given_name":"Bill","birth_date":"1969-01-31"}
{"family_name":"Clawson","given_name":"Curtis","birth_date":"1959-09-28"}
{"family_name":"Solomon","given_name":"Gerald","birth_date":"1930-08-14"}
{"family_name":"Rigell","given_name":"Edward","birth_date":"1960-05-28"}
{"family_name":"Crapo","given_name":"Michael","birth_date":"1951-05-20"}
{"family_name":"Hutto","given_name":"Earl","birth_date":"1926-05-12"}
```

```
{"family_name":"Ertel","given_name":"Allen","birth_date":"1937-11-07"}
{"family_name":"Minish","given_name":"Joseph","birth_date":"1916-09-01"}
{"family_name":"Andrews","given_name":"Robert","birth_date":"1957-08-04"}
```

split_fields

split_fields(paths, name1, name2, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

傳回新的 DynamicFrameCollection，其包含兩個 DynamicFrames。第一個 DynamicFrame 包含分割的所有節點，第二個包含其餘節點。

- paths – 字串清單，其各自為想要分割到新 DynamicFrame 的節點的完整路徑。
- name1 – 分割的 DynamicFrame 的名稱字串。
- name2 – 分割指定節點後剩餘的 DynamicFrame 的名稱字串。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與此轉換回報錯誤關聯的字串 (選用)。
- stageThreshold – 此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。
- totalThreshold –直到及包含此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。

範例：使用 split_fields 將選取的欄位分割為單獨的 **DynamicFrame**

此程式碼範例會使用 split_fields 方法，將指定欄位的清單分割為單獨的 DynamicFrame。

範例資料集

該範例使用稱為 l_root_contact_details 的 DynamicFrame，其來自名為 legislators_relationalized 的集合。

l_root_contact_details 具有以下結構描述和項目。

```
root
|-- id: long
|-- index: int
|-- contact_details.val.type: string
```

```
|-- contact_details.val.value: string

+---+-----+-----------------------+-------------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-------------------------+
|  1|    0|                  phone|             202-225-5265|
|  1|    1|                twitter|              kathyhochul|
|  2|    0|                  phone|             202-225-3252|
|  2|    1|                twitter|            repjackyrosen|
|  3|    0|                    fax|             202-225-1314|
|  3|    1|                  phone|             202-225-3772|
...
```

範例程式碼

```
# Example: Use split_fields to split selected
# fields into a separate DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Load the input DynamicFrame and inspect its schema
frame_to_split = legislators_relationalized.select("l_root_contact_details")
print("Inspect the input DynamicFrame schema:")
frame_to_split.printSchema()

# Split id and index fields into a separate DynamicFrame
split_fields_collection = frame_to_split.split_fields(["id", "index"], "left", "right")

# Inspect the resulting DynamicFrames
print("Inspect the schemas of the DynamicFrames created with split_fields:")
split_fields_collection.select("left").printSchema()
split_fields_collection.select("right").printSchema()
```

輸出

```
Inspect the input DynamicFrame's schema:
root
|-- id: long
```

```
|-- index: int
|-- contact_details.val.type: string
|-- contact_details.val.value: string

Inspect the schemas of the DynamicFrames created with split_fields:
root
|-- id: long
|-- index: int

root
|-- contact_details.val.type: string
|-- contact_details.val.value: string
```

split_rows

split_rows(comparison_dict, name1, name2, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

將 DynamicFrame 中一個或多個欄分割成新的 DynamicFrame。

該方法傳回新的 DynamicFrameCollection，其包含兩個 DynamicFrames。第一個 DynamicFrame 包含分割的所有列，第二個包含其餘節列。

- comparison_dict – 一個字典，其中索引鍵為欄位的路徑，而對於與欄位數值相比較的數值而言，此數值為另一種字典映射比較運算子。例如，{"age": {">": 10, "<": 20}} 分割所有資料列，其年齡欄中的值大於 10 且小於 20。
- name1 – 分割的 DynamicFrame 的名稱字串。
- name2 – 分割指定節點後剩餘的 DynamicFrame 的名稱字串。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與此轉換回報錯誤關聯的字串 (選用)。
- stageThreshold – 此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。
- totalThreshold –直到及包含此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。

範例：使用 split_rows 來分割 **DynamicFrame** 中的列

此程式碼範例使用 split_rows 方法，根據 id 欄位值來分割 DynamicFrame 中的列。

範例資料集

該範例使用稱為 l_root_contact_details 的 DynamicFrame，其選自名為 legislators_relationalized 的集合。

l_root_contact_details 具有以下結構描述和項目。

```
root
|-- id: long
|-- index: int
|-- contact_details.val.type: string
|-- contact_details.val.value: string

+---+-----+------------------------+------------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+------------------------+------------------------+
|  1|    0|                   phone|              202-225-5265|
|  1|    1|                 twitter|               kathyhochul|
|  2|    0|                   phone|              202-225-3252|
|  2|    1|                 twitter|             repjackyrosen|
|  3|    0|                     fax|              202-225-1314|
|  3|    1|                   phone|              202-225-3772|
|  3|    2|                 twitter|           MikeRossUpdates|
|  4|    0|                     fax|              202-225-1314|
|  4|    1|                   phone|              202-225-3772|
|  4|    2|                 twitter|           MikeRossUpdates|
|  5|    0|                     fax|              202-225-1314|
|  5|    1|                   phone|              202-225-3772|
|  5|    2|                 twitter|           MikeRossUpdates|
|  6|    0|                     fax|              202-225-1314|
|  6|    1|                   phone|              202-225-3772|
|  6|    2|                 twitter|           MikeRossUpdates|
|  7|    0|                     fax|              202-225-1314|
|  7|    1|                   phone|              202-225-3772|
|  7|    2|                 twitter|           MikeRossUpdates|
|  8|    0|                     fax|              202-225-1314|
+---+-----+------------------------+------------------------+
```

範例程式碼

```
# Example: Use split_rows to split up
# rows in a DynamicFrame based on value
```

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Retrieve the DynamicFrame to split
frame_to_split = legislators_relationalized.select("l_root_contact_details")

# Split up rows by ID
split_rows_collection = frame_to_split.split_rows({"id": {">": 10}}, "high", "low")

# Inspect the resulting DynamicFrames
print("Inspect the DynamicFrame that contains IDs < 10")
split_rows_collection.select("low").toDF().show()
print("Inspect the DynamicFrame that contains IDs > 10")
split_rows_collection.select("high").toDF().show()
```

輸出

```
Inspect the DynamicFrame that contains IDs < 10
+---+-----+-----------------------+-----------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-----------------------+
|  1|    0|                  phone|             202-225-5265|
|  1|    1|                twitter|              kathyhochul|
|  2|    0|                  phone|             202-225-3252|
|  2|    1|                twitter|            repjackyrosen|
|  3|    0|                    fax|             202-225-1314|
|  3|    1|                  phone|             202-225-3772|
|  3|    2|                twitter|          MikeRossUpdates|
|  4|    0|                    fax|             202-225-1314|
|  4|    1|                  phone|             202-225-3772|
|  4|    2|                twitter|          MikeRossUpdates|
|  5|    0|                    fax|             202-225-1314|
|  5|    1|                  phone|             202-225-3772|
|  5|    2|                twitter|          MikeRossUpdates|
|  6|    0|                    fax|             202-225-1314|
|  6|    1|                  phone|             202-225-3772|
|  6|    2|                twitter|          MikeRossUpdates|
|  7|    0|                    fax|             202-225-1314|
|  7|    1|                  phone|             202-225-3772|
```

```
|  7|    2|                 twitter|          MikeRossUpdates|
|  8|    0|                     fax|             202-225-1314|
+---+-----+------------------------+-------------------------+
only showing top 20 rows

Inspect the DynamicFrame that contains IDs > 10
+---+-----+------------------------+-------------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+------------------------+-------------------------+
| 11|    0|                   phone|             202-225-5476|
| 11|    1|                 twitter|            RepDavidYoung|
| 12|    0|                   phone|             202-225-4035|
| 12|    1|                 twitter|           RepStephMurphy|
| 13|    0|                     fax|             202-226-0774|
| 13|    1|                   phone|             202-225-6335|
| 14|    0|                     fax|             202-226-0774|
| 14|    1|                   phone|             202-225-6335|
| 15|    0|                     fax|             202-226-0774|
| 15|    1|                   phone|             202-225-6335|
| 16|    0|                     fax|             202-226-0774|
| 16|    1|                   phone|             202-225-6335|
| 17|    0|                     fax|             202-226-0774|
| 17|    1|                   phone|             202-225-6335|
| 18|    0|                     fax|             202-226-0774|
| 18|    1|                   phone|             202-225-6335|
| 19|    0|                     fax|             202-226-0774|
| 19|    1|                   phone|             202-225-6335|
| 20|    0|                     fax|             202-226-0774|
| 20|    1|                   phone|             202-225-6335|
+---+-----+------------------------+-------------------------+
only showing top 20 rows
```

unbox

unbox(path, format, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0, **options)

將 DynamicFrame 中的字串欄位拆箱 (重新格式化)，並傳回包含拆箱的 DynamicRecords 的新 DynamicFrame。

DynamicRecord 代表 DynamicFrame 中的邏輯記錄。它類似 Apache Spark DataFrame 中的一列，除了它是自我描述的，以及可用於不符合固定結構描述的資料。

- path – 要拆箱之字串節點的完整路徑。
- format – 格式化規格 (選用)。您可將其用於 Amazon S3 或支援多種格式的 AWS Glue 連線。如需了解受支援的格式，請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與此轉換回報錯誤關聯的字串 (選用)。
- stageThreshold – 此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。
- totalThreshold –直到及包含此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。
- options – 下列一或多個：
  - separator – 包含分隔符號字元的字串。
  - escaper – 包含逸出字元的字串。
  - skipFirst – 布林值，指出是否略過第一個執行個體。
  - withSchema：包含節點結構描述的 JSON 表示法的字串。結構描述的 JSON 表示法的格式由 StructType.json() 的輸出定義。
  - withHeader – 布林值，指出是否包含標頭。

範例：使用 unbox 將字串欄位拆箱到結構中

此程式碼範例使用 unbox 方法，將 DynamicFrame 中的字串欄位拆箱或重新格式化為結構類型的欄位。

範例資料集

此範例搭配使用稱為 mapped_with_string 的 DynamicFrame 與下列結構描述和項目。

請注意名為 AddressString 的欄位。這是範例拆箱為結構的欄位。

```
root
|-- Average Total Payments: string
|-- AddressString: string
|-- Average Covered Charges: string
|-- DRG Definition: string
|-- Average Medicare Payments: string
|-- Hospital Referral Region Description: string
|-- Address: struct
|    |-- Zip.Code: string
```

```
|    |-- City: string
|    |-- Array: array
|    |    |-- element: string
|    |-- State: string
|    |-- Street: string
|-- Provider Id: string
|-- Total Discharges: string
|-- Provider Name: string

+----------------------+--------------------+-----------------------
+--------------------+------------------------+-----------------------------------
+--------------------+-----------+---------------+--------------------+
|Average Total Payments|       AddressString|Average Covered Charges|      DRG
 Definition|Average Medicare Payments|Hospital Referral Region Description|
 Address|Provider Id|Total Discharges|       Provider Name|
+----------------------+--------------------+-----------------------
+--------------------+------------------------+-----------------------------------
+--------------------+-----------+---------------+--------------------+
|              $5777.24|{"Street": "1108 ...|             $32963.07|039 -
 EXTRACRANIA...|                $4763.73|                        AL - Dothan|[36301,
 DOTHAN, [...|      10001|             91|SOUTHEAST ALABAMA...|
|              $5787.57|{"Street": "2505 ...|             $15131.85|039 -
 EXTRACRANIA...|                $4976.71|                    AL - Birmingham|[35957,
 BOAZ, [25...|      10005|             14|MARSHALL MEDICAL ...|
|              $5434.95|{"Street": "205 M...|             $37560.37|039 -
 EXTRACRANIA...|                $4453.79|                    AL - Birmingham|[35631,
 FLORENCE,...|      10006|             24|ELIZA COFFEE MEMO...|
|              $5417.56|{"Street": "50 ME...|             $13998.28|039 -
 EXTRACRANIA...|                $4129.16|                    AL - Birmingham|[35235,
 BIRMINGHA...|      10011|             25|   ST VINCENT'S EAST|
...
```

範例程式碼

```
# Example: Use unbox to unbox a string field
# into a struct in a DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
```

```
glueContext = GlueContext(sc)

unboxed = mapped_with_string.unbox("AddressString", "json")
unboxed.printSchema()
unboxed.toDF().show()
```

輸出

```
root
|-- Average Total Payments: string
|-- AddressString: struct
|    |-- Street: string
|    |-- City: string
|    |-- State: string
|    |-- Zip.Code: string
|    |-- Array: array
|    |    |-- element: string
|-- Average Covered Charges: string
|-- DRG Definition: string
|-- Average Medicare Payments: string
|-- Hospital Referral Region Description: string
|-- Address: struct
|    |-- Zip.Code: string
|    |-- City: string
|    |-- Array: array
|    |    |-- element: string
|    |-- State: string
|    |-- Street: string
|-- Provider Id: string
|-- Total Discharges: string
|-- Provider Name: string

+----------------------+--------------------+-----------------------
+--------------------+-------------------------+-----------------------------------
+--------------------+-----------+----------------+--------------------+
|Average Total Payments|       AddressString|Average Covered Charges|      DRG
 Definition|Average Medicare Payments|Hospital Referral Region Description|
 Address|Provider Id|Total Discharges|       Provider Name|
+----------------------+--------------------+-----------------------
+--------------------+-------------------------+-----------------------------------
+--------------------+-----------+----------------+--------------------+
```

```
|              $5777.24|[1108 ROSS CLARK ...|              $32963.07|039 -
 EXTRACRANIA...|                $4763.73|                         AL - Dothan|[36301,
 DOTHAN, [...|      10001|              91|SOUTHEAST ALABAMA...|
|              $5787.57|[2505 U S HIGHWAY...|              $15131.85|039 -
 EXTRACRANIA...|                $4976.71|                     AL - Birmingham|[35957,
 BOAZ, [25...|      10005|              14|MARSHALL MEDICAL ...|
|              $5434.95|[205 MARENGO STRE...|              $37560.37|039 -
 EXTRACRANIA...|                $4453.79|                     AL - Birmingham|[35631,
 FLORENCE,...|      10006|              24|ELIZA COFFEE MEMO...|
|              $5417.56|[50 MEDICAL PARK ...|              $13998.28|039 -
 EXTRACRANIA...|                $4129.16|                     AL - Birmingham|[35235,
 BIRMINGHA...|      10011|              25|   ST VINCENT'S EAST|
|              $5658.33|[1000 FIRST STREE...|              $31633.27|039 -
 EXTRACRANIA...|                $4851.44|                     AL - Birmingham|[35007,
 ALABASTER...|      10016|              18|SHELBY BAPTIST ME...|
|              $6653.80|[2105 EAST SOUTH ...|              $16920.79|039 -
 EXTRACRANIA...|                $5374.14|                     AL - Montgomery|[36116,
 MONTGOMER...|      10023|              67|BAPTIST MEDICAL C...|
|              $5834.74|[2000 PEPPERELL P...|              $11977.13|039 -
 EXTRACRANIA...|                $4761.41|                     AL - Birmingham|[36801,
 OPELIKA, ...|      10029|              51|EAST ALABAMA MEDI...|
|              $8031.12|[619 SOUTH 19TH S...|              $35841.09|039 -
 EXTRACRANIA...|                $5858.50|                     AL - Birmingham|[35233,
 BIRMINGHA...|      10033|              32|UNIVERSITY OF ALA...|
|              $6113.38|[101 SIVLEY RD, H...|              $28523.39|039 -
 EXTRACRANIA...|                $5228.40|                     AL - Huntsville|[35801,
 HUNTSVILL...|      10039|             135| HUNTSVILLE HOSPITAL|
|              $5541.05|[1007 GOODYEAR AV...|              $75233.38|039 -
 EXTRACRANIA...|                $4386.94|                     AL - Birmingham|[35903,
 GADSDEN, ...|      10040|              34|GADSDEN REGIONAL ...|
|              $5461.57|[600 SOUTH THIRD ...|              $67327.92|039 -
 EXTRACRANIA...|                $4493.57|                     AL - Birmingham|[35901,
 GADSDEN, ...|      10046|              14|RIVERVIEW REGIONA...|
|              $5356.28|[4370 WEST MAIN S...|              $39607.28|039 -
 EXTRACRANIA...|                $4408.20|                         AL - Dothan|[36305,
 DOTHAN, [...|      10055|              45|    FLOWERS HOSPITAL|
|              $5374.65|[810 ST VINCENT'S...|              $22862.23|039 -
 EXTRACRANIA...|                $4186.02|                     AL - Birmingham|[35205,
 BIRMINGHA...|      10056|              43|ST VINCENT'S BIRM...|
|              $5366.23|[400 EAST 10TH ST...|              $31110.85|039 -
 EXTRACRANIA...|                $4376.23|                     AL - Birmingham|[36207,
 ANNISTON,...|      10078|              21|NORTHEAST ALABAMA...|
```

```
|              $5282.93|[1613 NORTH MCKEN...|              $25411.33|039 -
 EXTRACRANIA...|                 $4383.73|                        AL - Mobile|[36535,
 FOLEY, [1...|      10083|              15|SOUTH BALDWIN REG...|
|              $5676.55|[1201 7TH STREET ...|               $9234.51|039 -
 EXTRACRANIA...|                 $4509.11|                    AL - Huntsville|[35609,
 DECATUR, ...|      10085|              27|DECATUR GENERAL H...|
|              $5930.11|[6801 AIRPORT BOU...|              $15895.85|039 -
 EXTRACRANIA...|                 $3972.85|                        AL - Mobile|[36608,
 MOBILE, [...|      10090|              27| PROVIDENCE HOSPITAL|
|              $6192.54|[809 UNIVERSITY B...|              $19721.16|039 -
 EXTRACRANIA...|                 $5179.38|                    AL - Tuscaloosa|[35401,
 TUSCALOOS...|      10092|              31|D C H REGIONAL ME...|
|              $4968.00|[750 MORPHY AVENU...|              $10710.88|039 -
 EXTRACRANIA...|                 $3898.88|                        AL - Mobile|[36532,
 FAIRHOPE,...|      10100|              18|     THOMAS HOSPITAL|
|              $5996.00|[701 PRINCETON AV...|              $51343.75|039 -
 EXTRACRANIA...|                 $4962.45|                    AL - Birmingham|[35211,
 BIRMINGHA...|      10103|              33|BAPTIST MEDICAL C...|
+----------------------+--------------------+-----------------------
+--------------------+------------------------+-----------------------------------
+--------------------+-----------+---------------+--------------------+
only showing top 20 rows
```

聯集

union(frame1, frame2, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

聯盟二 DynamicFrames。返回 DynamicFrame 包含來自兩個輸入的所有記錄 DynamicFrames。這種轉換可能會從兩個 DataFrames 與等效數據的聯合返回不同的結果。如果您需要 Spark DataFrame 聯合行為，請考慮使用toDF.

- frame1— 首先 DynamicFrame 是工會。
- frame2— 第二 DynamicFrame 次聯合。
- transformation_ctx – (選用) 用於識別統計資料/狀態資訊的唯一字串
- info – (選用) 與轉換中的錯誤相關的任何字串
- stageThreshold – (選用) 在處理輸出錯誤之前，轉換中的最大錯誤數
- totalThreshold – (選用) 在處理輸出錯誤之前的最大錯誤數。

解巢狀

unnest(transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

對 DynamicFrame 中的巢狀化物件進行解除巢狀化，將其變為頂層元素，並傳回新的未巢狀化 DynamicFrame。

- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與此轉換回報錯誤關聯的字串 (選用)。
- stageThreshold – 此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。
- totalThreshold –直到及包含此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用)。預設值為零，表示此流程不會發生錯誤。

範例：使用 unnest 將巢狀化欄位轉換為頂層欄位

此程式碼範例使用 unnest 方法，將 DynamicFrame 中的所有巢狀化欄位壓平合併為頂層欄位。

範例資料集

此範例搭配使用稱為 mapped_medicare 的 DynamicFrame 與下列結構描述。請注意，Address 欄位是唯一包含巢狀化資料的欄位。

```
root
|-- Average Total Payments: string
|-- Average Covered Charges: string
|-- DRG Definition: string
|-- Average Medicare Payments: string
|-- Hospital Referral Region Description: string
|-- Address: struct
|    |-- Zip.Code: string
|    |-- City: string
|    |-- Array: array
|    |    |-- element: string
|    |-- State: string
|    |-- Street: string
|-- Provider Id: string
|-- Total Discharges: string
|-- Provider Name: string
```

範例程式碼

```
# Example: Use unnest to unnest nested
# objects in a DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Unnest all nested fields
unnested = mapped_medicare.unnest()
unnested.printSchema()
```

輸出

```
root
|-- Average Total Payments: string
|-- Average Covered Charges: string
|-- DRG Definition: string
|-- Average Medicare Payments: string
|-- Hospital Referral Region Description: string
|-- Address.Zip.Code: string
|-- Address.City: string
|-- Address.Array: array
|    |-- element: string
|-- Address.State: string
|-- Address.Street: string
|-- Provider Id: string
|-- Total Discharges: string
|-- Provider Name: string
```

unnest_ddb_json

解除專屬於 DynamoDB JSON 結構中 DynamicFrame 內的巢狀欄的巢狀化，並傳回新的解巢狀 DynamicFrame。結構類型陣列的欄將不是解巢狀狀態。請注意，這是一種特定類型的解除巢狀化轉換，其行為與常規 unnest 轉換不同，且資料必須已經位於 DynamoDB JSON 結構中。如需詳細資訊，請參閱 DynamoDB JSON。

unnest_ddb_json(transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與此轉換回報錯誤關聯的字串 (選用)。
- stageThreshold – 此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用：預設為 0，表示流程不會錯誤輸出)。
- totalThreshold – 直到及包含此轉換期間流程應錯誤輸出之前遇到的錯誤次數 (選用：預設為 0，表示流程不會錯誤輸出)。

例如，讀取 DynamoDB JSON 結構的匯出結構描述與以下類似：

```
root
|-- Item: struct
|    |-- ColA: struct
|    |    |-- S: string
|    |-- ColB: struct
|    |    |-- S: string
|    |-- ColC: struct
|    |    |-- N: string
|    |-- ColD: struct
|    |    |-- L: array
|    |    |    |-- element: null
```

unnest_ddb_json() 轉換會將此轉換為：

```
root
|-- ColA: string
|-- ColB: string
|-- ColC: string
|-- ColD: array
|    |-- element: null
```

下列程式碼範例示範如何使用 AWS Glue DynamoDB 匯出連接器、取消巢狀呼叫 DynamoDB JSON，以及如何列印分割區的數目：

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
```

```
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dynamicFrame = glue_context.create_dynamic_frame.from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.export": "ddb",
        "dynamodb.tableArn": "<test_source>",
        "dynamodb.s3.bucket": "<bucket name>",
        "dynamodb.s3.prefix": "<bucket prefix>",
        "dynamodb.s3.bucketOwner": "<account_id>",
    }
)
unnested = dynamicFrame.unnest_ddb_json()
print(unnested.getNumPartitions())

job.commit()
```

write

write(connection_type, connection_options, format, format_options, accumulator_size)

從此 DynamicFrame 的 GlueContext 類 取得指定連線類型的 DataSink(object)，並用其來格式化及寫入此 DynamicFrame 的內容。傳回依指定格式化和寫入的新 DynamicFrame。

- connection_type – 使用的連線類型。有效值包括 s3、mysql、postgresql、redshift、sqlserver 及 oracle。
- connection_options – 使用的連線選項 (選用)。如果是 connection_type 的 s3，會定義 Amazon S3 路徑。

  ```
  connection_options = {"path": "s3://aws-glue-target/temp"}
  ```

  如果是 JDBC 連線，必須定義幾項屬性。請注意，資料庫名稱必須是 URL 的一部分。它可以選擇性包含在連線選項中。

> **Warning**
>
> 不建議在指令碼中存放密碼。考慮使boto3用從 AWS Secrets Manager 或 AWS Glue 資料型錄擷取它們。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

- `format` – 格式化規格 (選用)。這用於 Amazon Simple Storage Service (Amazon S3) 或支援多種格式的 AWS Glue 連線。請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項 以了解受支援的格式。
- `format_options` – 指定格式的格式選項。請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項 以了解受支援的格式。
- `accumulator_size`：要使用的 accumulable 大小，以位元組為單位 (選用)。

— errors —

- assertErrorThreshold
- errorsAsDynamic框架
- errorsCount
- stageErrorsCount

assertErrorThreshold

`assertErrorThreshold( )` – 建立此 `DynamicFrame` 的轉換中的錯誤宣告。從基礎 `DataFrame` 傳回 `Exception`。

errorsAsDynamic框架

`errorsAsDynamicFrame( )` – 傳回 `DynamicFrame`，其內部有巢狀的錯誤記錄。

範例：使用 errorsAsDynamic Frame 檢視錯誤記錄

以下程式碼範例顯示如何使用 `errorsAsDynamicFrame` 方法來檢視 `DynamicFrame` 的錯誤記錄。

範例資料集

此範例使用下列資料集，您可以將其作為 JSON 上傳到 Amazon S3。請注意，第二條記錄的格式錯誤。當您使用 SparkSQL 時，格式錯誤的資料通常會中斷檔案剖析。但是，DynamicFrame 會辨識出格式錯誤問題，並將格式錯誤的行轉換為可以單獨處理的錯誤記錄。

```
{"id": 1, "name": "george", "surname": "washington", "height": 178}
{"id": 2, "name": "benjamin", "surname": "franklin",
{"id": 3, "name": "alexander", "surname": "hamilton", "height": 171}
{"id": 4, "name": "john", "surname": "jay", "height": 190}
```

範例程式碼

```
# Example: Use errorsAsDynamicFrame to view error records.
# Replace s3://DOC-EXAMPLE-S3-BUCKET/error_data.json with your location.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create errors DynamicFrame, view schema
errors = glueContext.create_dynamic_frame.from_options(
    "s3", {"paths": ["s3://DOC-EXAMPLE-S3-BUCKET/error_data.json"]}, "json"
)
print("Schema of errors DynamicFrame:")
errors.printSchema()

# Show that errors only contains valid entries from the dataset
print("errors contains only valid records from the input dataset (2 of 4 records)")
errors.toDF().show()

# View errors
print("Errors count:", str(errors.errorsCount()))
print("Errors:")
errors.errorsAsDynamicFrame().toDF().show()

# View error fields and error data
error_record = errors.errorsAsDynamicFrame().toDF().head()

error_fields = error_record["error"]
```

```
print("Error fields: ")
print(error_fields.asDict().keys())

print("\nError record data:")
for key in error_fields.asDict().keys():
    print("\n", key, ": ", str(error_fields[key]))
```

輸出

```
Schema of errors DynamicFrame:
root
|-- id: int
|-- name: string
|-- surname: string
|-- height: int

errors contains only valid records from the input dataset (2 of 4 records)
+---+------+----------+------+
| id|  name|   surname|height|
+---+------+----------+------+
|  1|george|washington|   178|
|  4|  john|       jay|   190|
+---+------+----------+------+

Errors count: 1
Errors:
+--------------------+
|               error|
+--------------------+
|[[  File "/tmp/20...|
+--------------------+

Error fields:
dict_keys(['callsite', 'msg', 'stackTrace', 'input', 'bytesread', 'source',
 'dynamicRecord'])

Error record data:

 callsite :  Row(site='  File "/tmp/2060612586885849088", line 549, in <module>\n
 sys.exit(main())\n  File "/tmp/2060612586885849088", line 523, in main\n    response
 = handler(content)\n  File "/tmp/2060612586885849088", line 197, in execute_request
\n    result = node.execute()\n  File "/tmp/2060612586885849088", line 103, in
 execute\n    exec(code, global_dict)\n  File "<stdin>", line 10, in <module>\n
```

```
 File "/opt/amazon/lib/python3.6/site-packages/awsglue/dynamicframe.py", line 625, in
 from_options\n    format_options, transformation_ctx, push_down_predicate, **kwargs)\n
  File "/opt/amazon/lib/python3.6/site-packages/awsglue/context.py", line 233, in
 create_dynamic_frame_from_options\n    source.setFormat(format, **format_options)\n',
 info='')

 msg :  error in jackson reader

 stackTrace :  com.fasterxml.jackson.core.JsonParseException: Unexpected character
 ('{' (code 123)): was expecting either valid name character (for unquoted name) or
 double-quote (for quoted) to start field name
 at [Source: com.amazonaws.services.glue.readers.BufferedStream@73492578; line: 3,
 column: 2]
 at com.fasterxml.jackson.core.JsonParser._constructError(JsonParser.java:1581)
 at
 com.fasterxml.jackson.core.base.ParserMinimalBase._reportError(ParserMinimalBase.java:533)
 at
 com.fasterxml.jackson.core.base.ParserMinimalBase._reportUnexpectedChar(ParserMinimalBase.java
 at
 com.fasterxml.jackson.core.json.UTF8StreamJsonParser._handleOddName(UTF8StreamJsonParser.java:
 at
 com.fasterxml.jackson.core.json.UTF8StreamJsonParser._parseName(UTF8StreamJsonParser.java:1650
 at
 com.fasterxml.jackson.core.json.UTF8StreamJsonParser.nextToken(UTF8StreamJsonParser.java:740)
 at com.amazonaws.services.glue.readers.JacksonReader$$anonfun$hasNextGoodToken
$1.apply(JacksonReader.scala:57)
 at com.amazonaws.services.glue.readers.JacksonReader$$anonfun$hasNextGoodToken
$1.apply(JacksonReader.scala:57)
 at scala.collection.Iterator$$anon$9.next(Iterator.scala:162)
 at scala.collection.Iterator$$anon$16.hasNext(Iterator.scala:599)
 at scala.collection.Iterator$$anon$16.hasNext(Iterator.scala:598)
 at scala.collection.Iterator$class.foreach(Iterator.scala:891)
 at scala.collection.AbstractIterator.foreach(Iterator.scala:1334)
 at com.amazonaws.services.glue.readers.JacksonReader$$anonfun
$1.apply(JacksonReader.scala:120)
 at com.amazonaws.services.glue.readers.JacksonReader$$anonfun
$1.apply(JacksonReader.scala:116)
 at
 com.amazonaws.services.glue.DynamicRecordBuilder.handleErr(DynamicRecordBuilder.scala:209)
 at
 com.amazonaws.services.glue.DynamicRecordBuilder.handleErrorWithException(DynamicRecordBuilder
 at
 com.amazonaws.services.glue.readers.JacksonReader.nextFailSafe(JacksonReader.scala:116)
 at com.amazonaws.services.glue.readers.JacksonReader.next(JacksonReader.scala:109)
```

```
 at com.amazonaws.services.glue.readers.JSONReader.next(JSONReader.scala:247)
 at
 com.amazonaws.services.glue.hadoop.TapeHadoopRecordReaderSplittable.nextKeyValue(TapeHadoopRec
 at org.apache.spark.rdd.NewHadoopRDD$$anon$1.hasNext(NewHadoopRDD.scala:230)
 at org.apache.spark.InterruptibleIterator.hasNext(InterruptibleIterator.scala:37)
 at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:409)
 at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:409)
 at scala.collection.Iterator$$anon$13.hasNext(Iterator.scala:462)
 at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:409)
 at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:409)
 at scala.collection.Iterator$$anon$13.hasNext(Iterator.scala:462)
 at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:409)
 at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:409)
 at org.apache.spark.sql.execution.SparkPlan$$anonfun$2.apply(SparkPlan.scala:255)
 at org.apache.spark.sql.execution.SparkPlan$$anonfun$2.apply(SparkPlan.scala:247)
 at org.apache.spark.rdd.RDD$$anonfun$mapPartitionsInternal$1$$anonfun$apply
$24.apply(RDD.scala:836)
 at org.apache.spark.rdd.RDD$$anonfun$mapPartitionsInternal$1$$anonfun$apply
$24.apply(RDD.scala:836)
 at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
 at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:324)
 at org.apache.spark.rdd.RDD.iterator(RDD.scala:288)
 at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
 at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:324)
 at org.apache.spark.rdd.RDD.iterator(RDD.scala:288)
 at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:90)
 at org.apache.spark.scheduler.Task.run(Task.scala:121)
 at org.apache.spark.executor.Executor$TaskRunner$$anonfun$10.apply(Executor.scala:408)
 at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1360)
 at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:414)
 at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
 at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
 at java.lang.Thread.run(Thread.java:750)


 input :

 bytesread :  252

 source :

 dynamicRecord :  Row(id=2, name='benjamin', surname='franklin')
```

errorsCount

`errorsCount( )` – 傳回 DynamicFrame 中的錯誤總數。

stageErrorsCount

`stageErrorsCount` – 傳回產生此 DynamicFrame 過程中發生的錯誤數量。

DynamicFrameCollection 類別

`DynamicFrameCollection` 為 DynamicFrame 類 物件的字典，其中索引鍵為 DynamicFrames 的名稱，值則為 DynamicFrame 物件。

__init__

`__init__(dynamic_frames, glue_ctx)`

- `dynamic_frames` – DynamicFrame 類 物件的字典。
- `glue_ctx` – GlueContext 類 物件。

鍵

`keys( )` – 傳回此集合裡的金鑰清單，通常包含了對應的 DynamicFrame 值之名稱。

值

`values(key)` – 傳回此集合裡的 DynamicFrame 值清單。

Select

`select(key)`

傳回的 DynamicFrame 會對應至指定的索引鍵 (通常為 DynamicFrame 的名稱)。

- `key` – DynamicFrameCollection 中的金鑰，通常代表 DynamicFrame 的名稱。

Map

`map(callable, transformation_ctx="")`

使用傳入的函數，根據此集合中的 DynamicFrames 建立並傳回新的 DynamicFrameCollection。

- `callable` – 此函數會以 `DynamicFrame` 和指定的轉換細節做為參數，並傳回 `DynamicFrame`。
- `transformation_ctx` – 由 callable 使用的轉換細節 (選用)。

Flatmap

`flatmap(f, transformation_ctx="")`

使用傳入的函數，根據此集合中的 `DynamicFrames` 建立並傳回新的 `DynamicFrameCollection`。

- `f` – 此函數以 `DynamicFrame` 做為參數並傳回 `DynamicFrame` 或 `DynamicFrameCollection`。
- `transformation_ctx` – 由函數使用的轉換細節 (選用)。

DynamicFrameWriter 類別

方法

- __init__
- from_options
- from_catalog
- from_jdbc_conf

__init__

`__init__(glue_context)`

- `glue_context` – 所要使用的 GlueContext 類。

from_options

`from_options(frame, connection_type, connection_options={}, format=None, format_options={}, transformation_ctx="")`

使用指定的連線和格式來撰寫 `DynamicFrame`。

- `frame` – 所要撰寫的 `DynamicFrame`。
- `connection_type` – 連線類型。有效值包括 `s3`、`mysql`、`postgresql`、`redshift`、`sqlserver` 及 `oracle`。

- connection_options – 連線選項，例如路徑和資料庫資料表 (選用)。如果是 connection_type 的 s3，會定義 Amazon S3 路徑。

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

如果是 JDBC 連線，必須定義幾項屬性。請注意，資料庫名稱必須是 URL 的一部分。它可以選擇性包含在連線選項中。

> **Warning**
>
> 不建議在指令碼中存放密碼。考慮使用 boto3 從 AWS Secrets Manager 或 AWS Glue Data Catalog 擷取它們。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

dbtable 屬性為 JDBC 資料表的名稱。若是支援資料庫內結構描述的 JDBC 資料存放區，請指定 schema.table-name。如果未提供結構描述，則會使用預設的 "public" 結構描述。

如需詳細資訊，請參閱 AWS Glue for Spark 中 ETL 的連線類型和選項。

- format – 格式化規格 (選用)。這用於 Amazon Simple Storage Service (Amazon S3) 或支援多種格式的 AWS Glue 連線。請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項 以了解受支援的格式。
- format_options – 指定格式的格式選項。請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項 以了解受支援的格式。
- transformation_ctx – 所要使用的轉換細節 (選用)。

from_catalog

from_catalog(frame, name_space, table_name, redshift_tmp_dir="", transformation_ctx="")

使用指定的目錄資料庫和資料表名稱，來撰寫 DynamicFrame。

- frame – 所要撰寫的 DynamicFrame。

- name_space – 所要使用的資料庫。
- table_name – 所要使用的 table_name。
- redshift_tmp_dir：所要使用的 Amazon Redshift 臨時目錄 (選用)。
- transformation_ctx – 所要使用的轉換細節 (選用)。
- additional_options – 提供給 AWS Glue 的額外選項。

若要寫入受 Lake Formation 管控的資料表，您可以使用下列其他選項：

- transactionId – (字串) 要寫入受管控資料表的交易 ID。此交易不能已遞交或中止，否則寫入將失敗。
- callDeleteObjectsOnCancel – (布林值，選用) 如果設定為 true (預設值)，則 AWS Glue 會在物件寫入至 Amazon S3 之後自動呼叫 DeleteObjectsOnCancel API。如需詳細資訊，請參閱《AWS Lake Formation 開發人員指南》中的 DeleteObjectsOnCancel。

Example 範例：寫入 Lake Formation 中的受管控資料表

```
txId = glueContext.start_transaction(read_only=False)
glueContext.write_dynamic_frame.from_catalog(
    frame=dyf,
    database = db,
    table_name = tbl,
    transformation_ctx = "datasource0",
    additional_options={"transactionId":txId})
...
glueContext.commit_transaction(txId)
```

from_jdbc_conf

from_jdbc_conf(frame, catalog_connection, connection_options={}, redshift_tmp_dir = "", transformation_ctx="")

使用指定的 JDBC 連線資訊來撰寫 DynamicFrame。

- frame – 所要撰寫的 DynamicFrame。
- catalog_connection – 所要使用的目錄連線。
- connection_options – 連線選項，例如路徑和資料庫資料表 (選用)。
- redshift_tmp_dir：所要使用的 Amazon Redshift 臨時目錄 (選用)。
- transformation_ctx – 所要使用的轉換細節 (選用)。

write_dynamic_frame 的範例

這個範例使用 S3 的 connection_type 和 connection_options 中的 POSIX 路徑參數在本機寫入輸出，這允許寫入本機儲存。

```
glueContext.write_dynamic_frame.from_options(\
frame = dyf_splitFields,\
connection_options = {'path': '/home/glue/GlueLocalOutput/'},\
connection_type = 's3',\
format = 'json')
```

DynamicFrameReader 類

— methods —

- __init__
- from_rdd
- from_options
- from_catalog

__init__

__init__(glue_context)

- glue_context – 所要使用的 GlueContext 類。

from_rdd

from_rdd(data, name, schema=None, sampleRatio=None)

DynamicFrame 從彈性分散式資料集 (RDD) 的讀取。

- data – 欲讀取的資料集。
- name – 欲讀取的名稱。
- schema - 欲讀取的結構描述 (選用)。
- sampleRatio – 取樣率 (選用)。

from_options

from_options(connection_type, connection_options={}, format=None, format_options={}, transformation_ctx="")

使用指定的連線和格式讀取 DynamicFrame。

- connection_type – 連線類型。有效值包括s3mysqlpostgresql、、redshift、sqlserver、oracle、dynamodb、和snowflake。
- connection_options – 連線選項，例如路徑和資料庫資料表 (選用)。如需詳細資訊，請參閱 Spark 的 AWS Glue 中 ETL 的連線類型和選項。如果是 connection_type 的 s3，Amazon S3 路徑定義在陣列中。

```
connection_options = {"paths": [ "s3://mybucket/object_a", "s3://mybucket/object_b"]}
```

如果是 JDBC 連線，必須定義幾項屬性。請注意，資料庫名稱必須是 URL 的一部分。它可以選擇性包含在連線選項中。

> **Warning**
>
> 不建議在指令碼中存放密碼。考慮使boto3用從 AWS Secrets Manager 或 AWS Glue 資料型錄擷取它們。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

若是執行平行讀取的 JDBC 連線，您可以設定 hashfield 選項。例如：

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path" , "hashfield": "month"}
```

如需詳細資訊，請參閱 從 JDBC 資料表中平行讀取。

- format – 格式化規格 (選用)。這用於 Amazon Simple Storage Service (Amazon S3) 或支援多種格式的 AWS Glue 連線。請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項 以了解受支援的格式。

- format_options – 指定格式的格式選項。請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項 以了解受支援的格式。
- transformation_ctx – 欲使用的轉換細節 (選用)。
- push_down_predicate – 篩選分割區，而無需列出和讀取資料集中的所有檔案。如需詳細資訊，請參閱使用 Pushdown 述詞預先篩選。

from_catalog

from_catalog(database, table_name, redshift_tmp_dir="", transformation_ctx="", push_down_predicate="", additional_options={})

使用指定的目錄命名空間和資料表名稱讀取 DynamicFrame。

- database – 欲讀取的資料庫。
- table_name – 欲讀取的資料表的名稱。
- redshift_tmp_dir - 要使用的 Amazon Redshift 暫時目錄 (如果不是從 Redshift 讀取資料，則為選用)。
- transformation_ctx – 欲使用的轉換細節 (選用)。
- push_down_predicate – 篩選分割區，而無需列出和讀取資料集中的所有檔案。如需詳細資訊，請參閱 使用 pushdown 述詞預先篩選。
- additional_options – 提供給 AWS Glue 的額外選項。
  - 若要使用執行平行讀取的 JDBC 連線，您可以設定 hashfield、hashexpression 或 hashpartitions 選項。例如：

    ```
    additional_options = {"hashfield": "month"}
    ```

    如需詳細資訊，請參閱 從 JDBC 資料表中平行讀取。
  - 若要傳遞目錄表達式以根據索引欄進行篩選，您可以參閱 catalogPartitionPredicate 選項。

    catalogPartitionPredicate — 您可以傳遞目錄表達式以根據索引欄進行篩選。這會將篩選下推至伺服器端。如需詳細資訊，請參閱 AWS Glue 分割區索引。注意 push_down_predicate 和 catalogPartitionPredicate 使用不同的語法。前者使用 Spark SQL 標準語法，後者使用 JSQL 剖析器。

    如需更多詳細資訊，請參閱 在 AWS Glue 中管理適用於 ETL 輸出的分割區。

GlueContext 類

包裝 Apache 的星火SparkContext對象，從而提供了與 Apache 星火平台進行交互的機制。

__init__

`__init__(sparkContext)`

- `sparkContext` – 欲使用的 Apache Spark 細節。

正在建立

- __init__
- getSource
- create_dynamic_frame_from_rdd
- create_dynamic_frame_from_catalog
- create_dynamic_frame_from_options
- create_sample_dynamic_frame_from_catalog
- create_sample_dynamic_frame_from_options
- add_ingestion_time_columns
- create_data_frame_from_catalog
- create_data_frame_from_options
- forEachBatch

getSource

`getSource(connection_type, transformation_ctx = "", **options)`

建立可用於從外部來源讀取 `DynamicFrames` 的 `DataSource` 物件。

- `connection_type` – 要使用的連線類型，例如 Amazon Simple Storage Service (Amazon S3)、Amazon Redshift 及 JDBC。有效值包括 `s3`、`mysql`、`postgresql`、`redshift`、`sqlserver`、`oracle` 和 `dynamodb`。
- `transformation_ctx` – 欲使用的轉換細節 (選用)。
- `options` – 選擇性的名稱/值對的集合。如需詳細資訊，請參閱 AWS Glue for Spark 中 ETL 的連線類型和選項。

以下是 getSource 的使用範例：

```
>>> data_source = context.getSource("file", paths=["/in/path"])
>>> data_source.setFormat("json")
>>> myFrame = data_source.getFrame()
```

create_dynamic_frame_from_rdd

create_dynamic_frame_from_rdd(data, name, schema=None, sample_ratio=None, transformation_ctx="")

傳回從 Apache Spark 彈性分散式資料集 (RDD) 建立的 DynamicFrame。

- data – 欲使用的資料來源。
- name – 欲使用的資料名稱。
- schema – 欲使用的結構描述 (選用)。
- sample_ratio – 欲使用的取樣率 (選用)。
- transformation_ctx – 欲使用的轉換細節 (選用)。

create_dynamic_frame_from_catalog

create_dynamic_frame_from_catalog(database, table_name, redshift_tmp_dir, transformation_ctx = "", push_down_predicate= "", additional_options = {}, catalog_id = None)

傳回使用 Data Catalog 資料庫和資料表名稱建立的 DynamicFrame。使用此方法時，您可以透format_options過additional_options引數在指定的 AWS Glue 資料目錄表格上提供透過資料表屬性，以及其他選項。

- Database – 欲讀取的資料庫。
- table_name – 欲讀取的資料表的名稱。
- redshift_tmp_dir – 所要使用的 Amazon Redshift 暫時目錄 (選用)。
- transformation_ctx – 欲使用的轉換細節 (選用)。
- push_down_predicate – 篩選分割區，而無需列出和讀取資料集中的所有檔案。如需支援的來源和限制，請參閱 AWS Glue ETL 中的透過下推將讀取最佳化。如需詳細資訊，請參閱 使用 pushdown 述詞預先篩選。

- additional_options – 選擇性的名稱/值對的集合。可能的選項包括 AWS Glue for Spark 中 ETL 的連線類型和選項 中列出的項目，除了 endpointUrl、streamName、bootstrap.servers、security.protocol、topicName、classi 以及delimiter。另一個支援的選項是 catalogPartitionPredicate：

  catalogPartitionPredicate — 您可以傳遞目錄表達式以根據索引欄進行篩選。這會將篩選下推至伺服器端。如需詳細資訊，請參閱 AWS Glue 分割區索引。注意 push_down_predicate 和 catalogPartitionPredicate 使用不同的語法。前者使用 Spark SQL 標準語法，後者使用 JSQL 剖析器。
- catalog_id — 要存取之 Data Catalog 的目錄 ID (帳戶 ID)。若無，會使用發起人的預設帳戶 ID。

create_dynamic_frame_from_options

create_dynamic_frame_from_options(connection_type, connection_options={}, format=None, format_options={}, transformation_ctx = "")

傳回使用指定的連線和格式建立的 DynamicFrame。

- connection_type – 連線類型，例如 Amazon S3、Amazon Redshift 及 JDBC。有效值包括 s3、mysql、postgresql、redshift、sqlserver、oracle 和 dynamodb。
- connection_options - 連線選項，例如路徑和資料庫資料表 (選用)。如果是 connection_type 的 s3，會定義 Amazon S3 路徑清單。

```
connection_options = {"paths": ["s3://aws-glue-target/temp"]}
```

如果是 JDBC 連線，必須定義幾項屬性。請注意，資料庫名稱必須是 URL 的一部分。它可以選擇性包含在連線選項中。

> **Warning**
>
> 不建議在指令碼中存放密碼。考慮使boto3用從 AWS Secrets Manager 或 AWS Glue 資料型錄擷取它們。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

dbtable 屬性為 JDBC 資料表的名稱。若是支援資料庫內結構描述的 JDBC 資料存放區，請指定 schema.table-name。如果未提供結構描述，則會使用預設的 "public" 結構描述。

如需詳細資訊，請參閱 AWS Glue for Spark 中 ETL 的連線類型和選項。

- format— 格式規格。這是用於 Amazon S3 或支援多種格式的 AWS Glue 連線。請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項 以了解受支援的格式。
- format_options – 指定格式的格式選項。請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項 以了解受支援的格式。
- transformation_ctx – 欲使用的轉換細節 (選用)。
- push_down_predicate – 篩選分割區，而無需列出和讀取資料集中的所有檔案。如需支援的來源和限制，請參閱 AWS Glue ETL 中的透過下推將讀取最佳化。如需詳細資訊，請參閱使用 Pushdown 述詞預先篩選。

create_sample_dynamic_frame_from_catalog

create_sample_dynamic_frame_from_catalog(database, table_name, num, redshift_tmp_dir, transformation_ctx = "", push_down_predicate= "", additional_options = {}, sample_options = {}, catalog_id = None)

傳回使用 Data Catalog 資料庫和資料表名稱建立的範例 DynamicFrame。DynamicFrame 僅包含來自資料來源的第一個 num 記錄。

- database – 欲讀取的資料庫。
- table_name – 欲讀取的資料表的名稱。
- num – 傳回的範例動態框架中記錄的最大數目。
- redshift_tmp_dir：所要使用的 Amazon Redshift 臨時目錄 (選用)。
- transformation_ctx – 欲使用的轉換細節 (選用)。
- push_down_predicate – 篩選分割區，而無需列出和讀取資料集中的所有檔案。如需詳細資訊，請參閱 使用 pushdown 述詞預先篩選。
- additional_options – 選擇性的名稱/值對的集合。可能的選項包括 AWS Glue for Spark 中 ETL 的連線類型和選項 中列出的項目，除了 endpointUrl、streamName、bootstrap.servers、security.protocol、topicName、classi 以及delimiter。
- sample_options – 用於控制取樣行為的參數 (選用)。Amazon S3 來源的目前可用參數：

- maxSamplePartitions – 取樣將讀取的分割區數目上限。預設值為 10
- maxSampleFilesPerPartition – 取樣將在一個分割區中讀取的檔案數目上限。預設值為 10。

  這些參數有助於減少檔案清單所耗用的時間。例如，假設資料集有 1000 個分割區，並且每個分割區都有 10 個檔案。如果您設定 maxSamplePartitions = 10 和 maxSampleFilesPerPartition = 10，而不是列出所有 10,000 個檔案，而是僅列出和讀取前 10 個分割區及每個分割區的前 10 個檔案 (總計為 10*10 = 100 個檔案)。
- catalog_id – 要存取之 Data Catalog 的目錄 ID (Data Catalog 的帳戶 ID)。依預設設定為 None。None 預設為服務中呼叫帳戶的目錄 ID。

create_sample_dynamic_frame_from_options

create_sample_dynamic_frame_from_options(connection_type, connection_options={}, num, sample_options={}, format=None, format_options={}, transformation_ctx = "")

傳回使用指定的連線和格式建立的範例 DynamicFrame。DynamicFrame 僅包含來自資料來源的第一個 num 記錄。

- connection_type – 連線類型，例如 Amazon S3、Amazon Redshift 及 JDBC。有效值包括 s3、mysql、postgresql、redshift、sqlserver、oracle 和 dynamodb。
- connection_options - 連線選項，例如路徑和資料庫資料表 (選用)。如需詳細資訊，請參閱 AWS Glue for Spark 中 ETL 的連線類型和選項。
- num – 傳回的範例動態框架中記錄的最大數目。
- sample_options – 用於控制取樣行為的參數 (選用)。Amazon S3 來源的目前可用參數：
  - maxSamplePartitions – 取樣將讀取的分割區數目上限。預設值為 10
  - maxSampleFilesPerPartition – 取樣將在一個分割區中讀取的檔案數目上限。預設值為 10。

    這些參數有助於減少檔案清單所耗用的時間。例如，假設資料集有 1000 個分割區，並且每個分割區都有 10 個檔案。如果您設定 maxSamplePartitions = 10 和 maxSampleFilesPerPartition = 10，而不是列出所有 10,000 個檔案，而是僅列出和讀取前 10 個分割區及每個分割區的前 10 個檔案 (總計為 10*10 = 100 個檔案)。
- format— 格式規格。這是用於 Amazon S3 或支援多種格式的 AWS Glue 連線。請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項 以了解受支援的格式。

- format_options – 指定格式的格式選項。請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項 以了解受支援的格式。
- transformation_ctx – 欲使用的轉換細節 (選用)。
- push_down_predicate – 篩選分割區，而無需列出和讀取資料集中的所有檔案。如需詳細資訊，請參閱 使用 pushdown 述詞預先篩選。

add_ingestion_time_columns

add_ingestion_time_columns(dataFrame, timeGranularity = "")

附加擷取時間欄 (如 ingest_year、ingest_month、ingest_day、ingest_hour、ingest_minute) 到輸入 DataFrame。當您指定以 Amazon S3 為目標的 Data Catalog 資料表時，此函數會在 AWS Glue 產生的指令碼中自動產生。此函數會自動使用輸出資料表上的擷取時間欄來更新分割區。這可讓輸出資料在擷取時間自動分割，而不需要輸入資料中的明確擷取時間欄。

- dataFrame – 要將擷取時間欄附加到的 dataFrame。
- timeGranularity – 時間欄的精密程度。有效值為 "day"、"hour" 和 "minute"。例如：如果 "hour" 被傳遞給函數，原始 dataFrame 會附加上 "ingest_year"、"ingest_month"、"ingest_day" 和 "ingest_hour" 時間欄。

傳回附加時間粒度欄後的資料框架。

範例：

```
dynamic_frame = DynamicFrame.fromDF(glueContext.add_ingestion_time_columns(dataFrame,
 "hour"))
```

create_data_frame_from_catalog

create_data_frame_from_catalog(database, table_name, transformation_ctx = "", additional_options = {})

傳回使用 Data Catalog 資料表的資訊建立的 DataFrame。

- database – 要從中讀取的 Data Catalog 資料庫。
- table_name – 要從中讀取的 Data Catalog 資料表的名稱。

- `transformation_ctx` – 欲使用的轉換細節 (選用)。
- `additional_options` – 選擇性的名稱/值對的集合。可能的選項包括 AWS Glue for Spark 中 ETL 的連線類型和選項 中列出用於串流來源的項目，例如 `startingPosition`、`maxFetchTimeInMs` 以及 `startingOffsets`。
  - `useSparkDataSource`— 當設定為 true 時，強制 AWS Glue 使用原生 Spark 資料來源 API 來讀取資料表。Spark Data Source API 支援下列格式：AVRO、二進位、CSV、JSON、ORC、Parquet 和文字。在 Data Catalog 資料表中，您可以使用 `classification` 屬性指定格式。若要進一步了解 Spark Data Source API，請參閱官方 Apache Spark 文件。

    將 `create_data_frame_from_catalog` 與 `useSparkDataSource` 一起使用具有以下好處：
    - 直接傳回 `DataFrame` 並提供 `create_dynamic_frame.from_catalog().toDF()` 的替代方案。
    - 支援原生格式的 AWS Lake Formation 表格層級權限控制。
    - 支援讀取資料湖格式，無需 AWS Lake Formation 表格層級權限控制。如需詳細資訊，請參閱 將資料湖架構與 AWS Glue ETL 任務搭配使用。

    啟用時`useSparkDataSource`，您也可以視需要在`additional_options`中新增任何 Spark 資料來源選項。 AWS Glue 將這些選項直接傳遞給 Spark 閱讀器。
  - `useCatalogSchema`— 設定為 true 時， AWS Glue 會將資料目錄結構描述套用至產生的結果`DataFrame`。否則，讀取器會從資料推斷結構描述。啟用 `useCatalogSchema` 時，也必須將 `useSparkDataSource` 設定為 true。

限制

使用 `useSparkDataSource` 選項時請考慮以下限制：

- 當您使用時`useSparkDataSource`， AWS Glue 會`DataFrame`在不同於原始 Spark 工作階段的個別 Spark 工作階段中建立新的。
- Spark DataFrame 分區過濾不適用於以下 AWS Glue 功能。
  - 任務書籤
  - 排除 Amazon S3 儲存類別
  - 目錄分割區述詞

  若要搭配這些功能使用分割區篩選，您可以使用 AWS Glue 下拉述詞。如需詳細資訊，請參閱 使用 pushdown 述詞預先篩選。篩選未分割資料欄不會受到影響。

下列範例指令碼示範使用 excludeStorageClasses 選項執行分割區篩選的不正確方法。

```
// Incorrect partition filtering using Spark filter with excludeStorageClasses
read_df = glueContext.create_data_frame.from_catalog(
    database=database_name,
    table_name=table_name,
    additional_options = {
      "useSparkDataSource": True,
      "excludeStorageClasses" : ["GLACIER", "DEEP_ARCHIVE"]
    }
)

//  Suppose year and month are partition keys.
//  Filtering on year and month won't work, the filtered_df will still
//  contain data with other year/month values.
filtered_df = read_df.filter("year == '2017 and month == '04' and 'state == 'CA'")
```

下列範例指令碼示範使用 excludeStorageClasses 選項，利用下推述詞來執行分割區篩選的正確方法。

```
// Correct partition filtering using the AWS Glue pushdown predicate
// with excludeStorageClasses
read_df = glueContext.create_data_frame.from_catalog(
    database=database_name,
    table_name=table_name,
    //  Use AWS Glue pushdown predicate to perform partition filtering
    push_down_predicate = "(year=='2017' and month=='04')"
    additional_options = {
      "useSparkDataSource": True,
      "excludeStorageClasses" : ["GLACIER", "DEEP_ARCHIVE"]
    }
)

//  Use Spark filter only on non-partitioned columns
filtered_df = read_df.filter("state == 'CA'")
```

範例：使用 Spark Data Source 讀取器來建立 CSV 資料表

```
//  Read a CSV table with '\t' as separator
read_df = glueContext.create_data_frame.from_catalog(
```

```
    database=<database_name>,
    table_name=<table_name>,
    additional_options = {"useSparkDataSource": True,  "sep": '\t'}
)
```

create_data_frame_from_options

`create_data_frame_from_options(connection_type, connection_options={}, format=None, format_options={}, transformation_ctx = "")`

此 API 現已棄用。請改用 `getSource()` API。傳回使用指定的連線和格式建立的 `DataFrame`。這個函數只能用於 AWS Glue 串流來源。

- `connection_type` - 串流連線類型。有效值包括 `kinesis` 與 `kafka`。
- `connection_options`— 連線選項，這些選項對於 Kinesis 和 Kafka 而言是不同的。您可以在 AWS Glue for Spark 中 ETL 的連線類型和選項 中找到每個串流資料來源的所有連線選項清單。請注意串流連線選項的下列不同處：
  - Kinesis 串流來源需要 `streamARN`、`startingPosition`、`inferSchema` 以及 `classification`。
  - Kafka 串流來源需要 `connectionName`、`topicName`、`startingOffsets`、`inferSchema` 以及 `classification`。
- `format`— 格式規格。這是用於 Amazon S3 或支援多種格式的 AWS Glue 連線。如需有關支援格式的資訊，請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項。
- `format_options` – 指定格式的格式選項。如需支援格式選項的詳細資訊，請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項。
- `transformation_ctx` – 欲使用的轉換細節 (選用)。

Amazon Kinesis 串流來源範例：

```
kinesis_options =
   { "streamARN": "arn:aws:kinesis:us-east-2:777788889999:stream/fromOptionsStream",
     "startingPosition": "TRIM_HORIZON",
     "inferSchema": "true",
     "classification": "json"
   }
data_frame_datasource0 =
 glueContext.create_data_frame.from_options(connection_type="kinesis",
 connection_options=kinesis_options)
```

Kafka 串流來源範例：

```
kafka_options =
    { "connectionName": "ConfluentKafka",
      "topicName": "kafka-auth-topic",
      "startingOffsets": "earliest",
      "inferSchema": "true",
      "classification": "json"
    }
data_frame_datasource0 =
 glueContext.create_data_frame.from_options(connection_type="kafka",
 connection_options=kafka_options)
```

forEachBatch

`forEachBatch(frame, batch_function, options)`

將傳入的 batch_function 套用至從串流來源讀取的每個微批次。

- frame— 包 DataFrame 含當前微批次。
- batch_function – 將套用至每個微批次的函數。
- options – 索引鍵/值配對的集合，其中包含如何處理微批次的相關資訊。下列選項是必要的：
  - windowSize – 處理每個批次的時間量。
  - checkpointLocation - 串流 ETL 任務的檢查點儲存位置。
  - batchMaxRetries – 如果失敗，可重試批次的次數上限。預設值為 3。此選項僅在 Glue 2.0 及以上版本上才可設定。

範例：

```
glueContext.forEachBatch(
    frame = data_frame_datasource0,
    batch_function = processBatch,
    options = {
        "windowSize": "100 seconds",
        "checkpointLocation": "s3://kafka-auth-dataplane/confluent-test/output/
checkpoint/"
    }
)

def processBatch(data_frame, batchId):
```

```
    if (data_frame.count() > 0):
        datasource0 = DynamicFrame.fromDF(
          glueContext.add_ingestion_time_columns(data_frame, "hour"),
          glueContext, "from_data_frame"
        )
        additionalOptions_datasink1 = {"enableUpdateCatalog": True}
        additionalOptions_datasink1["partitionKeys"] = ["ingest_yr", "ingest_mo",
"ingest_day"]
        datasink1 = glueContext.write_dynamic_frame.from_catalog(
          frame = datasource0,
          database = "tempdb",
          table_name = "kafka-auth-table-output",
          transformation_ctx = "datasink1",
          additional_options = additionalOptions_datasink1
        )
```

在 Amazon S3 中使用資料集

- purge_table
- purge_s3_path
- transition_table
- transition_s3_path

purge_table

purge_table(catalog_id=None, database="", table_name="", options={}, transformation_ctx="")

從 Amazon S3 中刪除指定目錄資料庫和資料表的檔案。如果刪除分割區中的所有檔案，該分割區也會從目錄中刪除。

如果您希望能夠復原已刪除的物件，您可以在 Amazon S3 儲存貯體上開啟物件版本控制。從未啟用物件版本控制的儲存貯體中刪除物件時，無法復原物件。如需如何復原已啟用版本控制之儲存貯體中已刪除物件的詳細資訊，請參閱 AWS Support 知識中心的如何擷取已刪除的 Amazon S3 物件？。

- `catalog_id` – 要存取之 Data Catalog 的目錄 ID (Data Catalog 的帳戶 ID)。依預設設定為 None。None 預設為服務中呼叫帳戶的目錄 ID。
- `database` – 所要使用的資料庫。
- `table_name` - 要使用的資料表名稱。

- options - 篩選要刪除之檔案和用於產生資訊清單檔案的選項。
  - retentionPeriod - 指定保留檔案的期間 (以小時為單位)。比保留期間新的檔案都會予以保留。依預設設定為 168 小時 (7 天)。
  - partitionPredicate - 滿足此述詞的分割區會被刪除。這些分割區中仍在保留期間內的檔案不會被刪除。設定為 "" – 預設為空值。
  - excludeStorageClasses - 不會刪除 excludeStorageClasses 集合中具有儲存體方案的檔案。預設為 Set() – 空集合。
  - manifestFilePath - 產生資訊清單檔案的選用路徑。所有已成功清除的檔案都會記錄在 Success.csv 中，失敗的則記錄在 Failed.csv 中
- transformation_ctx – 欲使用的轉換細節 (選用)。用於資訊清單檔案的路徑。

Example

```
glueContext.purge_table("database", "table", {"partitionPredicate": "(month=='march')",
 "retentionPeriod": 1, "excludeStorageClasses": ["STANDARD_IA"], "manifestFilePath":
 "s3://bucketmanifest/"})
```

purge_s3_path

purge_s3_path(s3_path, options={}, transformation_ctx="")

以遞迴方式刪除指定 Amazon S3 路徑中的檔案。

如果您希望能夠復原已刪除的物件，您可以在 Amazon S3 儲存貯體上開啟物件版本控制。從未開啟物件版本控制的儲存貯體中刪除物件時，無法復原物件。如需如何使用版本控制復原儲存貯體中已刪除物件的詳細資訊，請參閱如何擷取已刪除的 Amazon S3 物件？ 在 AWS Support 知識中心。

- s3_path - 要刪除之檔案的 Amazon S3 路徑，格式為 s3://<*bucket*>/<*prefix*>/
- options - 篩選要刪除之檔案和用於產生資訊清單檔案的選項。
  - retentionPeriod - 指定保留檔案的期間 (以小時為單位)。比保留期間新的檔案都會予以保留。依預設設定為 168 小時 (7 天)。
  - excludeStorageClasses - 不會刪除 excludeStorageClasses 集合中具有儲存體方案的檔案。預設為 Set() – 空集合。
  - manifestFilePath - 產生資訊清單檔案的選用路徑。所有已成功清除的檔案都會記錄在 Success.csv 中，失敗的則記錄在 Failed.csv 中
- transformation_ctx – 欲使用的轉換細節 (選用)。用於資訊清單檔案的路徑。

Example

```
glueContext.purge_s3_path("s3://bucket/path/", {"retentionPeriod": 1,
 "excludeStorageClasses": ["STANDARD_IA"], "manifestFilePath": "s3://bucketmanifest/"})
```

transition_table

`transition_table(database, table_name, transition_to, options={}, transformation_ctx="", catalog_id=None)`

針對指定之目錄的資料庫和資料表，轉換儲存在 Amazon S3 上之檔案的儲存體方案。

您可以在任意兩個儲存體方案之間轉換。對於 GLACIER 和 DEEP_ARCHIVE 儲存體方案，您可以轉換到這些方案。但是，您可以使用 S3 RESTORE 從 GLACIER 和 DEEP_ARCHIVE 儲存體方案轉換。

如果您執行的 AWS Glue ETL 任務會從 Amazon S3 讀取檔案或分割區，則您可排除部分 Amazon S3 儲存類別類型。如需詳細資訊，請參閱排除 Amazon S3 儲存體方案。

- `database` – 所要使用的資料庫。
- `table_name` - 要使用的資料表名稱。
- `transition_to` – 要轉移的 Amazon S3 儲存方案。
- `options` - 篩選要刪除之檔案和用於產生資訊清單檔案的選項。
  - `retentionPeriod` - 指定保留檔案的期間 (以小時為單位)。比保留期間新的檔案都會予以保留。依預設設定為 168 小時 (7 天)。
  - `partitionPredicate` - 滿足此述詞的分割區會被轉換。這些分割區中仍在保留期間內的檔案不會被轉換。設定為 `""` – 預設為空值。
  - `excludeStorageClasses` - 不會轉換 `excludeStorageClasses` 集合中具有儲存體方案的檔案。預設為 `Set()` – 空集合。
  - `manifestFilePath` - 產生資訊清單檔案的選用路徑。所有已成功轉換的檔案都會記錄在 `Success.csv` 中，失敗的則記錄在 `Failed.csv` 中
  - `accountId` – 要執行轉移轉換的 Amazon Web Services 帳戶 ID。對於這種轉換是強制性的。
  - `roleArn`— 執行轉換轉換的 AWS 角色。對於這種轉換是強制性的。
- `transformation_ctx` – 欲使用的轉換細節 (選用)。用於資訊清單檔案的路徑。
- `catalog_id` – 要存取之 Data Catalog 的目錄 ID (Data Catalog 的帳戶 ID)。依預設設定為 None。None 預設為服務中呼叫帳戶的目錄 ID。

Example

```
glueContext.transition_table("database", "table", "STANDARD_IA", {"retentionPeriod": 1,
 "excludeStorageClasses": ["STANDARD_IA"], "manifestFilePath": "s3://bucketmanifest/",
 "accountId": "12345678901", "roleArn": "arn:aws:iam::123456789012:user/example-
username"})
```

transition_s3_path

transition_s3_path(s3_path, transition_to, options={}, transformation_ctx="")

以遞迴方式轉換指定 Amazon S3 路徑中檔案的儲存體方案。

您可以在任意兩個儲存體方案之間轉換。對於 GLACIER 和 DEEP_ARCHIVE 儲存體方案，您可以轉換到這些方案。但是，您可以使用 S3 RESTORE 從 GLACIER 和 DEEP_ARCHIVE 儲存體方案轉換。

如果您執行的 AWS Glue ETL 任務會從 Amazon S3 讀取檔案或分割區，則您可排除部分 Amazon S3 儲存類別類型。如需詳細資訊，請參閱排除 Amazon S3 儲存體方案。

- s3_path - 要以格式 s3://<*bucket*>/<*prefix*>/ 轉換之檔案的 Amazon S3 路徑。
- transition_to – 要轉移的 Amazon S3 儲存方案。
- options - 篩選要刪除之檔案和用於產生資訊清單檔案的選項。
  - retentionPeriod - 指定保留檔案的期間 (以小時為單位)。比保留期間新的檔案都會予以保留。依預設設定為 168 小時 (7 天)。
  - partitionPredicate - 滿足此述詞的分割區會被轉換。這些分割區中仍在保留期間內的檔案不會被轉換。設定為 "" – 預設為空值。
  - excludeStorageClasses - 不會轉換 excludeStorageClasses 集合中具有儲存體方案的檔案。預設為 Set() – 空集合。
  - manifestFilePath - 產生資訊清單檔案的選用路徑。所有已成功轉換的檔案都會記錄在 Success.csv 中，失敗的則記錄在 Failed.csv 中
  - accountId – 要執行轉移轉換的 Amazon Web Services 帳戶 ID。對於這種轉換是強制性的。
  - roleArn— 執行轉換轉換的 AWS 角色。對於這種轉換是強制性的。
- transformation_ctx – 欲使用的轉換細節 (選用)。用於資訊清單檔案的路徑。

Example

```
glueContext.transition_s3_path("s3://bucket/prefix/", "STANDARD_IA",
 {"retentionPeriod": 1, "excludeStorageClasses": ["STANDARD_IA"],
 "manifestFilePath": "s3://bucketmanifest/", "accountId": "12345678901", "roleArn":
 "arn:aws:iam::123456789012:user/example-username"})
```

擷取

- extract_jdbc_conf

extract_jdbc_conf

`extract_jdbc_conf(connection_name, catalog_id = None)`

從 Data Catalog 中的 AWS Glue 連線物件傳回含索引鍵 (具有組態屬性) 的 `dict`。

- `user`：資料庫使用者名稱。
- `password`：資料庫密碼。
- `vendor`：指定廠商 (`mysql`、`postgresql`、`oracle`、`sqlserver` 等)。
- `enforceSSL`：布林字串，指示是否需要安全連線。
- `customJDBCCert`：使用指定 Amazon S3 路徑中的特定用戶端憑證。
- `skipCustomJDBCCertValidation`：布林字串，指示 `customJDBCCert` 必須由 CA 驗證。
- `customJDBCCertString`：有關自訂憑證的其他資訊，因驅動程式類型而異。
- `url`：(已棄用) 僅包含通訊協定、伺服器和連接埠的 JDBC URL。
- `fullUrl`：建立連線時輸入的 JDBC URL (適用於 AWS Glue 3.0 版或更新版本)。

擷取 JDBC 組態的範例：

```
jdbc_conf = glueContext.extract_jdbc_conf(connection_name="your_glue_connection_name")
print(jdbc_conf)
>>> {'enforceSSL': 'false', 'skipCustomJDBCCertValidation': 'false', 'url':
 'jdbc:mysql://myserver:3306', 'fullUrl': 'jdbc:mysql://myserver:3306/mydb',
 'customJDBCCertString': '', 'user': 'admin', 'customJDBCCert': '', 'password': '1234',
 'vendor': 'mysql'}
```

交易

- start_transaction
- commit_transaction
- cancel_transaction

start_transaction

`start_transaction(read_only)`

開始新交易。內部呼叫 Lake Formation startTransaction API。

- `read_only` – (布林值) 指出此交易應該是唯讀，還是讀取和寫入。使用唯讀交易 ID 進行的寫入將被拒絕。唯讀交易不需要遞交。

傳回交易 ID。

commit_transaction

`commit_transaction(transaction_id, wait_for_commit = True)`

嘗試遞交指定的交易。`commit_transaction` 可能會在交易完成遞交之前返回。內部呼叫 Lake Formation commitTransaction API。

- `transaction_id` – (字串) 要遞交的交易。
- `wait_for_commit` – (布林值) 決定 `commit_transaction` 是否立即傳回。預設值為 true。如為 False，`commit_transaction` 輪詢並等待，直到交易完成遞交。使用指數退避時，等待時間長度限制為 1 分鐘，最多可嘗試 6 次重試。

傳回一個布林值，指示遞交是否完成。

cancel_transaction

`cancel_transaction(transaction_id)`

嘗試取消指定的交易。如果交易先前已遞交，傳回 `TransactionCommittedException` 例外狀況。內部調用 Lake Formation CancelTransactionAPI。

- `transaction_id` – (字串) 要取消的交易。

寫入

- getSink
- write_dynamic_frame_from_options
- write_from_options
- write_dynamic_frame_from_catalog
- write_data_frame_from_catalog
- write_dynamic_frame_from_jdbc_conf
- write_from_jdbc_conf

getSink

getSink(connection_type, format = None, transformation_ctx = "", **options)

取得可用於將 DynamicFrames 寫入外部來源的 DataSink 物件。請先檢查 SparkSQL format 以確保取得預期的目的地。

- connection_type – 要使用的連線類型，例如 Amazon S3、Amazon Redshift 及 JDBC。有效值包括s3mysqlpostgresql、、redshift、sqlserver、oracle、kinesis、和kafka。
- format – 要使用的 SparkSQL 格式 (選用)。
- transformation_ctx – 欲使用的轉換細節 (選用)。
- options – 名稱/值對的集合，用來指定連線選項。一些可能的值為：
  - user 和 password：適用於授權
  - url：資料存放區的端點
  - dbtable：目標資料表的名稱。
  - bulkSize：插入操作的平行程度

您可以指定的選項取決於連線類型。如需其他值和範例，請參閱 AWS Glue for Spark 中 ETL 的連線類型和選項。

範例：

```
>>> data_sink = context.getSink("s3")
>>> data_sink.setFormat("json"),
>>> data_sink.writeFrame(myFrame)
```

write_dynamic_frame_from_options

`write_dynamic_frame_from_options(frame, connection_type, connection_options={}, format=None, format_options={}, transformation_ctx = "")`

使用指定的連線和格式來撰寫並傳回 DynamicFrame。

- `frame` – 所要撰寫的 DynamicFrame。
- `connection_type` – 連線類型，例如 Amazon S3、Amazon Redshift 及 JDBC。有效值包括`s3`、`mysql`、`postgresql`、`redshift`、`sqlserver`、`oracle`、`kinesis`、和`kafka`。
- `connection_options` – 連線選項，例如路徑和資料庫資料表 (選用)。如果是 `connection_type` 的 `s3`，會定義 Amazon S3 路徑。

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

如果是 JDBC 連線，必須定義幾項屬性。請注意，資料庫名稱必須是 URL 的一部分。它可以選擇性包含在連線選項中。

> **Warning**
>
> 不建議在指令碼中存放密碼。考慮使`boto3`用從 AWS Secrets Manager 或 AWS Glue 資料型錄擷取它們。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

`dbtable` 屬性為 JDBC 資料表的名稱。若是支援資料庫內結構描述的 JDBC 資料存放區，請指定 `schema.table-name`。如果未提供結構描述，則會使用預設的 "public" 結構描述。

如需詳細資訊，請參閱 AWS Glue for Spark 中 ETL 的連線類型和選項。

- `format`— 格式規格。這是用於 Amazon S3 或支援多種格式的 AWS Glue 連線。請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項 以了解受支援的格式。
- `format_options` – 指定格式的格式選項。請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項 以了解受支援的格式。

- transformation_ctx – 所要使用的轉換細節 (選用)。

write_from_options

write_from_options(frame_or_dfc, connection_type, connection_options={}, format={}, format_options={}, transformation_ctx = "")

寫入和傳回以指定的連線和格式資訊建立的 DynamicFrame 或 DynamicFrameCollection。

- frame_or_dfc – 所要撰寫的 DynamicFrame 或 DynamicFrameCollection。
- connection_type – 連線類型，例如 Amazon S3、Amazon Redshift 及 JDBC。有效值包括 s3、mysql、postgresql、redshift、sqlserver 及 oracle。
- connection_options – 連線選項，例如路徑和資料庫資料表 (選用)。如果是 connection_type 的 s3，會定義 Amazon S3 路徑。

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

如果是 JDBC 連線，必須定義幾項屬性。請注意，資料庫名稱必須是 URL 的一部分。它可以選擇性包含在連線選項中。

> **Warning**
>
> 不建議在指令碼中存放密碼。考慮使boto3用從 AWS Secrets Manager 或 AWS Glue 資料型錄擷取它們。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

dbtable 屬性為 JDBC 資料表的名稱。若是支援資料庫內結構描述的 JDBC 資料存放區，請指定 schema.table-name。如果未提供結構描述，則會使用預設的 "public" 結構描述。

如需詳細資訊，請參閱 AWS Glue for Spark 中 ETL 的連線類型和選項。

- format— 格式規格。這是用於 Amazon S3 或支援多種格式的 AWS Glue 連線。請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項 以了解受支援的格式。

- format_options – 指定格式的格式選項。請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項 以了解受支援的格式。
- transformation_ctx – 所要使用的轉換細節 (選用)。

write_dynamic_frame_from_catalog

write_dynamic_frame_from_catalog(frame, database, table_name, redshift_tmp_dir, transformation_ctx = "", additional_options = {}, catalog_id = None)

使用來自 Data Catalog 資料庫和資料表的資訊寫入並傳回 DynamicFrame。

- frame – 所要撰寫的 DynamicFrame。
- Database – 包含資料表的 Data Catalog 資料庫。
- table_name – 與目標關聯的 Data Catalog 資料表名稱。
- redshift_tmp_dir – 所要使用的 Amazon Redshift 暫時目錄 (選用)。
- transformation_ctx – 欲使用的轉換細節 (選用)。
- additional_options – 選擇性的名稱/值對的集合。
- catalog_id — 要存取之 Data Catalog 的目錄 ID (帳戶 ID)。若無，會使用發起人的預設帳戶 ID。

write_data_frame_from_catalog

write_data_frame_from_catalog(frame, database, table_name, redshift_tmp_dir, transformation_ctx = "", additional_options = {}, catalog_id = None)

使用來自 Data Catalog 資料庫和資料表的資訊寫入並傳回 DataFrame。此方法支援寫入資料湖格式 (Hudi、Iceberg 和 Delta Lake)。如需詳細資訊，請參閱 將資料湖架構與 AWS Glue ETL 任務搭配使用。

- frame – 所要撰寫的 DataFrame。
- Database – 包含資料表的 Data Catalog 資料庫。
- table_name – 與目標關聯的 Data Catalog 資料表名稱。
- redshift_tmp_dir：所要使用的 Amazon Redshift 臨時目錄 (選用)。
- transformation_ctx – 欲使用的轉換細節 (選用)。

- additional_options – 選擇性的名稱/值對的集合。
  - useSparkDataSink— 當設定為 true 時，強制 AWS Glue 使用原生 Spark 資料接收器 API 寫入資料表。當您啟用此選項時，您可以視需要將任何 Spark 資料來源選項新增至additional_options。 AWS Glue 將這些選項直接傳遞給 Spark 作家。
- catalog_id – 要存取之 Data Catalog 的目錄 ID (帳戶 ID)。如果您未指定值，則會使用發起人的預設帳戶 ID。

限制

使用 useSparkDataSink 選項時請考慮以下限制：

- 使用 useSparkDataSink 選項時，不支援 enableUpdateCatalog 選項。

範例：使用 Spark Data Source 寫入器寫入 Hudi 資料表

```
hudi_options = {
    'useSparkDataSink': True,
    'hoodie.table.name': <table_name>,
    'hoodie.datasource.write.storage.type': 'COPY_ON_WRITE',
    'hoodie.datasource.write.recordkey.field': 'product_id',
    'hoodie.datasource.write.table.name': <table_name>,
    'hoodie.datasource.write.operation': 'upsert',
    'hoodie.datasource.write.precombine.field': 'updated_at',
    'hoodie.datasource.write.hive_style_partitioning': 'true',
    'hoodie.upsert.shuffle.parallelism': 2,
    'hoodie.insert.shuffle.parallelism': 2,
    'hoodie.datasource.hive_sync.enable': 'true',
    'hoodie.datasource.hive_sync.database': <database_name>,
    'hoodie.datasource.hive_sync.table': <table_name>,
    'hoodie.datasource.hive_sync.use_jdbc': 'false',
    'hoodie.datasource.hive_sync.mode': 'hms'}

glueContext.write_data_frame.from_catalog(
    frame = <df_product_inserts>,
    database = <database_name>,
    table_name = <table_name>,
    additional_options = hudi_options
)
```

write_dynamic_frame_from_jdbc_conf

write_dynamic_frame_from_jdbc_conf(frame, catalog_connection, connection_options={}, redshift_tmp_dir = "", transformation_ctx = "", catalog_id = None)

使用指定的 JDBC 連線資訊撰寫並傳回 DynamicFrame。

- frame – 所要撰寫的 DynamicFrame。
- catalog_connection – 所要使用的目錄連線。
- connection_options – 連線選項，例如路徑和資料庫資料表 (選用)。如需詳細資訊，請參閱 AWS Glue for Spark 中 ETL 的連線類型和選項。
- redshift_tmp_dir – 所要使用的 Amazon Redshift 暫時目錄 (選用)。
- transformation_ctx – 所要使用的轉換細節 (選用)。
- catalog_id — 要存取之 Data Catalog 的目錄 ID (帳戶 ID)。若無，會使用發起人的預設帳戶 ID。

write_from_jdbc_conf

write_from_jdbc_conf(frame_or_dfc, catalog_connection, connection_options={}, redshift_tmp_dir = "", transformation_ctx = "", catalog_id = None)

使用指定的 JDBC 連線資訊撰寫並傳回 DynamicFrame 或 DynamicFrameCollection。

- frame_or_dfc – 所要撰寫的 DynamicFrame 或 DynamicFrameCollection。
- catalog_connection – 所要使用的目錄連線。
- connection_options – 連線選項，例如路徑和資料庫資料表 (選用)。如需詳細資訊，請參閱 AWS Glue for Spark 中 ETL 的連線類型和選項。
- redshift_tmp_dir – 所要使用的 Amazon Redshift 暫時目錄 (選用)。
- transformation_ctx – 所要使用的轉換細節 (選用)。
- catalog_id — 要存取之 Data Catalog 的目錄 ID (帳戶 ID)。若無，會使用發起人的預設帳戶 ID。

AWS Glue PySpark 轉換參考

AWS Glue 提供了以下內置的轉換，您可以在 PySpark ETL 操作中使用。您的數據從轉換傳遞到轉換稱為一個數據結構 DynamicFrame，這是一個 Apache 星火 SQL 的擴展DataFrame。DynamicFrame 包含您的資料，而您可以參考其結構描述以處理資料。

這些轉換大多數也作為 DynamicFrame 類別的方法存在。如需詳細資訊，請參閱DynamicFrame 轉換。

- GlueTransform base 類別
- ApplyMapping 類別
- DropFields 類別
- DropNullFields 類別
- ErrorsAsDynamicFrame 類別
- EvaluateDataQuality 類別
- FillMissingValues 類別
- Filter 類別
- FindIncrementalMatches 類別
- FindMatches 類別
- FlatMap 類別
- Join 類別
- Map 類別
- MapToCollection 類別
- mergeDynamicFrame
- Relationalize 類別
- RenameField 類別
- ResolveChoice 類別
- SelectFields 類別
- SelectFromCollection 類別
- 簡化 JSON 類別
- Spigot 類別
- SplitFields 類別
- SplitRows 類別

- Unbox 類別
- UnnestFrame 類別

GlueTransform base 類別

所有 awsglue.transforms 類別繼承的基底類別。

所有類別皆定義一項 __call__ 方法。依據預設，它們會覆寫下列的 GlueTransform 類別方法，或者透過使用類別名稱呼叫。

方法

- apply(cls, *args, **kwargs)
- name(cls)
- describeArgs(cls)
- describeReturn(cls)
- describeTransform(cls)
- describeErrors(cls)
- describe(cls)

apply(cls, *args, **kwargs)

藉由呼叫轉換類別並傳回結果以套用轉換。

- cls – self 類別物件。

name(cls)

傳回衍生的轉換類別名稱。

- cls – self 類別物件。

describeArgs(cls)

- cls – self 類別物件。

傳回字典清單，每個皆對應至一個具名引數，格式如下：

```
[
  {
    "name": "(name of argument)",
    "type": "(type of argument)",
    "description": "(description of argument)",
    "optional": "(Boolean, True if the argument is optional)",
    "defaultValue": "(Default value string, or None)(String; the default value, or
 None)"
  },
...
]
```

呼叫未實作的衍生轉換時，引發 NotImplementedError 例外。

describeReturn(cls)

- cls – self 類別物件。

傳回字典及有關傳回類型的資訊，格式如下：

```
{
  "type": "(return type)",
  "description": "(description of output)"
}
```

呼叫未實作的衍生轉換時，引發 NotImplementedError 例外。

describeTransform(cls)

傳回描述轉換的字串。

- cls – self 類別物件。

呼叫未實作的衍生轉換時，引發 NotImplementedError 例外。

describeErrors(cls)

- cls – self 類別物件。

傳回字典的清單，每個皆描述此轉換可能擲出的例外狀況，格式如下：

```
[
  {
    "type": "(type of error)",
    "description": "(description of error)"
  },
...
]
```

describe(cls)

- cls – self 類別物件。

以下列格式傳回物件：

```
{
  "transform" : {
    "name" : cls.name( ),
    "args" : cls.describeArgs( ),
    "returns" : cls.describeReturn( ),
    "raises" : cls.describeErrors( ),
    "location" : "internal"
  }
}
```

ApplyMapping 類別

在 DynamicFrame 套用映射。

範例

建議您使用 DynamicFrame.apply_mapping() 在 DynamicFrame 中套用映射。若要檢視程式碼範例，請參閱 範例：使用 apply_map 來重新命名欄位並變更欄位類型。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn

- describeTransform
- describeErrors
- 描述

__call__(frame, mappings, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

將宣告式映射套用於指定的 DynamicFrame。

- frame – 要套用映射的 DynamicFrame (必要)。
- mappings –映射元組的清單 (必要)。每個清單包括：(來源欄、來源類型、目標欄、目標類型)。

  如果來源欄的名稱中有一個小點 "."，則您必須在其前後加上反引號 "``"。例如，若要將 this.old.name (字串) 對應至 thisNewName，會使用以下元組：

  ```
  ("`this.old.name`", "string", "thisNewName", "string")
  ```

- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與轉換中的錯誤相關的字串 (選用)。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用)。預設為零。
- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用)。預設為零。

只傳回「映射」元組中指定的 DynamicFrame 欄位。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

DropFields 類別

在 DynamicFrame 內捨棄欄位。

範例

建議您使用 DynamicFrame.drop_fields() 方法從 DynamicFrame 中刪除欄位。若要檢視程式碼範例，請參閱 範例：使用 drop_fields 從 DynamicFrame 中移除欄位。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

__call__(frame, paths, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

在 DynamicFrame 內捨棄節點。

- frame – 要捨棄節點的 DynamicFrame (必要)。
- paths – 要捨棄之節點的完整路徑清單 (必要)。

- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與轉換中的錯誤相關的字串 (選用)。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用)。預設為零。
- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用)。預設為零。

傳回無指定欄位的新 DynamicFrame。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

DropNullFields 類別

捨棄 DynamicFrame 中類型為 NullType 的所有 null 欄位。在 DynamicFrame 資料集的每筆記錄中，皆存在缺少值或為空值的欄位。

範例

此範例使用 DropNullFields 建立新的 DynamicFrame，其中類型 NullType 的欄位已刪除。為了演示 DropNullFields，我們將類型為 null 的名為 empty_column 的新資料欄新增至已加載的 persons 資料集。

> **Note**
>
> 若要存取此範例中使用的資料集，請參閱 程式碼範例：加入和關聯化資料 並依照 步驟 1：在 Amazon S3 儲存貯體中網路爬取資料 中的說明進行。

```
# Example: Use DropNullFields to create a new DynamicFrame without NullType fields

from pyspark.context import SparkContext
from awsglue.context import GlueContext
from pyspark.sql.functions import lit
from pyspark.sql.types import NullType
from awsglue.dynamicframe import DynamicFrame
from awsglue.transforms import DropNullFields

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create DynamicFrame
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
)
print("Schema for the persons DynamicFrame:")
persons.printSchema()

# Add new column "empty_column" with NullType
persons_with_nulls = persons.toDF().withColumn("empty_column",
 lit(None).cast(NullType()))
persons_with_nulls_dyf = DynamicFrame.fromDF(persons_with_nulls, glueContext,
 "persons_with_nulls")
print("Schema for the persons_with_nulls_dyf DynamicFrame:")
persons_with_nulls_dyf.printSchema()

# Remove the NullType field
persons_no_nulls = DropNullFields.apply(persons_with_nulls_dyf)
```

```
print("Schema for the persons_no_nulls DynamicFrame:")
persons_no_nulls.printSchema()
```

輸出

```
Schema for the persons DynamicFrame:
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string

Schema for the persons_with_nulls_dyf DynamicFrame:
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
```

```
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string
|-- empty_column: null

null_fields ['empty_column']
Schema for the persons_no_nulls DynamicFrame:
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
```

```
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string
```

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

__call__(frame, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

刪除 DynamicFrame 中類型為 NullType 的所有 null 欄位。在 DynamicFrame 資料集的每筆記錄中，皆存在缺少值或為空值的欄位。

- frame – 要刪除其 null 欄位的 DynamicFrame (必要)。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與轉換中的錯誤相關的字串 (選用)。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用)。預設為零。
- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用)。預設為零。

傳回不含 null 欄位的新 DynamicFrame。

apply(cls, *args, **kwargs)

- cls – cls

name(cls)

- cls – cls

describeArgs(cls)

- cls – cls

describeReturn(cls)

- cls – cls

describeTransform(cls)

- cls – cls

describeErrors(cls)

- cls – cls

describe(cls)

- cls – cls

ErrorsAsDynamicFrame 類別

傳回一個 DynamicFrame，其中包含建立來源 DynamicFrame 時發生的錯誤的巢狀錯誤。

範例

建議您使用 DynamicFrame.errorsAsDynamicFrame() 方法擷取和檢視錯誤記錄。若要檢視程式碼範例，請參閱 範例：使用 errorsAsDynamic Frame 檢視錯誤記錄。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

__call__(frame)

傳回 DynamicFrame，其中包含與來源 DynamicFrame 有關的巢狀錯誤記錄。

- frame – 來源 DynamicFrame (必要)。

apply(cls, *args, **kwargs)

- cls – cls

name(cls)

- cls – cls

describeArgs(cls)

- cls – cls

describeReturn(cls)

- cls – cls

describeTransform(cls)

- cls – cls

describeErrors(cls)

- `cls` – cls

describe(cls)

- `cls` – cls

EvaluateDataQuality 類別

根據 DynamicFrame 評估資料品質規則集，並傳回包含評估結果的新 DynamicFrame。

範例

下列範例程式碼示範如何評估 DynamicFrame 的資料品質，然後檢視資料品質結果。

```
from awsglue.transforms import *
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsgluedq.transforms import EvaluateDataQuality

#Create Glue context
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Define DynamicFrame
legislatorsAreas = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="areas_json")

# Create data quality ruleset
ruleset = """Rules = [ColumnExists "id", IsComplete "id"]"""

# Evaluate data quality
dqResults = EvaluateDataQuality.apply(
    frame=legislatorsAreas,
    ruleset=ruleset,
    publishing_options={
        "dataQualityEvaluationContext": "legislatorsAreas",
        "enableDataQualityCloudWatchMetrics": True,
        "enableDataQualityResultsPublishing": True,
        "resultsS3Prefix": "DOC-EXAMPLE-BUCKET1",
    },
```

```
)


# Inspect data quality results
dqResults.printSchema()
dqResults.toDF().show()
```

輸出

```
root
|-- Rule: string
|-- Outcome: string
|-- FailureReason: string
|-- EvaluatedMetrics: map
|    |-- keyType: string
|    |-- valueType: double


+----------------------+-------+-------------+--------------------------------------+
|Rule                  |Outcome|FailureReason|EvaluatedMetrics                      |
+----------------------+-------+-------------+--------------------------------------+
|ColumnExists "id"     |Passed |null         |{}                                    |
|IsComplete "id"       |Passed |null         |{Column.first_name.Completeness -> 1.0}|
+----------------------+-------+-------------+--------------------------------------+
```

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, ruleset, publishing_options = {})

- `frame` – 您要評估資料品質的 `DynamicFrame`。

- `ruleset` – 字串格式的資料品質定義語言 (DQDL) 規則集。若要進一步了解 DQDL，請參閱 資料品質定義語言 (DQDL) 參考 指南。
- `publishing_options` – 指定以下用於發佈評估結果和指標的選項的字典：
  - `dataQualityEvaluationContext`— 字串，指定 AWS Glue 應在其下發佈 Amazon CloudWatch 度量和資料品質結果的命名空間。彙總的量度會顯示在中 CloudWatch，而完整的結果則會顯示在 AWS Glue Studio 介面中。
    - 必要：否
    - 預設值：`default_context`
  - `enableDataQualityCloudWatchMetrics`— 指定是否應將資料品質評估的結果發佈至 CloudWatch。您可以使用 `dataQualityEvaluationContext` 選項指定指標的命名空間。
    - 必要：否
    - 預設值：False
  - `enableDataQualityResultsPublishing` – 指定資料品質結果是否應顯示在 AWS Glue Studio 介面的 Data Quality (資料品質) 索引標籤上。
    - 必要：否
    - 預設值：True
  - `resultsS3Prefix`— 指定 AWS Glue 可以寫入資料品質評估結果的 Amazon S3 位置。
    - 必要：否
    - 預設值："" (空字串)

apply(cls, *args, **kwargs)

繼承自 `GlueTransform` apply。

name(cls)

繼承自 `GlueTransform` name。

describeArgs(cls)

繼承自 `GlueTransform` describeArgs。

describeReturn(cls)

繼承自 `GlueTransform` describeReturn。

describeTransform(cls)

繼承自 `GlueTransform` describeTransform。

describeErrors(cls)

繼承自 `GlueTransform` describeErrors。

describe(cls)

繼承自 `GlueTransform` describe。

FillMissingValues 類別

`FillMissingValues` 類別會在指定的 `DynamicFrame` 中找到 null 值和空字串，並使用機器學習方法 (例如線性迴歸和隨機樹系) 來預測缺少的值。ETL 任務會使用輸入資料集中的值來訓練機器學習模型，然後預測缺少的值應該是什麼。

> **Tip**
>
> 如果您使用增量資料集，則每個增量集都會用作機器學習模型的訓練資料，因此結果可能不準確。

若要匯入：

```
from awsglueml.transforms import FillMissingValues
```

方法

- 套用

apply(frame, missing_values_column, output_column ="", transformation_ctx ="", info ="", stageThreshold = 0, totalThreshold = 0)

在指定的欄中填入動態框架的缺少值，並在新的欄中傳回具有估計值的新框架。對於沒有缺少值的列，指定欄的值將被複製到新欄。

- `frame` – 在其中填入缺少值的 `DynamicFrame`。必要。
- `missing_values_column` – 包含缺少值的欄 (`null` 值和空字串)。必要。

- output_column – 新欄的名稱，該欄將包含所有缺少值的列的估計值。選擇性；預設值為 missing_values_column 的名稱，字尾為 "_filled"。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與轉換中的錯誤相關的字串 (選用)。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用；預設值為零)。
- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用；預設值為零)。

傳回具有一個額外欄的新 DynamicFrame，其中包含缺少值的列估計和其他列的目前值。

Filter 類別

建立新的 DynamicFrame，其中包含來自輸入 DynamicFrame 的符合指定述詞函數的記錄。

範例

建議您使用 DynamicFrame.filter() 方法篩選 DynamicFrame 中的記錄。若要檢視程式碼範例，請參閱 範例：使用篩選條件取得已篩選的欄位選取。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0))

傳回新 DynamicFrame，其藉由從符合指定述詞函數的輸入 DynamicFrame 選擇記錄所建置。

- frame – 要套用指定篩選條件函數的來源 DynamicFrame (必要)。
- f – 要套用到 DynamicFrame 中各個 DynamicRecord 的述詞函數。此函數必須以 DynamicRecord 做為引數並傳回 True，如果 DynamicRecord 符合篩選條件要求，否則將傳回 False (必要)。

DynamicRecord 代表 DynamicFrame 中的邏輯記錄。它類似 Spark DataFrame 中的一列，除了它是自我描述的，以及可用於不符合固定結構描述的資料。

- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與轉換中的錯誤相關的字串 (選用)。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用)。預設為零。
- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用)。預設為零。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

FindIncrementalMatches 類別

識別現有和增量 DynamicFrame 中的相符記錄，並使用指派給每個相符記錄群組的唯一識別碼來建立新的 DynamicFrame。

若要匯入：

```
from awsglueml.transforms import FindIncrementalMatches
```

方法

- 套用

apply(existingFrame, incrementalFrame, transformId, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0, enforcedMatches = none, computeMatchConfidenceScores = 0)

識別輸入 DynamicFrame 中的相符記錄，並使用指派給每個相符記錄群組的唯一識別碼來建立新的 DynamicFrame。

- existingFrame – 現有和預先符合的 DynamicFrame 以套用 FindIncrementalMatches 轉換。必要。
- incrementalFrame – 要套用 FindIncrementalMatches 轉換以比對 existingFrame 的增量 DynamicFrame。必要。
- transformId – 與 FindIncrementalMatches 轉換相關聯的唯一 ID，以套用於 DynamicFrames 中的記錄。必要。
- transformation_ctx – 用於識別統計資料/狀態資訊的唯一字串。選用。
- info – 與轉換中的錯誤相關的字串。選用。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限。選用。預設為零。
- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限。選用。預設為零。
- enforcedMatches – 用於強制符合的 DynamicFrame。選用。預設值為 None (無)。
- computeMatchConfidenceScores – 布林值，指出是否運算每個相符記錄群組的可信度分數。選用。預設值為 false。

傳回具有指派給每個相符記錄群組之唯一識別碼的新 DynamicFrame。

FindMatches 類別

識別輸入 DynamicFrame 中的相符記錄，並使用指派給每個相符記錄群組的唯一識別碼來建立新的 DynamicFrame。

若要匯入：

```
from awsglueml.transforms import FindMatches
```

方法

- 套用

apply(frame, transformId, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0, enforcedMatches = none, computeMatchConfidenceScores = 0)

識別輸入 DynamicFrame 中的相符記錄，並使用指派給每個相符記錄群組的唯一識別碼來建立新的 DynamicFrame。

- frame – 要套用 FindMatches 轉換的 DynamicFrame。必要。
- transformId – 與 FindMatches 轉換相關聯的唯一 ID，以套用於 DynamicFrame 中的記錄。必要。
- transformation_ctx – 用於識別統計資料/狀態資訊的唯一字串。選用。
- info – 與轉換中的錯誤相關的字串。選用。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限。選用。預設為零。
- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限。選用。預設為零。
- enforcedMatches – 用於強制符合的 DynamicFrame。選用。預設值為 None (無)。
- computeMatchConfidenceScores – 布林值，指出是否運算每個相符記錄群組的可信度分數。選用。預設值為 false。

傳回具有指派給每個相符記錄群組之唯一識別碼的新 DynamicFrame。

FlatMap 類別

將轉換套用至集合中的各個 DynamicFrame。結果不會扁平化為單一的 DynamicFrame，而是保留為集合。

FlatMap 的範例

下列範例程式碼片段示範如何在套用至 FlatMap 時對動態影格集合使用 ResolveChoice 轉換。用於輸入的資料位於 Amazon S3 地址 s3://bucket/path-for-data/sample.json 預留位置的 JSON 中，並包含下列資料。

範例 JSON 資料

```
[{
    "firstname": "Arnav",
    "lastname": "Desai",
    "address": {
        "street": "6 Anyroad Avenue",
        "city": "London",
        "state": "England",
        "country": "UK"
    },
    "phone": 17235550101,
    "affiliations": [
        "General Anonymous Example Products",
        "Example Independent Research",
        "Government Department of Examples"
    ]
},
{
    "firstname": "Mary",
    "lastname": "Major",
    "address": {
        "street": "7821 Spot Place",
        "city": "Centerville",
        "state": "OK",
        "country": "US"
    },
    "phone": 19185550023,
    "affiliations": [
        "Example Dot Com",
        "Example Independent Research",
        "Example.io"
    ]
},
{
    "firstname": "Paulo",
    "lastname": "Santos",
    "address": {
        "street": "123 Maple Street",
        "city": "London",
        "state": "Ontario",
        "country": "CA"
    },
    "phone": 12175550181,
```

```
    "affiliations": [
        "General Anonymous Example Products",
        "Example Dot Com"
    ]
}]
```

Example 將 ResolveChoice 套用至 DynamicFrameCollection 並顯示輸出。

```
#Read DynamicFrame
datasource = glueContext.create_dynamic_frame_from_options("s3", connection_options =
 {"paths":["s3://bucket/path/to/file/mysamplejson.json"]}, format="json")
datasource.printSchema()
datasource.show()

## Split to create a DynamicFrameCollection
split_frame=datasource.split_fields(["firstname","lastname","address"],"personal_info","busines
split_frame.keys()
print("---")

## Use FlatMap to run ResolveChoice
kwargs = {"choice": "cast:string"}
flat = FlatMap.apply(split_frame, ResolveChoice, frame_name="frame",
 transformation_ctx='tcx', **kwargs)
flat.keys()

##Select one of the DynamicFrames
personal_info = flat.select("personal_info")
personal_info.printSchema()
personal_info.show()
print("---")

business_info = flat.select("business_info")
business_info.printSchema()
business_info.show()
```

> **Important**
>
> 呼叫 `FlatMap.apply` 時，`frame_name` 參數必須是 `"frame"`。目前不接受其他值。

範例輸出

```
root
|-- firstname: string
|-- lastname: string
|-- address: struct
|    |-- street: string
|    |-- city: string
|    |-- state: string
|    |-- country: string
|-- phone: long
|-- affiliations: array
|    |-- element: string
---
{
    "firstname": "Mary",
    "lastname": "Major",
    "address": {
        "street": "7821 Spot Place",
        "city": "Centerville",
        "state": "OK",
        "country": "US"
    },
    "phone": 19185550023,
    "affiliations": [
        "Example Dot Com",
        "Example Independent Research",
        "Example.io"
    ]
}

{
    "firstname": "Paulo",
    "lastname": "Santos",
    "address": {
        "street": "123 Maple Street",
        "city": "London",
        "state": "Ontario",
        "country": "CA"
    },
    "phone": 12175550181,
    "affiliations": [
        "General Anonymous Example Products",
        "Example Dot Com"
```

```
    ]
}
---
root
|-- firstname: string
|-- lastname: string
|-- address: struct
|    |-- street: string
|    |-- city: string
|    |-- state: string
|    |-- country: string

{
    "firstname": "Mary",
    "lastname": "Major",
    "address": {
        "street": "7821 Spot Place",
        "city": "Centerville",
        "state": "OK",
        "country": "US"
    }
}

{
    "firstname": "Paulo",
    "lastname": "Santos",
    "address": {
        "street": "123 Maple Street",
        "city": "London",
        "state": "Ontario",
        "country": "CA"
    }
}
---
root
|-- phone: long
|-- affiliations: array
|    |-- element: string

{
    "phone": 19185550023,
    "affiliations": [
        "Example Dot Com",
        "Example Independent Research",
```

```
        "Example.io"
    ]
}

{
    "phone": 12175550181,
    "affiliations": [
        "General Anonymous Example Products",
        "Example Dot Com"
    ]
}
```

方法

- __call__
- 套用
- Name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

__call__(dfc, BaseTransform, frame_name, transformation_ctx = "", **base_kwargs)

對集合中的每個 DynamicFrame 套用轉換，並將結果扁平化。

- dfc – 要對其套用 flatmap 的 DynamicFrameCollection (必要)。
- BaseTransform – 從 GlueTransform 衍生的轉換功能，要套用到集合中的每個成員 (必要)，
- frame_name – 做為集合元素傳遞目標的引數名稱 (必要)。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- base_kwargs – 要傳遞到基本轉換的引數 (必要)。

傳回新的 DynamicFrameCollection，也就是對來源 DynamicFrameCollection 中個別 DynamicFrame 所套用轉換後所產生的集合。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

Join 類別

對兩個 DynamicFrames 執行對等性加入。

範例

建議您使用 DynamicFrame.join() 聯結 DynamicFrames。若要檢視程式碼範例，請參閱 範例：使用聯結合併 DynamicFrames。

方法

- __call__
- apply
- name

- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

__call__(frame1, frame2, keys1, keys2, transformation_ctx = "")

對兩個 DynamicFrames 執行對等性加入。

- frame1 – 第一個要加入的 DynamicFrame (必要)。
- frame2 – 第二個要加入的 DynamicFrame (必要)。
- keys1 – 第一個框架要加入的金鑰 (必要)。
- keys2 – 第二個框架要加入的金鑰 (必要)。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。

傳回建立的新 DynamicFrame，其建立透過聯結兩個 DynamicFrames。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

Map 類別

透過將函數套用到輸入 DynamicFrame 中的所有記錄，以建置新的 DynamicFrame。

範例

建議您使用 DynamicFrame.map() 方法將函數套用至 DynamicFrame 中的所有記錄。若要檢視程式碼範例，請參閱 範例：使用 map 將函數套用至 DynamicFrame 中的每個記錄。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

__call__(frame, f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

傳回透過將指定函數套用到原始 DynamicFrame 中所有 DynamicRecords 所產生的新 DynamicFrame。

- frame – 要套用映射函數的 DynamicFrame (必要)。
- f – 要套用到 DynamicFrame 中所有 DynamicRecords 的函數。此函數必須以 DynamicRecord 做為參數，並傳回以映射產生的新 DynamicRecord (必要)。

  DynamicRecord 代表 DynamicFrame 中的邏輯記錄。它類似 Apache Spark DataFrame 中的一列，除了它是自我描述的，以及可用於不符合固定結構描述的資料。

- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與轉換中的錯誤相關的字串 (選用)。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用)。預設為零。
- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用)。預設為零。

傳回透過將指定函數套用到原始 DynamicFrame 中所有 DynamicRecords 所產生的新 DynamicFrame。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

MapToCollection 類別

將轉換套用到指定的 DynamicFrameCollection 中的各個 DynamicFrame。

方法

- __call__
- 套用
- 名稱
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

__call__(dfc, BaseTransform, frame_name, transformation_ctx = "", **base_kwargs)

將轉換函式套用到指定的 DynamicFrameCollection 中的各個 DynamicFrame。

- dfc – 要套用轉換函數的 DynamicFrameCollection (必要)。
- callable – 要套用到各個成員集合的可呼叫轉換函數 (必要)。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。

傳回新的 DynamicFrameCollection，也就是對來源 DynamicFrameCollection 中個別 DynamicFrame 所套用轉換後所產生的集合。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 `GlueTransform` describeTransform。

describeErrors(cls)

繼承自 `GlueTransform` describeErrors。

describe(cls)

繼承自 `GlueTransform` describe。

Relationalize 類別

對 `DynamicFrame` 中的巢狀化結構描述進行壓平合併，並針對已攤平的框架，將陣列欄直轉橫。

範例

建議您使用 `DynamicFrame.relationalize()` 方法來關聯化 `DynamicFrame`。若要檢視程式碼範例，請參閱 範例：使用 relationalize 來壓平合併 `DynamicFrame` 中的巢狀化結構描述。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, staging_path=None, name='roottable', options=None, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

將 `DynamicFrame` 關聯化，並針對將巢狀欄解巢狀和將陣列欄直轉橫所產生的框架，來製作清單。使用在解除巢狀化階段中所產生的聯結鍵，將直轉橫的陣列欄聯結至根資料表。

- `frame` – 要進行關聯化的 DynamicFrame (必要)。
- `staging_path` – 該方法用來以 CSV 格式存放直轉橫資料表分區的路徑 (選用)。直轉橫資料表從這個路徑讀回。
- `name` – 根資料表的名稱 (選用)。
- `options` – 選用參數的字典。目前未使用。
- `transformation_ctx` – 用於識別狀態資訊的唯一字串 (選用)。
- `info` – 與轉換中的錯誤相關的字串 (選用)。
- `stageThreshold` – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用)。預設為零。
- `totalThreshold` – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用)。預設為零。

apply(cls, *args, **kwargs)

繼承自 `GlueTransform` apply。

name(cls)

繼承自 `GlueTransform` name。

describeArgs(cls)

繼承自 `GlueTransform` describeArgs。

describeReturn(cls)

繼承自 `GlueTransform` describeReturn。

describeTransform(cls)

繼承自 `GlueTransform` describeTransform。

describeErrors(cls)

繼承自 `GlueTransform` describeErrors。

describe(cls)

繼承自 `GlueTransform` describe。

RenameField 類別

重新命名 DynamicFrame 內的節點。

範例

建議您使用 DynamicFrame.rename_field() 方法重新命名 DynamicFrame 中的欄位。若要檢視程式碼範例，請參閱 範例：使用 rename_field 重新命名 DynamicFrame 中的欄位。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, old_name, new_name, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

重新命名 DynamicFrame 內的節點。

- frame - 要重新命名其中節點的 DynamicFrame (必要)。
- old_name – 要重新命名之節點的完整路徑 (必要)。

  如果舊名稱內有小點，RenameField 無法正常運作，除非在前後加上反引號 (``)。例如，若要將 this.old.name 換成 thisNewName，您可以用下列方式呼叫 RenameField：

  ```
  newDyF = RenameField(oldDyF, "`this.old.name`", "thisNewName")
  ```

- new_name – 新的名稱，包含完整路徑 (必要)。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與轉換中的錯誤相關的字串 (選用)。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用)。預設為零。

- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用)。預設為零。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

ResolveChoice 類別

解析 DynamicFrame 內的選擇類型。

範例

建議您使用 DynamicFrame.resolveChoice() 方法來處理 DynamicFrame 中包含多個類型的欄位。若要檢視程式碼範例，請參閱 範例：使用 resolveChoice 來處理包含多種類型的資料欄。

方法

- __call__
- apply

- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, specs = none, choice = "", transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

提供資訊以解析 DynamicFrame 內模棱兩可的類型。它會傳回產生的 DynamicFrame。

- frame – 要解析其中選擇類型的 DynamicFrame (必要)。
- specs – 要解析的特定模棱兩可項目的清單，形式皆為 tuple:(path, action)。path 值代表模棱兩可的特定元素，action 值則代表對應的解析動作。

  您只能使用 spec 和 choice 參數的其中一項。如果 spec 參數不是 None，則 choice 參數必須為空字串。相反地，如果 choice 不是空字串，則 spec 參數必須為 None。如果兩種參數皆未提供，則 AWS Glue 會試著剖析結構描述，並利用結構描述來解析模棱兩可的項目。

  可在 specs 元組的 action 部分中指定下列解析策略的其中一種：

  - cast - 可讓您指定轉換的目標類型 (例如 cast:int)。
  - make_cols – 透過將資料壓平合併來解析可能的模棱兩可項目。例如，如果 columnA 可能是 int 或 string，則在得出的 DynamicFrame 中，解析動作會產生名為 columnA_int 和 columnA_string 的兩個欄。
  - make_struct - 藉由以結構表示資料，來解決可能的模棱兩可項目。舉例來說，如果欄中的資料可能是 int 或 string，則使用 make_struct 動作會在結果的 DynamicFrame 中產生結構的欄，每個欄同時包含 int 和 string。
  - project - 在產生的 DynamicFrame 中只擷取指定種類的值，以此解析可能的模棱兩可項目。例如，如果 ChoiceType 欄中的資料可能是 int 或 string，指定 project:string 動作會從並非 string 類型產生的 DynamicFrame 捨棄欄。

  若 path 識別到陣列，在陣列的名稱後放置空白的方括號以避免模棱兩可的狀況。例如，假設您使用如下結構化的資料：

```
"myList": [
```

```
  { "price": 100.00 },
  { "price": "$100.00" }
]
```

您可以選取數值而不是價格字串版本，方法是將 path 設定為 "myList[].price"，且將 action 設定為 "cast:double"。

- choice – 當 specs 參數為 None 時的預設解析動作。如果 specs 參數不是 None，則此值只能為空字串，不能設定成其他的值。

  除了上述 specs 動作，此引數也支援下列動作：

  - MATCH_CATALOG – 嘗試將每個 ChoiceType 投射至指定 Data Catalog 資料表中的對應類型。
- database - 要搭配 MATCH_CATALOG 選擇使用的 AWS Glue Data Catalog 資料庫 (MATCH_CATALOG 需要)。
- table_name - 要搭配 MATCH_CATALOG 動作使用的 AWS Glue Data Catalog 資料表名稱 (MATCH_CATALOG 需要)。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與轉換中的錯誤相關的字串 (選用)。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用)。預設為零。
- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用)。預設為零。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

SelectFields 類別

SelectFields 類別建立新的 DynamicFrame 從現有 DynamicFrame，並僅保留您指定的欄位。SelectFields 提供類似 SQL SELECT 陳述式的功能。

範例

建議您使用 DynamicFrame.select_fields() 方法從 DynamicFrame 中選擇欄位。若要檢視程式碼範例，請參閱 範例：使用 select_fields 來用所選欄位建立新的 DynamicFrame。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

__call__(frame, paths, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

在 DynamicFrame 中取得欄位 (節點)。

- frame – 要在其中選擇欄位的 DynamicFrame (必要)。
- paths – 所要選擇的欄位的完整路徑清單 (必要)。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與轉換中的錯誤相關的字串 (選用)。

- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用)。預設為零。
- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用)。預設為零。

傳回僅包含指定欄位的新 DynamicFrame。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

SelectFromCollection 類別

在 DynamicFrameCollection 中選擇一個 DynamicFrame。

範例

此範例使用 SelectFromCollection 從 DynamicFrameCollection 中選取 DynamicFrame。

範例資料集

該範例從稱為 split_rows_collection 的 DynamicFrameCollection 中選取兩個 DynamicFrames。以下是 split_rows_collection 中的索引鍵清單。

```
dict_keys(['high', 'low'])
```

範例程式碼

```
# Example: Use SelectFromCollection to select
# DynamicFrames from a DynamicFrameCollection

from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import SelectFromCollection

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Select frames and inspect entries
frame_low = SelectFromCollection.apply(dfc=split_rows_collection, key="low")
frame_low.toDF().show()

frame_high = SelectFromCollection.apply(dfc=split_rows_collection, key="high")
frame_high.toDF().show()
```

輸出

```
+---+-----+-----------------------+-------------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-------------------------+
|  1|    0|                    fax|             202-225-3307|
|  1|    1|                  phone|             202-225-5731|
|  2|    0|                    fax|             202-225-3307|
|  2|    1|                  phone|             202-225-5731|
|  3|    0|                    fax|             202-225-3307|
|  3|    1|                  phone|             202-225-5731|
|  4|    0|                    fax|             202-225-3307|
|  4|    1|                  phone|             202-225-5731|
|  5|    0|                    fax|             202-225-3307|
|  5|    1|                  phone|             202-225-5731|
|  6|    0|                    fax|             202-225-3307|
|  6|    1|                  phone|             202-225-5731|
```

```
|  7|    0|                     fax|             202-225-3307|
|  7|    1|                   phone|             202-225-5731|
|  8|    0|                     fax|             202-225-3307|
|  8|    1|                   phone|             202-225-5731|
|  9|    0|                     fax|             202-225-3307|
|  9|    1|                   phone|             202-225-5731|
| 10|    0|                     fax|             202-225-6328|
| 10|    1|                   phone|             202-225-4576|
+---+-----+----------------------+-------------------------+
only showing top 20 rows

+---+-----+----------------------+-------------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+----------------------+-------------------------+
| 11|    0|                     fax|             202-225-6328|
| 11|    1|                   phone|             202-225-4576|
| 11|    2|                 twitter|           RepTrentFranks|
| 12|    0|                     fax|             202-225-6328|
| 12|    1|                   phone|             202-225-4576|
| 12|    2|                 twitter|           RepTrentFranks|
| 13|    0|                     fax|             202-225-6328|
| 13|    1|                   phone|             202-225-4576|
| 13|    2|                 twitter|           RepTrentFranks|
| 14|    0|                     fax|             202-225-6328|
| 14|    1|                   phone|             202-225-4576|
| 14|    2|                 twitter|           RepTrentFranks|
| 15|    0|                     fax|             202-225-6328|
| 15|    1|                   phone|             202-225-4576|
| 15|    2|                 twitter|           RepTrentFranks|
| 16|    0|                     fax|             202-225-6328|
| 16|    1|                   phone|             202-225-4576|
| 16|    2|                 twitter|           RepTrentFranks|
| 17|    0|                     fax|             202-225-6328|
| 17|    1|                   phone|             202-225-4576|
+---+-----+----------------------+-------------------------+
only showing top 20 rows
```

方法

- __call__
- apply
- name

- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(dfc, key, transformation_ctx = "")

從 DynamicFrameCollection 取得一個 DynamicFrame。

- dfc – 應該從其中選取 DynamicFrame 的 DynamicFrameCollection (必要)。
- key – 所要選擇的 DynamicFrame 金鑰 (必要)。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

簡化 JSON 類別

簡化中特別位於 DynamoDB JSON 結構中的巢狀資料行，並傳回新的簡化。DynamicFrame DynamicFrame

範例

我們建議您使用此方DynamicFrame.simplify_ddb_json()法來簡化特別位於 DynamoDB JSON 結構中的巢狀資料行。DynamicFrame若要檢視程式碼範例，請參閱 範例：使用簡化方式來叫 DynamoDB JSON 簡化。

Spigot 類別

將範例記錄寫入指定的目的地，以協助您驗證 AWS Glue 任務執行的轉換。

範例

建議您使用 DynamicFrame.spigot() 方法，將記錄子集從 DynamicFrame 寫入指定目的地。若要檢視程式碼範例，請參閱 範例：使用 spigot 將範例欄位從 DynamicFrame 寫入到 Amazon S3。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, path, options, transformation_ctx = "")

在轉換期間將範例記錄寫入到指定的目的地。

- frame – 欲 Spigot 的 DynamicFrame (必要)。
- path - 要寫入的目的地路徑 (必要)。

- options - 指定選項的 JSON 索引鍵/值對 (選用)。"topk" 選項指定應寫入前 k 個記錄。"prob" 選項指定挑選任何給定記錄的概率 (小數)。您可以使用它來選擇要寫入的記錄。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply

name(cls)

繼承自 GlueTransform name

describeArgs(cls)

繼承自 GlueTransform describeArgs

describeReturn(cls)

繼承自 GlueTransform describeReturn

describeTransform(cls)

繼承自 GlueTransform describeTransform

describeErrors(cls)

繼承自 GlueTransform describeErrors

describe(cls)

繼承自 GlueTransform describe

SplitFields 類別

以指定欄位將 DynamicFrame 分割為二。

範例

建議您使用 DynamicFrame.split_fields() 方法分割 DynamicFrame 中的欄位。若要檢視程式碼範例，請參閱 範例：使用 split_fields 將選取的欄位分割為單獨的 DynamicFrame。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, paths, name1 = none, name2 = none, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

將 DynamicFrame 中一個或多個欄位分割成新的 DynamicFrame，並且建立另一個包含剩餘欄位的新 DynamicFrame。

- frame – 要分割為二的來源 DynamicFrame (必要)。
- paths – 欲分割欄位的完整路徑清單 (必要)。
- name1 指派給 DynamicFrame 的名稱，其中包含要分割的欄位 (選用)。如果未提供名稱，則會使用來源架構的名稱並加上「1」。
- name2 – 指派給 DynamicFrame 的名稱，其中包含指定欄位分割後剩餘的欄位 (選用)。如果未提供名稱，則會使用來源架構的名稱並加上「2」。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與轉換中的錯誤相關的字串 (選用)。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用)。預設為零。
- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用)。預設為零。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

SplitRows 類別

建立 DynamicFrameCollection，其包含兩個 DynamicFrames。一個 DynamicFrame 僅包含要分割的指定資料列，另一個則包含所有剩餘的資料列。

範例

建議您使用 DynamicFrame.split_rows() 方法分割 DynamicFrame 中的資料列。若要檢視程式碼範例，請參閱 範例：使用 split_rows 來分割 DynamicFrame 中的列。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, comparison_dict, name1="frame1", name2="frame2", transformation_ctx = "", info = none, stageThreshold = 0, totalThreshold = 0)

將 DynamicFrame 中一個或多個欄分割成新的 DynamicFrame。

- frame – 要分割為二的來源 DynamicFrame (必要)。
- comparison_dict – 一個字典，其中索引鍵為欄位的完整路徑，而對於與欄位數值相比較的數值而言，此數值為另一種字典映射比較運算子。例如，{"age": {">": 10, "<": 20}} 會特別將介於 10 到 20 之間的 "age" (年齡) 數值與該範圍外的 "age" 列分割開來 (必要)。
- name1 – 指派給 DynamicFrame 的名稱，其中包含要分割的資料列 (選用)。
- name2 – 指派給 DynamicFrame 的名稱，其中包含指定資料列分割後剩餘的資料列 (選用)。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與轉換中的錯誤相關的字串 (選用)。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用)。預設為零。
- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用)。預設為零。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

Unbox 類別

將 DynamicFrame 中的字串欄位拆箱 (重新格式化)。

範例

建議您使用 DynamicFrame.unbox() 方法，對 DynamicFrame 中的欄位進行拆箱。若要檢視程式碼範例，請參閱 範例：使用 unbox 將字串欄位拆箱到結構中。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, path, format, transformation_ctx = "", info="", stageThreshold=0, totalThreshold=0, **options)

將 DynamicFrame 中的字串欄位拆箱。

- frame - 具有要拆箱之欄位的 DynamicFrame (必要)。
- path – 欲拆箱的 StringNode 之完整路徑 (必要)。
- format – 格式化規格 (選用)。這是用於 Amazon S3 或支援多種格式的 AWS Glue 連線。如需了解受支援的格式，請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與轉換中的錯誤相關的字串 (選用)。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用)。預設為零。

- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用)。預設為零。
- separator – 分隔符號符記 (選用)。
- escaper – 逸出符記 (選用)。
- skipFirst – 如果資料的第一行應略過則為 True，如果不應略過則為 False (選用)。
- withSchema – 這是一個字串，包含了要拆箱之資料的結構描述 (選用)。此字串應一律使用 StructType.json 來建立。
- withHeader – 如果被解壓縮的資料包含標頭則為 True，若無則為 False (選用)。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

UnnestFrame 類別

對 DynamicFrame 解除巢狀化，將巢狀化物件壓平合併為頂層元素，並且為陣列物件產生聯結鍵。

範例

我們建議您使用 DynamicFrame.unnest() 方法在 DynamicFrame 中壓平合併巢狀化結構。若要檢視程式碼範例，請參閱 範例：使用 unnest 將巢狀化欄位轉換為頂層欄位。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, transformation_ctx = "", info="", stageThreshold=0, totalThreshold=0)

對 DynamicFrame 解除巢狀化，將巢狀化物件壓平合併為頂層元素，並且為陣列物件產生聯結鍵。

- frame – 欲解巢狀的 DynamicFrame (必要)。
- transformation_ctx – 用於識別狀態資訊的唯一字串 (選用)。
- info – 與轉換中的錯誤相關的字串 (選用)。
- stageThreshold – 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用)。預設為零。
- totalThreshold – 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用)。預設為零。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

FlagDuplicatesInColumn 類

FlagDuplicatesInColumn轉換作業會傳回每一列中含有指定值的新資料欄，指出資料列的來源資料欄中的值是否與來源資料欄較早列中的值相符。找到相符項目時，會將它們標記為重複項目。不會標記初始出現位置，因為它與先前的列不相符。

範例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

datasource1 = spark.read.json("s3://${BUCKET}/json/zips/raw/data")

try:
    df_output = column.FlagDuplicatesInColumn.apply(
        data_frame=datasource1,
        spark_context=sc,
        source_column="city",
        target_column="flag_col",
        true_string="True",
        false_string="False"
    )
except:
```

```
    print("Unexpected Error happened ")
    raise
```

輸出

FlagDuplicatesInColumn轉換將會在 `df_col' 中添加一個新的列 'df 輸出'。 DataFrame該列將包含一個字符串值，指示相應的行是否在 `city` 列或沒有重複的值。如果一行具有重複的「城市」值，則 `flag_col` 將包含 'true' 字符串 '值「真」。如果一行具有唯一的「城市」值，則「FLAG_COL」將包含「假」字符串值。

由此產生的 `df_output` DataFrame 將包含來自原始「資料庫 1」的所有欄位，再加上額外的 'flag_col' 欄，表示重複的 ` DataFrame城市' 值。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（閃光上下文，數據框架，源列，目標列，真實字符串 = 默認 _ 真實 _ 字符串，假字符串 = 默認 _ 假字符串）

FlagDuplicatesInColumn轉換作業會傳回每一列中含有指定值的新資料欄，指出資料列的來源資料欄中的值是否與來源資料欄較早列中的值相符。找到相符項目時，會將它們標記為重複項目。不會標記初始出現位置，因為它與先前的列不相符。

- `source_column`— 來源資料欄的名稱。
- `target_column`— 目標資料行的名稱。
- `true_string`— 當來源資料行值複製該欄中較早的值時，要在目標資料行中插入的字串。
- `false_string`— 當來源資料行值與該資料行中較早的值不同時，要在目標資料行中插入的字串。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

FormatPhoneNumber 類

轉FormatPhoneNumber換指令會傳回一欄，其中電話號碼字串會轉換成格式化值。

範例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
```

```
        ("408-341-5669",),
        ("4083415669",)
    ],
    ["phone"],
)

try:
    df_output = column_formatting.FormatPhoneNumber.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="phone",
        default_region="US"
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

輸出

輸出將是：

```
```
+---------------+
| phone|
+---------------+
|(408) 341-5669|
|(408) 341-5669|
+---------------+
```
```

FormatPhoneNumber轉型採用「源列」作為「電話」和「默認區域」作為「美國」`。

轉換成功地將兩個電話號碼（無論其初始格式如何）格式化為標準的美國格式 `(408) 341-5669`。

方法

- __call__
- apply
- name
- describeArgs

- describeReturn
- describeTransform
- describeErrors
- describe

__call__（閃光上下文，數據框架，源列，電話號碼格式 = 無，默認 _ 區域 = 無，默認 _ 區域 _ 列 = 無）

轉FormatPhoneNumber換指令會傳回一欄，其中電話號碼字串會轉換成格式化值。

- source_column – 現有資料欄的名稱。
- phone_number_format— 將電話號碼轉換為的格式。如果未指定格式，則預設為E.164國際識別的標準電話號碼格式。有效值包括以下項目：
  - E164（省略 E 之後的期間）
- default_region— 由兩個或三個大寫字母組成的有效地區代碼，當號碼本身中沒有國家/地區代碼時，指定電話號碼的區域。最多可defaultRegionColumn以提供defaultRegion或之一。
- default_region_column— 進階資料類型的欄名稱Country。當號碼本身沒有國家/地區代碼時，指定列中的區域代碼用於確定電話號碼的國家/地區代碼。最多可defaultRegionColumn以提供defaultRegion或之一。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

FormatCase 類

FormatCase轉換將列中的每個字符串更改為指定的大小寫類型。

範例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

datasource1 = spark.read.json("s3://${BUCKET}/json/zips/raw/data")

try:
    df_output = data_cleaning.FormatCase.apply(
        data_frame=datasource1,
        spark_context=sc,
        source_column="city",
        case_type="LOWER"
    )
except:
    print("Unexpected Error happened ")
    raise
```

輸出

FormatCase轉換會根據 'CASE_TYPE = "小寫"' 參數，將 `city` 資料欄中的值轉換為小寫。由此產生的 `df_output` DataFrame 將包含來自原始「資料庫 1」的所有資料欄，但是 DataFrame以小寫形式顯示的「城市」欄值。

方法

- __call__

- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（閃光上下文，數據框架，源列，案例類型）

FormatCase轉換將列中的每個字符串更改為指定的大小寫類型。

- source_column – 現有資料欄的名稱。
- case_type— 支援的案例類型為CAPITALLOWER、UPPER、、SENTENCE。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

FillWithMode 類別

FillWithMode轉換會根據您指定的電話號碼格式來格式化欄。您也可以指定平局斷路器邏輯，其中一些值是相同的。例如，請考慮下列值：1 2 2 3 3 4

MINIMUM導致FillWithMode返回 2 作為模式值的模式類型。如果模式類型為MAXIMUM，則模式為 3。對於AVERAGE，模式為 2.5。

範例

```
from awsglue.context import *
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (105.111, 13.12),
        (1055.123, 13.12),
        (None, 13.12),
        (13.12, 13.12),
        (None, 13.12),
    ],
    ["source_column_1", "source_column_2"],
)

try:
    df_output = data_quality.FillWithMode.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column_1",
        mode_type="MAXIMUM"
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

輸出

給定代碼的輸出將是：

```
```
+---------------+---------------+
|source_column_1|source_column_2|
+---------------+---------------+
105.111	13.12
1055.123	13.12
1055.123	13.12
13.12	13.12
1055.123	13.12
+---------------+---------------+
```
```

從 'awsglu.data_質量` 模塊的`FillWithMode`轉換被應用到 '輸入 _df`。 DataFrame它用該列中的非空值的最大值（`Mode_TYPE =「最大值」）替換`source_column_1`列中的「空」值。

在這種情況下，該列中的最大`source_column_1`值是「1055.123」。因此，中的「空值」值會在輸出「df_out」中`source_column_1`被「1055.123」取代。 DataFrame

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（閃光上下文，數據框架，源列，模式類型）

轉`FillWithMode`換格式化列中字符串的情況。

- source_column – 現有資料欄的名稱。
- mode_type— 如何解決數據中的平局值。此值必須是MINIMUM、NONEAVERAGE、或之一MAXIMUM。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

FlagDuplicateRows 類

FlagDuplicateRows轉換作業會傳回每一列中指定值的新資料欄，指出該資料列是否與資料集中較早的資料列完全相符。找到相符項目時，會將它們標記為重複項目。不會標記初始出現位置，因為它與先前的列不相符。

範例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
```

```
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (105.111, 13.12),
        (13.12, 13.12),
        (None, 13.12),
        (13.12, 13.12),
        (None, 13.12),
    ],
    ["source_column_1", "source_column_2"],
)

try:
    df_output = data_quality.FlagDuplicateRows.apply(
        data_frame=input_df,
        spark_context=sc,
        target_column="flag_row",
        true_string="True",
        false_string="False",
        target_index=1
    )
except:
    print("Unexpected Error happened ")
    raise
```

輸出

輸出將是一個 PySpark DataFrame 附加列，`flag_row`該列根據`source_column_1`列指示行是否重複。產生的「df_輸出」 DataFrame 將包含下列資料列：

```
```
+---------------+---------------+--------+
|source_column_1|source_column_2|flag_row|
+---------------+---------------+--------+
105.111	13.12	False
13.12	13.12	True
null	13.12	True
13.12	13.12	True
```
```

```
| null| 13.12| True|
+---------------+---------------+--------+
```

該flag_row列指示行是否重複。「真實字串」設定為「真」，而「假字串」則設定為「假」。該 `target_index` 被設置為 1，這意味著該flag_row列將被插入到輸出的第二個位置（索引 1）。DataFrame

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（閃光上下文，數據框架，目標 _ 列，真 _ 字符串 = 默認 _ 真實 _ 字符串，假字符串 = 默認 _ 假字符串，目標 _ 索引 = 無）

FlagDuplicateRows轉換作業會傳回每一列中指定值的新資料欄，指出該資料列是否與資料集中較早的資料列完全相符。找到相符項目時，會將它們標記為重複項目。不會標記初始出現位置，因為它與先前的列不相符。

- true_string— 如果列與較早的列相符，則要插入的值。
- false_string— 如果列是唯一的，則要插入的值。
- target_column— 插入資料集中的新資料行名稱。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

RemoveDuplicates 類

如果選取的來源資料欄中遇到重複的值，則RemoveDuplicates轉換指令會刪除整個資料列。

範例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (105.111, 13.12),
        (13.12, 13.12),
        (None, 13.12),
        (13.12, 13.12),
        (None, 13.12),
    ],
    ["source_column_1", "source_column_2"],
)
```

```
try:
    df_output = data_quality.RemoveDuplicates.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column_1"
    )
except:
    print("Unexpected Error happened ")
    raise
```

輸出

輸出將是 PySpark DataFrame 根據source_column_1列刪除重複項。產生的「df_輸出」DataFrame 將包含下列資料列：

```
```
+---------------+---------------+
|source_column_1|source_column_2|
+---------------+---------------+
105.111	13.12
13.12	13.12
null	13.12
+---------------+---------------+
```
```

請注意，source_column_1值為「13.12`」和「null」的列只會在輸出中出現一次 DataFrame，因為重複項已根據欄移除。source_column_1

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（閃光上下文，數據框架，源列）

如果選取的來源資料欄中遇到重複的值，則RemoveDuplicates轉換指令會刪除整個資料列。

- source_column – 現有資料欄的名稱。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

MonthName 類

MonthName轉換會從代表日期的字串中建立包含月份名稱的新資料欄。

範例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *
```

```
sc = SparkContext()
spark = SparkSession(sc)

spark.conf.set("spark.sql.legacy.timeParserPolicy", "LEGACY")

input_df = spark.createDataFrame(
    [
        ("20-2018-12",),
        ("2018-20-12",),
        ("20182012",),
        ("12202018",),
        ("20122018",),
        ("20-12-2018",),
        ("12/20/2018",),
        ("02/02/02",),
        ("02 02 2009",),
        ("02/02/2009",),
        ("August/02/2009",),
        ("02/june/2009",),
        ("02/2020/june",),
        ("2013-02-21 06:35:45.658505",),
        ("August 02 2009",),
        ("2013/02/21",),
        (None,),
    ],
    ["column_1"],
)

try:
    df_output = datetime_functions.MonthName.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="column_1",
        target_column="target_column"
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

輸出

輸出將是：

```
+------------+------------+
| column_1|target_column|
+------------+------------+
|20-2018-12 | December |
|2018-20-12 | null |
| 20182012| null |
| 12202018| null |
| 20122018| null |
|20-12-2018 | December |
|12/20/2018 | December |
| 02/02/02 | February |
|02 02 2009 | February |
|02/02/2009 | February |
|August/02/2009| August |
|02/june/2009| null |
|02/2020/june| null |
|2013-02-21 06:35:45.658505| February |
|August 02 2009| August |
| 2013/02/21| February |
| null | null |
+------------+------------+
```

MonthName轉換採用「源列」作為「列 1」和「目標列」作為「目標列」`。它試圖從「列 1」列中的日期/時間字符串中提取月份名稱，並將其放置在「目標列」列中。如果日期/時間字串的格式無法辨識或無法剖析，則「target_column」值會設定為「空值」。

此次轉換成功地從各種日期/時間格式中提取月份名稱，例如「二零一八年十二月二十二日」、「二零零九年二月二十二日」、「2013-02-21 06:35：45.658 505」和「二零零九年八月二日」。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors

- describe

__call__（閃光上下文，數據框架，目標列，源列 = 無，值 = 無）

MonthName轉換會從代表日期的字串中建立包含月份名稱的新資料欄。

- source_column – 現有資料欄的名稱。
- value— 要評估的字元字串。
- target_column— 新建立欄的名稱。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

IsEven 類別

IsEven轉換會在新資料欄中傳回 Boolean 值，指出來源資料欄或值是否為偶數。如果來源資料行或值是十進位，則結果為 false。

範例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [(5,), (0,), (-1,), (2,), (None,)],
    ["source_column"],
)

try:
    df_output = math_functions.IsEven.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column",
        target_column="target_column",
        value=None,
        true_string="Even",
        false_string="Not even",
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

輸出

輸出將是：

```
```
+-------------+-------------+
|source_column|target_column|
+-------------+-------------+
5	Not even
0	Even
-1	Not even
2	Even
null	null
```
```

```
+------------+------------+
```
```

IsEven轉換需要「源列」作為「源列」和「目標列」作為「目標列」。它會檢查「源 _ 列」中的值是否為偶數。如果該值是偶數，它將「目標列」值設置為「真實字符串」「偶數」。如果該值是奇數，它將「目標列」值設置為「假字符串」「不偶數」。如果「來源欄」值為「空」，則「目標欄」值會設定為「空值」。

轉換可正確識別偶數（0 和 2），並將「目標列」值設置為「偶數」。對於奇數（5 和 -1），它將「目標列」值設置為「不偶數」。對於「來源列」中的「空值」，「目標列」值設置為「空」。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（閃光上下文，數據框架，目標列，源列 = 無，真實字符串 = 默認 _ 真實 _ 字符串，假字符串 = 默認 _ 假字符串，值 = 無）

IsEven轉換會在新資料欄中傳回 Boolean 值，指出來源資料欄或值是否為偶數。如果來源資料行或值是十進位，則結果為 false。

- `source_column` – 現有資料欄的名稱。
- `target_column`— 要建立的新欄名稱。
- `true_string`— 字串，指出值是否為偶數。
- `false_string`— 字串，指出值是否為偶數。

apply(cls, *args, **kwargs)

繼承自 `GlueTransform` apply。
```

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

CryptographicHash 類別

CryptographicHash轉換會將演算法套用至資料行中的雜湊值。

範例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

secret = "${SECRET}"
sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (1, "1234560000"),
        (2, "1234560001"),
        (3, "1234560002"),
        (4, "1234560003"),
        (5, "1234560004"),
        (6, "1234560005"),
```

```
        (7, "1234560006"),
        (8, "1234560007"),
        (9, "1234560008"),
        (10, "1234560009"),
    ],
    ["id", "phone"],
)

try:
    df_output = pii.CryptographicHash.apply(
        data_frame=input_df,
        spark_context=sc,
        source_columns=["id", "phone"],
        secret_id=secret,
        algorithm="HMAC_SHA256",
        output_format="BASE64",
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

輸出

輸出將是：

```
+---+------------+--------------------+--------------------+
| id| phone | id_hashed | phone_hashed |
+---+------------+--------------------+--------------------+
| 1| 1234560000 | QUI1zXTJiXmfIb... | juDBAmiRnnO3g... |
| 2| 1234560001 | ZAUWiZ3dVTzCo... | vC8lgUqBVDMNQ... |
| 3| 1234560002 | ZP4VvZWkqYifu... | Kl3QAkgswYpzB... |
| 4| 1234560003 | 3u8vO3wQ8EQfj... | CPBzK1P8PZZkV... |
| 5| 1234560004 | eWkQJk4zAOIzx... | aLf7+mHcXqbLs... |
| 6| 1234560005 | xtI9fZCJZCvsa... | dy2DFgdYWmr0p... |
| 7| 1234560006 | iW9hew7jnHuOf... | wwfGMCOEv6oOv... |
| 8| 1234560007 | H9V1pqvgkFhfS... | g9WKhagIXy9ht... |
| 9| 1234560008 | xDhEuHaxAUbU5... | b3uQLKPY+Q5vU... |
| 10| 1234560009 | GRN6nFXkxk349... | VJdsKt8VbxBbt... |
+---+------------+--------------------+--------------------+
```

此轉換會使用指定的演算法和密鑰來計算 `id` 和 `phone` 資料欄中值的加密雜湊值，並以 Base64 格式編碼雜湊。產生的「df_output」 DataFrame 包含來自原始「輸入」df` 的所有資料欄 DataFrame，加上附加的「id_hashed`」和「已連結」資料欄以及計算出來的雜湊值。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（閃光上下文，數據框架，源列，秘密 ID，算法 = 無，秘密版本 = 無，創建秘密 _ 如果缺失 = 假，輸出格式 = 無，實體 _ 類型 _ 過濾器 = 無）

`CryptographicHash`轉換會將演算法套用至資料行中的雜湊值。

- `source_columns`— 現有資料行的陣列。
- `secret_id`— Secrets Manager 秘密密鑰的 ARN. 雜湊型訊息驗證碼 (HMAC) 前置碼演算法中用來雜湊來源資料行的金鑰。
- `secret_version` - 選用。預設為最新的密碼版本。
- `entity_type_filter`— 實體類型的選擇性陣列。可用於僅加密自由文字資料行中偵測到的 PII。
- `create_secret_if_missing`-可選布爾值。如果為 true，則會嘗試代表呼叫者建立密碼。
- `algorithm`— 用於雜湊資料的演算法。有效的枚舉值：MD5，沙 1，SHA256，SHA512，HMAC_MD5，HMAC_SHA1，HMAC_SHA256，哈 512。

apply(cls, *args, **kwargs)

繼承自 `GlueTransform` apply。

name(cls)

繼承自 `GlueTransform` name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

解密類

轉Decrypt換解密 AWS Glue 內部。您也可以使用 AWS 加密 SDK 在 AWS Glue 外部解密您的資料。如果提供的 KMS 金鑰 ARN 不符合用來加密資料行的項目，解密作業就會失敗。

範例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

kms = "${KMS}"
sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (1, "1234560000"),
        (2, "1234560001"),
        (3, "1234560002"),
        (4, "1234560003"),
        (5, "1234560004"),
        (6, "1234560005"),
        (7, "1234560006"),
        (8, "1234560007"),
```

```
        (9, "1234560008"),
        (10, "1234560009"),
    ],
    ["id", "phone"],
)

try:
    df_encrypt = pii.Encrypt.apply(
        data_frame=input_df,
        spark_context=sc,
        source_columns=["phone"],
        kms_key_arn=kms
    )
    df_decrypt = pii.Decrypt.apply(
        data_frame=df_encrypt,
        spark_context=sc,
        source_columns=["phone"],
        kms_key_arn=kms
    )
    df_decrypt.show()
except:
    print("Unexpected Error happened ")
    raise
```

輸出

輸出將是一個原 PySpark DataFrame 始的 `id` 列和解密的「電話」列：

```
```
+---+------------+
| id| phone|
+---+------------+
1	1234560000
2	1234560001
3	1234560002
4	1234560003
5	1234560004
6	1234560005
7	1234560006
8	1234560007
9	1234560008
10	1234560009
+---+------------+
```
```

```
```

Encrypt轉換程式會將「來源欄」作為「[電話」] `和「kms_key_arn」作為環境變數的值。轉換作業會使用指定的 KMS 金鑰加密 `phone` 資料行中的值。然後將加密的 DataFrame `df_加密` 傳遞給從 'awsglue.pii` 模塊的轉Decrypt換。它需要「來源列」作為「[" 電話 "] `和 `kms_key_arn '作為環境變量的值。轉換作業會使用相同的 KMS 金鑰，將 `phone` 資料行中的加密值解密。產生的「df_crypt`」DataFrame 包含原始的「ID」列和解密的「電話」列。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（閃光上下文，數據框架，源列，kms_key_arn）

轉Decrypt換解密 AWS Glue 內部。您也可以使用 AWS 加密 SDK 在 AWS Glue 外部解密您的資料。如果提供的 KMS 金鑰 ARN 不符合用來加密資料行的項目，解密作業就會失敗。

- `source_columns`— 現有資料行的陣列。
- `kms_key_arn`— 用來解密來源資料行的 AWS 金鑰管理服務金鑰的金鑰 ARN。

apply(cls, *args, **kwargs)

繼承自 `GlueTransform` apply。

name(cls)

繼承自 `GlueTransform` name。

describeArgs(cls)

繼承自 `GlueTransform` describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

加密類

Encrypt轉換作業會使用 AWS 金鑰管理服務金鑰加密來源資料行。轉Encrypt換每個儲存格最多可加密 128 MiB。它將嘗試保留解密的格式。若要保留資料類型，資料類型中繼資料必須序列化為小於 1KB。否則，您必須將preserve_data_type參數設定為 false。資料類型中繼資料將以純文字儲存在加密內容中。

範例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

kms = "${KMS}"
sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (1, "1234560000"),
        (2, "1234560001"),
        (3, "1234560002"),
        (4, "1234560003"),
        (5, "1234560004"),
        (6, "1234560005"),
        (7, "1234560006"),
        (8, "1234560007"),
        (9, "1234560008"),
```

```
        (10, "1234560009"),
    ],
    ["id", "phone"],
)

try:
    df_encrypt = pii.Encrypt.apply(
        data_frame=input_df,
        spark_context=sc,
        source_columns=["phone"],
        kms_key_arn=kms
    )
except:
    print("Unexpected Error happened ")
    raise
```

輸出

輸出將是一個 PySpark DataFrame 包含原始 `id` 列和一個包含 `phone` 列的加密值的附加列。

```
```
+---+------------+-------------------------+
| id| phone | phone_encrypted |
+---+------------+-------------------------+
1	1234560000	EncryptedData1234...abc
2	1234560001	EncryptedData5678...def
3	1234560002	EncryptedData9012...ghi
4	1234560003	EncryptedData3456...jkl
5	1234560004	EncryptedData7890...mno
6	1234560005	EncryptedData1234...pqr
7	1234560006	EncryptedData5678...stu
8	1234560007	EncryptedData9012...vwx
9	1234560008	EncryptedData3456...yz0
10	1234560009	EncryptedData7890...123
+---+------------+-------------------------+
```
```

Encrypt轉換程式會將「來源欄」作為「[電話」] 和「kms_key_arn」作為「$ {KMS}」環境變數的值。轉換作業會使用指定的 KMS 金鑰加密 `phone` 資料行中的值。結果「df_encrypted」 DataFrame 包含原始的「ID」欄、原始的「電話」欄，以及另一個名為「phone_encrypted」的資料欄，其中包含「電話」欄的加密值。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（閃光上下文，數據框架，源列，kms_key_arn，實體類型 _ 過濾器 = 無，前置數據 _ 類型 = 無）

Encrypt轉換作業會使用 AWS 金鑰管理服務金鑰加密來源資料行。

- source_columns— 現有資料行的陣列。
- kms_key_arn— 用來加密來源資料行的 AWS 金鑰管理服務金鑰的金鑰 ARN。
- entity_type_filter— 可選的實體類型陣列。可用於僅加密自由文字資料行中偵測到的 PII。
- preserve_data_type-可選布爾值。預設為 true。如果為 false，數據類型將不被存儲。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

IntToIp 類

轉IntToIp換指令會將來源資料行或其他值的整數值轉換為目標資料行中對應的 IPv4 值，並在新資料行中傳回結果。

範例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (3221225473,),
        (0,),
        (1,),
        (100,),
        (168430090,),
        (4294967295,),
        (4294967294,),
        (4294967296,),
        (-1,),
        (None,),
    ],
    ["source_column_int"],
)

try:
```

```
    df_output = web_functions.IntToIp.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column_int",
        target_column="target_column",
        value=None
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

輸出

輸出將是：

```
```
+---------------+---------------+
|source_column_int|target_column|
+---------------+---------------+
3221225473	192.0.0.1
0	0.0.0.0
1	0.0.0.1
100	0.0.0.100
168430090	10.0.0.10
4294967295	255.255.255.255
4294967294	255.255.255.254
4294967296	null
-1	null
null	null
+---------------+---------------+
```
```

此IntToIp.apply轉換會將「來源欄」作為「來源欄」，「目標欄」為「目標欄」，並將「來源欄」欄中的整數值轉換為對應的 IPv4 位址表示法，並將結果儲存在「目標資料欄」欄中。

對於 IPv4 位址範圍內的有效整數值 (0 至 4294967295)，轉換會成功地將它們轉換成它們的 IPv4 位址表示 (例如，192.0.0.1、0.0.0、10.0.10、255.255.255.255)。

對於超出有效範圍的整數值 (例如 4294967296、-1)，「目標欄」值會設定為「空值」。對於「源列」列中的「空」值，「目標列」值也設置為空值。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（閃光上下文，數據框架，目標列，源列 = 無，值 = 無）

轉IntToIp換指令會將來源資料行或其他值的整數值轉換為目標資料行中對應的 IPv4 值，並在新資料行中傳回結果。

- sourceColumn – 現有資料欄的名稱。
- value— 要評估的字元字串。
- targetColumn— 要建立的新欄名稱。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 GlueTransform describe。

IpToInt 類

IpToInt轉換作業會將來源資料行或其他值的網際網路通訊協定第 4 版 (IPv4) 值轉換為目標資料行中對應的整數值，並在新資料欄中傳回結果。

範例

對於 AWS Glue 4.0 及更新版本，建立或更新工作引數 key: --enable-glue-di-transforms, value: true

```
from pyspark.context import SparkContext
from awsgluedi.transforms import *

sc = SparkContext()

input_df = spark.createDataFrame(
    [
        ("192.0.0.1",),
        ("10.10.10.10",),
        ("1.2.3.4",),
        ("1.2.3.6",),
        ("http://12.13.14.15",),
        ("https://16.17.18.19",),
        ("1.2.3.4",),
        (None,),
        ("abc",),
        ("abc.abc.abc.abc",),
        ("321.123.123.123",),
        ("244.4.4.4",),
        ("255.255.255.255",),
    ],
    ["source_column_ip"],
)

    df_output = web_functions.IpToInt.apply(
        data_frame=input_df,
```

```
        spark_context=sc,
        source_column="source_column_ip",
        target_column="target_column",
        value=None
    )
    df_output.show()
```

輸出

輸出將是：

```
```
+----------------+---------------+
|source_column_ip| target_column|
+----------------+---------------+
192.0.0.1	3221225473
10.10.10.10	168427722
1.2.3.4	16909060
1.2.3.6	16909062
http://12.13.14.15	null
https://16.17.18.19	null
1.2.3.4	16909060
null	null
abc	null
abc.abc.abc.abc	null
321.123.123.123	null
244.4.4.4	4102444804
255.255.255.255	4294967295
+----------------+---------------+
```
```

此IpToInt轉換會將「來源欄」作為「來源欄」，「目標欄」為「目標欄」，然後將「來源欄」中的有效 IPv4 位址字串轉換為對應的 32 位元整數表示，並將結果儲存在「目標資料欄」資料欄中。

對於有效的 IPv4 位址字串 (例如，「192.0.0.1」、「10.10.10」、「1.2.4 3.4」)，轉換會成功地將它們轉換成整數表示 (例如，3221225473、168427722、16909060)。對於不是有效的 IPv4 地址的字符串（例如，網址，非 IP 字符串（如「abc」），無效的 IP 格式（如「abc.abc.abc」），「目標列」值被設置為「空」。對於「源列」列中的「空」值，「目標列」值也設置為空值。

方法

- __call__

- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（閃光上下文，數據框架，目標列，源列 = 無，值 = 無）

IpToInt轉換作業會將來源資料行或其他值的網際網路通訊協定第 4 版 (IPv4) 值轉換為目標資料行中對應的整數值，並在新資料欄中傳回結果。

- sourceColumn – 現有資料欄的名稱。
- value— 要評估的字元字串。
- targetColumn— 要建立的新欄名稱。

apply(cls, *args, **kwargs)

繼承自 GlueTransform apply。

name(cls)

繼承自 GlueTransform name。

describeArgs(cls)

繼承自 GlueTransform describeArgs。

describeReturn(cls)

繼承自 GlueTransform describeReturn。

describeTransform(cls)

繼承自 GlueTransform describeTransform。

describeErrors(cls)

繼承自 GlueTransform describeErrors。

describe(cls)

繼承自 `GlueTransform` describe。

資料整合轉換

對於 AWS Glue 4.0 及更新版本，請使用`key: --enable-glue-di-transforms, value: true`.

工作指令碼範例：

```
from pyspark.context import SparkContext

from awsgluedi.transforms import *
sc = SparkContext()

input_df = spark.createDataFrame(
    [(5,), (0,), (-1,), (2,), (None,)],
    ["source_column"],
)

try:
    df_output = math_functions.IsEven.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column",
        target_column="target_column",
        value=None,
        true_string="Even",
        false_string="Not even",
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

使用筆記本工作階段

```
%idle_timeout 2880
%glue_version 4.0
%worker_type G.1X
%number_of_workers 5
```

```
%region eu-west-1
```

```
%%configure
{
    "--enable-glue-di-transforms": "true"
}
```

```
from pyspark.context import SparkContext
from awsgluedi.transforms import *

sc = SparkContext()

input_df = spark.createDataFrame(
    [(5,), (0,), (-1,), (2,), (None,)],
    ["source_column"],
)

try:
    df_output = math_functions.IsEven.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column",
        target_column="target_column",
        value=None,
        true_string="Even",
        false_string="Not even",
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

範例工作階段使 AWS CLI

```
aws glue create-session --default-arguments "--enable-glue-di-transforms=true"
```

DI 轉換：

- FlagDuplicatesInColumn 類
- FormatPhoneNumber 類
- FormatCase 類

- FillWithMode 類別
- FlagDuplicateRows 類
- RemoveDuplicates 類
- MonthName 類
- IsEven 類別
- CryptographicHash 類別
- 解密類
- 加密類
- IntToIp 類
- IpToInt 類

Maven：將插件與星火應用程序捆綁在一起

您可以通過在 Maven 中添加插件依賴關係，同時在本地開發 Spark 應用程序`pom.xml`時，將轉換依賴關係與 Spark 應用程序和 Spark 發行版（3.3 版）捆綁在一起。

```
<repositories>
   ...
    <repository>
        <id>aws-glue-etl-artifacts</id>
        <url>https://aws-glue-etl-artifacts.s3.amazonaws.com/release/ </url>
    </repository>
</repositories>
...
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>AWSGlueTransforms</artifactId>
    <version>4.0.0</version>
</dependency>
```

您也可以直接從 AWS Glue Maven 工件下載二進製文件，並將它們包含在 Spark 應用程序中，如下所示。

```
#!/bin/bash
sudo wget -v https://aws-glue-etl-artifacts.s3.amazonaws.com/release/com/amazonaws/
AWSGlueTransforms/4.0.0/AWSGlueTransforms-4.0.0.jar -P /usr/lib/spark/jars/
```

在 Scala 中進行 AWS Glue ETL 指令碼程式設計

您可以在 GitHub 網站上的 AWS Glue 範例儲存庫中，找到適用於 AWS Glue 的 Scala 程式碼範例與公用程式。

AWS Glue 支援 PySpark Scala 方言的延伸模組，適用於編寫擷取、轉換和載入 (ETL) 任務的指令碼。以下部分說明如何使用 ETL 指令碼的 AWS Glue Scala 程式庫和 AWS Glue API，並提供該程式庫的參考文件。

內容

使用 Scala 以程式設計 AWS Glue ETL 指令碼

您可以使用 AWS Glue 主控台來自動產生 Scala 擷取、轉換和載入 (ETL) 程式並視需要修改，再將其指派到任務。或者，您也可以從頭開始撰寫自己的程式。如需更多資訊，請參閱 設定 Spark 工作的工作屬性 AWS Glue。AWS Glue 會在執行相關的任務前在伺服器編譯 Scala 程式。

為了確保您的程式編譯無誤且如預期般執行，您必須在任務中執行該程式前，在 REPL (Read-Eval-Print Loop) 或 Jupyter 筆記本的開發端點上將其載入並測試。由於編譯處理會在伺服器上進行，您將無法詳細查看在其中發生的任何問題。

在開發端點上使用 Jupyter 筆記本測試 Scala ETL 程式

若要在 AWS Glue 開發端點上測試 Scala 程式，請如 新增開發端點 所述來設定開發端點。

接著，將其連線至在本機電腦或 Amazon EC2 筆記本伺服器遠端執行中的 Jupyter 筆記本。若要安裝本機版本的 Jupyter 筆記本，請遵循 教學課程：JupyterLab 中的 Jupyter 筆記本 中的說明進行。

在筆記本上執行 Scala 程式碼與執行 PySpark 程式碼之間的唯一差別，是您應使用以下項目以在筆記本上開始每一段落：

```
%spark
```

這可防止筆記本伺服器將 Spark 解譯器的 PySpark 類別設為預設。

在 Scala REPL 測試 Scala ETL 程式

您可以使用 AWS Glue Scala REPL 在開發端點上測試 Scala 程式。請遵循教學課程：使用 SageMaker 筆記本中的指示，但在 SSH-to-REPL 命令的結尾，將 `-t gluepyspark` 取代為 `-t glue-spark-shell`。這會呼叫 AWS Glue Scala REPL。

若要在完成時關閉 REPL，輸入 `sys.exit`。

Scala 指令碼範例 - 串流 ETL

Example

下列指令碼範例會連線到 Amazon Kinesis Data Streams，使用來自 Data Catalog 的結構描述剖析資料串流，將串流聯結至 Amazon S3 上的靜態資料集，然後將聯結的結果以 Parquet 格式輸出至 Amazon S3。

```
// This script connects to an Amazon Kinesis stream, uses a schema from the data
 catalog to parse the stream,
// joins the stream to a static dataset on Amazon S3, and outputs the joined results to
 Amazon S3 in parquet format.
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import java.util.Calendar
import org.apache.spark.SparkContext
import org.apache.spark.sql.Dataset
import org.apache.spark.sql.Row
import org.apache.spark.sql.SaveMode
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.from_json
import org.apache.spark.sql.streaming.Trigger
import scala.collection.JavaConverters._

object streamJoiner {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
```

```
    val glueContext: GlueContext = new GlueContext(spark)
    val sparkSession: SparkSession = glueContext.getSparkSession
    import sparkSession.implicits._
    // @params: [JOB_NAME]
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val staticData = sparkSession.read          // read() returns type DataFrameReader
      .format("csv")
      .option("header", "true")
      .load("s3://awsexamplebucket-streaming-demo2/inputs/productsStatic.csv")  //
load() returns a DataFrame

    val datasource0 = sparkSession.readStream   // readstream() returns type
DataStreamReader
      .format("kinesis")
      .option("streamName", "stream-join-demo")
      .option("endpointUrl", "https://kinesis.us-east-1.amazonaws.com")
      .option("startingPosition", "TRIM_HORIZON")
      .load                                     // load() returns a DataFrame

    val selectfields1 = datasource0.select(from_json($"data".cast("string"),
glueContext.getCatalogSchemaAsSparkSchema("stream-demos", "stream-join-demo2")) as
"data").select("data.*")

    val datasink2 = selectfields1.writeStream.foreachBatch { (dataFrame: Dataset[Row],
batchId: Long) => {   //foreachBatch() returns type DataStreamWriter
      val joined = dataFrame.join(staticData, "product_id")
      val year: Int = Calendar.getInstance().get(Calendar.YEAR)
      val month :Int = Calendar.getInstance().get(Calendar.MONTH) + 1
      val day: Int = Calendar.getInstance().get(Calendar.DATE)
      val hour: Int = Calendar.getInstance().get(Calendar.HOUR_OF_DAY)

      if (dataFrame.count() > 0) {
        joined.write                            // joined.write returns type
DataFrameWriter
          .mode(SaveMode.Append)
          .format("parquet")
          .option("quote", " ")
          .save("s3://awsexamplebucket-streaming-demo2/output/" + "/year=" +
"%04d".format(year) + "/month=" + "%02d".format(month) + "/day=" + "%02d".format(day)
+ "/hour=" + "%02d".format(hour) + "/")
      }
    }
```

```
    }  // end foreachBatch()
      .trigger(Trigger.ProcessingTime("100 seconds"))
      .option("checkpointLocation", "s3://awsexamplebucket-streaming-demo2/
checkpoint/")
      .start().awaitTermination()              // start() returns type StreamingQuery
    Job.commit()
  }
}
```

AWS Glue Scala 程式庫中的 API

AWS Glue 支援 PySpark Scala 方言的延伸模組，適用於編寫擷取、轉換和載入 (ETL) 任務的指令碼。以下幾節會說明 AWS Glue Scala 程式庫中的 API。

com.amazonaws.services.glue

在 Scala 程式庫中 com.amazonaws.services.glueAWS Glue 套件包含下列 API：

- ChoiceOption
- DataSink
- DataSource 特徵
- DynamicFrame
- DynamicRecord
- GlueContext
- MappingSpec
- ResolveSpec

com.amazonaws.services.glue.ml

在 AWS Glue Scala 程式庫中 com.amazonaws.services.glue.ml 套件包含下列 API：

- FillMissingValues
- FindIncrementalMatches
- FindMatches

com.amazonaws.services.glue.dq

AWS Glue Scala 程式庫中的 com.amazonaws.services.glue.dq 套件包含下列 API：

- EvaluateDataQuality

com.amazonaws.services.glue.types

在 Scala 程式庫中 com.amazonaws.services.glue.typesAWS Glue 套件包含下列 API：

- ArrayNode
- BinaryNode
- BooleanNode
- ByteNode
- DateNode
- DecimalNode
- DoubleNode
- DynamicNode
- FloatNode
- IntegerNode
- LongNode
- MapLikeNode
- MapNode
- NullNode
- ObjectNode
- ScalarNode
- ShortNode
- StringNode
- TimestampNode

com.amazonaws.services.glue.util

在 Scala 程式庫中 com.amazonaws.services.glue.utilAWS Glue 套件包含下列 API：

- GlueArgParser
- 任務

AWS Glue Scala ChoiceOption API

主題

Package: com.amazonaws.services.glue

ChoiceOption 特徵

```
trait ChoiceOption extends Serializable
```

ChoiceOption 物件

ChoiceOption

```
object ChoiceOption
```

用來解析所有 ChoiceType 節點 (DynamicFrame 中) 適用選擇的一般策略。

- `val CAST`
- `val MAKE_COLS`
- `val MAKE_STRUCT`
- `val MATCH_CATALOG`
- `val PROJECT`

Def apply

```
def apply(choice: String): ChoiceOption
```

案例類別 ChoiceOptionWithResolver

```
case class ChoiceOptionWithResolver(name: String, choiceResolver: ChoiceResolver)
 extends ChoiceOption {}
```

案例類別 MatchCatalogSchemaChoiceOption

```
case class MatchCatalogSchemaChoiceOption() extends ChoiceOption {}
```

Abstract DataSink 類別

主題

- Def writeDynamicFrame
- Def pyWriteDynamicFrame
- Def writeDataFrame
- Def pyWriteDataFrame
- Def setCatalogInfo
- Def supportsFormat
- Def setFormat
- Def withFormat
- Def setAccumulableSize
- Def getOutputErrorRecordsAccumulable
- Def errorsAsDynamicFrame
- DataSink 物件

Package: com.amazonaws.services.glue

```
abstract class DataSink
```

對 DataSource 的寫入類比。DataSink 會封裝 DynamicFrame 可寫入的目的地和格式。

Def writeDynamicFrame

```
def writeDynamicFrame( frame : DynamicFrame,
                       callSite : CallSite = CallSite("Not provided", "")
                     ) : DynamicFrame
```

Def pyWriteDynamicFrame

```
def pyWriteDynamicFrame( frame : DynamicFrame,
                         site : String = "Not provided",
                         info : String = "" )
```

Def writeDataFrame

```
def writeDataFrame(frame: DataFrame,
                   glueContext: GlueContext,
                   callSite: CallSite = CallSite("Not provided", "")
                   ): DataFrame
```

Def pyWriteDataFrame

```
def pyWriteDataFrame(frame: DataFrame,
                     glueContext: GlueContext,
                     site: String = "Not provided",
                     info: String = ""
                     ): DataFrame
```

Def setCatalogInfo

```
def setCatalogInfo(catalogDatabase: String,
                   catalogTableName : String,
                   catalogId : String = "")
```

Def supportsFormat

```
def supportsFormat( format : String ) : Boolean
```

Def setFormat

```
def setFormat( format : String,
               options : JsonOptions
             ) : Unit
```

Def withFormat

```
def withFormat( format : String,
                options : JsonOptions = JsonOptions.empty
              ) : DataSink
```

Def setAccumulableSize

```
def setAccumulableSize( size : Int ) : Unit
```

Def getOutputErrorRecordsAccumulable

```
def getOutputErrorRecordsAccumulable : Accumulable[List[OutputError], OutputError]
```

Def errorsAsDynamicFrame

```
def errorsAsDynamicFrame : DynamicFrame
```

DataSink 物件

```
object DataSink
```

Def recordMetrics

```
def recordMetrics( frame : DynamicFrame,
                   ctxt : String
                 ) : DynamicFrame
```

AWS Glue Scala DataSource 特徵

Package: com.amazonaws.services.glue

用於產生 DynamicFrame 的高階界面。

```
trait DataSource {

  def getDynamicFrame : DynamicFrame
```

```
  def getDynamicFrame( minPartitions : Int,
                       targetPartitions : Int
                     ) : DynamicFrame
  def getDataFrame : DataFrame

  /** @param num: the number of records for sampling.
    * @param options: optional parameters to control sampling behavior. Current
 available parameter for Amazon S3 sources in options:
    *  1. maxSamplePartitions: the maximum number of partitions the sampling will
 read.
    *  2. maxSampleFilesPerPartition: the maximum number of files the sampling will
 read in one partition.
    */
  def getSampleDynamicFrame(num:Int, options: JsonOptions = JsonOptions.empty):
 DynamicFrame

  def glueContext : GlueContext

  def setFormat( format : String,
                 options : String
               ) : Unit

  def setFormat( format : String,
                 options : JsonOptions
               ) : Unit

  def supportsFormat( format : String ) : Boolean

  def withFormat( format : String,
                  options : JsonOptions = JsonOptions.empty
                ) : DataSource
}
```

AWS Glue Scala DynamicFrame API

Package: com.amazonaws.services.glue

內容

- Def count
- Def dropField
- Def dropFields
- Def dropNulls
- 高清 errorsAsDynamic框架
- Def filter
- Def getName
- 防禦 getNumPartitions
- 高清 getSchemaIf計算
- 防禦 isSchemaComputed
- 防禦 javaToPython
- Def join
- Def map
- 防禦 mergeDynamicFrames
- Def printSchema
- Def recomputeSchema
- Def relationalize
- Def renameField
- Def repartition
- Def resolveChoice
- Def schema
- Def selectField
- Def selectFields
- Def show
- 定義簡化
- Def spigot
- Def splitFields
- Def splitRows
- 防禦 stageErrorsCount
- Def toDF

- Def unbox
- Def unnest
- Def unnestDDBJson
- 防禦 withFrameSchema
- Def withName
- 防禦 withTransformationContext

- DynamicFrame 物件
  - Def apply
  - Def emptyDynamicFrame
  - Def fromPythonRDD
  - Def ignoreErrors
  - Def inlineErrors
  - Def newFrameWithErrors

AWS Glue斯卡拉 DynamicFrame级

Package: com.amazonaws.services.glue

```
class DynamicFrame extends Serializable with Logging  (
           val glueContext : GlueContext,
           _records : RDD[DynamicRecord],
           val name : String = s"",
           val transformationContext : String = DynamicFrame.UNDEFINED,
           callSite : CallSite = CallSite("Not provided", ""),
           stageThreshold : Long = 0,
           totalThreshold : Long = 0,
           prevErrors : => Long = 0,
           errorExpr : => Unit = {} )
```

DynamicFrame 是自我描述 DynamicRecord 物件的分散式集合。

DynamicFrame 旨在為 ETL (擷取、轉換和載入) 操作提供靈活的資料模型。它們不需要結構描述即可建立，並可用於讀取和轉換內含雜亂或不一致值和類型的資料。您可以為需要結構描述的操作隨需運算結構描述。

DynamicFrame 提供各種轉換以進行資料洗滌和 ETL。它們還支援與 SparkSQL 之間的轉換，DataFrames 以便與現有的程式碼以及 DataFrames 提供的許多分析作業整合。

以下參數在許多 AWS Glue 轉換之間共用以建構 DynamicFrame：

- transformationContext – 此 DynamicFrame 的識別碼。transformationContext 做為在執行之間持續存在之任務書籤狀態的金鑰使用。
- callSite – 提供錯誤報告的內容資訊。從 Python 呼叫時，會自動設定這些值。
- stageThreshold – 此 DynamicFrame 運算在擲回例外狀況之前允許的最大錯誤記錄數，不包含於先前 DynamicFrame 中存在的記錄。
- totalThreshold – 在擲回例外狀況之前，最大的錯誤記錄總計 (包括之前框架的數量)。

Val errorsCount

```
val errorsCount
```

此 DynamicFrame 中的錯誤記錄數量。這包括之前操作的錯誤。

Def applyMapping

```
def applyMapping( mappings : Seq[Product4[String, String, String, String]],
                  caseSensitive : Boolean = true,
                  transformationContext : String = "",
                  callSite : CallSite = CallSite("Not provided", ""),
                  stageThreshold : Long = 0,
                  totalThreshold : Long = 0
                ) : DynamicFrame
```

- mappings – 用來建構新 DynamicFrame 的映射序列。
- caseSensitive – 是否將來源欄位視為區分大小寫。將此設定設為 false 可能有助於整合區分大小寫的存放區，例如 AWS Glue Data Catalog。

依據映射序列選取、投影及投射欄位。

每個映射皆由來源欄位和類型以及目標欄位和類型所組成。映射可能會指定為四元組 (source_path、source_type、 target_path、target_type) 或包含相同資訊的 MappingSpec 物件。

映射除了可用來進行簡單的投影與投射，還可以用來將欄位巢狀化或解除巢狀化 (藉由使用「.」(句點) 分隔路徑元件來達成)。

例如，假設您有內含結構描述如下的 DynamicFrame。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- address: struct
  |    |-- state: string
  |    |-- zip: int
}}}
```

您可以進行以下呼叫來將 state 和 zip 欄位解除巢狀化。

```
{{{
  df.applyMapping(
   Seq(("name", "string", "name", "string"),
       ("age", "int", "age", "int"),
       ("address.state", "string", "state", "string"),
       ("address.zip", "int", "zip", "int")))
}}}
```

產生的結構描述如下。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- state: string
  |-- zip: int
}}}
```

您也可以使用 applyMapping 來將欄位重新巢狀化。例如，以下會反轉之前的轉換並在目標中建立名為 address 的結構。

```
{{{
  df.applyMapping(
   Seq(("name", "string", "name", "string"),
       ("age", "int", "age", "int"),
       ("state", "string", "address.state", "string"),
       ("zip", "int", "address.zip", "int")))
}}}
```

可使用反引號 (``) 來括住包含「.」(句點) 字元的欄位名稱。

> **Note**
>
> 您目前無法使用 applyMapping 方法來映射於陣列下的巢狀欄位。

防禦 assertErrorThreshold

```
def assertErrorThreshold : Unit
```

強制運算與驗證錯誤記錄數低於 stageThreshold 與 totalThreshold 的動作。如果任一條件失敗，將會擲出例外狀況。

Def count

```
lazy
def count
```

傳回此 DynamicFrame 中的元素數量。

Def dropField

```
def dropField( path : String,
               transformationContext : String = "",
               callSite : CallSite = CallSite("Not provided", ""),
               stageThreshold : Long = 0,
               totalThreshold : Long = 0
             ) : DynamicFrame
```

傳回已移除指定欄位的新 DynamicFrame。

Def dropFields

```
def dropFields( fieldNames : Seq[String],   // The column names to drop.
                transformationContext : String = "",
                callSite : CallSite = CallSite("Not provided", ""),
                stageThreshold : Long = 0,
                totalThreshold : Long = 0
              ) : DynamicFrame
```

傳回已移除指定欄位的新 DynamicFrame。

您可以使用這個方法來刪除巢狀欄位 (包括陣列中的巢狀欄位)，但不能丟棄特定陣列元素。

Def dropNulls

```
def dropNulls( transformationContext : String = "",
               callSite : CallSite = CallSite("Not provided", ""),
               stageThreshold : Long = 0,
               totalThreshold : Long = 0 )
```

傳回新的 DynamicFrame 並移除所有 null 欄位。

> **Note**
>
> 這只會移除 NullType 類型的欄位。其他欄位中的個別 null 值不會被移除或修改。

高清 errorsAsDynamic框架

```
def errorsAsDynamicFrame
```

傳回包含此 DynamicFrame 錯誤記錄的新 DynamicFrame。

Def filter

```
def filter( f : DynamicRecord => Boolean,
            errorMsg : String = "",
            transformationContext : String = "",
            callSite : CallSite = CallSite("Not provided"),
            stageThreshold : Long = 0,
            totalThreshold : Long = 0
          ) : DynamicFrame
```

建構新的 DynamicFrame，其中僅包含函數「f」傳回 true 的那些記錄。篩選條件函數「f」不應使輸入記錄產生變化。

Def getName

```
def getName : String
```

傳回此 DynamicFrame 的名稱。

防禦 getNumPartitions

```
def getNumPartitions
```

傳回此 DynamicFrame 中的分割區數量。

高清 getSchemaIf計算

```
def getSchemaIfComputed : Option[Schema]
```

如果結構描述已經計算，即傳回結構描述。如果結構描述尚未計算，則不掃描資料。

防禦 isSchemaComputed

```
def isSchemaComputed : Boolean
```

如果此 DynamicFrame 的結構描述已經計算，即傳回 true，否則傳回 false。如果此方法傳回 false，則呼叫 schema 方法需要另一個結構描述來在此 DynamicFrame 中傳遞記錄。

防禦 javaToPython

```
def javaToPython : JavaRDD[Array[Byte]]
```

Def join

```
def join( keys1 : Seq[String],
          keys2 : Seq[String],
          frame2 : DynamicFrame,
          transformationContext : String = "",
          callSite : CallSite = CallSite("Not provided", ""),
          stageThreshold : Long = 0,
          totalThreshold : Long = 0
        ) : DynamicFrame
```

- keys1 – 此 DynamicFrame 中要用於聯結的欄位。
- keys2 – frame2 中要用於聯結的欄位。長度必須與 keys1 相同。
- frame2 – 要據以加入的 DynamicFrame。

傳回使用指定金鑰以 frame2 執行對等聯結的結果。

Def map

```
def map( f : DynamicRecord => DynamicRecord,
         errorMsg : String = "",
         transformationContext : String = "",
         callSite : CallSite = CallSite("Not provided", ""),
         stageThreshold : Long = 0,
         totalThreshold : Long = 0
       ) : DynamicFrame
```

傳回藉由將指定函數「f」套用至此 DynamicFrame 中各個記錄而建構的新 DynamicFrame。

此方法會在套用指定的函數之前複製每個記錄，因此可以安全地改變記錄。如果映射函數在指定的記錄擲出例外狀況，會將該記錄標示為錯誤，而會將堆疊追蹤儲存為錯誤記錄中的欄位。

防禦 mergeDynamicFrames

```
def mergeDynamicFrames( stageDynamicFrame: DynamicFrame,  primaryKeys: Seq[String],
 transformationContext: String = "",
                         options: JsonOptions = JsonOptions.empty, callSite: CallSite =
 CallSite("Not provided"),
                         stageThreshold: Long = 0, totalThreshold: Long = 0):
 DynamicFrame
```

- stageDynamicFrame – 要合併的暫存 DynamicFrame。
- primaryKeys – 要從來源和暫存 DynamicFrame 比對記錄的主索引鍵欄位清單。
- transformationContext – 用來擷取目前轉換之中繼資料的唯一字串 (選用)。
- options – JSON 名稱值組的字串，可提供此轉換的額外資料。
- callSite – 用於提供錯誤報告的內容資訊。
- stageThreshold – A Long。在給定轉換中的錯誤數量，其處理需要輸出錯誤。
- totalThreshold – A Long。在此轉換之前 (包括在此轉換中) 的錯誤總數，其處理需要輸出錯誤。

根據指定的主索引鍵來合併此 DynamicFrame 與暫存 DynamicFrame 以識別記錄。重複的記錄 (具有相同主索引鍵的記錄) 不會被刪除重複資料。如果暫存影格中沒有相符的記錄，則會保留來源中的所

有記錄 (包括重複項)。如果暫存影格具有相符的記錄，則暫存影格中的記錄會覆寫 AWS Glue 中來源的記錄。

在下列情況下，傳回的 DynamicFrame 包含記錄 A：

1. 如果 A 同時存在於來源影格和暫存影格，則會傳回暫存影格中的 A。
2. 如果 A 位於來源資料表中而 A.primaryKeys 不在 stagingDynamicFrame 中 (這表示 A 未在暫存資料表中更新)。

來源影格和暫存影格不需要具有相同的結構描述。

Example

```
val mergedFrame: DynamicFrame = srcFrame.mergeDynamicFrames(stageFrame, Seq("id1",
 "id2"))
```

Def printSchema

```
def printSchema : Unit
```

以人類可讀取的格式，將此 DynamicFrame 的結構描述列印至 stdout。

Def recomputeSchema

```
def recomputeSchema : Schema
```

強制結構描述重新計算。這需要掃描資料，但如果目前的結構描述中有一些欄位不存在於資料中，則可能會「限鎖」結構描述。

傳回重新計算的結構描述。

Def relationalize

```
def relationalize( rootTableName : String,
                   stagingPath : String,
                   options : JsonOptions = JsonOptions.empty,
                   transformationContext : String = "",
                   callSite : CallSite = CallSite("Not provided"),
                   stageThreshold : Long = 0,
                   totalThreshold : Long = 0
                 ) : Seq[DynamicFrame]
```

- rootTableName – 在輸出中用於基本 DynamicFrame 的名稱。藉由旋轉陣列所建立的 DynamicFrame 會以此做為字首。
- stagingPath – Amazon Simple Storage Service (Amazon S3) 路徑，用來寫入中繼資料。
- options – 關聯化選項和組態。目前未使用。

將所有巢狀結構平面化並將陣列旋轉為單獨的資料表。

您可以使用此操作來準備深度巢狀資料，以將該資料擷取至關聯式資料庫。巢狀結構以相同於 Unnest 轉換的方式平面化。此外，系統會將陣列旋轉為單獨的資料表，每個陣列元素都將成為資料列。例如，假設您有含以下資料的 DynamicFrame。

```
{"name": "Nancy", "age": 47, "friends": ["Fred", "Lakshmi"]}
{"name": "Stephanie", "age": 28, "friends": ["Yao", "Phil", "Alvin"]}
{"name": "Nathan", "age": 54, "friends": ["Nicolai", "Karen"]}
```

執行下列程式碼。

```
{{{
  df.relationalize("people", "s3:/my_bucket/my_path", JsonOptions.empty)
}}}
```

這會產生兩個資料表。第一個資料表名為「people」，並包含下列項目。

```
{{{
  {"name": "Nancy", "age": 47, "friends": 1}
  {"name": "Stephanie", "age": 28, "friends": 2}
  {"name": "Nathan", "age": 54, "friends": 3)
}}}
```

在此，friends 陣列已替換為自動產生的聯結索引鍵。建立名為 people.friends 的個別資料表，內含以下內容。

```
{{{
  {"id": 1, "index": 0, "val": "Fred"}
  {"id": 1, "index": 1, "val": "Lakshmi"}
  {"id": 2, "index": 0, "val": "Yao"}
  {"id": 2, "index": 1, "val": "Phil"}
  {"id": 2, "index": 2, "val": "Alvin"}
  {"id": 3, "index": 0, "val": "Nicolai"}
```

```
  {"id": 3, "index": 1, "val": "Karen"}
}}}
```

在此資料表中，「id」是一種聯結索引鍵，可識別陣列元素來自哪些記錄，「index」會參照原始陣列中的位置，而「val」則是實際的陣列項目。

relationalize 方法會傳回藉由將此程序遞迴套用至所有陣列而建立的一系列 DynamicFrame。

> **Note**
>
> AWS Glue 程式庫會為新表格自動產生聯結索引鍵。為了確保聯結索引鍵在任務執行中是唯一的，您必須啟用任務書籤。

Def renameField

```
def renameField( oldName : String,
                 newName : String,
                 transformationContext : String = "",
                 callSite : CallSite = CallSite("Not provided", ""),
                 stageThreshold : Long = 0,
                 totalThreshold : Long = 0
               ) : DynamicFrame
```

- oldName – 欄位的原始名稱。
- newName – 欄位的新名稱。

傳回已重新命名指定欄位的新 DynamicFrame。

您可以使用這個方法來重新命名巢狀欄位。例如，以下程式碼會將地址結構中的 state 重新命名為 state_code。

```
{{{
  df.renameField("address.state", "address.state_code")
}}}
```

Def repartition

```
def repartition( numPartitions : Int,
```

```
                transformationContext : String = "",
                callSite : CallSite = CallSite("Not provided", ""),
                stageThreshold : Long = 0,
                totalThreshold : Long = 0
              ) : DynamicFrame
```

傳回包含 numPartitions 分割區的新 DynamicFrame。

Def resolveChoice

```
def resolveChoice( specs : Seq[Product2[String, String]] = Seq.empty[ResolveSpec],
                   choiceOption : Option[ChoiceOption] = None,
                   database : Option[String] = None,
                   tableName : Option[String] = None,
                   transformationContext : String = "",
                   callSite : CallSite = CallSite("Not provided", ""),
                   stageThreshold : Long = 0,
                   totalThreshold : Long = 0
                 ) : DynamicFrame
```

- choiceOption – 套用到所有未在規格序列中列出之 ChoiceType 欄位的動作。
- database – 搭配 match_catalog 動作使用的 Data Catalog 資料庫。
- tableName – 搭配 match_catalog 動作使用的 Data Catalog 資料表。

使用更為特定的類型取代一或多個 ChoiceType 以傳回新 DynamicFrame。

有兩種方式可以使用 resolveChoice。第一種是指定一系列的特定的欄以及解析它們的方式。這些是指定為由 (欄位、動作) 配對所組成的元組。

可行的動作如下：

- cast:type – 嘗試將所有值投射至指定類型。
- make_cols – 將每個不同的類型轉換為具有 columnName_type 名稱的欄位。
- make_struct – 將欄位轉換為每個不同類型皆有金鑰的結構。
- project:type – 僅保留指定類型的值。

resolveChoice 的其他模式可為所有 ChoiceType 指定單一解析度。您可以在 ChoiceType 的完整清單在執行之前是未知的情況下使用此模式。除了以上列出的動作，此模式也支援下列動作：

- match_catalogChoiceType – 嘗試將每個 投射至指定目錄資料表中的對應類型。

範例：

藉由投射至 int 以解析 user.id 欄位，並且讓 address 欄位僅保留結構。

```
{{{
  df.resolveChoice(specs = Seq(("user.id", "cast:int"), ("address", "project:struct")))
}}}
```

藉由將每個選擇轉換單獨的欄位以解析所有 ChoiceType。

```
{{{
  df.resolveChoice(choiceOption = Some(ChoiceOption("make_cols")))
}}}
```

藉由投射至指定目錄資料表中的類型以解析所有 ChoiceType。

```
{{{
  df.resolveChoice(choiceOption = Some(ChoiceOption("match_catalog")),
                   database = Some("my_database"),
                   tableName = Some("my_table"))
}}}
```

Def schema

```
def schema : Schema
```

傳回此 DynamicFrame 的結構描述。

傳回的結構描述會保證包含於此 DynamicFrame 中之記錄存在的每個欄位。但在少數情況下，它也可能包含額外的欄位。您可以使用 Unnest 方法，依據此 DynamicFrame 中的記錄來「限縮」結構描述。

Def selectField

```
def selectField( fieldName : String,
                 transformationContext : String = "",
                 callSite : CallSite = CallSite("Not provided", ""),
                 stageThreshold : Long = 0,
```

```
                totalThreshold : Long = 0
              ) : DynamicFrame
```

以 DynamicFrame 傳回單一欄位。

Def selectFields

```
def selectFields( paths : Seq[String],
                  transformationContext : String = "",
                  callSite : CallSite = CallSite("Not provided", ""),
                  stageThreshold : Long = 0,
                  totalThreshold : Long = 0
                ) : DynamicFrame
```

- paths – 要選取的欄位名稱序列。

傳回包含指定欄位的新 DynamicFrame。

> **Note**
>
> 您只能使用 selectFields 方法來選取最上層欄位。您可以使用 applyMapping 方法來選取巢狀欄位。

Def show

```
def show( numRows : Int = 20 ) : Unit
```

- numRows – 要列印的資料列數。

以 JSON 格式列印此 DynamicFrame 的資料列。

定義簡化

使用 DynamoDB 匯出連接器匯出的 AWS Glue DynamoDB 會產生特定巢狀結構的 JSON 檔案。如需詳細資訊，請參閱資料物件。 simplifyDDBJson簡化此類型資料中 DynamicFrame 的巢狀資料行，並傳回新的簡化 DynamicFrame。如果 List 類型中包含多種類型或 Map 類型，則 List 中的元素將不會簡化。此方法僅支援使用 DynamoDB 資料匯出 JSON 格式的資料。請考慮unnest對其他類型的資料執行類似的變更。

```
def simplifyDDBJson() : DynamicFrame
```

此方法不採用任何參數。

範例輸入

請考慮由 DynamoDB 匯出產生的下列結構描述：

```
root
|-- Item: struct
|    |-- parentMap: struct
|    |    |-- M: struct
|    |    |    |-- childMap: struct
|    |    |    |    |-- M: struct
|    |    |    |    |    |-- appName: struct
|    |    |    |    |    |    |-- S: string
|    |    |    |    |    |-- packageName: struct
|    |    |    |    |    |    |-- S: string
|    |    |    |    |    |-- updatedAt: struct
|    |    |    |    |    |    |-- N: string
|    |-- strings: struct
|    |    |-- SS: array
|    |    |    |-- element: string
|    |-- numbers: struct
|    |    |-- NS: array
|    |    |    |-- element: string
|    |-- binaries: struct
|    |    |-- BS: array
|    |    |    |-- element: string
|    |-- isDDBJson: struct
|    |    |-- BOOL: boolean
|    |-- nullValue: struct
|    |    |-- NULL: boolean
```

範例程式碼

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContextimport scala.collection.JavaConverters._
```

```
object GlueApp {

  def main(sysArgs: Array[String]): Unit = {
    val glueContext = new GlueContext(SparkContext.getOrCreate())
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType = "dynamodb",
      options = JsonOptions(Map(
        "dynamodb.export" -> "ddb",
        "dynamodb.tableArn" -> "ddbTableARN",
        "dynamodb.s3.bucket" -> "exportBucketLocation",
        "dynamodb.s3.prefix" -> "exportBucketPrefix",
        "dynamodb.s3.bucketOwner" -> "exportBucketAccountID",
      ))
    ).getDynamicFrame()

    val simplified = dynamicFrame.simplifyDDBJson()
    simplified.printSchema()

    Job.commit()
  }

}
```

範例輸出

simplifyDDBJson 轉換將此簡化為：

```
root
|-- parentMap: struct
|    |-- childMap: struct
|    |    |-- appName: string
|    |    |-- packageName: string
|    |    |-- updatedAt: string
|-- strings: array
|    |-- element: string
|-- numbers: array
|    |-- element: string
|-- binaries: array
|    |-- element: string
|-- isDDBJson: boolean
```

```
|-- nullValue: null
```

Def spigot

```
def spigot( path : String,
            options : JsonOptions = new JsonOptions("{}"),
            transformationContext : String = "",
            callSite : CallSite = CallSite("Not provided"),
            stageThreshold : Long = 0,
            totalThreshold : Long = 0
          ) : DynamicFrame
```

傳遞轉換以傳回相同的記錄，但副作用是寫出部分記錄。

- path – 以 s3://bucket//path 格式將輸出寫入至 Amazon S3 中的路徑。
- options – 描述取樣行為的選用 JsonOptions 映射。

傳回包含與此相同記錄的 DynamicFrame。

在預設情況下，寫入 100 任意記錄到 path 指定的位置。您可以使用 options 對應來自訂此行為。有效索引鍵包括下列：

- topk – 指定寫出的記錄總數。預設為 100。
- prob – 指定包含個別記錄的機率 (以小數表示)。預設值為 1。

例如，以下呼叫取樣資料集的方式是以 20% 的可能性選取每個記錄，並在已寫入 200 個記錄之後停止。

```
{{{
  df.spigot("s3://my_bucket/my_path", JsonOptions(Map("topk" -&gt; 200, "prob" -&gt;
 0.2)))
}}}
```

Def splitFields

```
def splitFields( paths : Seq[String],
                 transformationContext : String = "",
                 callSite : CallSite = CallSite("Not provided", ""),
                 stageThreshold : Long = 0,
                 totalThreshold : Long = 0
```

```
            ) : Seq[DynamicFrame]
```

- paths – 要包含在第一個 DynamicFrame 中的路徑。

傳回兩個 DynamicFrame 的序列。第一個 DynamicFrame 包含指定的路徑，第二個包含所有其他欄。

範例

此範例取自 AWS Glue Data Catalog 資料legislators庫中資料庫中的persons表格 DynamicFrame 建立的，並 DynamicFrame 將其分割為兩個，其中指定的欄位會進入第一個欄位 DynamicFrame ，剩餘欄位會進入第二 DynamicFrame個欄位。然後，該示例從結果 DynamicFrame 中選擇第一個。

```
val InputFrame = glueContext.getCatalogSource(database="legislators",
 tableName="persons",
transformationContext="InputFrame").getDynamicFrame()

val SplitField_collection = InputFrame.splitFields(paths=Seq("family_name", "name",
 "links.note",
"links.url", "gender", "image", "identifiers.scheme", "identifiers.identifier",
 "other_names.lang",
"other_names.note", "other_names.name"), transformationContext="SplitField_collection")

val ResultFrame = SplitField_collection(0)
```

Def splitRows

```
def splitRows( paths : Seq[String],
               values : Seq[Any],
               operators : Seq[String],
               transformationContext : String,
               callSite : CallSite,
               stageThreshold : Long,
               totalThreshold : Long
             ) : Seq[DynamicFrame]
```

根據比較欄位與常數的述詞來分割列。

- paths – 用於比較的欄位。
- values – 用於比較的常數值。

- operators – 用於比較的運算子。

傳回兩個 DynamicFrame 的序列。第一個包含述詞為 true 的列，第二個包含述詞為 false 的列。

使用三個序列指定述詞：「paths」包含 (可能為巢狀) 欄位名稱、「values」包含要比較的常數值，以及「operators」包含用於比較的運算子。這三個序列的長度必須相同：第 n 個運算子會用於比較第 n 個欄位與第 n 個值。

每個運算子都必須是「!=」、「=」、「<=」、「<」、「>=」或「>」其中之一。

舉例來說，以下呼叫會分割 DynamicFrame，因此第一個輸出框架會包含來自美國超過 65 人的記錄，第二個會包含所有其他記錄。

```
{{{
  df.splitRows(Seq("age", "address.country"), Seq(65, "USA"), Seq("&gt;=", "="))
}}}
```

防禦 stageErrorsCount

```
def stageErrorsCount
```

傳回運算此 DynamicFrame 時建立的錯誤記錄的數量。這會排除之前傳遞至此 DynamicFrame 做為輸入之操作的錯誤。

Def toDF

```
def toDF( specs : Seq[ResolveSpec] = Seq.empty[ResolveSpec] ) : DataFrame
```

以相同的結構描述和記錄，將此 DynamicFrame 轉換為 Apache Spark SQL DataFrame。

> **Note**
>
> 由於 DataFrame 不支援 ChoiceType，因此這個方法會自動將 ChoiceType 欄轉換成 StructType。如需有關解析選擇的詳細資訊和選項，請參閱resolveChoice。

Def unbox

```
def unbox( path : String,
           format : String,
```

```
            optionString : String = "{}",
            transformationContext : String = "",
            callSite : CallSite = CallSite("Not provided"),
            stageThreshold : Long = 0,
            totalThreshold : Long = 0
          ) : DynamicFrame
```

- path – 要剖析的欄。必須為字串或二進位。
- format – 用於剖析的格式。
- optionString – 傳送格式的選項，例如 CSV 分隔符號。

根據指定的格式，剖析嵌入字串或二進位欄位。剖析的欄位是具有原始資料欄名稱結構的巢狀欄位。

例如，假設您有 CSV 檔案與內嵌 JSON 欄位。

```
name, age, address
Sally, 36, {"state": "NE", "city": "Omaha"}
...
```

完成初始剖析後，您會取得具有下列結構描述的 DynamicFrame。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- address: string
}}}
```

您可以呼叫地址欄位上的 unbox 以剖析特定元件。

```
{{{
  df.unbox("address", "json")
}}}
```

如此將提供我們具有下列結構描述的 DynamicFrame。

```
{{{
  root
  |-- name: string
  |-- age: int
```

```
  |-- address: struct
  |    |-- state: string
  |    |-- city: string
}}}
```

Def unnest

```
def unnest( transformationContext : String = "",
            callSite : CallSite = CallSite("Not Provided"),
            stageThreshold : Long = 0,
            totalThreshold : Long = 0
          ) : DynamicFrame
```

傳回其所有巢狀結構皆已平面化的新 DynamicFrame。使用「.」(句點) 字元建構名稱。

例如，假設您有內含結構描述如下的 DynamicFrame。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- address: struct
  |    |-- state: string
  |    |-- city: string
}}}
```

以下呼叫將會解巢狀地址結構。

```
{{{
  df.unnest()
}}}
```

產生的結構描述如下。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- address.state: string
  |-- address.city: string
}}}
```

此方法也會解巢狀陣列中的巢狀結構。但因為歷史因素，這類欄位的名稱會附加封閉陣列和「.val」的名稱。

Def unnestDDBJson

```
unnestDDBJson(transformationContext : String = "",
            callSite : CallSite = CallSite("Not Provided"),
            stageThreshold : Long = 0,
            totalThreshold : Long = 0): DynamicFrame
```

解除專屬於 DynamoDB JSON 結構中 DynamicFrame 內的巢狀欄的巢狀化，並傳回新的解巢狀 DynamicFrame。結構類型陣列的欄將不是解巢狀狀態。請注意，這是一種特定類型的解除巢狀化轉換，其行為與常規 unnest 轉換不同，且資料必須已經位於 DynamoDB JSON 結構中。如需詳細資訊，請參閱 DynamoDB JSON。

例如，讀取 DynamoDB JSON 結構的匯出結構描述與以下類似：

```
root
|-- Item: struct
|    |-- ColA: struct
|    |    |-- S: string
|    |-- ColB: struct
|    |    |-- S: string
|    |-- ColC: struct
|    |    |-- N: string
|    |-- ColD: struct
|    |    |-- L: array
|    |    |    |-- element: null
```

unnestDDBJson() 轉換會將此轉換為：

```
root
|-- ColA: string
|-- ColB: string
|-- ColC: string
|-- ColD: array
|    |-- element: null
```

下列程式碼範例演示如何使用 AWS Glue DynamoDB 匯出連接器，呼叫 DynamoDB JSON 解巢狀，並列印分割區數量：

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._


object GlueApp {

  def main(sysArgs: Array[String]): Unit = {
    val glueContext = new GlueContext(SparkContext.getOrCreate())
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType = "dynamodb",
      options = JsonOptions(Map(
        "dynamodb.export" -> "ddb",
        "dynamodb.tableArn" -> "<test_source>",
        "dynamodb.s3.bucket" -> "<bucket name>",
        "dynamodb.s3.prefix" -> "<bucket prefix>",
        "dynamodb.s3.bucketOwner" -> "<account_id of bucket>",
      ))
    ).getDynamicFrame()

    val unnested = dynamicFrame.unnestDDBJson()
    print(unnested.getNumPartitions())

    Job.commit()
  }

}
```

防禦 withFrameSchema

```
def withFrameSchema( getSchema : () => Schema ) : DynamicFrame
```

- getSchema – 傳回結構描述以供使用的函數。指定為零參數函數以延遲可能昂貴的運算。

將此 DynamicFrame 的結構描述設定為指定的值。這主要用於內部以避免昂貴的結構描述重新計算。傳入的結構描述必須包含存在於資料中的所有資料欄位。

Def withName

```
def withName( name : String ) : DynamicFrame
```

- name – 要使用的新名稱。

傳回此具有新名稱的 DynamicFrame 的副本。

防禦 withTransformationContext

```
def withTransformationContext( ctx : String ) : DynamicFrame
```

傳回此具有指定轉換內容的 DynamicFrame 的副本。

DynamicFrame 物件

Package: com.amazonaws.services.glue

```
object DynamicFrame
```

Def apply

```
def apply( df : DataFrame,
           glueContext : GlueContext
         ) : DynamicFrame
```

Def emptyDynamicFrame

```
def emptyDynamicFrame( glueContext : GlueContext ) : DynamicFrame
```

Def fromPythonRDD

```
def fromPythonRDD( rdd : JavaRDD[Array[Byte]],
```

```
                    glueContext : GlueContext
                  ) : DynamicFrame
```

Def ignoreErrors

```
def ignoreErrors( fn : DynamicRecord => DynamicRecord ) : DynamicRecord
```

Def inlineErrors

```
def inlineErrors( msg : String,
                  callSite : CallSite
                ) : (DynamicRecord => DynamicRecord)
```

Def newFrameWithErrors

```
def newFrameWithErrors( prevFrame : DynamicFrame,
                        rdd : RDD[DynamicRecord],
                        name : String = "",
                        transformationContext : String = "",
                        callSite : CallSite,
                        stageThreshold : Long,
                        totalThreshold : Long
                      ) : DynamicFrame
```

AWS Glue Scala DynamicRecord 類別

主題

- Def readFields
- Def clone
- Def schema
- Def getRoot
- Def toJson
- Def getFieldNode
- Def getField
- Def hashCode
- Def equals
- DynamicRecord 物件
- RecordTraverser 特徵

Package: com.amazonaws.services.glue

```
class DynamicRecord extends Serializable with Writable with Cloneable
```

DynamicRecord 本身是描述資料結構，其代表要處理的資料集內的資料列。其為自我描述，這表示您可以透過檢查記錄本身來取得由 DynamicRecord 所呈現的資料列資料結構。DynamicRecord 與 Apache Spark 中的 Row 類似。

Def addField

```
def addField( path : String,
              dynamicNode : DynamicNode
            ) : Unit
```

將 DynamicNode 新增到指定的路徑。

- path — 要新增的欄位路徑。
- dynamicNode — 要在指定路徑中新增的 DynamicNode。

Def dropField

```
def dropField(path: String, underRename: Boolean = false): Option[DynamicNode]
```

如果指定的路徑中沒有任何陣列，則從指定的路徑放入 DynamicNode 並傳回放入的節點。

- path — 要放入的欄位路徑。
- underRenamedropField — 若在更名轉換期間呼叫 則為 true，否則為 false (預設為 false)。

傳回 scala.Option Option (DynamicNode)。

Def setError

```
def setError( error : Error )
```

將此記錄設定為錯誤記錄，如 error 參數所指定。

傳回 DynamicRecord。

Def isError

```
def isError
```

檢查此記錄是否是錯誤記錄。

Def getError

```
def getError
```

如果記錄是錯誤記錄，則取得 Error。如果此記錄是錯誤記錄，傳回 scala.Some Some (錯誤)，否則傳回 scala.None。

Def clearError

```
def clearError
```

將 Error 設定為 scala.None.None。

Def write

```
override def write( out : DataOutput ) : Unit
```

Def readFields

```
override def readFields( in : DataInput ) : Unit
```

Def clone

```
override def clone : DynamicRecord
```

將此記錄複製到新 DynamicRecord 並將其傳回。

Def schema

```
def schema
```

檢查記錄以取得 Schema。

Def getRoot

```
def getRoot : ObjectNode
```

取得記錄的根 ObjectNode。

Def toJson

```
def toJson : String
```

取得記錄的 JSON 字串。

Def getFieldNode

```
def getFieldNode( path : String ) : Option[DynamicNode]
```

在指定的 path 取得欄位值做為 DynamicNode 的選項。

如果欄位存在，傳回 scala.Some Some (DynamicNode)，否則傳回 scala.None.None。

Def getField

```
def getField( path : String ) : Option[Any]
```

在指定的 path 取得欄位值做為 DynamicNode 的選項。

傳回 scala.Some Some (值)。

Def hashCode

```
override def hashCode : Int
```

Def equals

```
override def equals( other : Any )
```

DynamicRecord 物件

```
object DynamicRecord
```

Def apply

```
def apply( row : Row,
           schema : SparkStructType )
```

套用將 Apache Spark SQL Row 轉換為 DynamicRecord 的方法。

- row — Spark SQL Row。
- schema — 該資料列的 Schema。

傳回 DynamicRecord。

RecordTraverser 特徵

```
trait RecordTraverser {
  def nullValue(): Unit
  def byteValue(value: Byte): Unit
  def binaryValue(value: Array[Byte]): Unit
  def booleanValue(value: Boolean): Unit
  def shortValue(value: Short) : Unit
```

```
  def intValue(value: Int) : Unit
  def longValue(value: Long) : Unit
  def floatValue(value: Float): Unit
  def doubleValue(value: Double): Unit
  def decimalValue(value: BigDecimal): Unit
  def stringValue(value: String): Unit
  def dateValue(value: Date): Unit
  def timestampValue(value: Timestamp): Unit
  def objectStart(length: Int): Unit
  def objectKey(key: String): Unit
  def objectEnd(): Unit
  def mapStart(length: Int): Unit
  def mapKey(key: String): Unit
  def mapEnd(): Unit
  def arrayStart(length: Int): Unit
  def arrayEnd(): Unit
}
```

AWS Glue斯卡 GlueContext 拉

Package: com.amazonaws.services.glue

```
class GlueContext extends SQLContext(sc) (
           @transient val sc : SparkContext,
           val defaultSourcePartitioner : PartitioningStrategy )
```

GlueContext 是讀取和寫入 DynamicFrame 至 Amazon Simple Storage Service (Amazon S3)、AWS Glue Data Catalog 、JDBC 等的進入點。此類別提供公用程式函數來建立 DataSource 特徵 和 DataSink 物件，從而用於讀取和寫入 DynamicFrame。

如果從來源建立的分割區數低於分割區的閾值下限 (預設 10)，您也可以使用 GlueContext 來設定在 DynamicFrame 中的分割區目標數 (預設 20)。

高清 addIngestionTime列

```
def addIngestionTimeColumns(
         df : DataFrame,
         timeGranularity : String = "") : dataFrame
```

附加擷取時間欄 (如 ingest_year、ingest_month、ingest_day、ingest_hour、ingest_minute) 到輸入

DataFrame。當您指定以 Amazon S3 為目標的 Data Catalog 資料表時，此函數會在 AWS Glue 產生的指令碼中自動產生。此函數會自動使用輸出資料表上的擷取時間欄來更新分割區。這可讓輸出資料在擷取時間自動分割，而不需要輸入資料中的明確擷取時間欄。

- dataFrame – 要將擷取時間欄附加到的 dataFrame。
- timeGranularity – 時間欄的精密程度。有效值為 "day"、"hour" 和 "minute"。例如：如果 "hour" 被傳遞給函數，原始 dataFrame 會附加上 "ingest_year"、"ingest_month"、"ingest_day" 和 "ingest_hour" 時間欄。

傳回附加時間粒度欄後的資料框架。

範例：

```
glueContext.addIngestionTimeColumns(dataFrame, "hour")
```

高清 createDataFrame FromOptions

```
def createDataFrameFromOptions( connectionType : String,
                          connectionOptions : JsonOptions,
                          transformationContext : String = "",
                          format : String = null,
                          formatOptions : JsonOptions = JsonOptions.empty
                        ) : DataSource
```

傳回使用指定的連線和格式建立的 DataFrame。此功能僅與 AWS Glue 串流來源搭配使用。

- connectionType – 串流連線類型。有效值包括 kinesis 與 kafka。
- connectionOptions – 連線選項，這些選項對於 Kinesis 和 Kafka 而言是不同的。您可以在 AWS Glue for Spark 中 ETL 的連線類型和選項 中找到每個串流資料來源的所有連線選項清單。請注意串流連線選項的下列不同處：
  - Kinesis 串流來源需要 streamARN、startingPosition、inferSchema 以及 classification。
  - Kafka 串流來源需要 connectionName、topicName、startingOffsets、inferSchema 以及 classification。
- transformationContext – 要使用的轉換細節 (選用)。
- format – 格式化規格 (選用)。這是用於 Amazon S3 或支援多種格式的 AWS Glue 連線。如需有關支援格式的資訊，請參閱 AWS Glue for Spark 中的輸入與輸出的資料格式選項

- formatOptions – 指定格式的格式選項。如需支援格式選項的詳細資訊，請參閱 資料格式選項。

Amazon Kinesis 串流來源範例：

```
val data_frame_datasource0 =
glueContext.createDataFrameFromOptions(transformationContext = "datasource0",
 connectionType = "kinesis",
connectionOptions = JsonOptions("""{"streamName": "example_stream", "startingPosition":
 "TRIM_HORIZON", "inferSchema": "true", "classification": "json"}}""""))
```

Kafka 串流來源範例：

```
val data_frame_datasource0 =
glueContext.createDataFrameFromOptions(transformationContext = "datasource0",
 connectionType = "kafka",
connectionOptions = JsonOptions("""{"connectionName": "example_connection",
 "topicName": "example_topic", "startingPosition": "earliest", "inferSchema": "false",
 "classification": "json", "schema":"`column1` STRING, `column2` STRING"}""""))
```

forEachBatch

forEachBatch(frame, batch_function, options)

將傳入的 batch_function 套用至從串流來源讀取的每個微批次。

- frame— 包 DataFrame 含當前微批次。
- batch_function – 將套用至每個微批次的函數。
- options – 索引鍵/值配對的集合，其中包含如何處理微批次的相關資訊。下列選項是必要的：
  - windowSize – 處理每個批次的時間量。
  - checkpointLocation - 串流 ETL 任務的檢查點儲存位置。
  - batchMaxRetries – 如果失敗，可重試批次的次數上限。預設值為 3。此選項僅在 Glue 2.0 及以上版本上才可設定。

範例：

```
glueContext.forEachBatch(data_frame_datasource0, (dataFrame: Dataset[Row], batchId:
 Long) =>
   {
```

```
      if (dataFrame.count() > 0)
        {
          val datasource0 = DynamicFrame(glueContext.addIngestionTimeColumns(dataFrame,
 "hour"), glueContext)
          // @type: DataSink
          // @args: [database = "tempdb", table_name = "fromoptionsoutput",
 stream_batch_time = "100 seconds",
          //      stream_checkpoint_location = "s3://from-options-testing-eu-central-1/
fromOptionsOutput/checkpoint/",
          //      transformation_ctx = "datasink1"]
          // @return: datasink1
          // @inputs: [frame = datasource0]
          val options_datasink1 = JsonOptions(
             Map("partitionKeys" -> Seq("ingest_year", "ingest_month","ingest_day",
 "ingest_hour"),
             "enableUpdateCatalog" -> true))
          val datasink1 = glueContext.getCatalogSink(
             database = "tempdb",
             tableName = "fromoptionsoutput",
             redshiftTmpDir = "",
             transformationContext = "datasink1",
             additionalOptions = options_datasink1).writeDynamicFrame(datasource0)
        }
   }, JsonOptions("""{"windowSize" : "100 seconds",
         "checkpointLocation" : "s3://from-options-testing-eu-central-1/
fromOptionsOutput/checkpoint/"}"""))
```

高清 getCatalogSink

```
def getCatalogSink( database : String,
        tableName : String,
        redshiftTmpDir : String = "",
        transformationContext : String = ""
        additionalOptions: JsonOptions = JsonOptions.empty,
        catalogId: String = null
) : DataSink
```

建立 DataSink，以便寫入 Data Catalog 中定義之資料表中指定的位置。

- `database` — Data Catalog 中的資料庫名稱。
- `tableName` — Data Catalog 中的資料表名稱。
- `redshiftTmpDir` — 要與特定資料目的地搭配使用的臨時暫存目錄。設定為 預設為空值。

- transformationContext — 與由任務書籤使用之目的地關聯的轉換內容。設定為 預設為空值。
- additionalOptions – 提供給 AWS Glue 的額外選項。
- catalogId — 要存取之 Data Catalog 的目錄 ID (帳戶 ID)。為 null 時，會使用發起人的預設帳戶 ID。

傳回 DataSink。

高清 getCatalogSource

```
def getCatalogSource( database : String,
                      tableName : String,
                      redshiftTmpDir : String = "",
                      transformationContext : String = ""
                      pushDownPredicate : String = " "
                      additionalOptions: JsonOptions = JsonOptions.empty,
                      catalogId: String = null
                    ) : DataSource
```

建立 DataSource 特徵，以便從 Data Catalog 中的資料表定義中讀取資料。

- database — Data Catalog 中的資料庫名稱。
- tableName — Data Catalog 中的資料表名稱。
- redshiftTmpDir — 要與特定資料目的地搭配使用的臨時暫存目錄。設定為 預設為空值。
- transformationContext — 與由任務書籤使用之目的地關聯的轉換內容。設定為 預設為空值。
- pushDownPredicate – 篩選分割區，而無需列出和讀取資料集中的所有檔案。如需詳細資訊，請參閱 使用 pushdown 述詞預先篩選。
- additionalOptions – 選擇性的名稱/值對的集合。可能的選項包括 AWS Glue for Spark 中 ETL 的連線類型和選項 中列出的項目，除了 endpointUrl、streamName、bootstrap.servers、security.protocol、topicName、classi 以及delimiter。另一個支援的選項是 catalogPartitionPredicate：

  catalogPartitionPredicate — 您可以傳遞目錄表達式以根據索引欄進行篩選。這會將篩選下推至伺服器端。如需詳細資訊，請參閱 AWS Glue 分割區索引。注意 push_down_predicate 和 catalogPartitionPredicate 使用不同的語法。前者使用 Spark SQL 標準語法，後者使用 JSQL 剖析器。
- catalogId — 要存取之 Data Catalog 的目錄 ID (帳戶 ID)。為 null 時，會使用發起人的預設帳戶 ID。

傳回 DataSource。

串流來源範例

```
val data_frame_datasource0 = glueContext.getCatalogSource(
    database = "tempdb",
    tableName = "test-stream-input",
    redshiftTmpDir = "",
    transformationContext = "datasource0",
    additionalOptions = JsonOptions("""{
        "startingPosition": "TRIM_HORIZON", "inferSchema": "false"}""")
    ).getDataFrame()
```

def getJDBCSink

```
def getJDBCSink( catalogConnection : String,
                 options : JsonOptions,
                 redshiftTmpDir : String = "",
                 transformationContext : String = "",
                 catalogId: String = null
               ) : DataSink
```

建立 DataSink，以便寫入 Data Catalog 中 Connection 物件所指定的 JDBC 資料庫。此 Connection 物件擁有用來對 JDBC 目的地連線的資訊 (包括 URL、使用者名稱、密碼、VPC、子網路和安全群組)。

- catalogConnection — Data Catalog 中的連線名稱，其中包含要做為寫入目的地之 JDBC URL。
- options — JSON 名稱值組的字串，可提供寫入 JDBC 資料存放區所需的其他資訊。其中包含：
  - dbtable (必要) — JDBC 資料表的名稱。若是支援資料庫內結構描述的 JDBC 資料存放區，請指定 schema.table-name。如果未提供結構描述，則會使用預設的 "public" 結構描述。以下範例說明 options 參數，它會指向資料庫 test_db 中名為 test 的結構描述和名為 test_table 的資料表。

    ```
    options = JsonOptions("""{"dbtable": "test.test_table", "database": "test_db"}""")
    ```

  - database (必要) — JDBC 資料庫的名稱。
  - 任何其他選項都會直接傳遞至 SparkSQL JDBC 寫入器。如需詳細資訊，請參閱 Spark 的 Redshift 資料來源。

- redshiftTmpDir — 要與特定資料目的地搭配使用的臨時暫存目錄。設定為 預設為空值。
- transformationContext — 與由任務書籤使用之目的地關聯的轉換內容。設定為 預設為空值。
- catalogId — 要存取之 Data Catalog 的目錄 ID (帳戶 ID)。為 null 時，會使用發起人的預設帳戶 ID。

範例程式碼：

```
getJDBCSink(catalogConnection = "my-connection-name", options =
 JsonOptions("""{"dbtable": "my-jdbc-table", "database": "my-jdbc-db"}"""),
 redshiftTmpDir = "", transformationContext = "datasink4")
```

傳回 DataSink。

def getSink

```
def getSink( connectionType : String,
             connectionOptions : JsonOptions,
             transformationContext : String = ""
           ) : DataSink
```

建立將資料DataSink寫入像 Amazon Simple Storage Service (Amazon S3)、JDBC 或 AWS Glue 資料型錄或 Apache 卡夫卡或 Amazon Kinesis 資料串流這樣的目的地。

- connectionType — 連線的類型。請參閱the section called “連線參數”。
- connectionOptions — JSON 名稱值組的字串，可提供與資料目的地建立連線的額外資料。請參閱the section called “連線參數”。
- transformationContext — 與由任務書籤使用之目的地關聯的轉換內容。設定為 預設為空值。

傳回 DataSink。

高清 getSinkWith格式

```
def getSinkWithFormat( connectionType : String,
                       options : JsonOptions,
                       transformationContext : String = "",
                       format : String = null,
                       formatOptions : JsonOptions = JsonOptions.empty
```

```
                    ) : DataSink
```

建立一DataSink個將資料寫入 Amazon S3、JDBC 或資料型錄等目的地，或者阿帕奇卡夫卡或 Amazon Kinesis 資料串流。還可以設定要寫出至目標的資料的格式。

- connectionType — 連線的類型。請參閱the section called “連線參數”。
- options — JSON 名稱值組的字串，可提供與資料目的地建立連線的額外資料。請參閱the section called “連線參數”。
- transformationContext — 與由任務書籤使用之目的地關聯的轉換內容。設定為 預設為空值。
- format — 要從目的地寫出的資料格式。
- formatOptions — JSON 名稱值組的字串，會提供在目的地格式化資料的其他選項。請參閱資料格式選項。

傳回 DataSink。

def getSource

```
def getSource( connectionType : String,
               connectionOptions : JsonOptions,
               transformationContext : String = ""
               pushDownPredicate
             ) : DataSource
```

建立DataSource 特徵可從 Amazon S3、JDBC 或 AWS Glue 資料型錄等來源讀取資料的資料。也支援 Kafka 和 Kinesis 串流資料來源。

- connectionType — 資料來源的類型。請參閱the section called “連線參數”。
- connectionOptions — JSON 名稱值組的字串，可提供與資料來源建立連線的額外資料。如需詳細資訊，請參閱 the section called “連線參數”。

  Kinesis 串流來源需要下列連線選項：streamARN、startingPosition、inferSchema 及 classification。

  Kafka 串流來源需要以下連線選項：connectionName、topicName、startingOffsets、inferSchema 及 classification。
- transformationContext — 與由任務書籤使用之目的地關聯的轉換內容。設定為 預設為空值。

- pushDownPredicate — 分割區欄上的述詞。

傳回 DataSource。

Amazon Kinesis 串流來源範例：

```
val kinesisOptions = jsonOptions()
data_frame_datasource0 = glueContext.getSource("kinesis",
 kinesisOptions).getDataFrame()

private def jsonOptions(): JsonOptions = {
    new JsonOptions(
      s"""{"streamARN": "arn:aws:kinesis:eu-central-1:123456789012:stream/
fromOptionsStream",
         |"startingPosition": "TRIM_HORIZON",
         |"inferSchema": "true",
         |"classification": "json"}""".stripMargin)
}
```

Kafka 串流來源範例：

```
val kafkaOptions = jsonOptions()
val data_frame_datasource0 = glueContext.getSource("kafka",
 kafkaOptions).getDataFrame()

private def jsonOptions(): JsonOptions = {
    new JsonOptions(
      s"""{"connectionName": "ConfluentKafka",
         |"topicName": "kafka-auth-topic",
         |"startingOffsets": "earliest",
         |"inferSchema": "true",
         |"classification": "json"}""".stripMargin)
 }
```

高清 getSourceWith格式

```
def getSourceWithFormat( connectionType : String,
                         options : JsonOptions,
                         transformationContext : String = "",
                         format : String = null,
                         formatOptions : JsonOptions = JsonOptions.empty
```

```
                    ) : DataSource
```

建立DataSource 特徵可從 Amazon S3、JDBC 或 AWS Glue 資料型錄等來源讀取資料的資料，並設定儲存在來源中的資料格式。

- connectionType – 資料來源的類型。請參閱the section called “連線參數”。
- options – JSON 名稱值組的字串，可提供與資料來源建立連線的額外資料。請參閱the section called “連線參數”。
- transformationContext – 與由任務書籤使用之目的地關聯的轉換內容。設定為 預設為空值。
- format – 來源中所存放資料的格式。當 connectionType 為「s3」時，您也可以指定 format。可以是「avro」、「csv」、「grokLog」、「ion」、「json」、「xml」、「parquet」或「orc」其中之一。
- formatOptions – JSON 名稱值組的字串，會提供在來源剖析資料的其他選項。請參閱資料格式選項。

傳回 DataSource。

範例

DynamicFrame 從 Amazon S3 上以逗號分隔值 (CSV) 檔案為檔案的資料來源建立：

```
val datasource0 = glueContext.getSourceWithFormat(
    connectionType="s3",
    options =JsonOptions(s"""{"paths": [ "s3://csv/nycflights.csv"]}"""),
    transformationContext = "datasource0",
    format = "csv",
    formatOptions=JsonOptions(s"""{"withHeader":"true","separator": ","}""")
    ).getDynamicFrame()
```

DynamicFrame 從使用 JDBC 連接的 PostgreSQL 的數據源創建一個：

```
val datasource0 = glueContext.getSourceWithFormat(
    connectionType="postgresql",
    options =JsonOptions(s"""{
      "url":"jdbc:postgresql://databasePostgres-1.rds.amazonaws.com:5432/testdb",
      "dbtable": "public.company",
      "redshiftTmpDir":"",
      "user":"username",
      "password":"password123"
```

```
    }"""),
    transformationContext = "datasource0").getDynamicFrame()
```

使用 JDBC 連接 DynamicFrame 從一個 MySQL 的數據源創建一個：

```
val datasource0 = glueContext.getSourceWithFormat(
   connectionType="mysql",
   options =JsonOptions(s"""{
     "url":"jdbc:mysql://databaseMysql-1.rds.amazonaws.com:3306/testdb",
     "dbtable": "athenatest_nycflights13_csv",
     "redshiftTmpDir":"",
     "user":"username",
     "password":"password123"
   }"""),
   transformationContext = "datasource0").getDynamicFrame()
```

高清 getSparkSession

```
def getSparkSession : SparkSession
```

取得與此 GlueContext 相關聯的 SparkSession 物件。使用此 SparkSession 物件可註冊表格和 UDF，以便與DataFrame建立來源 DynamicFrames搭配使用。

傳回 SparkSession。

def startTransaction

```
def startTransaction(readOnly: Boolean):String
```

開始新交易。內部呼叫 Lake Formation startTransaction API。

- readOnly – (布林值) 指出此交易應該是唯讀，還是讀取和寫入。使用唯讀交易 ID 進行的寫入將被拒絕。唯讀交易不需要遞交。

傳回交易 ID。

def commitTransaction

```
def commitTransaction(transactionId: String, waitForCommit: Boolean): Boolean
```

嘗試遞交指定的交易。commitTransaction 可能會在交易完成遞交之前返回。內部呼叫 Lake Formation commitTransaction API。

- transactionId – (字串) 要遞交的交易。
- waitForCommit – (布林值) 決定 commitTransaction 是否立即傳回。預設值為 true。如為 False，commitTransaction 輪詢並等待，直到交易完成遞交。使用指數退避時，等待時間長度限制為 1 分鐘，最多可嘗試 6 次重試。

傳回一個布林值，指示遞交是否完成。

def cancelTransaction

```
def cancelTransaction(transactionId: String): Unit
```

嘗試取消指定的交易。內部調用 Lake Formation CancelTransactionAPI。

- transactionId – (字串) 要取消的交易。

如果交易先前已遞交，傳回 TransactionCommittedException 例外狀況。

def 此

```
def this( sc : SparkContext,
          minPartitions : Int,
          targetPartitions : Int )
```

使用指定的 SparkContext、最小分割區和分割區目標來建立 GlueContext 物件。

- sc — SparkContext。
- minPartitions — 分割區最小數。
- targetPartitions — 分割區目標數。

傳回 GlueContext。

def 此

```
def this( sc : SparkContext )
```

透過提供的 SparkContext 建立 GlueContext 物件。將分割區的最小值設為 10，目標分割區設為 20。

- sc — SparkContext。

傳回 GlueContext。

def 此

```
def this( sparkContext : JavaSparkContext )
```

透過提供的 JavaSparkContext 建立 GlueContext 物件。將分割區的最小值設為 10，目標分割區設為 20。

- sparkContext — JavaSparkContext。

傳回 GlueContext。

MappingSpec

Package: com.amazonaws.services.glue

MappingSpec 案例類別

```
case class MappingSpec( sourcePath: SchemaPath,
                        sourceType: DataType,
                        targetPath: SchemaPath,
                        targetType: DataTyp
                       ) extends Product4[String, String, String, String] {
  override def _1: String = sourcePath.toString
  override def _2: String = ExtendedTypeName.fromDataType(sourceType)
  override def _3: String = targetPath.toString
  override def _4: String = ExtendedTypeName.fromDataType(targetType)
}
```

- sourcePath — 來源欄位的 SchemaPath。
- sourceType — 來源欄位的 DataType。
- targetPath — 目標欄位的 SchemaPath。
- targetType — 目標欄位的 DataType。

MappingSpec 指定從來源路徑和來源資料類型到目標路徑和目標資料類型的映射。在來源框架中來源路徑的值會顯示在目標路徑的目標框架中。來源資料類型會轉換到目標資料類型。

它從 Product4 擴展，讓您可以處理任何 Product4 (在 applyMapping 介面中)。

MappingSpec 物件

```
object MappingSpec
```

MappingSpec 物件具有下列成員：

Val orderingByTarget

```
val orderingByTarget: Ordering[MappingSpec]
```

Def apply

```
def apply( sourcePath : String,
           sourceType : DataType,
           targetPath : String,
           targetType : DataType
         ) : MappingSpec
```

建立 MappingSpec。

- sourcePath — 來源路徑的字串顯示方式。
- sourceType — 來源 DataType。
- targetPath — 目標路徑的字串顯示方式。
- targetType — 目標 DataType。

傳回 MappingSpec。

Def apply

```
def apply( sourcePath : String,
           sourceTypeString : String,
           targetPath : String,
           targetTypeString : String
```

```
        ) : MappingSpec
```

建立 MappingSpec。

- sourcePath — 來源路徑的字串顯示方式。
- sourceType — 來源資料類型的字串顯示方式。
- targetPath — 目標路徑的字串顯示方式。
- targetType — 目標資料類型的字串顯示方式。

傳回 MappingSpec。

Def apply

```
def apply( product : Product4[String, String, String, String] ) : MappingSpec
```

建立 MappingSpec。

- product — 來源路徑、來源資料類型、目標路徑和目標資料類型的 Product4。

傳回 MappingSpec。

AWS Glue Scala ResolveSpec API

主題

- ResolveSpec 物件
- ResolveSpec 案例類別

Package: com.amazonaws.services.glue

ResolveSpec 物件

ResolveSpec

```
object ResolveSpec
```

Def

```
def apply( path : String,
```

```
            action : String
          ) : ResolveSpec
```

建立 ResolveSpec。

- path — 以字串表示、需要解析的選擇欄位。
- action — 解析動作。動作可以是以下其中之一：Project、KeepAsStruct 或 Cast。

傳回 ResolveSpec。

Def

```
def apply( product : Product2[String, String] ) : ResolveSpec
```

建立 ResolveSpec。

- product — 以下項目的 Product2：來源路徑、解析動作。

傳回 ResolveSpec。

ResolveSpec 案例類別

```
case class ResolveSpec extends Product2[String, String]  (
            path : SchemaPath,
            action : String )
```

建立 ResolveSpec。

- path — 需要解析之選擇欄位的 SchemaPath。
- action — 解析動作。動作可以是以下其中之一：Project、KeepAsStruct 或 Cast。

ResolveSpec def 方法

```
def _1 : String
```

```
def _2 : String
```

AWS Glue Scala ArrayNode API

Package: com.amazonaws.services.glue.types

ArrayNode 案例類別

ArrayNode

```
case class ArrayNode extends DynamicNode  (
           value : ArrayBuffer[DynamicNode] )
```

ArrayNode def 方法

```
def add( node : DynamicNode )
```

```
def clone
```

```
def equals( other : Any )
```

```
def get( index : Int ) : Option[DynamicNode]
```

```
def getValue
```

```
def hashCode : Int
```

```
def isEmpty : Boolean
```

```
def nodeType
```

```
def remove( index : Int )
```

```
def this
```

```
def toIterator : Iterator[DynamicNode]
```

```
def toJson : String
```

```
def update( index : Int,
            node : DynamicNode )
```

AWS Glue Scala BinaryNode API

Package: com.amazonaws.services.glue.types

BinaryNode 案例類別

BinaryNode

```
case class BinaryNode extends ScalarNode(value, TypeCode.BINARY)  (
           value : Array[Byte] )
```

BinaryNode val 欄位

- `ordering`

BinaryNode def 方法

```
def clone
```

```
def equals( other : Any )
```

```
def hashCode : Int
```

AWS Glue Scala BooleanNode API

Package: com.amazonaws.services.glue.types

BooleanNode 案例類別

BooleanNode

```
case class BooleanNode extends ScalarNode(value, TypeCode.BOOLEAN)  (
           value : Boolean )
```

BooleanNode val 欄位

- ordering

BooleanNode def 方法

```
def equals( other : Any )
```

AWS Glue Scala ByteNode API

Package: com.amazonaws.services.glue.types

ByteNode 案例類別

ByteNode

```
case class ByteNode extends ScalarNode(value, TypeCode.BYTE)  (
           value : Byte )
```

ByteNode val 欄位

- ordering

ByteNode def 方法

```
def equals( other : Any )
```

AWS Glue Scala DateNode API

Package: com.amazonaws.services.glue.types

DateNode 案例類別

DateNode

```
case class DateNode extends ScalarNode(value, TypeCode.DATE)  (
           value : Date )
```

DateNode val 欄位

- ordering

DateNode def 方法

```
def equals( other : Any )
```

```
def this( value : Int )
```

AWS Glue Scala DecimalNode API

Package: com.amazonaws.services.glue.types

DecimalNode 案例類別

DecimalNode

```
case class DecimalNode extends ScalarNode(value, TypeCode.DECIMAL)  (
           value : BigDecimal )
```

DecimalNode val 欄位

- `ordering`

DecimalNode def 方法

```
def equals( other : Any )
```

```
def this( value : Decimal )
```

AWS Glue Scala DoubleNode API

Package: com.amazonaws.services.glue.types

DoubleNode 案例類別

DoubleNode

```
case class DoubleNode extends ScalarNode(value, TypeCode.DOUBLE)  (
           value : Double )
```

DoubleNode val 欄位

- `ordering`

DoubleNode def 方法

```
def equals( other : Any )
```

AWS Glue Scala DynamicNode API

主題

- DynamicNode 類別
- DynamicNode 物件

Package: com.amazonaws.services.glue.types

DynamicNode 類別

DynamicNode

```
class DynamicNode extends Serializable with Cloneable
```

DynamicNode def 方法

```
def getValue : Any
```

取得純值並繫結至目前的記錄：

```
def nodeType : TypeCode
```

```
def toJson : String
```

偵錯方式：

```
def toRow( schema : Schema,
           options : Map[String, ResolveOption]
         ) : Row
```

```
def typeName : String
```

DynamicNode 物件

DynamicNode

```
object DynamicNode
```

DynamicNode def 方法

```
def quote( field : String,
           useQuotes : Boolean
         ) : String
```

```
def quote( node : DynamicNode,
           useQuotes : Boolean
         ) : String
```

EvaluateDataQuality 類別

AWS Glue 的 AWS Glue Data Quality 目前為預覽版本，並可能會有所變更。

套件：com.amazonaws.services.glue.dq

```
object EvaluateDataQuality
```

Def apply

```
def apply(frame: DynamicFrame,
            ruleset: String,
            publishingOptions: JsonOptions = JsonOptions.empty): DynamicFrame
```

根據 DynamicFrame 評估資料品質規則集，並傳回包含評估結果的新 DynamicFrame。若要進一步了解 AWS Glue Data Quality，請參閱AWS Glue 資料品質。

- frame – 您要評估資料品質的 DynamicFrame。
- ruleset – 字串格式的資料品質定義語言 (DQDL) 規則集。若要進一步了解 DQDL，請參閱 資料品質定義語言 (DQDL) 參考 指南。

- publishingOptions – 指定以下用於發佈評估結果和指標的選項的字典：
  - dataQualityEvaluationContext – 字串，指定 AWS Glue 應在其下發佈 Amazon CloudWatch 指標和資料品質結果的命名空間。彙總的指標會顯示在 CloudWatch 中，而完整的結果則會顯示在 AWS Glue Studio 介面中。
    - 必要：否
    - 預設值：default_context
  - enableDataQualityCloudWatchMetrics – 指定是否應將資料品質評估的結果發佈至 CloudWatch。您可以使用 dataQualityEvaluationContext 選項指定指標的命名空間。
    - 必要：否
    - 預設值：False
  - enableDataQualityResultsPublishing – 指定資料品質結果是否應顯示在 AWS Glue Studio 介面的 Data Quality (資料品質) 索引標籤上。
    - 必要：否
    - 預設值：True
  - resultsS3Prefix – 指定 AWS Glue 可以寫入資料品質評估結果的 Amazon S3 位置。
    - 必要：否
    - 預設值："" (空字串)

範例

下列範例程式碼示範如何在執行 SelectFields 轉換之前評估 DynamicFrame 的資料品質。指令碼會在嘗試轉換之前驗證所有資料品質規則是否通過。

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
import com.amazonaws.services.glue.dq.EvaluateDataQuality

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
```

```
    // @params: [JOB_NAME]
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    // Create DynamicFrame with data
    val Legislators_Area = glueContext.getCatalogSource(database="legislators",
 tableName="areas_json", transformationContext="S3bucket_node1").getDynamicFrame()

    // Define data quality ruleset
    val DQ_Ruleset = """
      Rules = [ColumnExists "id"]
    """

    // Evaluate data quality
    val DQ_Results = EvaluateDataQuality.apply(frame=Legislators_Area,
 ruleset=DQ_Ruleset, publishingOptions=JsonOptions("""{"dataQualityEvaluationContext":
 "Legislators_Area", "enableDataQualityMetrics": "true",
 "enableDataQualityResultsPublishing": "true"}"""))
    assert(DQ_Results.filter(_.getField("Outcome").contains("Failed")).count == 0,
 "Failing DQ rules for Legislators_Area caused the job to fail.")

    // Script generated for node Select Fields
    val SelectFields_Results = Legislators_Area.selectFields(paths=Seq("id", "name"),
 transformationContext="Legislators_Area")

    Job.commit()
  }
}
```

AWS Glue Scala FloatNode API

Package: com.amazonaws.services.glue.types

FloatNode 案例類別

FloatNode

```
case class FloatNode extends ScalarNode(value, TypeCode.FLOAT)  (
           value : Float )
```

FloatNode val 欄位

- `ordering`

FloatNode def 方法

```
def equals( other : Any )
```

FillMissingValues 類別

套件：com.amazonaws.services.glue.ml

```
object FillMissingValues
```

Def apply

```
def apply(frame: DynamicFrame,
          missingValuesColumn: String,
          outputColumn: String = "",
          transformationContext: String = "",
          callSite: CallSite = CallSite("Not provided", ""),
          stageThreshold: Long = 0,
          totalThreshold: Long = 0): DynamicFrame
```

在指定的欄中填入動態框架的缺少值，並在新的欄中傳回具有估計值的新框架。對於沒有缺少值的列，指定欄的值將被複製到新欄。

- `frame` — 在其中填入缺少值的 DynamicFrame。必要。
- `missingValuesColumn` — 包含缺少值的欄 (`null` 值和空字串)。必要。
- `outputColumn` — 新欄的名稱，該欄將包含所有缺少值的列的估計值。選擇性；預設為 `missingValuesColumn` 的值，字尾為 `"_filled"`。
- `transformationContext` — 用於識別狀態資訊的唯一字串 (選用)。
- `callSite` — 用於提供錯誤報告的內容資訊 (選用)。
- `stageThreshold` — 在錯誤輸出之前，轉換作業中可發生錯誤的次數上限 (選用；預設值為零)。
- `totalThreshold` — 在處理錯誤輸出之前，整體作業可發生錯誤的次數上限 (選用；預設值為零)。

傳回具有一個額外欄的新動態框架，其中包含缺少值的列估計和其他列的目前值。

FindMatches 類別

套件：com.amazonaws.services.glue.ml

```
object FindMatches
```

Def apply

```
def apply(frame: DynamicFrame,
            transformId: String,
            transformationContext: String = "",
            callSite: CallSite = CallSite("Not provided", ""),
            stageThreshold: Long = 0,
            totalThreshold: Long = 0,
            enforcedMatches: DynamicFrame = null): DynamicFrame,
   computeMatchConfidenceScores: Boolean
```

在輸入框架中尋找相符項目並傳回一個新框架，其中包含每個相符群組唯一 ID 的新欄。

- `frame` — 要在其中尋找相符項目的 DynamicFrame。必要。
- `transformId` — 與 FindMatches 轉換相關聯的唯一 ID，以套用於輸入影格。必要。
- `transformationContext` — 此 DynamicFrame 的識別碼。`transformationContext` 做為在執行之間持續存在之任務書籤狀態的金鑰使用。選用。
- `callSite` — 用於提供錯誤報告的內容資訊。從 Python 呼叫時，會自動設定這些值。選用。
- `stageThreshold` — 此 DynamicFrame 運算在擲回例外狀況之前允許的最大錯誤記錄數，不包含於先前 DynamicFrame 中存在的記錄。選用。預設為零。
- `totalThreshold` — 在擲回例外狀況之前，最大的錯誤記錄總計 (包括之前框架的數量)。選用。預設為零。
- `enforcedMatches` — 強制相符的框架。選用。預設值為 `null`。
- `computeMatchConfidenceScores` – 布林值，指出是否運算每個相符記錄群組的可信度分數。選用。預設值為 false。

傳回具有指派給每個相符記錄群組之唯一識別碼的新動態框架。

FindIncrementalMatches 類別

套件：com.amazonaws.services.glue.ml

```
object FindIncrementalMatches
```

Def apply

```
apply(existingFrame: DynamicFrame,
            incrementalFrame: DynamicFrame,
            transformId: String,
            transformationContext: String = "",
            callSite: CallSite = CallSite("Not provided", ""),
            stageThreshold: Long = 0,
            totalThreshold: Long = 0,
            enforcedMatches: DynamicFrame = null): DynamicFrame,
   computeMatchConfidenceScores: Boolean
```

在現有和增量框架中尋找相符項目，並傳回一個新框架，其中包含每個相符群組唯一 ID 的新欄。

- existingframe — 已為每個群組指派相符 ID 的現有框架。必要。
- incrementalframe — 用來尋找與現有框架相符的增量框架。必要。
- transformId — 與 FindIncrementalMatches 轉換相關聯的唯一 ID，以套用於輸入影格。必要。
- transformationContext — 此 DynamicFrame 的識別碼。transformationContext 做為在執行之間持續存在之任務書籤狀態的金鑰使用。選用。
- callSite — 用於提供錯誤報告的內容資訊。從 Python 呼叫時，會自動設定這些值。選用。
- stageThreshold — 此 DynamicFrame 運算在擲回例外狀況之前允許的最大錯誤記錄數，不包含於先前 DynamicFrame 中存在的記錄。選用。預設為零。
- totalThreshold — 在擲回例外狀況之前，最大的錯誤記錄總計 (包括之前框架的數量)。選用。預設為零。
- enforcedMatches — 強制相符的框架。選用。預設值為 null。
- computeMatchConfidenceScores – 布林值，指出是否運算每個相符記錄群組的可信度分數。選用。預設值為 false。

傳回具有指派給每個相符記錄群組之唯一識別碼的新動態框架。

AWS Glue Scala IntegerNode API

Package: com.amazonaws.services.glue.types

IntegerNode 案例類別

IntegerNode

```
case class IntegerNode extends ScalarNode(value, TypeCode.INT)  (
           value : Int )
```

IntegerNode val 欄位

- ordering

IntegerNode def 方法

```
def equals( other : Any )
```

AWS Glue Scala LongNode API

Package: com.amazonaws.services.glue.types

LongNode 案例類別

LongNode

```
case class LongNode extends ScalarNode(value, TypeCode.LONG)  (
           value : Long )
```

LongNode val 欄位

- ordering

LongNode def 方法

```
def equals( other : Any )
```

AWS Glue Scala MapLikeNode API

Package: com.amazonaws.services.glue.types

MapLikeNode 類別

MapLikeNode

```
class MapLikeNode extends DynamicNode  (
```

```
            value : mutable.Map[String, DynamicNode] )
```

MapLikeNode def 方法

```
def clear : Unit
```

```
def get( name : String ) : Option[DynamicNode]
```

```
def getValue
```

```
def has( name : String ) : Boolean
```

```
def isEmpty : Boolean
```

```
def put( name : String,
         node : DynamicNode
       ) : Option[DynamicNode]
```

```
def remove( name : String ) : Option[DynamicNode]
```

```
def toIterator : Iterator[(String, DynamicNode)]
```

```
def toJson : String
```

```
def toJson( useQuotes : Boolean ) : String
```

範例：給定此 JSON：

```
{"foo": "bar"}
```

如果 `useQuotes == true`，`toJson` 會產生 `{"foo": "bar"}`。如果 `useQuotes == false`，`toJson` 會產生 `{foo: bar}` @return。

AWS Glue Scala MapNode API

Package: com.amazonaws.services.glue.types

MapNode 案例類別

MapNode

```
case class MapNode extends MapLikeNode(value)  (
           value : mutable.Map[String, DynamicNode] )
```

MapNode def 方法

```
def clone
```

```
def equals( other : Any )
```

```
def hashCode : Int
```

```
def nodeType
```

```
def this
```

AWS Glue Scala NullNode API

主題

- NullNode 類別
- NullNode 案例物件

Package: com.amazonaws.services.glue.types

NullNode 類別

NullNode

```
class NullNode
```

NullNode 案例物件

NullNode

```
case object NullNode extends NullNode
```

AWS Glue Scala ObjectNode API

主題

- ObjectNode 物件
- ObjectNode 案例類別

Package: com.amazonaws.services.glue.types

ObjectNode 物件

ObjectNode

```
object ObjectNode
```

ObjectNode def 方法

```
def apply( frameKeys : Set[String],
           v1 : mutable.Map[String, DynamicNode],
           v2 : mutable.Map[String, DynamicNode],
           resolveWith : String
         ) : ObjectNode
```

ObjectNode 案例類別

ObjectNode

```
case class ObjectNode extends MapLikeNode(value)  (
           val value : mutable.Map[String, DynamicNode] )
```

ObjectNode def 方法

```
def clone
```

```
def equals( other : Any )
```

```
def hashCode : Int
```

```
def nodeType
```

```
def this
```

AWS Glue Scala ScalarNode API

主題

- ScalarNode 類別
- ScalarNode 物件

Package: com.amazonaws.services.glue.types

ScalarNode 類別

ScalarNode

```
class ScalarNode extends DynamicNode  (
           value : Any,
           scalarType : TypeCode )
```

ScalarNode def 方法

```
def compare( other : Any,
             operator : String
           ) : Boolean
```

```
def getValue
```

```
def hashCode : Int
```

```
def nodeType
```

```
def toJson
```

ScalarNode 物件

ScalarNode

```
object ScalarNode
```

ScalarNode def 方法

```
def apply( v : Any ) : DynamicNode
```

```
def compare( tv : Ordered[T],
             other : T,
             operator : String
           ) : Boolean
```

```
def compareAny( v : Any,
                y : Any,
                o : String )
```

```
def withEscapedSpecialCharacters( jsonToEscape : String ) : String
```

AWS Glue Scala ShortNode API

Package: com.amazonaws.services.glue.types

ShortNode 案例類別

ShortNode

```
case class ShortNode extends ScalarNode(value, TypeCode.SHORT)  (
           value : Short )
```

ShortNode val 欄位

- `ordering`

ShortNode def 方法

```
def equals( other : Any )
```

AWS Glue Scala StringNode API

Package: com.amazonaws.services.glue.types

StringNode 案例類別

StringNode

```
case class StringNode extends ScalarNode(value, TypeCode.STRING)  (
           value : String )
```

StringNode val 欄位

- ordering

StringNode def 方法

```
def equals( other : Any )
```

```
def this( value : UTF8String )
```

AWS Glue Scala TimestampNode API

Package: com.amazonaws.services.glue.types

TimestampNode 案例類別

TimestampNode

```
case class TimestampNode extends ScalarNode(value, TypeCode.TIMESTAMP)  (
           value : Timestamp )
```

TimestampNode val 欄位

- ordering

TimestampNode def 方法

```
def equals( other : Any )
```

```
def this( value : Long )
```

AWS Glue Scala GlueArgParser API

Package: com.amazonaws.services.glue.util

GlueArgParser 物件

GlueArgParser

```
object GlueArgParser
```

此與 AWSGlueDataplanePython 套件中的 utils.getResolvedOptions Python 版本完全一致。

GlueArgParser def 方法

```
def getResolvedOptions( args : Array[String],
                        options : Array[String]
                      ) : Map[String, String]
```

```
def initParser( userOptionsSet : mutable.Set[String] ) : ArgumentParser
```

Example 擷取傳遞至任務的引數

若要擷取任務引數，您可使用 getResolvedOptions 方法。請考慮以下範例，其會擷取名為 aws_region 的任務引數。

```
val args = GlueArgParser.getResolvedOptions(sysArgs,
 Seq("JOB_NAME","aws_region").toArray)
Job.init(args("JOB_NAME"), glueContext, args.asJava)
val region = args("aws_region")
println(region)
```

AWS Glue Scala 任務 API

Package: com.amazonaws.services.glue.util

任務物件

任務

```
object Job
```

Job def 方法

```
def commit
```

```
def init( jobName : String,
          glueContext : GlueContext,
          args : java.util.Map[String, String] = Map[String, String]().asJava
        ) : this.type
```

```
def init( jobName : String,
          glueContext : GlueContext,
          endpoint : String,
          args : java.util.Map[String, String]
        ) : this.type
```

```
def isInitialized
```

```
def reset
```

```
def runId
```

AWS Glue for Spark ETL 指令碼程式設計的功能和最佳化

以下各節說明通常會套用於任何語言的 AWS Glue for Spark ETL (擷取、轉換和載入) 程式設計的技術和值。

主題

- AWS Glue for Spark 中 ETL 的連線類型和選項
- AWS Glue for Spark 中的輸入與輸出的資料格式選項
- AWS Glue 資料目錄支援 Spark SQL 任務
- 使用任務書籤

- 在 AWS Glue Studio 外部使用敏感數據檢測
- AWS Glue 視覺化任務 API

AWS Glue for Spark 中 ETL 的連線類型和選項

在 AWS Glue for Spark 中，可採用多種 PySpark 和 Scala 方法及轉換使用 `connectionType` 參數指定連線類型。這些使用 `connectionOptions` 或 `options` 參數指定連線選項。

`connectionType` 參數可以接受如下表所示的值。以下各節將說明每種類型的相關 `connectionOptions` (或 `options`) 參數值。除非另有說明，否則當連線做為來源或接收器使用時，參數即會套用。

如需有關示範設定和使用連線選項的範例程式碼，請參閱每個連線類型的首頁。

connectionType	連線到
dynamodb	Amazon DynamoDB 資料庫
kinesis	Amazon Kinesis Data Streams
s3	Amazon Simple Storage Service (Amazon S3)
documentdb	Amazon DocumentDB (with MongoDB compatibility) 資料庫
opensearch	Amazon OpenSearch Service。
redshift	Amazon Redshift 資料庫
kafka	Kafka 或 Amazon Managed Streaming for Apache Kafka
azurecosmos	Azure Cosmos for NoSQL。
azuresql	Azure SQL。
bigquery	Google BigQuery。
mongodb	MongoDB 資料庫 (包含 MongoDB Atlas)。
sqlserver	Microsoft SQL 伺服器資料庫 (請參閱 JDBC 連線)
mysql	MySQL 資料庫 (請參閱 JDBC 連線)

connectionType	連線到
oracle	Oracle 資料庫 (請參閱 JDBC 連線)
postgresql	PostgreSQL 資料庫 (請參閱 JDBC 連線)
saphana	SAP HANA。
Snowflake	Snowflake 資料湖
teradata	Teradata Vantage。
vertica	Vertica。
custom.*	Spark、Athena 或 JDBC 資料存放區 (請參閱 自訂和 AWS Marketplace connectionType 值
marketplace.*	Spark、Athena 或 JDBC 資料存放區 (請參閱 自訂和 AWS Marketplace connectionType 值)

DynamoDB 連線

可以使用 AWS Glue for Spark 在 AWS Glue 的 DynamoDB 中讀取和寫入資料表。可以使用連接到 AWS Glue 工作的 IAM 許可連線到 DynamoDB。AWS Glue 支援將資料寫入到另一個 AWS 帳戶的 DynamoDB 資料表中。如需更多詳細資訊，請參閱 the section called “跨帳戶跨區域存取 DynamoDB 資料表”。

除了 AWS Glue DynamoDB ETL 連接器之外，還可使用 DynamoDB 匯出連接器從 DynamoDB 中讀取，該連接器可調用 DynamoDB `ExportTableToPointInTime` 請求，並將其存放在您提供的 Amazon S3 位置中，格式為 DynamoDB JSON。AWS Glue 則會透過讀取 Amazon S3 匯出位置中的資料來建立 DynamicFrame 物件。

AWS Glue 1.0 或更新版本中會提供 DynamoDB 寫入器。AWS Glue 2.0 或更新版本中會提供 AWS Glue DynamoDB 匯出連接器。

如需 DynamoDB 的詳細資訊，請參閱 Amazon DynamoDB 文件。

> **Note**
>
> DynamoDB ETL 讀取器不支援篩選條件或下推述詞。

設定 DynamoDB 連線

若要從 AWS Glue 連線至 DynamoDB，請將許可授予給與 AWS Glue 工作相關聯的 IAM 角色，以便與 DynamoDB 互動。如需有關在 DynamoDB 中進行讀取或寫入所需許可的詳細資訊，請參閱 IAM 文件中的 DynamoDB 的動作、資源和條件金鑰。

在下列情況中，可能需要其他組態：

- 使用 DynamoDB 匯出連接器時，將需要設定 IAM，以便工作可以請求 DynamoDB 資料表匯出。此外，需要確定用於匯出的 Amazon S3 儲存貯體，並在 IAM 中提供適當的許可，以便 DynamoDB 可寫入到該儲存貯體，並且 AWS Glue 工作可從中讀取。如需詳細資訊，請參閱請求在 DynamoDB 中匯出資料表。
- 如果 AWS Glue 工作具有特定的 Amazon VPC 連線需求，請使用 NETWORK AWS Glue 連線類型來提供網路選項。由於 DynamoDB 的存取權由 IAM 授權，因此不需要使用 AWS Glue DynamoDB 連線類型。

對 DynamoDB 進行讀取和寫入

下列程式碼範例示範如何讀取（透過 ETL 連接器）及寫入 DynamoDB 資料表。其展示從一個資料表讀取以及寫入另一個資料表。

Python

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame.from_options(
    connection_type="dynamodb",
    connection_options={"dynamodb.input.tableName": test_source,
        "dynamodb.throughput.read.percent": "1.0",
        "dynamodb.splits": "100"
    }
)
```

```
print(dyf.getNumPartitions())

glue_context.write_dynamic_frame_from_options(
    frame=dyf,
    connection_type="dynamodb",
    connection_options={"dynamodb.output.tableName": test_sink,
        "dynamodb.throughput.write.percent": "1.0"
    }
)

job.commit()
```

Scala

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._


object GlueApp {

  def main(sysArgs: Array[String]): Unit = {
    val glueContext = new GlueContext(SparkContext.getOrCreate())
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType = "dynamodb",
      options = JsonOptions(Map(
        "dynamodb.input.tableName" -> test_source,
        "dynamodb.throughput.read.percent" -> "1.0",
        "dynamodb.splits" -> "100"
      ))
    ).getDynamicFrame()

    print(dynamicFrame.getNumPartitions())

    val dynamoDbSink: DynamoDbDataSink =  glueContext.getSinkWithFormat(
      connectionType = "dynamodb",
```

```
      options = JsonOptions(Map(
        "dynamodb.output.tableName" -> test_sink,
        "dynamodb.throughput.write.percent" -> "1.0"
      ))
    ).asInstanceOf[DynamoDbDataSink]

    dynamoDbSink.writeDynamicFrame(dynamicFrame)

    Job.commit()
  }

}
```

使用 DynamoDB 匯出連接器

在 DynamoDB 資料表大小大於 80 GB 時，匯出連接器的效能會比 ETL 連接器更佳。此外，由於匯出請求是在 AWS Glue 任務中 (在 Spark 程序之外) 執行，您可以啟用 AWS Glue 任務的自動擴展，以在匯出請求期間儲存 DPU 使用量。若使用匯出連接器，您也無需為 Spark 執行器平行處理原則或 DynamoDB 輸送量讀取百分比設定分割數。

> **Note**
>
> DynamoDB 對呼叫 `ExportTableToPointInTime` 請求具有特定要求。如需詳細資訊，請參閱請求在 DynamoDB 中匯出資料表。例如，需要在資料表上啟用時間點還原 (PITR) 才能使用此連接器。DynamoDB 連接器也支援 AWS KMS 加密，以便將 DynamoDB 匯出至 Amazon S3。在 AWS Glue 任務設定中提供安全組態，可為 DynamoDB 的匯出進行 AWS KMS 加密。KMS 金鑰必須位於與 Amazon S3 儲存貯體相同的區域中。
> 請注意，DynamoDB 匯出和 Amazon S3 儲存皆會產生額外費用和成本。在任務運作完成後，Amazon S3 中匯出的資料仍會存在，因此您可以重新使用該資料，無需進行其他 DynamoDB 匯出。使用此連接器的要求為資料表啟用時間點復原 (PITR)。
> DynamoDB ETL 連接器或匯出連接器皆不支援要在 DynamoDB 來源中套用的篩選條件或 pushdown 述詞。

以下程式碼範例演示如何讀取 (透過匯出連接器) 及列印分割區數。

Python

```
import sys
```

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame.from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.export": "ddb",
        "dynamodb.tableArn": test_source,
        "dynamodb.s3.bucket": bucket_name,
        "dynamodb.s3.prefix": bucket_prefix,
        "dynamodb.s3.bucketOwner": account_id_of_bucket,
    }
)
print(dyf.getNumPartitions())

job.commit()
```

Scala

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._


object GlueApp {

  def main(sysArgs: Array[String]): Unit = {
    val glueContext = new GlueContext(SparkContext.getOrCreate())
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val dynamicFrame = glueContext.getSourceWithFormat(
```

```
      connectionType = "dynamodb",
      options = JsonOptions(Map(
        "dynamodb.export" -> "ddb",
        "dynamodb.tableArn" -> test_source,
        "dynamodb.s3.bucket" -> bucket_name,
        "dynamodb.s3.prefix" -> bucket_prefix,
        "dynamodb.s3.bucketOwner" -> account_id_of_bucket,
      ))
    ).getDynamicFrame()

    print(dynamicFrame.getNumPartitions())

    Job.commit()
  }

}
```

這些範例會演示如何讀取 (透過匯出連接器) 及列印具有 dynamodb 分類的 AWS Glue Data Catalog 資料表：

Python

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dynamicFrame = glue_context.create_dynamic_frame.from_catalog(
        database=catalog_database,
        table_name=catalog_table_name,
        additional_options={
            "dynamodb.export": "ddb",
            "dynamodb.s3.bucket": s3_bucket,
            "dynamodb.s3.prefix": s3_bucket_prefix
        }
    )
```

```
print(dynamicFrame.getNumPartitions())

job.commit()
```

Scala

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._


object GlueApp {

  def main(sysArgs: Array[String]): Unit = {
    val glueContext = new GlueContext(SparkContext.getOrCreate())
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val dynamicFrame = glueContext.getCatalogSource(
        database = catalog_database,
        tableName = catalog_table_name,
        additionalOptions = JsonOptions(Map(
            "dynamodb.export" -> "ddb",
            "dynamodb.s3.bucket" -> s3_bucket,
            "dynamodb.s3.prefix" -> s3_bucket_prefix
        ))
    ).getDynamicFrame()
    print(dynamicFrame.getNumPartitions())
)
```

簡化 DynamoDB 匯出 JSON 的使用

DynamoDB 會使用 AWS Glue DynamoDB 匯出連接器進行匯出，這會產生具有特定巢套結構的 JSON 檔案。如需詳細資訊，請參閱資料物件。AWS Glue 會提供 DynamicFrame 轉換，其可將這些結構解除巢狀，使其成為下游應用程式更易於使用的形式。

您可以使用兩種方式中的一種來調用此轉換。呼叫要從 DynamoDB 進行讀取的方法時，可以使用值 "true" 來設定連線選項 "dynamodb.simplifyDDBJson"。也可以將轉換作為 AWS Glue 程式庫中獨立提供的方法進行呼叫。

請考慮由 DynamoDB 匯出產生的下列結構描述：

```
root
|-- Item: struct
|    |-- parentMap: struct
|    |    |-- M: struct
|    |    |    |-- childMap: struct
|    |    |    |    |-- M: struct
|    |    |    |    |    |-- appName: struct
|    |    |    |    |    |    |-- S: string
|    |    |    |    |    |-- packageName: struct
|    |    |    |    |    |    |-- S: string
|    |    |    |    |    |-- updatedAt: struct
|    |    |    |    |    |    |-- N: string
|    |-- strings: struct
|    |    |-- SS: array
|    |    |    |-- element: string
|    |-- numbers: struct
|    |    |-- NS: array
|    |    |    |-- element: string
|    |-- binaries: struct
|    |    |-- BS: array
|    |    |    |-- element: string
|    |-- isDDBJson: struct
|    |    |-- BOOL: boolean
|    |-- nullValue: struct
|    |    |-- NULL: boolean
```

simplifyDDBJson 轉換將此簡化為：

```
root
|-- parentMap: struct
|    |-- childMap: struct
|    |    |-- appName: string
|    |    |-- packageName: string
|    |    |-- updatedAt: string
|-- strings: array
|    |-- element: string
|-- numbers: array
```

```
|    |-- element: string
|-- binaries: array
|    |-- element: string
|-- isDDBJson: boolean
|-- nullValue: null
```

> **Note**
>
> AWS Glue 3.0 及更新版本中會提供 `simplifyDDBJson`。`unnestDDBJson` 轉換也可用於簡化 DynamoDB 匯出 JSON。我們鼓勵使用者從 `unnestDDBJson` 轉換為 `simplifyDDBJson`。

在 DynamoDB 操作中設定平行處理

為了提高效能，可以調整 DynamoDB 連接器的某些參數。調整平行處理參數時，您的目標是最大限度地利用佈建的 AWS Glue 工作者。然後，如果需要更高的效能，建議透過增加 DPU 數量來橫向擴展您的工作。

可以在使用 ETL 連接器時，使用 `dynamodb.splits` 參數來變更 DynamoDB 讀取操作中的平行處理。使用匯出連接器進行讀取時，無需為 Spark 執行器平行處理設定分割數。可以使用 `dynamodb.output.numParallelTasks` 來變更 DynamoDB 寫入操作中的平行處理。

使用 DynamoDB ETL 連接器進行讀取

建議您根據工作組態中設定的最大工作者數目和以下 `numSlots` 計算來計算 `dynamodb.splits`。如果進行自動擴展，則實際可用的工作者數量可能會在該上限下變更。如需有關設定工作者數目上限的詳細資訊，請參閱 the section called “配置星火工作屬性” 中的工作者數目 (`NumberOfWorkers`)。

- `numExecutors = NumberOfWorkers - 1`

  針對內容，為 Spark 驅動程式保留一個執行程式；其他執行程式則用於處理資料。
- `numSlotsPerExecutor =`

  AWS Glue 3.0 and later versions
  - 4，若 `WorkerType` 為 `G.1X`
  - 8，若 `WorkerType` 為 `G.2X`
  - 16，若 `WorkerType` 為 `G.4X`
  - 32，若 `WorkerType` 為 `G.8X`

AWS Glue 2.0 and legacy versions

- 8，若 WorkerType 為 G.1X
- 16，若 WorkerType 為 G.2X

- numSlots = numSlotsPerExecutor * numExecutors

建議您將 dynamodb.splits 設定為可用插槽的數量 numSlots。

寫入到 DynamoDB

dynamodb.output.numParallelTasks 參數使用以下計算來確定每個 Spark 任務的 WCU：

```
permittedWcuPerTask = ( TableWCU * dynamodb.throughput.write.percent ) /
dynamodb.output.numParallelTasks
```

如果組態準確地表示寫入到 DynamoDB 的 Spark 任務數目，則 DynamoDB 寫入器的運作效果最佳。在某些情況下，可能需要覆寫預設計算來提高寫入效能。如果不指定此參數，將透過下列公式自動計算每個 Spark 任務允許的 WCU：

-
  - numPartitions = dynamicframe.getNumPartitions()
  - numSlots (如本節先前所定義)
  - numParallelTasks = min(numPartitions, numSlots)
- 範例 1。DPU=10, WorkerType=Standard. 輸入 DynamicFrame 有 100 個 RDD 分割區。
  - numPartitions = 100
  - numExecutors = (10 - 1) * 2 - 1 = 17
  - numSlots = 4 * 17 = 68
  - numParallelTasks = min(100, 68) = 68
- 範例 2。DPU=10, WorkerType=Standard. 輸入 DynamicFrame 有 20 個 RDD 分割區。
  - numPartitions = 20
  - numExecutors = (10 - 1) * 2 - 1 = 17
  - numSlots = 4 * 17 = 68
  - numParallelTasks = min(20, 68) = 20

> **Note**
>
> 舊版 AWS Glue 和使用標準工作者的工作需要不同的方法來計算插槽數量。如果需要調整這些工作的效能，建議轉換為支援的 AWS Glue 版本。

DynamoDB 連線選項參考

指定 Amazon DynamoDB 的連線。

來源連線和接收器連線的連線選項不同。

"connectionType": "dynamodb" 搭配 ETL 連接器作為來源

在使用 AWS Glue DynamoDB ETL 連接器時，使用以下連線選項搭配 `"connectionType": "dynamodb"` 作為來源：

- `"dynamodb.input.tableName"`：(必要) 要讀取的 DynamoDB。
- `"dynamodb.throughput.read.percent"`：(選用) 要使用的讀取容量單位 (RCU) 百分比。預設值設定為「0.5」。可接受的值從「0.1」(含) 到「1.5」(含)。
  - `0.5` 代表預設讀取率，這表示 AWS Glue 會嘗試使用資料表一半的讀取容量。如果將值增加到 `0.5` 以上，AWS Glue 會增加請求率；如果將值減少到 `0.5` 以下，則會降低讀取請求率。(根據不同因素，例如在 DynamoDB 資料表中是否有統一金鑰分佈等，實際讀取率可能會有所不同。)
  - 當 DynamoDB 資料表處於隨需模式時，AWS Glue 處理資料表的讀取容量為 40000。若要匯出大型資料表，建議您將 DynamoDB 資料表切換為隨需模式。
- `"dynamodb.splits"`：(選用) 定義讀取此 DynamoDB 資料表的同時，要將資料表分割成多少個區塊。預設值已設為「1」。可接受的值從「1」(含) 到「1,000,000」(含)。

  1 表示無平行處理。我們強烈建議您使用以下公式，指定較大的值以獲得更好的效能。如需有關適當設定某個值的詳細資訊，請參閱 the section called “DynamoDB 平行處理”。
- `"dynamodb.sts.roleArn"`：(選用) 跨帳戶存取所要擔任的 IAM 角色 ARN。此參數適用於 AWS Glue 1.0 或以上版本。
- `"dynamodb.sts.roleSessionName"`：(選用) STS 工作階段名稱。預設設定為 "glue-dynamodb-read-sts-session"。此參數適用於 AWS Glue 1.0 或以上版本。

"connectionType": "dynamodb" 搭配作為來源的 AWS Glue DynamoDB 匯出連接器

在使用 AWS Glue DynamoDB 匯出連接器時 (該連接器僅可用於 AWS Glue 版本 2.0 和更新版本)，請搭配使用以下連接選項與 "connectionType": "dynamodb" 作為來源：

- "dynamodb.export"：(必要) 字串值：
  - 如果設為 ddb，則會啟用 AWS Glue DynamoDB 匯出連接器，並將會在 AWS Glue 任務期間呼叫新的 ExportTableToPointInTimeRequest。系統將使用從 dynamodb.s3.bucket 和 dynamodb.s3.prefix 傳遞的位置產生新的匯出。
  - 如果設為 s3，則會啟用 AWS Glue DynamoDB 匯出連接器，但會略過建立新的 DynamoDB 匯出，而是使用 dynamodb.s3.bucket 和 dynamodb.s3.prefix 作為該資料表之前匯出的 Amazon S3 位置。
- "dynamodb.tableArn"：(必要) 要讀取的 DynamoDB。
- "dynamodb.unnestDDBJson"：(選用) 預設值：false。有效值：布林值。如果設為 true，系統會對出現在匯出中的 DynamoDB JSON 結構執行解巢狀轉換。將 "dynamodb.unnestDDBJson" 和 "dynamodb.simplifyDDBJson" 同時設定為 true 是錯誤做法。在 AWS Glue 3.0 及更新版本中，建議您在簡化 DynamoDB Map 類型時使用 "dynamodb.simplifyDDBJson" 以獲得更好的行為。如需更多詳細資訊，請參閱 the section called “簡化 DynamoDB 匯出 JSON 的使用”。
- "dynamodb.simplifyDDBJson"：(選用) 預設值：false。有效值：布林值。如果設為 true，系統會執行轉換，以簡化出現在匯出中的 DynamoDB JSON 結構的結構描述。這與 "dynamodb.unnestDDBJson" 選項具有相同的用途，但是對 DynamoDB Map 類型或甚至是 DynamoDB 資料表中的巢狀 Map 類型提供了更好的支援。AWS Glue 3.0 及更新版本中具有此選項。將 "dynamodb.unnestDDBJson" 和 "dynamodb.simplifyDDBJson" 同時設定為 true 是錯誤做法。如需更多詳細資訊，請參閱 the section called “簡化 DynamoDB 匯出 JSON 的使用”。
- "dynamodb.s3.bucket"：（選用）指定要在其中執行 DynamoDB ExportTableToPointInTime 程序的 Amazon S3 儲存貯體位置。匯出的檔案格式為 DynamoDB JSON。
  - "dynamodb.s3.prefix"：（選用）指定 Amazon S3 儲存貯體內的 Amazon S3 前綴位置，DynamoDB ExportTableToPointInTime 負載將儲存於其中。如果 dynamodb.s3.prefix 和 dynamodb.s3.bucket 均未指定，則這些值將預設為 AWS Glue 任務組態中指定的暫時目錄位置。如需詳細資訊，請參閱 AWS Glue 使用的特殊參數。
  - "dynamodb.s3.bucketOwner"：指定跨帳戶存取 Amazon S3 所需的儲存貯體擁有者。
- "dynamodb.sts.roleArn"：(選用) DynamoDB 資料表的跨帳戶存取和/或跨區域存取所要擔任的 IAM 角色 ARN。注意：相同的 IAM 角色 ARN 將用於存取為 ExportTableToPointInTime 要求中指定的 Amazon S3 位置。

- "dynamodb.sts.roleSessionName"：(選用) STS 工作階段名稱。預設設定為 "glue-dynamodb-read-sts-session"。
- "dynamodb.exportTime" (選用) 有效值：表示 ISO-8601 瞬間的字串。應進行匯出的時間點。
- "dynamodb.sts.region"：(如果使用區域端點進行跨區域呼叫，則為必填項) 託管您要讀取的 DynamoDB 資料表的區域。

"connectionType": "dynamodb" 搭配 ETL 連接器做為接收器

使用下列有 "connectionType": "dynamodb" 的連線選項作為接收器：

- "dynamodb.output.tableName"：(必要) 要寫入的 DynamoDB 資料表。
- "dynamodb.throughput.write.percent"：(選用) 要使用的寫入容量單位 (WCU) 百分比。預設值設定為「0.5」。可接受的值從「0.1」(含) 到「1.5」(含)。
  - 0.5 代表預設寫入率，這代表 AWS Glue 會嘗試使用資料表一半的寫入容量。如果將值增加到 0.5 以上，AWS Glue 會增加請求率；如果將值減少到 0.5 以下，則會降低寫入請求率。(根據不同因素，例如在 DynamoDB 資料表中是否有統一金鑰分佈等，實際寫入率可能會有所不同。)
  - 當 DynamoDB 資料表處於隨需模式時，AWS Glue 會以 40000 處理資料表的寫入容量。若要匯入大型資料表，建議您將 DynamoDB 資料表切換為隨需模式。
- "dynamodb.output.numParallelTasks"：(選用) 定義同時寫入 DynamoDB 的平行任務數量。用於計算每個 Spark 任務的寬鬆 WCU。在大多數情況下，AWS Glue 會計算此值的合理預設值。如需更多詳細資訊，請參閱 the section called “DynamoDB 平行處理”。
- "dynamodb.output.retry"：(選用) 定義有來自 DynamoDB 的 ProvisionedThroughputExceededException 時，要執行多少次重試。預設設定為 "10"。
- "dynamodb.sts.roleArn"：(選用) 跨帳戶存取所要擔任的 IAM 角色 ARN。
- "dynamodb.sts.roleSessionName"：(選用) STS 工作階段名稱。預設設定為 "glue-dynamodb-write-sts-session"。

跨帳戶跨區域存取 DynamoDB 資料表

AWS Glue ETL 任務同時支援跨區域及跨帳戶存取 DynamoDB 資料表。AWS GlueETL 任務支援從另一個 AWS 帳戶的 DynamoDB 資料表讀取資料，以及將資料寫入另一個 AWS 帳戶 DynamoDB 資料表。AWS Glue 也支援從另一個區域中的 DynamoDB 資料表讀取，以及寫入另一個區域中的 DynamoDB 資料表。本節提供設定存取權的指示，並提供範例指令碼。

本節中的程序參考 IAM 教學課程，用於建立 IAM 角色並授與角色存取權。本教學課程也討論如何擔任角色，但在這裡，您將改為在 AWS Glue 中使用任務指令碼來擔任角色。本教學課程也包含一般跨帳

戶實務的相關資訊。如需更多資訊，請參閱 IAM 使用者指南中的教學課程：使用 IAM 角色跨 AWS 帳戶委派存取。

建立角色

跟隨教學課程中的步驟 1，在帳戶 A 中建立 IAM 角色。在定義角色的許可時，您可以選擇連接現有政策，例如 AmazonDynamoDBReadOnlyAccess，或 AmazonDynamoDBFullAccess 以允許角色讀取/寫入 DynamoDB。下列範例顯示建立名為 DynamoDBCrossAccessRole 的角色，具備許可政策 AmazonDynamoDBFullAccess。

授予角色存取權

跟隨 IAM 使用者指南中的教學課程中的步驟 2，以允許帳戶 B 切換至新建立的角色。下列範例會建立使用下列陳述式的新政策：

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": "<DynamoDBCrossAccessRole's ARN>"
    }
}
```

然後，您可以將此政策連接到您要用來存取 DynamoDB 的群組/角色/使用者。

在 AWS Glue 任務指令碼中擔任角色

現在，您可以登入帳戶 B 並建立 AWS Glue 任務。若要建立任務，請參閱設定 Spark 工作的工作屬性 AWS Glue中的指示。

在任務指令碼中，您需要使用 dynamodb.sts.roleArn 參數來擔任 DynamoDBCrossAccessRole 角色。擔任此角色可讓您取得臨時憑證，這些憑證需要用於存取帳戶 B 中的 DynamoDB。請檢閱這些範例指令碼。

對於跨區域的跨帳戶讀取（ETL 連接器）：

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
```

```
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame_from_options(
    connection_type="dynamodb",
    connection_options={
    "dynamodb.region": "us-east-1",
    "dynamodb.input.tableName": "test_source",
    "dynamodb.sts.roleArn": "<DynamoDBCrossAccessRole's ARN>"
    }
)
dyf.show()
job.commit()
```

對於跨區域的跨帳戶讀取（ETL 連接器）：

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame_from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.export": "ddb",
        "dynamodb.tableArn": "<test_source ARN>",
        "dynamodb.sts.roleArn": "<DynamoDBCrossAccessRole's ARN>"
    }
)
dyf.show()
job.commit()
```

對於跨區域的讀取和跨帳戶寫入：

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame_from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.region": "us-east-1",
        "dynamodb.input.tableName": "test_source"
    }
)
dyf.show()

glue_context.write_dynamic_frame_from_options(
    frame=dyf,
    connection_type="dynamodb",
    connection_options={
        "dynamodb.region": "us-west-2",
        "dynamodb.output.tableName": "test_sink",
        "dynamodb.sts.roleArn": "<DynamoDBCrossAccessRole's ARN>"
    }
)

job.commit()
```

Kinesis 連線

您可以使用儲存在 Data Catalog 資料表中的資訊，或提供資訊以直接存取資料串流，藉此從Amazon Kinesis Data Streams 中讀取和寫入。您可以從 Kinesis 讀取信息到火花 DataFrame，然後將其轉換為 AWS Glue DynamicFrame。您可以 DynamicFrames 使用 JSON 格式寫入 Kinesis。如果您直接存取資料串流，則請使用這些選項來提供如何存取資料串流的相關資訊。

如果您使用 `getCatalogSource` 或 `create_data_frame_from_catalog` 來取用來自 Kinesis 串流來源的記錄，則該任務具有 Data Catalog 資料庫和資料表名稱資訊，並可以使用它來獲取一些從 Kinesis 串流來源讀取的基本參數。如果使用

getSource、getSourceWithFormat、createDataFrameFromOptions 或 create_data_frame_from_options，則您必須使用此處描述的連線選項來指定這些基本參數。

您可以使用 GlueContext 類別中指定方法的下列引數來指定 Kinesis 的連線選項。

- Scala
  - connectionOptions：與 getSource、createDataFrameFromOptions、getSink 搭配使用
  - additionalOptions：與 getCatalogSource、getCatalogSink 搭配使用。
  - options：與 getSourceWithFormat、getSinkWithFormat 搭配使用。
- Python
  - connection_options：與 create_data_frame_from_options、write_dynamic_frame_from_options 搭配使用。
  - additional_options：與 create_data_frame_from_catalog、write_dynamic_frame_from_catalog 搭配使用。
  - options：與 getSource、getSink 搭配使用。

如需有關串流 ETL 任務的注意事項和限制，請參閱 the section called “串流 ETL 注意事項和限制”。

設定 Kinesis

若要連線到 AWS Glue Spark 工作中的 Kinesis 資料串流，您需要一些先決條件：

- 如果讀取， AWS Glue 工作必須具有 Kinesis 資料串流的讀取存取權層級 IAM 權限。
- 如果要寫入， AWS Glue 工作必須具有 Kinesis 資料串流的寫入存取權層級 IAM 權限。

在某些情況下，您需要設定其他先決條件：

- 如果您的 AWS Glue 任務設定了「其他網路連線」(通常是用來連接到其他資料集)，而其中一個連線提供 Amazon VPC 網路選項，則會引導您的任務透過 Amazon VPC 進行通訊。在這種情況下，您還需要將 Kinesis 資料串流設定為透過 Amazon VPC 進行通訊。為此，您可以建立 Amazon VPC 與 Kinesis 資料串流之間的介面 VPC 端點。如需詳細資訊，請參閱 Using Kinesis Data Streams with Interface VPC Endpoints。

- 在另一個帳戶中指定 Amazon Kinesis Data Streams 時，您必須設定角色和政策以允許跨帳戶存取。如需詳細資訊，請參閱範例：從不同帳戶中的 Kinesis 串流讀取。

如需有關串流 ETL 任務先決條件的詳細資訊，請參閱 the section called “串流 ETL 任務”。

範例：從 Kinesis 串流讀取

範例：從 Kinesis 串流讀取

搭配 the section called “forEachBatch” 使用。

Amazon Kinesis 串流來源範例：

```
kinesis_options =
   { "streamARN": "arn:aws:kinesis:us-east-2:777788889999:stream/fromOptionsStream",
     "startingPosition": "TRIM_HORIZON",
     "inferSchema": "true",
     "classification": "json"
   }
data_frame_datasource0 =
 glueContext.create_data_frame.from_options(connection_type="kinesis",
 connection_options=kinesis_options)
```

範例：寫入 Kinesis 串流

範例：從 Kinesis 串流讀取

搭配 the section called “forEachBatch” 使用。

Amazon Kinesis 串流來源範例：

```
kinesis_options =
   { "streamARN": "arn:aws:kinesis:us-east-2:777788889999:stream/fromOptionsStream",
     "startingPosition": "TRIM_HORIZON",
     "inferSchema": "true",
     "classification": "json"
   }
data_frame_datasource0 =
 glueContext.create_data_frame.from_options(connection_type="kinesis",
 connection_options=kinesis_options)
```

Kinesis 連線選項參考

指定 Amazon Kinesis Data Streams 的連線選項。

針對 Kinesis 串流資料來源使用下列的連線選項：

- "streamARN" (必要) 用於讀取/寫入。Kinesis 資料串流的 ARN。
- "classification" (讀取時為必要) 用於讀取。記錄中資料使用的檔案格式。除非透過資料型錄提供，否則為必要。
- "streamName" – (選用) 用於讀取。要從中讀取的 Kinesis 資料串流名稱。與 endpointUrl 搭配使用。
- "endpointUrl" – (選用) 用於讀取。預設："https://kinesis.us-east-1.amazonaws.com"。Kinesis 串流的 AWS 端點。除非您要連線到特殊區域，否則無需變更此設定。
- "partitionKey" – (選用) 用於寫入。在產生記錄時使用的 Kinesis 分割區索引鍵。
- "delimiter" (選用) 用於讀取。當 classification 為 CSV 時使用的值分隔符號。預設值為 ","。
- "startingPosition"：(選用) 用於讀取。Kinesis 資料串流中要從中讀取資料的起始位置。可能的值包括 "latest"、"trim_horizon"、"earliest" 或 yyyy-mm-ddTHH:MM:SSZ 模式中 UTC 格式的時間戳記字串 (其中 Z 代表以 +/- 表示的 UTC 時區偏移。例如："2023-04-04T08:00:00-04:00")。預設值為 "latest"。注意：只有 AWS Glue 4.0 版或更新版本"startingPosition"才支援 UTC 格式的時間戳記字串。
- "failOnDataLoss"：(選用) 如果有任何作用中的碎片遺失或過期，則任務失敗。預設值為 "false"。
- "awsSTSRoleARN"：(選用) 用於讀取/寫入。角色的 Amazon 資源名稱 (ARN) 假定使用 AWS Security Token Service (AWS STS)。此角色必須具有描述或讀取 Kinesis 資料串流記錄操作的許可。存取不同帳戶中的資料串流時，您必須使用此參數。搭配 "awsSTSSessionName" 使用。
- "awsSTSSessionName"：(選用) 用於讀取/寫入。使用 AWS STS擔任角色之工作階段的識別符。存取不同帳戶中的資料串流時，您必須使用此參數。搭配 "awsSTSRoleARN" 使用。
- "awsSTSEndpoint": (選擇性) 以假定角色連線到 Kinesis 時要使用的 AWS STS 端點。這允許在 VPC 中使用區域 AWS STS 端點，而預設全域端點無法執行此操作。
- "maxFetchTimeInMs"：(選用) 用於讀取。工作執行程式從 Kinesis 資料串流讀取目前批次記錄所花費的時間上限，以毫秒 (毫秒) 為單位。在這段時間內可能會進行多個 GetRecords API 呼叫。預設值為 1000。
- "maxFetchRecordsPerShard"：(選用) 用於讀取。每個微批次在 Kinesis 資料串流中，每個碎片可擷取的最大記錄數。注意：如果串流工作已讀取 Kinesis 的額外記錄 (在相同的 Get-record 呼叫中)，用戶端可能會超過此限制。如果maxFetchRecordsPerShard需要嚴格，那麼它需要是maxRecordPerRead. 預設值為 100000。

- "maxRecordPerRead"：(選用) 用於讀取。要從每個 getRecords 操作的 Kinesis 資料串流中擷取的記錄數量上限。預設值為 10000。
- "addIdleTimeBetweenReads"：(選用) 用於讀取。增加兩個連續 getRecords 操作之間的時間延遲。預設值為 "False"。此選項僅在 Glue 2.0 及以上版本上才可設定。
- "idleTimeBetweenReadsInMs"：(選用) 用於讀取。兩個連續 getRecords 操作的最小延遲時間，以毫秒為單位指定。預設值為 1000。此選項僅在 Glue 2.0 及以上版本上才可設定。
- "describeShardInterval"：(選用) 用於讀取。指令碼考慮重新分片的兩個 ListShards API 呼叫之間的最小時間間隔。如需詳細資訊，請參閱 Amazon Kinesis Data Streams 開發人員指南中的重新分片的策略。預設值為 1s。
- "numRetries"：(選用) 用於讀取。Kinesis Data Streams API 請求的重試數上限。預設值為 3。
- "retryIntervalMs"：(選用) 用於讀取。重試 Kinesis Data Streams API 呼叫之前的冷卻時間期間 (以毫秒為單位)。預設值為 1000。
- "maxRetryIntervalMs"：(選用) 用於讀取。Kinesis Data Streams API 呼叫之兩次重試之間的最大冷卻時間期間 (以毫秒為單位)。預設值為 10000。
- "avoidEmptyBatches"：(選用) 用於讀取。避免建立空白微批次任務，方法是在批次開始之前檢查 Kinesis 資料串流中是否有未讀取的資料。預設值為 "False"。
- "schema"：(在 inferSchema 設定為 false 時為必要) 用於讀取。用於處理承載的結構描述。如果分類為 avro，提供的架構必須採用 Avro 架構格式。如果分類不是 avro，提供的架構必須採用 DDL 架構格式。

  以下是架構範例。

  Example in DDL schema format

  ```
  `column1` INT, `column2` STRING , `column3` FLOAT
  ```

  Example in Avro schema format

  ```
  {
    "type":"array",
    "items":
    {
      "type":"record",
      "name":"test",
      "fields":
      [
        {
          "name":"_id",
  ```

```
        "type":"string"
      },
      {
        "name":"index",
        "type":
        [
          "int",
          "string",
          "float"
        ]
      }
    ]
  }
}
```

- "inferSchema"：(選用) 用於讀取。預設值為 'false'。如果設為 'true'，將在執行時間時從 foreachbatch 承載偵測架構。
- "avroSchema"：(已棄用) 用於讀取。使用 Avro 格式時，用於指定 Avro 資料架構的參數。此參數現已棄用。使用 schema 參數。
- "addRecordTimestamp"：(選用) 用於讀取。當此選項設定為 'true' 時，資料輸出將包含一個名為 "__src_timestamp" 的額外資料欄，其指示串流收到相應記錄的時間。預設值為 'false'。在 AWS Glue 4.0 版或更新版中支援此選項。
- "emitConsumerLagMetrics"：(選用) 用於讀取。當該選項設置為 'true' 時，對於每個批次，它將發出流接收到的最舊記錄到達時間之間的持續時間的AWS Glue指標。 CloudWatch該指標的名稱是「膠合. 驅動程序. maxConsumerLagInMs」. 預設值為 'false'。在 AWS Glue 4.0 版或更新版中支援此選項。
- "fanoutConsumerARN"：(選用) 用於讀取。Kinesis 串流取用者的 ARN，適用於 streamARN 中指定的串流。用於啟用 Kinesis 連線的強化廣發功能模式。如需有關使用強化廣發功能使用 Kinesis 串流的詳細資訊，請參閱 the section called “在 Kinesis 串流任務中使用強化廣發功能”。
- "recordMaxBufferedTime" – (選用) 用於寫入。預設：1000 (毫秒)。在等待寫入時，記錄受到緩衝的最長時間。
- "aggregationEnabled" – (選用) 用於寫入。預設：true。指定是否應在將記錄傳送至 Kinesis 前先彙整記錄。
- "aggregationMaxSize" – (選用) 用於寫入。預設：51200 (位元組)。若記錄大於此限制，則其會略過彙整工具。請注意，Kinesis 會強制執行 50 KB 的記錄大小限制。若您將此值設定為超過 50 KB，Kinesis 將會拒絕過大的記錄。

- "aggregationMaxCount" – (選用) 用於寫入。預設：4294967295。要匯入彙整記錄的項目數量上限。
- "producerRateLimit" – (選用) 用於寫入。預設：150 (%)。作為後端限制的百分比來限制從單一生產者 (例如您的任務) 傳送的每個碎片輸送量。
- "collectionMaxCount" – (選用) 用於寫入。預設：500。要打包到 PutRecords 請求中的最大項目數。
- "collectionMaxSize" – (選用) 用於寫入。預設：5242880 (位元組)。與 PutRecords 請求一起發送的最大數據量。

在 Kinesis 串流任務中使用強化廣發功能

強化廣發功能取用者能夠從 Kinesis 串流接收記錄，其專用輸送量可能高於一般取用者。這是透過最佳化用來提供資料給 Kinesis 取用者 (例如您的任務) 的傳輸通訊協定來完成。如需有關 Kinesis 強化廣發功能的詳細資訊，請參閱 Kinesis 文件。

在強化廣發功能模式下，maxRecordPerRead 和 idleTimeBetweenReadsInMs 連線選項不再適用，因為使用強化廣發功能時無法設定這些參數。重試的組態選項會如上所述執行。

使用下列程序來啟用和停用串流任務的強化廣發功能。您應該為每個會使用串流資料的任務註冊串流取用者。

若要在任務上啟用強化廣發功能使用服務：

1. 使用 Kinesis API 為您的任務註冊串流取用者。按照 Kinesis 文件中的說明，使用 Kinesis Data Streams API 向強化廣發功能註冊取用者。您只需要按照第一步：呼叫 RegisterStreamConsumer 進行操作。您的請求應傳回一個 ARN，即 *consumerARN*。
2. 在連線方法引數中將連線選項 fanoutConsumerARN 設定為 *consumerARN*。
3. 重新啟動您的任務。

若要停用任務上的強化廣發功能取用服務：

1. 從您的方法呼叫中移除 fanoutConsumerARN 連線選項。
2. 重新啟動您的任務。
3. 按照 Kinesis 文件中的說明取消註冊取用者。這些說明適用於主控台，但也可以透過 Kinesis API 來實現。如需有關透過 Kinesis API 進行串流取用者取消註冊的詳細資訊，請參閱 Kinesis 文件中的 DeregisterStreamConsumer。

Amazon S3 連線

您可以使用 AWS Glue for Spark 在 Amazon S3 中讀取和寫入檔案。AWSGlue for Spark 支援許多存放在 Amazon S3 中的開箱即用的資料格式，包括 CSV、Avro、JSON、Orc 和 Parquet.。如需有關支援資料格式的詳細資訊，請參閱 the section called “資料格式選項”。每種資料格式可支援不同的 AWS Glue 功能集。如需有關功能支援的詳細資訊，請參閱資料格式頁面。此外，您還可以讀取和寫入 Hudi、Iceberg 和 Delta Lake 資料湖架構中存放的版本控制檔案。如需有關資料湖架構的詳細資訊，請參閱 the section called “資料湖架構”。

藉由 AWS Glue，您可以在寫入時將 Amazon S3 物件分割為資料夾結構，然後透過分割區擷取它，使用簡單的組態提升效能。您還可以設定組態，在轉換資料時將小檔案分組在一起以提高效能。您可以在 Amazon S3 中讀取和寫入 `bzip2` 與 `gzip` 封存。

主題

- 設定 S3 連線
- Amazon S3 連線選項參考
- 資料格式的已作廢連線語法
- 排除 Amazon S3 儲存體方案
- 在 AWS Glue 中管理適用於 ETL 輸出的分割區
- 讀取在大型群組中的輸入檔案
- Amazon S3 的 VPC 端點

設定 S3 連線

若要在具有 Spark 任務的 AWS Glue 中連線到 Amazon S3，您需要滿足一些先決條件：

- AWS Glue 任務必須具有相關 Amazon S3 儲存貯體的 IAM 許可。

在某些情況下，您需要設定其他先決條件：

- 設定跨帳戶存取權時，Amazon S3 儲存貯體上適當的存取權控制。
- 基於安全理由，您可以選擇透過 Amazon VPC 路由 Amazon S3 請求。這個方法可能會帶來頻寬和可用性方面的挑戰。如需詳細資訊，請參閱 the section called “Amazon S3 的 VPC 端點”。

Amazon S3 連線選項參考

指定 Amazon S3 的連線。

Amazon S3 管理檔案而非資料表，因此除了指定本文件中提供的連線屬性之外，您還需要指定有關檔案類型的其他組態。您可以透過資料格式選項指定這些資訊。如需有關格式選項的詳細資訊，請參閱 the section called “資料格式選項”。您也可以透過整合 AWS Glue Data Catalog 來指定這些資訊。

如需連線選項和格式選項之間區分的範例，請考慮 the section called “create_dynamic_frame_from_options” 方法如何採用 `connection_type`、`connection_options`、`format` 和 `format_options`。本節特別討論提供給 `connection_options` 的參數。

使用下列有 `"connectionType": "s3"` 的連線選項：

- `"paths"`：(必要) 要讀取的 Amazon S3 路徑清單。
- `"exclusions"`：(選用) 包含要排除之 Unix 樣式 glob 模式 JSON 清單的字串。例如，`"[\"**.pdf\"]"` 會排除所有 PDF 檔案。如需 AWS Glue 支援的 glob 語法的詳細資訊，請參閱包含和排除模式。
- `"compressionType"`：或 `"compression"`：(選用) 指定資料的壓縮方式。將 `"compressionType"` 用於 Amazon S3 來源，以及將 `"compression"` 用於 Amazon S3 目標。一般來說，如果資料具有標準副檔名，則不需要此項目。可能值為 `"gzip"` 和 `"bzip2"`。特定格式可能支援其他壓縮格式。如需有關功能支援的詳細資訊，請參閱資料格式頁面。
- `"groupFiles"`：(選用) 當輸入含有超過 50,000 個檔案時，預設會開啟分組檔案。若要在少於 50,000 個檔案時開啟分組，請將此參數設定為 `"inPartition"`。若要在超過 50,000 個檔案時停用分組，請將此參數設定為 `"none"`。
- `"groupSize"`：(選用) 目標群組大小 (以位元組為單位)。系統會根據輸入資料大小和叢集大小來計算預設值。當輸入檔案數少於 50,000 個時，`"groupFiles"` 必須設定為 `"inPartition"` 才能讓此設定生效。
- `"recurse"`：(選用) 如果設定為 true，則會遞迴讀取指定路徑下所有子目錄中的檔案。
- `"maxBand"`：(選用，進階) 此選項可控制 `s3` 清單可能會在多長時間 (以毫秒為單位) 後變得一致。使用 `JobBookmarks` 來考量 Amazon S3 最終一致性時，會特別追蹤修改時間戳記落在最後 `maxBand` 毫秒內的檔案。使用者大多不需要設定此選項。預設值為 900000 毫秒或 15 分鐘。
- `"maxFilesInBand"`：(選用、進階) 此選項會指定從最後 `maxBand` 秒內所要儲存的檔案數上限。如果超過此數量，系統就會略過額外的檔案，等下一個任務執行到來再處理。使用者大多不需要設定此選項。
- `"isFailFast"`：(選用) 此選項決定 AWS Glue ETL 任務是否擲回讀取器剖析例外狀況。如果設定為 `true`，則如果 Spark 任務的四次重試都無法正確剖析資料，任務就會快速失敗。
- `"catalogPartitionPredicate"`：(選用) 用於讀取。SQL `WHERE` 子句的內容。從具有大量分割區的資料型錄資料表讀取時使用。從資料型錄索引擷取相符的分割區。與 `push_down_predicate`

一起使用，這是 the section called “create_dynamic_frame_from_catalog” 方法 (和其他類似方法) 上的一個選項。如需詳細資訊，請參閱 the section called “目錄分割述詞”。

- `"partitionKeys"`：(選用) 用於寫入。資料欄標籤字串陣列。AWSGlue 將按照此組態指定的方式對資料進行分割。如需詳細資訊，請參閱 the section called “寫入分割區”。
- `"excludeStorageClasses"`：(選用) 用於讀取。指定 Amazon S3 儲存類別的字串陣列。AWSGlue 會根據此組態排除 Amazon S3 物件。如需詳細資訊，請參閱 the section called “排除 Amazon S3 儲存體方案”。

資料格式的已作廢連線語法

某些資料格式可以使用特定連線類型語法來存取。此語法已作廢。建議您改用 the section called “資料格式選項” 中提供的 `s3` 連線類型和格式選項來指定格式。

"connectionType": "Orc"

指定以 Apache Hive 最佳化資料列單欄式 (ORC) 檔案格式存放在 Amazon S3 之檔案的連線。

使用下列有 `"connectionType": "orc"` 的連線選項：

- `paths`：(必要) 要讀取的 Amazon S3 路徑清單。
- (其他選項名稱/值對)：任何其他選項 (包括格式化選項) 都會直接傳遞至 SparkSQL `DataSource`。

"connectionType": "parquet"

指定以 Apache Parquet 檔案格式存放在 Amazon S3 之檔案的連線。

使用下列有 `"connectionType": "parquet"` 的連線選項：

- `paths`：(必要) 要讀取的 Amazon S3 路徑清單。
- (其他選項名稱/值對)：任何其他選項 (包括格式化選項) 都會直接傳遞至 SparkSQL `DataSource`。

排除 Amazon S3 儲存體方案

如果您執行的 AWS Glue ETL 任務會從 Amazon Simple Storage Service (Amazon S3) 讀取檔案或分割區，則您可排除部分 Amazon S3 儲存類別類型。

Amazon S3 會提供下列儲存體方案：

- `STANDARD` — 適用於經常存取資料的一般用途儲存體。

- INTELLIGENT_TIERING — 適用於存取模式不明或不斷變化的資料。
- STANDARD_IA 和 ONEZONE_IA — 適用於長期存放但不常存取的資料。
- GLACIER、DEEP_ARCHIVE 和 REDUCED_REDUNDANCY — 適用於長期存檔和數位儲存。

如需詳細資訊，請參閱 Amazon S3 開發人員指南中的 Amazon S3 儲存體方案。

本節中的範例會示範如何排除 GLACIER 和 DEEP_ARCHIVE 儲存體方案。這些方案可讓您列出相關檔案，但只有在檔案經過還原的情況下，才能進行讀取。(如需詳細資訊，請參閱 Amazon S3 開發人員指南中的還原封存物件。)

透過使用儲存體方案排除功能，您即可確保 AWS Glue 任務會在具有這些儲存體方案分割區的資料表上執行。如果沒有進行排除，則從這些方案中讀取資料的任務就會失敗，並出現以下錯誤訊息：AmazonS3Exception: The operation is not valid for the object's storage class (AmazonS3Exception：本操作不適用於該物件的儲存體方案)。

在 AWS Glue 中篩選 Amazon S3 儲存類別有幾種不同方法。

主題

- 在建立動態框架時排除 Amazon S3 儲存體方案
- 在資料目錄資料表上排除 Amazon S3 儲存體方案

在建立動態框架時排除 Amazon S3 儲存體方案

若要在建立動態框架時排除 Amazon S3 儲存類別，請使用 additionalOptions 中的 excludeStorageClasses。AWS Glue 會自動使用自己的 Amazon S3 Lister 實作來列出對應至特定儲存體方案的檔案，並加以排除。

下列 Python 和 Scala 範例會示範在建立動態框架時，排除 GLACIER 和 DEEP_ARCHIVE 儲存體方案的方法。

Python 範例：

```
glueContext.create_dynamic_frame.from_catalog(
    database = "my_database",
    tableName = "my_table_name",
    redshift_tmp_dir = "",
    transformation_ctx = "my_transformation_context",
    additional_options = {
        "excludeStorageClasses" : ["GLACIER", "DEEP_ARCHIVE"]
    }
```

```
)
```

Scala 範例：

```
val* *df = glueContext.getCatalogSource(
    nameSpace, tableName, "", "my_transformation_context",
    additionalOptions = JsonOptions(
        Map("excludeStorageClasses" -> List("GLACIER", "DEEP_ARCHIVE"))
    )
).getDynamicFrame()
```

在資料目錄資料表上排除 Amazon S3 儲存體方案

您可以將 AWS Glue ETL 任務要排除的儲存類別指定為 AWS Glue Data Catalog 中的資料表參數。若要在 `CreateTable` 操作中包含此參數，則可使用 AWS Command Line Interface (AWS CLI) 或以程式設計方式使用 API。如需詳細資訊，請參閱資料表結構和 CreateTable。

您也可以在 AWS Glue 主控台上指定要排除的儲存體方案。

排除 Amazon S3 儲存體方案 (主控台)

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在左側的導覽窗格中，選擇 Tables (資料表)。
3. 在清單中選擇資料表名稱，然後選擇 Edit table (編輯資料表)。
4. 在 Table properties (資料表屬性) 中，新增 **`excludeStorageClasses`** 做為索引鍵，並將 **`[\"GLACIER\",\"DEEP_ARCHIVE\"]`** 做為值。
5. 選擇 Apply (套用)。

在 AWS Glue 中管理適用於 ETL 輸出的分割區

分割區是組織資料集的一項重要技術，您可以有效率地對它們進行查詢。分割區會根據一或多個欄位的不同值來組織階層目錄結構組織中的資料。

例如，您可以決定在 Amazon Simple Storage Service (Amazon S3) 中以日期來分割應用程式日誌，並依年、月和天來劃分。與單一天資料對應的檔案會放在字首 (例如 `s3://my_bucket/logs/year=2018/month=01/day=23/`)。系統 (如 Amazon Athena、Amazon Redshift Spectrum 與現在的 AWS Glue) 可以使用這些分割區來依分割區值篩選資料，而不必從 Amazon S3 讀取所有基礎資料。

爬蟲程式不僅推斷檔案類型和結構描述，他們在填入 AWS Glue Data Catalog 時也會自動識別資料集的分割區結構。產生的分割區資料行可用於在 AWS Glue ETL 任務中查詢，或查詢 Amazon Athena 之類的引擎。

在您編目資料表後，您可以查看爬蟲程式建立的分割區。在 AWS Glue 主控台，於左側導覽窗格選擇 Tables (資料表)。選擇爬蟲程式建立的資料表，然後選擇 View Partitions (檢視分割區)。

對於樣式為 `key=val` 的 Apache Hive 樣式分割區路徑，爬蟲程式會自動使用金鑰名稱填入欄位名稱。否則，它會使用預設名稱，如 `partition_0`、`partition_1`，以此類推。您可以在主控台上變更預設名稱。請導覽至資料表執行此操作。在索引索引標籤下檢查索引是否存在。如果存在，您需要刪除索引才能繼續 (之後可以使用新資料欄名稱重新建立索引)。然後，選擇編輯結構描述，並在該處修改分割區資料欄的名稱。

在您的 ETL 指令碼中，所以您可以在分割區欄上篩選。因為分割區資訊儲存在 Data Catalog 中，請使用 `from_catalog` API 呼叫，以將分割區欄包含在 `DynamicFrame` 中。例如，使用 `create_dynamic_frame.from_catalog` 代替 `create_dynamic_frame.from_options`。

磁碟分割是減少資料掃描的最佳化技術。如需有關識別此技術何時適用的程序的詳細資訊，請參閱《AWS 規定指南》中 AWS Glue for Apache Spark 任務效能調校最佳實務指引中的減少資料掃描數量。

使用 pushdown 述詞預先篩選

在許多情況下，您可以使用 pushdown 述詞來在分割區上篩選，而無需列出和讀取資料集中的所有檔案。您不須在 DynamicFrame 中讀取整個資料集，然後才進行篩選，您可以直接在 Data Catalog 中分割區中繼資料上套用篩選。然後，您只需要列出與讀取在 DynamicFrame 中實際需要的項目。

例如，您可以在 Python 中寫入下列各項。

```
glue_context.create_dynamic_frame.from_catalog(
    database = "my_S3_data_set",
    table_name = "catalog_data_table",
    push_down_predicate = my_partition_predicate)
```

此會建立在 Data Catalog 中載入並滿足述詞表達式的 DynamicFrame。根據您載入的資料子集大小，這可以節省大量的處理時間。

述詞表達式可以是 Spark SQL 支援的任何布林值表達式。您可以在 Spark SQL 查詢中放入 `WHERE` 子句的任何項目。例如，述詞表達式 `pushDownPredicate = "(year=='2017' and`

month=='04')" 僅讀取 Data Catalog 中 year 等於 2017 且 month 等於 04 的分割區。如需更多詳細資訊，請參閱「Apache Spark SQL 文件」，特別是「Scala SQL 函數參考」。

使用目錄分割述詞的伺服器端篩選

push_down_predicate 選項會在列出目錄中的所有分割區之後，以及列出 Amazon S3 中的這些分割區的檔案之前套用。如果您有資料表的很多分割區，目錄分割區列表仍會產生額外的時間負荷。若要解決此額外負荷，您可以使用伺服器端分割區清除搭配 catalogPartitionPredicate 選項，以使用 AWS Glue Data Catalog 中的分割區索引。當您在一個資料表中有數百萬個分割區時，這會使分割區篩選速度更快。如果您的 catalogPartitionPredicate 需要目錄分割區索引尚未支援的述詞語法，您可以同時在 additional_options 中使用 push_down_predicate 和 catalogPartitionPredicate。

Python：

```
dynamic_frame = glueContext.create_dynamic_frame.from_catalog(
    database=dbname,
    table_name=tablename,
    transformation_ctx="datasource0",
    push_down_predicate="day>=10 and customer_id like '10%'",
    additional_options={"catalogPartitionPredicate":"year='2021' and month='06'"}
)
```

Scala：

```
val dynamicFrame = glueContext.getCatalogSource(
    database = dbname,
    tableName = tablename,
    transformationContext = "datasource0",
    pushDownPredicate="day>=10 and customer_id like '10%'",
    additionalOptions = JsonOptions("""{
        "catalogPartitionPredicate": "year='2021' and month='06'"}""")
    ).getDynamicFrame()
```

> **Note**
>
> push_down_predicate 和 catalogPartitionPredicate 使用不同的語法。前者使用 Spark SQL 標準語法，後者使用 JSQL 剖析器。

寫入分割區

在預設情況下，在對 DynamicFrame 寫入時，不會對其進行分割。會在指定輸出路徑的最高層級寫入所有輸出檔。先前，將 DynamicFrame 寫到分割區的唯一方法是將它轉換為 Spark SQL DataFrame 再進行寫入。

DynamicFrames 現支援使用一系列的金鑰來進行原生分割，在您建立目的地時使用 `partitionKeys` 選項。例如，以下 Python 程式碼會將資料集寫出至格式為 Parquet 的 Amazon S3 中，以類型欄位寫入分割的目錄中。您可以在此使用其他系統 (例如 Amazon Athena) 處理這些分割區。

```
glue_context.write_dynamic_frame.from_options(
    frame = projectedEvents,
    connection_type = "s3",
    connection_options = {"path": "$outpath", "partitionKeys": ["type"]},
    format = "parquet")
```

讀取在大型群組中的輸入檔案

您可以設定資料表的屬性，來啟用 AWS Glue ETL 任務，以讓檔案從 Amazon S3 資料存放區進行讀取時進行分組。這些屬性可讓每個 ETL 任務將一組輸入檔案輸入到單一記憶體分割區，這在 Amazon S3 資料存放區中有大量小型檔案時特別有用。在您設定特定屬性時，您會指示 AWS Glue 以在 Amazon S3 資料分割區中分組檔案並設定要讀取的群組大小。您也可以在從 Amazon S3 資料存放區讀取時，使用 `create_dynamic_frame.from_options` 方法設定這些選項。

若要啟用資料表分組檔案，您須在資料表結構的參數欄位中設定金鑰值對。使用 JSON 符號來設定資料表的參數欄位值。關於編輯資料表屬性的詳細資訊，請參閱 [檢視與編輯資料表的詳細資訊]。

您可以使用此方法，以啟用在 Data Catalog 中與 Amazon S3 資料存放區的資料表群組。

groupFiles

將 groupFiles 設為 `inPartition`，可將檔案分組在 Amazon S3 資料分割區中。如果超過 50,000 個輸入檔案，則 AWS Glue 會自動分組，如下例所示。

```
'groupFiles': 'inPartition'
```

groupSize

將 groupSize 設定為群組目標大小 (以位元組為單位)。groupSize 屬性是選用的，如果沒有提供，AWS Glue 會計算在叢集中使用所有 CPU 核心的大小，同時仍降低 ETL 任務的總數與記憶體內分割區。

例如，下列會將群組大小設定為 1 MB。

```
'groupSize': '1048576'
```

請注意，您應使用計算的結果來設定 `groupsize`。例如：1024 * 1024 = 1048576。

recurse

將 recurse (遞迴) 設定為 `True` 以在指定 `paths` 作為路徑陣列時，遞迴讀取所有子目錄中的檔案。如果 `paths` 是 Amazon S3 中的物件金鑰陣列或輸入格式為 parquet/orc，則您不需要設定遞迴，如下列範例所示。

```
'recurse':True
```

如果您直接使用 `create_dynamic_frame.from_options` 方法從 Amazon S3 進行讀取，則新增這些連線選項。例如，以下會嘗試將檔案分組到 1 MB 群組。

```
df = glueContext.create_dynamic_frame.from_options("s3", {'paths': ["s3://s3path/"],
 'recurse':True, 'groupFiles': 'inPartition', 'groupSize': '1048576'}, format="json")
```

> **Note**
>
> `groupFiles`從以下資料格式建立的 DynamicFrames 支援：csv、ion、grokLog、json 和 xml。avro、parquet 和 orc 不支援此選項。

Amazon S3 的 VPC 端點

基於安全考量，許多 AWS 客戶會在 Amazon Virtual Private Cloud (Amazon VPC) 環境中執行其應用程式。運用 Amazon VPC，您可以將 Amazon EC2 執行個體啟動到 Virtual Private Cloud 上，此 Virtual Private Cloud 與其他網路 (包括公有網際網路) 在邏輯上隔離。使用 Amazon VPC，您可以控制其 IP 地址範圍、子網路、路由表、網路閘道及安全設定。

> **Note**
>
> 如果您是在 2013 年 12 月 4 日之後建立 AWS 帳戶，則每個 AWS 區域中都會有預設的 VPC。您可以立即開始使用預設的 VPC，不需進行任何額外的設定。
> 如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的您的 VPC 和子網路。

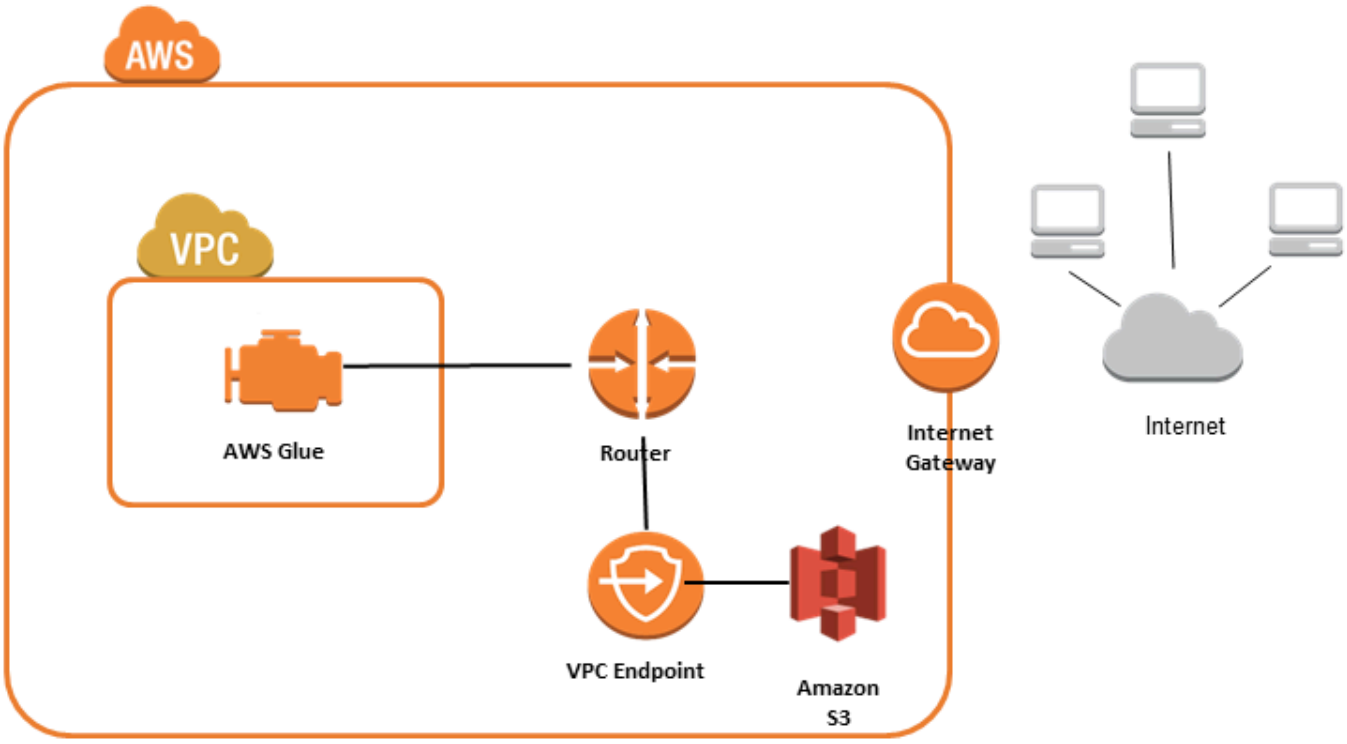
許多客戶對於透過公有的網際網路來傳送和接收資料，都會有合法的隱私與安全性疑慮。客戶可以使用虛擬私有網路 (VPN)，來轉傳透過自己企業網路基礎設施的所有 Amazon S3 網路傳輸資料，以解決前述的問題。但是這個方法可能會帶來頻寬和可用性的挑戰。

Amazon S3 的 VPC 端點可以減低這些挑戰的阻礙。Amazon S3 的 VPC 端點，可讓 AWS Glue 使用私有 IP 地址來存取 Amazon S3，而不需接觸到公開的網際網路。AWS Glue 不需公有 IP 地址，您的 VPC 中也不需要網際網路閘道、NAT 裝置或虛擬私有閘道。您可以使用端點規則來控制對 Amazon S3 的存取。您 VPC 與 AWS 服務之間的流量都會保持在 Amazon 網路的範圍內。

當您建立 Amazon S3 使用的 VPC 端點時，區域 (例如 s3.us-west-2.amazonaws.com) 內傳送到 Amazon S3 端點的所有要求，都會轉傳到 Amazon 網路中的私有 Amazon S3 端點。您不需要修改 VPC 中在 Amazon EC2 執行個體上執行的應用程式，端點的名稱會保持不變，但轉傳至 Amazon S3 的作業會完全在 Amazon 網路中進行，不會存取公有的網際網路。

如需 VPC 端點的完整資訊，請參閱《Amazon VPC 使用者指南》中的 VPC 端點。

下圖顯示 AWS Glue 可如何使用 VPC 端點來存取 Amazon S3。



設定 Amazon S3 的存取

1. 登入 AWS Management Console，並在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
2. 在左側導覽窗格中選擇 Endpoints (端點)。
3. 選擇 Create Endpoint (建立端點)，然後按照步驟來建立閘道類型的 Amazon S3 VPC 端點。

Amazon DocumentDB 連線

可以使用 AWS Glue for Spark 在 Amazon DocumentDB 中讀取和寫入資料表。可以使用透過 AWS Glue 連線儲存在 AWS Secrets Manager 中的憑證來連線至 Amazon DocumentDB。

如需 Amazon DocumentDB 的詳細資訊，請參閱 Amazon DocumentDB 文件。

> **Note**
>
> 使用 AWS Glue 連接器時，目前不支援 Amazon DocumentDB 彈性叢集。如需有關彈性叢集的詳細資訊，請參閱 Using Amazon DocumentDB elastic clusters。

讀取和寫入 Amazon DocumentDB 集合

> **Note**
>
> 建立連接至 Amazon DocumentDB 的 ETL 任務時，如為 `Connections` 任務屬性，您必須指定連線物件，以指定執行 Amazon DocumentDB 的 Virtual Private Cloud (VPC)。如為連線物件，連線類型必須為 `JDBC`，而 `JDBC URL` 必須為 `mongo://<DocumentDB_host>:27017`。

> **Note**
>
> 這些程式碼範例專為 AWS Glue 3.0 而開發。若要遷移至 AWS Glue 4.0，請參閱 the section called “MongoDB”。`uri` 參數已變更。

> **Note**
>
> 使用 Amazon DocumentDB 時，在某些情況下 `retryWrites` 必須設定為 false，例如當編寫的文件指定 `_id` 時。如需詳細資訊，請參閱 Amazon DocumentDB 文件中的 MongoDB 的功能差異。

下列 Python 指令碼示範使用連線類型和連線選項，以讀取和寫入至 Amazon DocumentDB。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext, SparkConf
from awsglue.context import GlueContext
from awsglue.job import Job
import time
```

```
## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

job = Job(glueContext)
job.init(args['JOB_NAME'], args)

output_path = "s3://some_bucket/output/" + str(time.time()) + "/"
documentdb_uri = "mongodb://<mongo-instanced-ip-address>:27017"
documentdb_write_uri = "mongodb://<mongo-instanced-ip-address>:27017"

read_docdb_options = {
    "uri": documentdb_uri,
    "database": "test",
    "collection": "coll",
    "username": "username",
    "password": "1234567890",
    "ssl": "true",
    "ssl.domain_match": "false",
    "partitioner": "MongoSamplePartitioner",
    "partitionerOptions.partitionSizeMB": "10",
    "partitionerOptions.partitionKey": "_id"
}

write_documentdb_options = {
    "retryWrites": "false",
    "uri": documentdb_write_uri,
    "database": "test",
    "collection": "coll",
    "username": "username",
    "password": "pwd"
}

# Get DynamicFrame from  DocumentDB
dynamic_frame2 =
 glueContext.create_dynamic_frame.from_options(connection_type="documentdb",

 connection_options=read_docdb_options)

# Write DynamicFrame to MongoDB and DocumentDB
```

```
glueContext.write_dynamic_frame.from_options(dynamic_frame2,
 connection_type="documentdb",

 connection_options=write_documentdb_options)

job.commit()
```

下列 Scala 指令碼示範使用連線類型和連線選項，以讀取和寫入至 Amazon DocumentDB。

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamicFrame
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {
  val DOC_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
  val DOC_WRITE_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
  lazy val documentDBJsonOption = jsonOptions(DOC_URI)
  lazy val writeDocumentDBJsonOption = jsonOptions(DOC_WRITE_URI)
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    // Get DynamicFrame from DocumentDB
    val resultFrame2: DynamicFrame = glueContext.getSource("documentdb",
 documentDBJsonOption).getDynamicFrame()

    // Write DynamicFrame to DocumentDB
    glueContext.getSink("documentdb", writeJsonOption).writeDynamicFrame(resultFrame2)

    Job.commit()
  }

  private def jsonOptions(uri: String): JsonOptions = {
    new JsonOptions(
      s"""{"uri": "${uri}",
```

```
          |"database":"test",
          |"collection":"coll",
          |"username": "username",
          |"password": "pwd",
          |"ssl":"true",
          |"ssl.domain_match":"false",
          |"partitioner": "MongoSamplePartitioner",
          |"partitionerOptions.partitionSizeMB": "10",
          |"partitionerOptions.partitionKey": "_id"}""".stripMargin)
  }
}
```

Amazon DocumentDB 連線選項參考

指定 Amazon DocumentDB (with MongoDB compatibility) 的連線。

來源連線和接收器連線的連線選項不同。

"connectionType": "Documentdb" as source

使用下列有 `"connectionType": "documentdb"` 的連線選項作為來源：

- `"uri"`：(必要) 讀取的 Amazon DocumentDB 主機，格式為 `mongodb://<host>:<port>`。
- `"database"`：(必要) 要從中讀取的 Amazon DocumentDB 資料庫。
- `"collection"`：(必要) 要從中讀取的 Amazon DocumentDB 集合。
- `"username"`：(必要) Amazon DocumentDB 使用者名稱。
- `"password"`：(必要) Amazon DocumentDB 密碼。
- `"ssl"`：(如果使用 SSL，則為必要) 如果您的連線使用 SSL，則必須加入此選項並具備值 `"true"`。
- `"ssl.domain_match"`：(如果使用 SSL，則為必要) 如果您的連線使用 SSL，則必須加入此選項並具備值 `"false"`。
- `"batchSize"`：(選用)：每個批次傳回的文件數目，於內部批次的游標內使用。
- `"partitioner"`：(選用)：從 Amazon DocumentDB 讀取輸入資料的分割區類別名稱。連接器提供下列分割區：
  - `MongoDefaultPartitioner` (預設) (不支援 AWS Glue 4.0)
  - `MongoSamplePartitioner` (不支援 AWS Glue 4.0)
  - `MongoShardedPartitioner`
  - `MongoSplitVectorPartitioner`

  - MongoPaginateByCountPartitioner
  - MongoPaginateBySizePartitioner (不支援 AWS Glue 4.0)
- "partitionerOptions" (選用)：指定分割區的選項。每個分割區都支援下列選項：
  - MongoSamplePartitioner: partitionKey, partitionSizeMB, samplesPerPartition
  - MongoShardedPartitioner: shardkey
  - MongoSplitVectorPartitioner：partitionKey，partitionSizeMB
  - MongoPaginateByCountPartitioner: partitionKey, numberOfPartitions
  - MongoPaginateBySizePartitioner：partitionKey，partitionSizeMB

  如需有關這些選項的詳細資訊，請參閱 MongoDB 文件中的分割區組態。

"connectionType": "Documentdb" as sink

使用下列有 "connectionType": "documentdb" 的連線選項作為接收器：

- "uri"：(必要) 寫入的 Amazon DocumentDB 主機，格式為 mongodb://<host>:<port>。
- "database"：(必要) 要寫入的 Amazon DocumentDB 資料庫。
- "collection"：(必要) 要寫入的 Amazon DocumentDB 集合。
- "username"：(必要) Amazon DocumentDB 使用者名稱。
- "password"：(必要) Amazon DocumentDB 密碼。
- "extendedBsonTypes"：(選用) 如果 true，在寫入資料到 Amazon DocumentDB 時允許延伸的 BSON 類型。預設值為 true。
- "replaceDocument"：(選用) 如果 true，則在儲存包含 _id 欄位的資料集時取代整個文件。若為 false，則文件中僅有與資料集中欄位相符的欄位會更新。預設值為 true。
- "maxBatchSize"：(選用)：儲存資料時大量操作的批次大小上限。預設為 512。
- "retryWrites":(選用) 如果 AWS Glue 發生網路錯誤，系統會自動重試特定寫入操作一次。

OpenSearch 服務連接

您可以在 AWS Glue 4.0 及更新版本中使用 Spark 的 AWS Glue 來讀取和寫入 OpenSearch 服務中的資料表。您可以使用 OpenSearch 查詢定義要從 OpenSearch 服務讀取的內容。您可以使用AWS Secrets Manager透過 AWS Glue 連線儲存在中的 HTTP 基本驗證認證連線到 OpenSearch 服務。此功能與無伺 OpenSearch 服器服務不相容。

如需有關 Amazon OpenSearch 服務的詳細資訊，請參閱 Amazon OpenSearch 服務文件。

設定 OpenSearch 服務連線

若要從 AWS Glue 連線到 OpenSearch 服務，您必須在AWS Secrets Manager密碼中建立並儲存您的 OpenSearch 服務認證，然後將該機密與 OpenSearch Service AWS Glue 連線建立關聯。

先決條件：

- 依照 Amazon 服務文件中的指示，識別網域端*##AoS port* 和連接埠、AoSport，或按照 Amazon 服務文件中的指示建立資源。 OpenSearch 如需有關建立網域的詳細資訊，請參閱 Amazon OpenSearch 服務文件中的建立和管理 Amazon OpenSearch 服務網域。

  Amazon OpenSearch 服務域端點將具有以下默認表單 https://search-*domainName*-*unstructuredIdContent*。 *##. #*馬遜. 如需識別網域端點的詳細資訊，請參閱 Amazon OpenSearch 服務文件中的建立和管理 Amazon OpenSearch 服務網域。

  識別或產生您網域的 HTTP 基本身分驗證憑證 *aosUser* 和 *aosPassword*。

若要設定 OpenSearch 服務的連線：

1. 在中AWS Secrets Manager，使用您的 OpenSearch 服務認證建立密碼。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *secretName*，以便進行下一個步驟。
   - 在選取鍵/值組時，請使用 *aosUser* 值來建立 `opensearch.net.http.auth.user` 金鑰對。
   - 在選取鍵/值組時，請使用 *aosPassword* 值來建立 `opensearch.net.http.auth.pass` 金鑰對。
2. 在 AWS Glue 主控台中，依照 the section called “新增 AWS Glue 連線” 中的步驟建立連線。建立連線之後，請保留連線名稱 *connectionName*，以便未來在 AWS Glue 中使用。
   - 選取連線類型時，請選取 OpenSearch 服務。
   - 選取網域端點時，請提供 *aosEndpoint*。
   - 選取連接埠時，請提供 *aosPort*。
   - 選取 AWS 機密時，請提供 *secretName*。

建立 AWS Glue OpenSearch 服務連線後，您必須先執行下列步驟，才能執行 AWS Glue 工作：

- 授予與您 AWS Glue 任務許可相關聯的 IAM 角色，以讀取 *secretName*。
- 在您的 AWS Glue 任務組態中，提供 *connectionName* 作為其他網路連線。

從 OpenSearch 服務索引讀取

先決條件：

- 您想從中讀取的 OpenSearch 服務索引，*AoSin* dex。
- 設定為提供驗證和網路位置資訊的 AWS Glue OpenSearch 服務連線。若要取得此功能，請完成上一個程序中的步驟：設定與 OpenSearch 服務的連線。您將會需要 AWS Glue 連線的名稱，*connectionName*。

這個例子從 Amazon OpenSearch 服務讀取索引。您將需要提供 pushdown 參數。

例如：

```
opensearch_read = glueContext.create_dynamic_frame.from_options(
    connection_type="opensearch",
    connection_options={
        "connectionName": "connectionName",
        "opensearch.resource": "aosIndex",
        "pushdown": "true",
    }
)
```

您也可以提供查詢字串，以篩選 DynamicFrame. 您將需要設定 opensearch.query。

opensearch.query 可接受 URL 查詢參數字串 *queryString* 或查詢 DSL JSON 物件 *queryObject*。如需有關查詢 DSL 的詳細資訊，請參閱文件中的查詢 DSL。 OpenSearch若要提供 URL 查詢參數字串，請在查詢前面加上 ?q= 字首 (如同您會在完整 URL 中加上字首)。若要提供查詢 DSL 物件，請將 JSON 物件字串逸出後再進行。

例如：

```
            queryObject = "{ "query": { "multi_match": { "query": "Sample", "fields":
 [ "sample" ] } } }"
            queryString = "?q=queryString"

            opensearch_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="opensearch",
    connection_options={
        "connectionName": "connectionName",
```

```
        "opensearch.resource": "aosIndex",
        "opensearch.query": queryString,
        "pushdown": "true",
    }
)
```

如需如何在特定語法之外建立查詢的詳細資訊，請參閱 OpenSearch 文件中的查詢字串語法。

從包含陣列類型資料的 OpenSearch 集合讀取時，您必須使用opensearch.read.field.as.array.include參數在方法呼叫中指定哪些欄位是陣列類型。

例如，在閱讀下列文件時，您會遇到 genre 和 actor 陣列欄位：

```
{
    "_index": "movies",
    "_id": "2",
    "_version": 1,
    "_seq_no": 0,
    "_primary_term": 1,
    "found": true,
    "_source": {
        "director": "Frankenheimer, John",
        "genre": [
            "Drama",
            "Mystery",
            "Thriller",
            "Crime"
        ],
        "year": 1962,
        "actor": [
            "Lansbury, Angela",
            "Sinatra, Frank",
            "Leigh, Janet",
            "Harvey, Laurence",
            "Silva, Henry",
            "Frees, Paul",
            "Gregory, James",
            "Bissell, Whit",
            "McGiver, John",
            "Parrish, Leslie",
            "Edwards, James",
            "Flowers, Bess",
            "Dhiegh, Khigh",
```

```
            "Payne, Julie",
            "Kleeb, Helen",
            "Gray, Joe",
            "Nalder, Reggie",
            "Stevens, Bert",
            "Masters, Michael",
            "Lowell, Tom"
        ],
        "title": "The Manchurian Candidate"
    }
}
```

在此情況下，您會在方法呼叫中包含那些欄位名稱。例如：

```
"opensearch.read.field.as.array.include": "genre,actor"
```

如果在文件結構中將陣列欄位巢狀化，請使用點標記法來表示："genre,actor,foo.bar.baz"。這將透過內含嵌入文件 bar 的 foo 嵌入文件，指定來源文件中包含的 baz 陣列。

寫入 OpenSearch 服務資料表

此範例會將現有 DynamicFrame####的資訊寫入服務。 OpenSearch 如果索引已有資訊，AWSGlue 會從您的 DynamicFrame. 您將需要提供 pushdown 參數。

先決條件：

- 您想要寫入的 OpenSearch 服務表。您將需要資料表的識別資訊。我們稱此為 *tableName*。
- 設定為提供驗證和網路位置資訊的 AWS Glue OpenSearch 服務連線。若要取得此功能，請完成上一個程序中的步驟：設定與 OpenSearch 服務的連線。您將會需要 AWS Glue 連線的名稱，*connectionName*。

例如：

```
glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="opensearch",
    connection_options={
      "connectionName": "connectionName",
      "opensearch.resource": "aosIndex",
    },
```

```
)
```

OpenSearch 服務連線選項參考

- `connectionName` – 必要。用於讀取/寫入。AWSGlue OpenSearch 服務連線的名稱，設定為為您的連線方法提供驗證和網路位置資訊。
- `opensearch.resource` – 必要。用於讀取/寫入。有效值： OpenSearch 索引名稱。將會與您的連線方法互動的索引名稱。
- `opensearch.query` – 用於讀取。有效值：字串逸出的 JSON，或此字串以 ? 開頭時，URL 的搜尋部分。篩選讀取時應擷取的內容的 OpenSearch 查詢。如需有關使用此參數的詳細資訊，請參閱上一節 the section called “從 OpenSearch 服務中讀取”。
- `pushdown`：如有需要。用於讀取。有效值：布林值。指示 Spark 傳遞讀取查詢下來，以 OpenSearch 便數據庫只返回相關文檔。
- `opensearch.read.field.as.array.include`：如果讀取的是陣列類型資料，則需要。用於讀取。有效值：以逗號分隔的欄位名稱清單。指定要從 OpenSearch 文件中讀取為陣列的欄位。如需有關使用此參數的詳細資訊，請參閱上一節 the section called “從 OpenSearch 服務中讀取”。

Redshift 連線

您可以使用 Spark 的 AWS Glue 從 Amazon Redshift 資料庫中讀取和寫入資料表。連接到 Amazon Redshift 資料庫時， AWS Glue 會使用亞馬遜紅移 SQL `COPY` 和`UNLOAD`命令，透過 Amazon S3 移動資料以達到最大輸送量。在 AWS Glue 4.0 及更新版本中，您可以使用適用於 Apache Spark 的 Amazon Redshift 整合來讀取和寫入，其中包含 Amazon Redshift 特有的最佳化和功能，而不是透過舊版連線時可用的功能。

了解 AWS Glue 如何讓 Amazon Redshift 使用者更輕鬆地遷移到 AWS Glue 以進行無伺服器資料整合和 ETL。

設定 Redshift 連線

若要在 AWS Glue 中使用 Amazon Redshift 叢集，您需要一些先決條件：

- 從資料庫讀取和寫入資料庫時用於暫存空間的 Amazon S3 目錄。
- 一個可在您的 Amazon 紅移叢集、 AWS Glue 任務和 Amazon S3 目錄之間進行通訊的亞馬遜 VPC。
- AWS Glue 任務和 Amazon Redshift 叢集上的適當 IAM 許可。

設定 IAM 角色

設定 Amazon Redshift 叢集的角色

您的 Amazon Redshift 叢集必須能夠讀取和寫入 Amazon S3，才能與 AWS Glue 任務整合。若要允許這項功能，您可以將 IAM 角色與想要連線的 Amazon Redshift 叢集建立關聯。您的角色應具有允許從 Amazon S3 臨時目錄讀取和寫入該目錄的政策。您的角色應具有允許 `redshift.amazonaws.com` 服務 `AssumeRole` 的信任關係。

將 IAM 角色與 Amazon Redshift 建立關聯

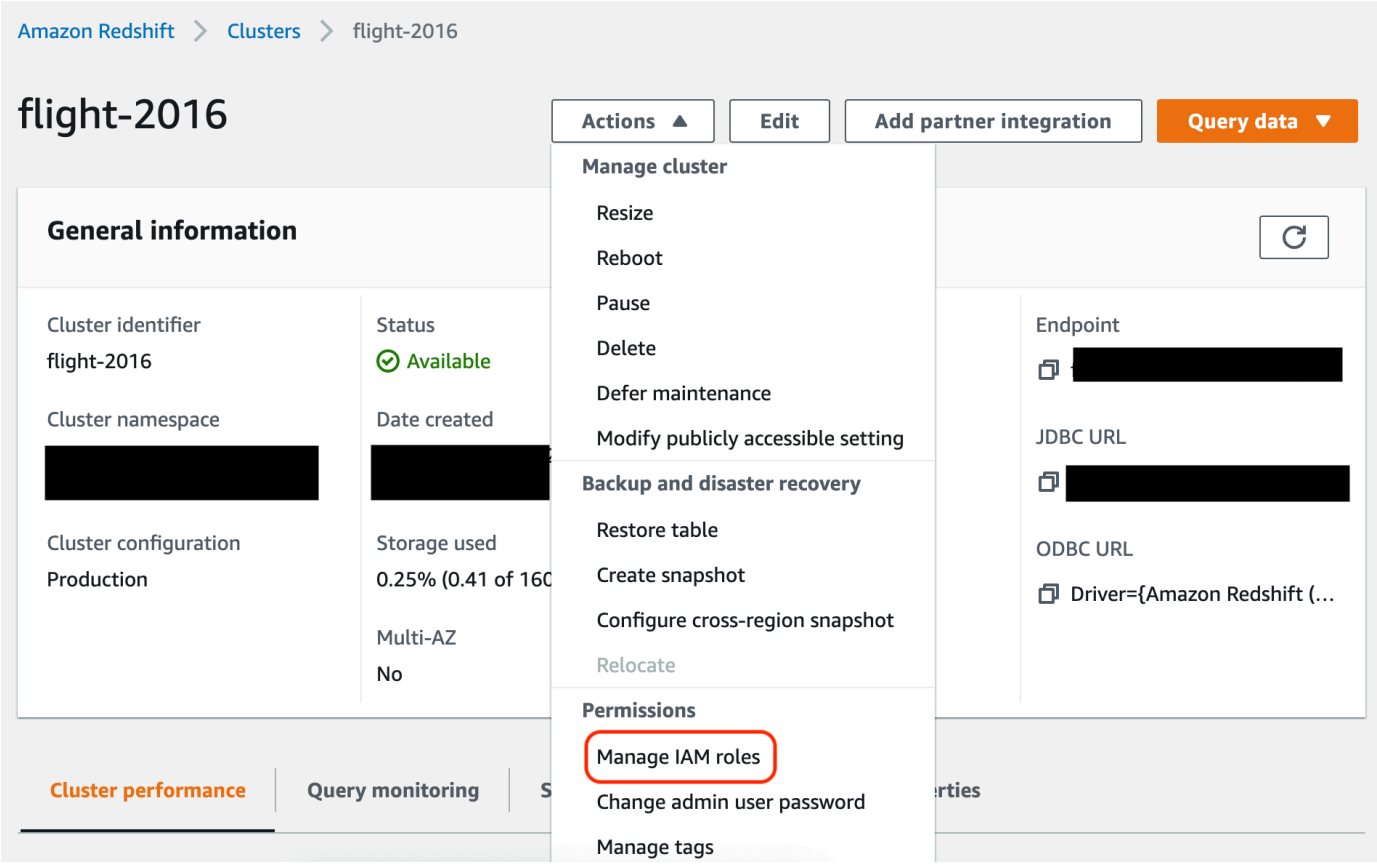
1. 先決條件：用於檔案暫存空間的 Amazon S3 儲存貯體或目錄。
2. 確定您的 Amazon Redshift 叢集需要哪些 Amazon S3 許可。在 Amazon Redshift 叢集間移動資料時， AWS Glue 任務會針對 Amazon Redshift 發出複製和卸載陳述式。如果您的任務修改了 Amazon Redshift 中的資料表， AWS Glue 也會發出 CREATE 程式庫陳述式。如需 Amazon Redshift 執行這些陳述式所需的特定 Amazon S3 許可的相關資訊，請參閱 Amazon Redshift 文件：存取其他資源的許可。 AWS
3. 在 IAM 主控台中，建立具有必要許可的 IAM 政策。如需有關建立政策的詳細資訊，請參閱建立 IAM 政策。
4. 在 IAM 主控台中，建立角色和信任關係，以允許 Amazon Redshift 擔任該角色。遵循 IAM 文件中的指示建立 AWS 服務的角色 (主控台)

   - 當系統要求選擇 AWS 服務使用案例時，請選擇「Redshift-可自訂」。
   - 當系統要求您附加政策時，請選擇您先前定義的政策。

   > **Note**
   >
   > 如需有關為 Amazon Redshift 設定角色的詳細資訊，請參閱 Amazon Redshift 文件中的授權亞馬遜紅移代表您存取其他 AWS 服務。

5. 在 Amazon Redshift 主控台中，將角色與您的 Amazon Redshift 叢集建立關聯。請按照 Amazon Redshift 文件中的說明進行操作。

   在 Amazon Redshift 主控台中選取反白顯示的選項，以進行此設定：



> **Note**
>
> 根據預設，AWS Glue 任務會傳遞使用您指定用來執行任務的角色建立的 Amazon Redshift 臨時登入資料。我們不建議使用這些憑證。基於安全考量，這些憑證會在 1 小時後過期。

設定 AWS Glue 工作的角色

AWS Glue 任務需要角色才能存取 Amazon S3 儲存貯體。您不需要 Amazon Redshift 叢集的 IAM 許可，您的存取是由 Amazon VPC 中的連線和資料庫憑證來控制。

設定 Amazon VPC

設定 Amazon Redshift 資料存放區存取

1. 登入 AWS Management Console 並開啟 Amazon Redshift 主控台，網址為 https://console.aws.amazon.com/redshiftv2/。
2. 在左側導覽窗格中選擇 Clusters (叢集)。

3. 選擇您想要從 AWS Glue 存取的叢集名稱。
4. 在 Cluster Properties (叢集屬性) 區段中，選擇 VPC security groups (VPC 安全群組) 中的安全群組，以允許 AWS Glue 使用。記錄所選的安全群組名稱，供日後參考。選擇安全群組後，將開啟 Amazon EC2 主控台安全群組清單。
5. 選擇要修改的安全群組並導覽至 Inbound (傳入) 索引標籤。
6. 新增自我參考規則，以允許 AWS Glue 元件進行通訊。具體來說，新增或確認有類型 `All TCP`、通訊協定為 `TCP`，連接埠範圍包含所有連接埠，且其來源與群組 ID 為相同安全群組名稱的規則。

   傳入規則類似如下：

Type	通訊協定	連接埠範圍	來源
所有 TCP	TCP	0–65535	database-security-group

   例如：

7. 同時新增一個規則，以用於傳出流量。您可以開啟傳出流量到所有連接埠，例如：

Type	通訊協定	連接埠範圍	目的地
所有流量	ALL	ALL	0.0.0.0/0

   或建立 Type (類型) `All TCP`、Protocol (通訊協定) 為 `TCP`、Port Range (連接埠範圍) 包含所有連接埠，且其 Destination (目的地) 與 Group ID (群組 ID) 為相同安全群組名稱的自我參考規則。如果使用 Amazon S3 VPC 端點，也可以針對 Amazon S3 存取新增 HTTPS 規則。安全群組規則中需要 *`s3-prefix-list-id`* 才能允許從虛擬私人雲端到 Amazon S3 VPC 端點的流量。

   例如：

Type	通訊協定	連接埠範圍	目的地
所有 TCP	TCP	0–65535	*`security-group`*

Type	通訊協定	連接埠範圍	目的地
HTTPS	TCP	443	*S3-prefix-list-id*

設置 AWS Glue

您必須建立可提供 Amazon VPC 連線資訊的 AWS Glue 資料型錄連線。

在主控台中將亞馬 Amazon Redshift Amazon VPC 連線設定為 AWS Glue

1. 依照下列步驟建立資料型錄連線：the section called “新增 AWS Glue 連線”。建立連線之後，請保留連線名稱 *connectionName*，以便進行下一個步驟。

   - 選取連線類型時，請選取 Amazon Redshift。
   - 選取 Redshift 叢集時，請依名稱選取您的叢集。
   - 為叢集上的 Amazon Redshift 使用者提供預設連線資訊。
   - 系統會自動進行 Amazon VPC 設定。

   > **Note**
   >
   > 當您透過 AWS SDK 建立 Amazon Redshift 連線時，您需要手動為 Amazon VPC 提供 `PhysicalConnectionRequirements`。

2. 在您的 AWS Glue 工作組態中，提供####作為其他網路連線。

範例：從 Amazon Redshift 資料表讀取

您可以從 Amazon Redshift 叢集和 Amazon Redshift Serverless 環境讀取。

先決條件：您想要讀取的 Amazon Redshift 資料表。請按照上一節中的步驟進行操作，the section called “設定 Redshift”之後您應該為臨時目錄、*temp-s3-dir* 和 IAM 角色（在帳戶中）擁有 Amazon S3 URI。*rs-role-namerole-account-id*

Using the Data Catalog

其他先決條件：您想要讀取的 Amazon Redshift 資料表的資料型錄資料庫和資料表。如需有關資料型錄的詳細資訊，請參閱 資料探索與編目。在為您的 Amazon Redshift 表格建立項目之後，您將識別與*redshift-dc-database-name*和*redshift-table-name*的連線。

組態：在函數選項中，您需使用 database 和 table_name 參數識別資料型錄資料表。您需使用 redshift_tmp_dir 識別 Amazon S3 臨時目錄。您還將提供*rs-role-name*使用additional_options參數中的aws_iam_role鍵。

```
 glueContext.create_dynamic_frame.from_catalog(
    database = "redshift-dc-database-name",
    table_name = "redshift-table-name",
    redshift_tmp_dir = args["temp-s3-dir"],
    additional_options = {"aws_iam_role": "arn:aws:iam::role-account-id:role/rs-
role-name"})
```

Connecting directly

其他先決條件:您將需要您的 Amazon Redshift 表的名稱 (*redshift-table-name*. 您將需要存放該資料表之 Amazon Redshift 叢集的 JDBC 連線資訊。您將提供*##*，*##*，*###*和*##*的連接信息。*redshift-database-name*

使用 Amazon Redshift 叢集時，您可以從 Amazon Redshift 主控台擷取連線資訊。使用 Amazon Redshift Serverless 時，請參閱 Amazon Redshift 文件中的 Connecting to Amazon Redshift Serverless。

組態：在函數選項中，您需使用 url、dbtable、user 和 password 識別連線參數。您需使用 redshift_tmp_dir 識別 Amazon S3 臨時目錄。您可以在使用 from_options 時透過 aws_iam_role 指定 IAM 角色。語法與透過資料型錄連線類似，但您可以將參數放入 connection_options 地圖中。

將密碼硬編碼為 AWS Glue 腳本是不好的做法。考慮將密碼存儲在中， AWS Secrets Manager 並使用適用於 Python 的 SDK (Boto3) 在腳本中檢索密碼。

```
my_conn_options = {
    "url": "jdbc:redshift://host:port/redshift-database-name",
    "dbtable": "redshift-table-name",
```

```
    "user": "username",
    "password": "password",
    "redshiftTmpDir": args["temp-s3-dir"],
    "aws_iam_role": "arn:aws:iam::account id:role/rs-role-name"
}

df = glueContext.create_dynamic_frame.from_options("redshift", my_conn_options)
```

範例：寫入 Amazon Redshift 資料表

您可以寫入 Amazon Redshift 叢集和 Amazon Redshift Serverless 環境。

先決條件：Amazon Redshift 叢集並按照上一節中的步驟進行操作，the section called “設定 Redshift”之後您應該擁有用於暫存目錄、臨時目錄、*## s3-dir* 和 IAM 角色 (在帳戶中) 的 Amazon S3 URI。*rs-role-namerole-account-id*您還需要一個 DynamicFrame，其內容要寫入資料庫。

Using the Data Catalog

其他先決條件：您想要寫入的 Amazon Redshift 叢集和資料表的資料型錄資料庫。如需有關資料型錄的詳細資訊，請參閱 資料探索與編目。您將使用識別您的連接*redshift-dc-database-name*和目標表*redshift-table-name*。

組態：在函數選項中，您需使用 database 參數識別資料型錄資料庫，然後為資料表提供 table_name。您需使用 redshift_tmp_dir 識別 Amazon S3 臨時目錄。您還將提供*rs-role-name*使用additional_options參數中的aws_iam_role鍵。

```
 glueContext.write_dynamic_frame.from_catalog(
    frame = input dynamic frame,
    database = "redshift-dc-database-name",
    table_name = "redshift-table-name",
    redshift_tmp_dir = args["temp-s3-dir"],
    additional_options = {"aws_iam_role": "arn:aws:iam::account-id:role/rs-role-
name"})
```

Connecting through a AWS Glue connection

您可以使用 write_dynamic_frame.from_options 方法直接連線到 Amazon Redshift。但是，您可以使用 from_jdbc_conf 方法參考存放在資料型錄連線中的連線詳細資訊，而不是直接

將連線詳細資訊插入指令碼。您無需為資料庫進行網路爬取或建立資料型錄資料表即可執行此操作。如需有關資料型錄連線的詳細資訊，請參閱 連線至資料。

其他先決條件：資料庫的資料型錄連線、您想要讀取的 Amazon Redshift 資料表

組態：您將識別與資料目錄連線*dc-connection-name*。您將使*redshift-table-name*用和識別您的 Amazon Redshift 資料庫和資料表。*redshift-database-name*您需提供包含 `catalog_connection` 的資料型錄連線資訊，以及包含 `dbtable` 和 `database` 的 Amazon Redshift 資訊。語法與透過資料型錄連線類似，但您可以將參數放入 `connection_options` 地圖中。

```
my_conn_options = {
    "dbtable": "redshift-table-name",
    "database": "redshift-database-name",
    "aws_iam_role": "arn:aws:iam::role-account-id:role/rs-role-name"
}

glueContext.write_dynamic_frame.from_jdbc_conf(
    frame = input dynamic frame,
    catalog_connection = "dc-connection-name",
    connection_options = my_conn_options,
    redshift_tmp_dir = args["temp-s3-dir"])
```

Amazon Redshift 連線選項參考

用於所有 AWS Glue JDBC 連線的基本連線選項，用來設定資訊`url`，例如，`user`且在所有 JDBC 類型之間都`password`是一致的。如需有關標準 JDBC 參數的詳細資訊，請參閱 the section called “JDBC 連線參數”。

Amazon Redshift 連線類型需要一些額外的連接選項：

- `"redshiftTmpDir"`：(必要) 從資料庫複製時，可用來暫存臨時資料的 Amazon S3 路徑。
- `"aws_iam_role"`：(選用) IAM 角色的 ARN。 AWS Glue 任務會將此角色傳遞給 Amazon Redshift 叢集，以授與完成任務指示所需的叢集許可。

AWS Glue 4.0+ 提供其他連接選項

您也可以透過 Glue 連線選項，傳遞新的 Amazon Redshift 連 AWS 接器的選項。如需支援的連接器選項完整清單，請參閱 Amazon Redshift integration for Apache Spark (Apache Spark 的 Amazon Redshift 整合) 中的 Spark SQL 參數部分。

為方便起見，我們在此重申某些新選項：

名稱	必要	預設	描述
autopushdown	否	TRUE	擷取和分析 SQL 操作的 Spark 邏輯計畫，以套用述詞和查詢下推。這些操作會轉換成 SQL 查詢，然後在 Amazon Redshift 中執行以提高效能。
autopushdown.s3_result_cache	否	FALSE	快取 SQL 查詢以卸載記憶體中 Amazon S3 路徑對應的資料，因此不需要在相同的 Spark 工作階段中再次執行相同的查詢。僅在啟用 `autopushdown` 時提供支援。
unload_s3_format	否	PARQUET	PARQUET – 以 Parquet 格式卸載查詢結果。 TEXT – 以管道分隔的文字格式卸載查詢結果。
sse_kms_key	否	N/A	在UNL0AD作業期間用於加密的 AWS SSE-

名稱	必要	預設	描述
			KMS 金鑰，而不是的預設加密。 AWS
extracopyoptions	否	N/A	載入資料時要附加至 Amazon Redshift COPY 命令的額外選項清單，例如 TRUNCATECOLUMNS 或 MAXERROR n (如需其他選項，請參閱 COPY：選用參數)。 請注意，由於這些選項會附加到 COPY 命令的結尾，因此只能使用在命令結尾上具有意義的選項。這應該涵蓋大多數可能的使用案例。
csvnullstring (實驗性)	否	NULL	使用 CSV tempformat 時為 Null 值寫入的字串值。這應該是不會出現在實際資料中的值。

這些新參數可以透過以下方式使用。

提升效能的全新選項

新的連接器引入了一些新的效能提升選項：

- autopushdown：預設為啟用。
- autopushdown.s3_result_cache：預設為停用。
- unload_s3_format：預設為 PARQUET。

如需有關使用這些選項的資訊，請參閱 Apache Spark 的 Amazon Redshift 整合。建議您在混合讀取和寫入操作時不要開啟 autopushdown.s3_result_cache，因為快取的結果可能包含過時的資訊。依預設，UNLOAD 命令的選項 unload_s3_format 會設定為 PARQUET，以提高效能並降低儲存成本。若要使用 UNLOAD 命令預設行為，請將選項重設為 TEXT。

新的讀取加密選項

預設情況下，從 Amazon Redshift 資料表讀取資料時，由 AWS Glue 使用之臨時資料夾中的資料會使用 SSE-S3 加密來加密。若要使用客戶管理金鑰 AWS Key Management Service (AWS KMS) 來加密您的資料，您可("sse_kms_key" # kmsKey)以設定 KSMKey 作為金鑰 ID 的來源 AWS KMS，而不是 3.0 AWS Glue 版("extraunloadoptions" # s"ENCRYPTED KMS_KEY_ID '$kmsKey'")中的舊版設定選項。

```
datasource0 = glueContext.create_dynamic_frame.from_catalog(
  database = "database-name",
  table_name = "table-name",
  redshift_tmp_dir = args["TempDir"],
  additional_options = {"sse_kms_key":"<KMS_KEY_ID>"},
  transformation_ctx = "datasource0"
)
```

支援 IAM 型 JDBC URL

新的連接器支援 IAM 型 JDBC URL，因此您不需要傳入使用者/密碼或機密。透過 IAM 型 JDBC URL，連接器會使用任務執行時間角色來存取 Amazon Redshift 資料來源。

步驟 1：將下列最小必要政策附加至您的 AWS Glue 任務執行時間角色。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "redshift:GetClusterCredentials",
            "Resource": [
                "arn:aws:redshift:<region>:<account>:dbgroup:<cluster name>/*",
                "arn:aws:redshift:*:<account>:dbuser:*/*",
                "arn:aws:redshift:<region>:<account>:dbname:<cluster name>/<database
 name>"
            ]
```

```
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "redshift:DescribeClusters",
            "Resource": "*"
        }
    ]
}
```

步驟 2：使用 IAM 型 JDBC URL (如下所示)。使用您要連線的 Amazon Redshift 使用者名稱指定新選項 DbUser。

```
conn_options = {
     // IAM-based JDBC URL
    "url": "jdbc:redshift:iam://<cluster name>:<region>/<database name>",
    "dbtable": dbtable,
    "redshiftTmpDir": redshiftTmpDir,
    "aws_iam_role": aws_iam_role,
    "DbUser": "<Redshift User name>" // required for IAM-based JDBC URL
    }

redshift_write = glueContext.write_dynamic_frame.from_options(
    frame=dyf,
    connection_type="redshift",
    connection_options=conn_options
)

redshift_read = glueContext.create_dynamic_frame.from_options(
    connection_type="redshift",
    connection_options=conn_options
)
```

> **Note**
>
> DynamicFrame 目前只支援 GlueContext.create_dynamic_frame.from_options 工作流程中具有 DbUser 的 IAM 型 JDBC URL。

從 AWS Glue 3.0 版本遷移至第 4.0 版

在 AWS Glue 4.0 中，ETL 任務可以存取新的 Amazon Redshift Spark 連接器和具有不同選項和組態的新 JDBC 驅動程式。新 Amazon Redshift 連接器和驅動程式在寫入時考量效能，並維持資料的交易一致性。這些產品會記錄在 Amazon Redshift 文件中。如需詳細資訊，請參閱：

- Apache Spark 的 Amazon Redshift 整合
- Amazon Redshift JDBC 驅動程式 2.1 版

資料表/欄名稱和識別符限制

新的 Amazon Redshift Spark 連接器和驅動程式對 Redshift 資料表名稱的要求更加嚴格。如需詳細資訊，請參閱 Names and identifiers (名稱和識別符) 以定義您的 Amazon Redshift 資料表名稱。任務書籤工作流程可能無法使用不符合規則和具有特定字元 (例如空格) 的資料表名稱。

如果舊版資料表名稱不符合名稱和識別符規則，並且看到書籤問題 (任務會重新處理舊版 Amazon Redshift 資料表資料)，建議您重新命名資料表名稱。如需詳細資訊，請參閱 ALTER TABLE 範例。

Dataframe 中的預設 tempformat 變更

AWS Glue 3.0 版 Spark 連接器在寫入 Amazon Redshift 時將 `tempformat` 預設為 CSV。為了保持一致，在 AWS Glue 3.0 版本中， `DynamicFrame` 仍然將 `tempformat` 預設為使用 `CSV`。如果之前已將 Spark Dataframe API 直接與 Amazon Redshift Spark 連接器搭配使用，您可以在 `DataframeReader/Writer` 選項中將 `tempformat` 明確設定為 CSV。否則，`tempformat` 會在新 Spark 連接器中預設為 `AVRO`。

行為變更：將 Amazon Redshift 資料類型 REAL 映射到 Spark 資料類型 FLOAT 而不是 DOUBLE

在 AWS Glue 3.0 版本中，Amazon Redshift `REAL` 會轉換為 Spark `DOUBLE` 類型。新的 Amazon Redshift Spark 連接器已經更新行為，以便 Amazon Redshift `REAL` 類型轉換為 Spark `FLOAT` 類型，並轉換回來。如果您仍有希望將 Amazon Redshift `REAL` 類型映射至 Spark `DOUBLE` 類型的舊版使用案例，則可以使用下列因應措施：

- 對於 `DynamicFrame`，將 `Float` 類型映射至具有 `DynamicFrame.ApplyMapping` 的 `Double` 類型。對於 `Dataframe`，您需要使用 `cast`。

程式碼範例：

```
dyf_cast = dyf.apply_mapping([('a', 'long', 'a', 'long'), ('b', 'float', 'b',
 'double')])
```

Kafka 連線

指定連接到 Kafka 叢集或 Amazon Managed Streaming for Apache Kafka 叢集的連線。

您可以使用存儲在「數據目錄」表中的信息或提供信息直接訪問數據流來讀取和寫入卡夫卡數據流。您可以從卡夫卡讀取信息到火花 DataFrame，然後將其轉換為 Glue。 AWS DynamicFrame您可以以 JSON 格式寫信給 DynamicFrames 卡夫卡。如果您直接存取資料串流，則請使用這些選項來提供如何存取資料串流的相關資訊。

如果您使用getCatalogSource或create_data_frame_from_catalog使用卡夫卡流源中的記錄，getCatalogSink或者write_dynamic_frame_from_catalog將記錄寫入卡夫卡，並且該作業具有數據目錄數據庫和表名信息，則可以使用該信息來獲取從卡夫卡流源讀取一些基本參數。如果使用getSource、、getCatalogSink、或getSourceWithFormat，createDataFrameFromOptions或 getSinkWithFormat create_data_frame_from_optionswrite_dynamic_frame_from_catalog，則必須使用此處描述的連接選項來指定這些基本參數。

您可以使用類中的指定方法下面的參數指定卡夫卡的連接選項。GlueContext

- Scala
  - connectionOptions：與 getSource、createDataFrameFromOptions、getSink 搭配使用
  - additionalOptions：與 getCatalogSource、getCatalogSink 搭配使用。
  - options：與 getSourceWithFormat、getSinkWithFormat 搭配使用。
- Python
  - connection_options：與 create_data_frame_from_options、write_dynamic_frame_from_options 搭配使用。
  - additional_options：與 create_data_frame_from_catalog、write_dynamic_frame_from_catalog 搭配使用。
  - options：與 getSource、getSink 搭配使用。

如需有關串流 ETL 任務的注意事項和限制，請參閱 the section called “串流 ETL 注意事項和限制”。

設定 Kafka

連接到通過互聯網可用的卡夫卡流沒有 AWS 先決條件。

您可以建立 AWS Glue Kafka 連線來管理您的連線認證。如需詳細資訊，請參閱 the section called “為 Kafka 資料串流建立連線”。在您的 AWS Glue 工作組態中，提供 *ConnectionName* 做為其他網路連線，然後在您的方法呼叫中，提供####給參數。connectionName

在某些情況下，您需要設定其他先決條件：

- 如果搭配 IAM 身分驗證使用 Amazon Managed Streaming for Apache Kafka，您會需要適當的 IAM 組態。
- 如果搭配 Amazon VPC 使用 Amazon Managed Streaming for Apache Kafka，您會需要適當的 Amazon VPC 組態。您必須建立可提供 Amazon VPC 連線資訊的 AWS Glue 連線。您需要工作組態，才能將 AWS Glue 連線納入為其他網路連線。

如需有關串流 ETL 任務先決條件的詳細資訊，請參閱 the section called “串流 ETL 任務”。

範例：從 Kafka 串流讀取

搭配 the section called “forEachBatch” 使用。

Kafka 串流來源範例：

```
kafka_options =
    { "connectionName": "ConfluentKafka",
      "topicName": "kafka-auth-topic",
      "startingOffsets": "earliest",
      "inferSchema": "true",
      "classification": "json"
    }
data_frame_datasource0 =
 glueContext.create_data_frame.from_options(connection_type="kafka",
 connection_options=kafka_options)
```

範例：寫入卡夫卡串流

寫信給卡夫卡的例子：

使用該getSink方法的示例：

```
data_frame_datasource0 =
glueContext.getSink(
 connectionType="kafka",
 connectionOptions={
  JsonOptions("""{
   "connectionName": "ConfluentKafka",
   "classification": "json",
   "topic": "kafka-auth-topic",
   "typeOfData": "kafka"}
  """)},
 transformationContext="dataframe_ApacheKafka_node1711729173428")
 .getDataFrame()
```

使用該write_dynamic_frame.from_options方法的示例：

```
kafka_options =
    { "connectionName": "ConfluentKafka",
      "topicName": "kafka-auth-topic",
      "classification": "json"
    }
data_frame_datasource0 =
 glueContext.write_dynamic_frame.from_options(connection_type="kafka",
 connection_options=kafka_options)
```

Kafka 連線選項參考

閱讀時，請使用以下連接選項"connectionType": "kafka"：

- "bootstrap.servers" (必要) 自舉伺服器 URL 的清單，例如 b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094。此選項必須在 API 呼叫中指定，或在 Data Catalog 的資料表中繼資料中定義。
- "security.protocol"(必要) 用來與代理程式通訊的協定。可能的值為 "SSL" 或 "PLAINTEXT"。
- "topicName" (必要) 要訂閱的主題清單 (以逗號分隔)。您必須指定 "topicName"、"assign" 或 "subscribePattern" 其中一個。
- "assign"：(必要) JSON 字串，指定要消耗的特定 TopicPartitions。您必須指定 "topicName"、"assign" 或 "subscribePattern" 其中一個。

  範例：'{"topicA":[0,1],"topicB":[2,4]}'

- "subscribePattern"：(必要) 識別要訂閱的主題清單的 Java regex 字串。您必須指定 "topicName"、"assign" 或 "subscribePattern" 其中一個。

  範例：'topic.*'
- "classification" (必要) 記錄中資料使用的檔案格式。除非透過資料型錄提供，否則為必要。
- "delimiter" (選用) 當 classification 為 CSV 時使用的值分隔符號。預設值為 ","。
- "startingOffsets"：(選用) 要從中讀取資料的 Kafka 主題的起始位置。可能的值為 "earliest" 或 "latest"。預設值為 "latest"。
- "startingTimestamp": (選用，僅適用於 AWS Glue 4.0 版或更新版本) Kafka 主題中要讀取資料的記錄時間戳記。可能的值是 yyyy-mm-ddTHH:MM:SSZ 模式中 UTC 格式的時間戳記字串 (其中 Z 代表以 +/- 表示的 UTC 時區偏移。例如："2023-04-04T08:00:00-04:00")。

  注意： AWS Glue 串流指令集的「連線選項」清單中只能有一個「開始偏移」或「開始時間戳記」，包括這兩個屬性會導致工作失敗。
- "endingOffsets"：(選用) 批次查詢結束時的終點。可能值為 "latest" 或指定每個 TopicPartition 結束偏移的 JSON 字串。

  對於 JSON 字串，格式為 {"topicA":{"0":23,"1":-1},"topicB":{"0":-1}}。值 -1 作為偏移代表 "latest"。
- "pollTimeoutMs"：(選用) 在 Spark 任務執行器中從 Kafka 輪詢資料的逾時 (以毫秒為單位)。預設值為 512。
- "numRetries"：(選用) 擷取 Kafka 位移失敗之前，要重試的次數。預設值為 3。
- "retryIntervalMs"：(選用) 重試擷取 Kafka 偏移量之前等待的時間 (毫秒)。預設值為 10。
- "maxOffsetsPerTrigger"：(選用) 每個觸發間隔所處理之偏移數目上限的速率限制。指定的偏移總數會按比例跨 topicPartitions 或不同磁碟區而分割。預設值為 null，這表示消費者讀取所有偏移，直到已知的最新偏移。
- "minPartitions"：(選用) 從 Kafka 讀取所需的分割區最小數量。預設值為 null，這表示 Spark 分割區的數量等於 Kafka 分割區的數量。
- "includeHeaders"：(選用) 是否包含 Kafka 標頭。當選項設定為「true」時，資料輸出將包含一個名為「glue_streaming_kafka_headers」的額外欄，其類型為 Array[Struct(key: String, value: String)]。預設值為 "false"。此選項能在 AWS Glue 3.0 版或更新版中使用。
- "schema"：(當 InferSchema 設定為 false 時為必要) 用於處理承載的架構。如果分類為 avro，提供的架構必須採用 Avro 架構格式。如果分類不是 avro，提供的架構必須採用 DDL 架構格式。

  以下是架構範例。

Example in DDL schema format

```
'column1' INT, 'column2' STRING , 'column3' FLOAT
```

Example in Avro schema format

```
{
"type":"array",
"items":
{
"type":"record",
"name":"test",
"fields":
[
  {
    "name":"_id",
    "type":"string"
  },
  {
    "name":"index",
    "type":
    [
      "int",
      "string",
      "float"
    ]
  }
]
}
}
```

- "inferSchema"：(選用) 預設值為 'false'。如果設為 'true'，將在執行時間時從 foreachbatch 承載偵測架構。
- "avroSchema"：(已棄用) 使用 Avro 格式時，用於指定 Avro 資料架構的參數。此參數現已棄用。使用 schema 參數。
- "addRecordTimestamp"：(選用) 當此選項設定為 'true' 時，資料輸出將包含一個名為 "__src_timestamp" 的額外資料欄，其指示主題收到相應記錄的時間。預設值為 'false'。在 AWS Glue 4.0 版或更新版中支援此選項。

- "emitConsumerLagMetrics": (選擇性) 當選項設為 'true' 時，對於每個批次，它會發出主題接收到的最舊記錄到達時間之間的持續時間的AWS Glue指標。 CloudWatch該指標的名稱是「膠合. 驅動程序. maxConsumerLagInMs」. 預設值為 'false'。在 AWS Glue 4.0 版或更新版中支援此選項。

寫入時，請使用以下連接選項"connectionType": "kafka"：

- "connectionName"（必要）用於連接到卡夫卡集群的 AWS Glue 連接名稱（類似於卡夫卡源）。
- "topic"(必要) 如果主題欄存在，則除非設定了主題組態選項，否則在將指定資料列寫入 Kafka 時，會使用其值作為主題。也就是說，組topic態選項會覆寫主題欄。
- "partition"(選擇性) 如果指定了有效的分割區編號，partition將在傳送記錄時使用。

  如果沒有指定分區，但存key在一個分區，將使用密鑰的散列來選擇一個分區。

  如果key既不存在partition也不存在，則當至少為分區產生 batch .size 字節時，將根據粘性分區這些更改選擇分區。
- "key"(選擇性) 如partition果為 null，則用於分割。
- "classification"(選擇性) 記錄中資料使用的檔案格式。我們只支持 JSON，CSV 和阿夫羅。

  使用 Avro 格式，我們可以提供自定義的 AvroSchema 進行序列化，但請注意，這也需要在源代碼上提供反序列化。否則，默認情況下它使用 Apache AvroSchema 進行序列化。

此外，您可以根據需要通過更新卡夫卡生產者配置參數微調卡夫卡水槽。請注意，連接選項上沒有允許列出，所有鍵值對都保留在接收器上。

但是，有一個小的拒絕列表的選項不會生效。如需詳細資訊，請參閱 Kafka 特定組態。

Azure Cosmos DB 連線

您可以使用 AWS Glue for Spark，從使用 AWS Glue 4.0 及更新版本之 NoSQL API 的 Azure Cosmos DB 讀取和寫入現有容器。您可以使用 SQL 查詢定義要從 Azure Cosmos DB 讀取的內容。您可以使用透過 AWS Glue 連線儲存於 AWS Secrets Manager 的 Azure Cosmos DB 索引鍵連線至 Azure Cosmos DB。

如需有關 Azure Cosmos DB for NoSQL 的詳細資訊，請參閱 Azure 文件。

設定 Azure Cosmos DB 連線

若要從 AWS Glue 連線至 Azure Cosmos DB，您將需要在 AWS Secrets Manager 密碼中建立並儲存 Azure Cosmos DB 索引鍵，然後將該密碼與 Azure Cosmos DB AWS Glue 連線建立關聯。

先決條件：

- 在 Azure 中，您將需要識別或產生 Azure Cosmos DB 索引鍵 `cosmosKey`，以供 AWS Glue 使用。如需詳細資訊，請參閱《Azure 文件》中的安全存取 Azure Cosmos DB 中的資料。

設定連至 Azure Cosmos DB 的連線：

1. 在 AWS Secrets Manager 中，使用 Azure Cosmos DB 索引鍵建立密碼。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *`secretName`*，以便進行下一個步驟。
   - 在選取鍵/值組時，請使用 *`cosmosKey`* 值來建立 `spark.cosmos.accountKey` 金鑰對。
2. 在 AWS Glue 主控台中，依照 the section called “新增 AWS Glue 連線” 中的步驟建立連線。建立連線之後，請保留連線名稱 *`connectionName`*，以便未來在 AWS Glue 中使用。
   - 選取連線類型時，請選取 Azure Cosmos DB。
   - 選取 AWS 機密時，請提供 *`secretName`*。

建立 AWS Glue Azure Cosmos DB 連線之後，您將需要執行下列步驟，才能執行 AWS Glue 工作：

- 授予與您 AWS Glue 任務許可相關聯的 IAM 角色，以讀取 *`secretName`*。
- 在您的 AWS Glue 任務組態中，提供 *`connectionName`* 作為其他網路連線。

從 Azure Cosmos DB for NoSQL 容器讀取

先決條件：

- 您想要讀取的 Azure Cosmos DB for NoSQL 容器。您將需要容器的識別資訊。

  An Azure Cosmos NoSQL 容器由資料庫和容器識別。連線至 Azure Cosmos for NoSQL API 時，您必須提供資料庫名稱 *`cosmosDBName`* 和容器名稱 *`cosmosContainerName`*。
- 完成設定的 AWS Glue Azure Cosmos DB 連線，可提供驗證和網路位置資訊。若要取得此功能，請完成上一個程序設定連至 Azure Cosmos DB 的連線中的步驟。您將會需要 AWS Glue 連線的名稱，*`connectionName`*。

例如：

```
azurecosmos_read = glueContext.create_dynamic_frame.from_options(
    connection_type="azurecosmos",
    connection_options={
    "connectionName": connectionName,
    "spark.cosmos.database": cosmosDBName,
    "spark.cosmos.container": cosmosContainerName,
    }
)
```

您也可提供 SELECT SQL 查詢，以篩選傳回 DynamicFrame 的結果。您將需要設定 `query`。

例如：

```
azurecosmos_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="azurecosmos",
    connection_options={
        "connectionName": "connectionName",
        "spark.cosmos.database": cosmosDBName,
        "spark.cosmos.container": cosmosContainerName,
        "spark.cosmos.read.customQuery": "query"
    }
)
```

寫入 Azure Cosmos DB for NoSQL 容器

此範例會從現有的 DynamicFrame *dynamicFrame* 將資訊寫入 Azure Cosmos DB。如果容器已具有資訊，則 AWS Glue 會從 DynamicFrame 附加資料。如果容器中的資訊與寫入的資訊具有不同的結構描述，就會發生錯誤。

先決條件：

- 您想要寫入的 Azure Cosmos DB 資料表。您將需要容器的識別資訊。您必須先建立容器，再呼叫連線方法。

  An Azure Cosmos NoSQL 容器由資料庫和容器識別。連線至 Azure Cosmos for NoSQL API 時，您必須提供資料庫名稱 *cosmosDBName* 和容器名稱 *cosmosContainerName*。
- 完成設定的 AWS Glue Azure Cosmos DB 連線，可提供驗證和網路位置資訊。若要取得此功能，請完成上一個程序設定連至 Azure Cosmos DB 的連線中的步驟。您將會需要 AWS Glue 連線的名稱，*connectionName*。

例如：

```
azurecosmos_write = glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="azurecosmos",
    connection_options={
    "connectionName": connectionName,
    "spark.cosmos.database": cosmosDBName,
    "spark.cosmos.container": cosmosContainerName
)
```

Azure Cosmos DB 連線選項參考

- `connectionName` – 必要。用於讀取/寫入。完成設定之 AWS Glue Azure Cosmos DB 連線的名稱，可向連線方法提供驗證和網路位置資訊。
- `spark.cosmos.database` – 必要。用於讀取/寫入。有效值:資料庫名稱。Azure Cosmos DB for NoSQL 資料庫名稱。
- `spark.cosmos.container` – 必要。用於讀取/寫入。有效值:容器名稱。Azure Cosmos DB for NoSQL 容器名稱。
- `spark.cosmos.read.customQuery` – 用於讀取。有效值：SELECT SQL 查詢。自訂查詢以選取要讀取的文件。

Azure SQL 連線

您可以使用 AWS Glue for Spark 從 AWS Glue 4.0 及更新版本中的 Azure SQL 受管執行個體讀取和寫入資料表。您可以使用 SQL 查詢定義要從 Azure SQL 讀取的內容。您可以使用透過 AWS Glue 連線儲存於 AWS Secrets Manager 的使用者和密碼憑證來連線至 Azure SQL。

如需有關 Azure SQL 的詳細資訊，請參閱 Azure SQL 文件。

設定 Azure SQL 連線

若要從 AWS Glue 連線至 Azure SQL，您將需要在 AWS Secrets Manager 密碼中建立並儲存 Azure SQL 憑證，然後將該密碼與 Azure SQL AWS Glue 連線建立關聯。

設定連至 Azure SQL 的連線：

1. 在 AWS Secrets Manager 中，使用您的 Azure SQL 憑證建立密碼。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *secretName*，以便進行下一個步驟。

- 在選取鍵/值組時，請使用 *azuresqlUsername* 值來建立 user 金鑰對。
- 在選取鍵/值組時，請使用 *azuresqlPassword* 值來建立 password 金鑰對。

2. 在 AWS Glue 主控台中，依照 the section called “新增 AWS Glue 連線” 中的步驟建立連線。建立連線之後，請保留連線名稱 *connectionName*，以便未來在 AWS Glue 中使用。

   - 選取連線類型時，請選取 Azure SQL。
   - 提供 Azure SQL URL時，請提供 JDBC 端點 URL。

     此 URL 必須採用下列格式：jdbc:sqlserver://*databaseServerName*:*databasePort*;databaseName=*azuresqlDB*

     AWS Glue 需要具有下列 URL 屬性：

     - databaseName：要連線之 Azure SQL 的預設資料庫。

     如需有關 Azure SQL 受控執行個體之 JDBC URL 的詳細資訊，請參閱 Microsoft 文件。
   - 選取 AWS 機密時，請提供 *secretName*。

建立 AWS Glue Azure SQL 連線之後，您將需要執行下列步驟，才能執行 AWS Glue 工作：

- 授予與您 AWS Glue 任務許可相關聯的 IAM 角色，以讀取 *secretName*。
- 在您的 AWS Glue 任務組態中，提供 *connectionName* 作為其他網路連線。

從 Azure SQL 資料表讀取

先決條件：

- 您想要讀取的 Azure SQL 資料表。您將需要資料表的識別資訊 *databaseName* 和 *tableIdentifier*。

  Azure SQL 資料表由資料庫、結構描述及資料表名稱識別。連線至 Azure SQL 時，您必須提供資料庫名稱和資料表名稱。如果結構描述不是預設的 "public"，您也必須提供結構描述。資料庫會透過 *connectionName* 中的 URL 屬性提供，而結構描述和資料表名稱會透過 dbtable 提供。
- 完成設定的 AWS Glue Azure SQL 連線，可提供驗證資訊。完成上一個程序設定連至 Azure SQL 的連線的步驟，以設定驗證資訊。您將會需要 AWS Glue 連線的名稱，*connectionName*。

例如：

```
azuresql_read_table = glueContext.create_dynamic_frame.from_options(
    connection_type="azuresql",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableIdentifier"
    }
)
```

您也可提供 SELECT SQL 查詢，以篩選傳回 DynamicFrame 的結果。您將需要設定 `query`。

例如：

```
azuresql_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="azuresql",
    connection_options={
        "connectionName": "connectionName",
        "query": "query"
    }
)
```

寫入 Azure SQL 資料表

此範例會從現有的 DynamicFrame *dynamicFrame* 將資訊寫入 Azure SQL。如果資料表已具有資訊，則 AWS Glue 會從 DynamicFrame 附加資料。

先決條件：

- 您想要寫入的 Azure SQL 資料表。您將需要資料表的識別資訊 *databaseName* 和 *tableIdentifier*。

  Azure SQL 資料表由資料庫、結構描述及資料表名稱識別。連線至 Azure SQL 時，您必須提供資料庫名稱和資料表名稱。如果結構描述不是預設的 "public"，您也必須提供結構描述。資料庫會透過 *connectionName* 中的 URL 屬性提供，而結構描述和資料表名稱會透過 `dbtable` 提供。
- Azure SQL 驗證資訊。完成上一個程序設定連至 Azure SQL 的連線的步驟，以設定驗證資訊。您將會需要 AWS Glue 連線的名稱，*connectionName*。

例如：

```
azuresql_write = glueContext.write_dynamic_frame.from_options(
    connection_type="azuresql",
```

```
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableIdentifier"
    }
)
```

Azure SQL 連線選項參考

- connectionName – 必要。用於讀取/寫入。完成設定之 AWS Glue Azure SQL 連線的名稱，可向連線方法提供驗證資訊。
- databaseName：用於讀取/寫入。有效值:Azure SQL 資料庫名稱。要連線之 Azure SQL 的資料庫名稱。
- dbtable：除非已提供 query，否則為寫入和讀取的必要項目。用於讀取/寫入。有效值：Azure SQL 資料表的名稱，或句點分隔的結構描述/資料表名稱組合。用於指定識別要連線之資料表的資料表和結構描述。預設結構描述為 "public"。如果您的資料表位於非預設的結構描述中，請在表單 *schemaName*.*tableName* 中提供此資訊。
- query – 用於讀取。定義從 Azure SQL 讀取時應擷取之內容的 Transact-SQL SELECT 查詢。如需詳細資訊，請參閱 Microsoft 文件。

BigQuery 連線

您可以使用 AWS Glue for Spark 從 AWS Glue 4.0 及更新版本中的 Google BigQuery 讀取和寫入資料表。您可以使用 Google SQL 查詢，從 BigQuery 讀取。您可以使用透過 AWS Glue 連線儲存在 AWS Secrets Manager 中的憑證來連線至 BigQuery。

如需有關 Google BigQuery 的詳細資訊，請參閱 Google Cloud BigQuery 網站。

設定 BigQuery 連線

若要從 AWS Glue 連線至 Google BigQuery，必須在 AWS Secrets Manager 秘密中建立並儲存您的 Google Cloud Platform 憑證，然後將該秘密與 Google BigQuery AWS Glue 連線建立關聯。

設定連至 BigQuery 的連線：

1. 在 Google Cloud Platform 中建立並識別相關資源：
   - 建立或識別 GCP 專案，其中應包含您要連線之 BigQuery 資料表。
   - 啟用 BigQuery API。如需詳細資訊，請參閱 Use the BigQuery Storage Read API to read table data。

2. 在 Google Cloud Platform 中建立和匯出服務帳戶憑證：

   您可以使用 BigQuery 憑證精靈來加速此步驟：Create credentials。

   若要在 GCP 中建立服務帳戶，請依照 Create service accounts 中提供的教學課程進行操作。

   - 在選取專案時，請選取包含您 BigQuery 資料表的專案。
   - 在為您的服務帳戶選取 GCP IAM 角色時，請新增或建立角色，其會授予供執行 BigQuery 任務的適當許可，以讀取、寫入或建立 BigQuery 資料表。

   若要為您的服務帳戶建立憑證，請依照 Create a service account key 中提供的教學課程進行操作。

   - 在選取金鑰類型時，請選取 JSON。

   您現在應已下載 JSON 檔案，該檔案中包含您服務帳戶的憑證。其看起來與下列類似：

   ```
   {
     "type": "service_account",
     "project_id": "******",
     "private_key_id": "******",
     "private_key": "******",
     "client_email": "******",
     "client_id": "******",
     "auth_uri": "https://accounts.google.com/o/oauth2/auth",
     "token_uri": "https://oauth2.googleapis.com/token",
     "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
     "client_x509_cert_url": "******",
     "universe_domain": "googleapis.com"
   }
   ```

3. 對您下載的憑證檔案進行 base64 編碼。在 AWS CloudShell 工作階段或類似工作階段中，您可以透過執行 `cat` *`credentialsFile.json`* `| base64 -w 0` 來從命令列進行此動作。保留此命令的輸出，*`credentialString`*。

4. 在 AWS Secrets Manager 中使用您的 Google Cloud Platform 憑證建立秘密。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *`secretName`*，以便進行下一個步驟。

   - 在選取鍵值對時，請使用 *`credentialString`* 值來建立 `credentials` 鍵對。

5. 在 AWS Glue Data Catalog 中，依照 the section called “新增 AWS Glue 連線” 中的步驟建立連線。建立連線之後，請保留連線名稱 *connectionName*，以便進行下一個步驟。
   - 選取連線類型時，請選取 Google BigQuery。
   - 選取 AWS 機密時，請提供 *secretName*。
6. 授予與您 AWS Glue 任務許可相關聯的 IAM 角色，以讀取 *secretName*。
7. 在您的 AWS Glue 任務組態中，提供 *connectionName* 作為其他網路連線。

從 BigQuery 資料表讀取

先決條件：

- 您要從中讀取的 BigQuery 資料表。您將會需要 BigQuery 資料表和資料集名稱，形式為 `[dataset].[table]`。我們稱此為 *tableName*。
- BigQuery 資料表的計費專案。您將會需要專案的名稱，*parentProject*。若無計費父系專案，請使用包含資料表的專案。
- BigQuery 身分驗證資訊。請完成使用 AWS Glue 來管理您的連線憑證相關步驟，以設定您的驗證資訊。您將會需要 AWS Glue 連線的名稱，*connectionName*。

例如：

```
bigquery_read = glueContext.create_dynamic_frame.from_options(
    connection_type="bigquery",
    connection_options={
        "connectionName": "connectionName",
        "parentProject": "parentProject",
        "sourceType": "table",
        "table": "tableName",
    }
```

您也可提供查詢，以篩選傳回至 DynamicFrame 的結果。您將需要設定 `query`、`sourceType`、`viewsEnabled` 和 `materializationDataset`。

例如：

其他先決條件：

您必須建立或識別 BigQuery 資料集 *materializationDataset*，BigQuery 才能為您的查詢寫入具體化視觀表。

您將需授予適當的 GCP IAM 許可給您的服務帳戶，以在 *materializationDataset* 中建立資料表。

```
glueContext.create_dynamic_frame.from_options(
            connection_type="bigquery",
            connection_options={
                "connectionName": "connectionName",
                "materializationDataset": materializationDataset,
                "parentProject": "parentProject",
                "viewsEnabled": "true",
                "sourceType": "query",
                "query": "select * from bqtest.test"
            }
        )
```

寫入 BigQuery 資料表

此範例會直接寫入 BigQuery 服務。此外，BigQuery 也支援「間接」寫入方法。如需有關設定間接寫入的詳細資訊，請參閱the section called “透過 Google BigQuery 使用間接寫入”。

先決條件：

- 您要寫入的 BigQuery 資料表。您將會需要 BigQuery 資料表和資料集名稱，形式為 `[dataset].[table]`。您也可以提供自動建立的新資料表名稱。我們稱此為 *tableName*。
- BigQuery 資料表的計費專案。您將會需要專案的名稱，*parentProject*。若無計費父系專案，請使用包含資料表的專案。
- BigQuery 身分驗證資訊。請完成使用 AWS Glue 來管理您的連線憑證相關步驟，以設定您的驗證資訊。您將會需要 AWS Glue 連線的名稱，*connectionName*。

例如：

```
bigquery_write = glueContext.write_dynamic_frame.from_options(
    frame=frameToWrite,
    connection_type="bigquery",
    connection_options={
        "connectionName": "connectionName",
        "parentProject": "parentProject",
        "writeMethod": "direct",
```

```
        "table": "tableName",
    }
)
```

BigQuery 連線選項參考

- `project` – 預設：Google Cloud 服務帳戶預設值。用於讀取/寫入。與您資料表相關聯的 Google Cloud 專案名稱。
- `table`：(必要) 用於讀取/寫入。BigQuery 資料表的名稱，格式為 `[[project:]dataset.]`。
- `dataset` – 未透過 `table` 選項定義時為必要。用於讀取/寫入。包含您 BigQuery 資料表的資料集名稱。
- `parentProject` – 預設：Google Cloud 服務帳戶預設值。用於讀取/寫入。與用於計費之 `project` 相關聯的 Google Cloud 專案名稱。
- `sourceType` – 用於讀取。讀取時需要。有效值：`table`，`query` 會通知 AWS Glue 您將藉由資料表讀取還是藉由查詢讀取。
- `materializationDataset` – 用於讀取。有效值：字串。用來儲存檢視之實體化的 BigQuery 資料集名稱。
- `viewsEnabled` – 用於讀取。預設：false。有效值：true、false。設定 BigQuery 是否要使用檢視。
- `query` – 用於讀取。`viewsEnabled` 為 true 時使用。GoogleSQL DQL 查詢。
- `temporaryGcsBucket` – 用於寫入。當 `writeMethod` 設定為預設 (`indirect`) 時，此為必要項。在寫入 BigQuery 時，用來儲存您資料之中繼格式的 Google Cloud Storage 儲存貯體名稱。
- `writeMethod`：預設：`indirect`。有效值：`direct`、`indirect`。用於寫入。指定用來寫入資料的方法。
  - 若設定為 `direct`，您的連接器將使用 BigQuery Storage Write API 來寫入。
  - 若設定為 `indirect`，您的連接器會寫入 Google Cloud Storage，然後使用載入操作將其傳輸至 BigQuery。您的 Google Cloud 服務帳戶需要適當的 GCS 許可。

透過 Google BigQuery 使用間接寫入

此範例會使用間接寫入，即將資料寫入 Google Cloud Storage，然後將資料複製至 Google BigQuery。

先決條件：

您將需要一個臨時的 Google Cloud Storage 儲存貯體，*`temporaryBucket`*。

AWS Glue GCP 服務帳戶的 GCP IAM 角色需要適當的 GCS 許可，才能存取 *temporaryBucket*。

其他組態：

使用 BigQuery 設定間接寫入：

1. 評估 the section called “設定 BigQuery” 並找出或重新下載您的 GCP 憑證 JSON 檔案。識別 *secretName*，這是您任務中使用之 Google BigQuery AWS Glue 連線的 AWS Secrets Manager 秘密。
2. 將您的憑證 JSON 檔案上傳至適當安全的 Amazon S3 位置。保留前往該檔案的路徑，*s3secretpath*，以供後續步驟使用。
3. 編輯 *secretName*，新增 `spark.hadoop.google.cloud.auth.service.account.json.keyfile` 金鑰。將值設定為 *s3secretpath*。
4. 授予您的 AWS Glue 任務 Amazon S3 IAM 許可，以存取 *s3secretpath*。

您現在可將暫時 GCS 儲存貯體位置提供給寫入方法。由於過去 `indirect` 為預設，您無需提供 `writeMethod`。

```
bigquery_write = glueContext.write_dynamic_frame.from_options(
    frame=frameToWrite,
    connection_type="bigquery",
    connection_options={
        "connectionName": "connectionName",
        "parentProject": "parentProject",
        "temporaryGcsBucket": "temporaryBucket",
        "table": "tableName",
    }
)
```

JDBC 連線

某些 (通常是關聯式) 資料庫類型透過 JDBC 標準支援連線。如需有關 JDBC 的詳細資訊，請參閱 Java JDBC API 文件。AWSGlue 以原生方式支援透過其 JDBC 連接器連線到某些資料庫，而 JDBC 程式庫在 AWS Glue Spark 任務中提供。使用 AWS Glue 程式庫連線到這些資料庫類型時，您可以存取一組標準選項。

JDBC connectionType 的值包括以下項目：

- `"connectionType": "sqlserver"`：指定 Microsoft SQL Server 資料庫的連線。
- `"connectionType": "mysql"`：指定 MySQL 資料庫的連線。
- `"connectionType": "oracle"`：指定 Oracle 資料庫的連線。
- `"connectionType": "postgresql"`：指定 PostgreSQL 資料庫的連線。
- `"connectionType": "redshift"`：指定 Amazon Redshift 資料庫的連線。如需更多詳細資訊，請參閱 the section called “Redshift 連線”。

下表列出 AWS Glue 支援的 JDBC 驅動程式版本。

產品	適用於 Glue 4.0 的 JDBC 驅動程式版本	適用於 Glue 3.0 的 JDBC 驅動程式版本	適用於 Glue 0.9、1.0、2.0 的 JDBC 驅動程式版本
Microsoft SQL Server	9.4.0	7.x	6.x
MySQL	8.0.23	8.0.23	5.1
Oracle Database	21.7	21.1	11.2
PostgreSQL	42.3.6	42.2.18	42.1.x
MongoDB	4.7.2	4.0.0	2.0.0
Amazon Redshift *	redshift-jdbc42-2.1.0.16	redshift-jdbc41-1.2.12.1017	redshift-jdbc41-1.2.12.1017

* 對於 Amazon Redshift 連線類型，JDBC 連線的連線選項中包含的所有其他選項名稱/值對 (包含格式化選項)，都會直接傳遞至基礎 SparkSQL DataSource。在 AWS Glue 4.0 和更新版本的 AWS Glue with Spark 任務中，Amazon Redshift 的 AWS Glue 原生連接器會使用 Apache Spark 的 Amazon Redshift 整合。如需詳細資訊，請參閱 Apache Spark 的 Amazon Redshift 整合。在舊版中，請參閱 Spark 的 Amazon Redshift 資料來源。

若要將 Amazon VPC 設定為使用 JDBC 連線到 Amazon RDS 資料存放區，請參閱 the section called “設置 Amazon VPC 以連接到 Amazon RDS 數據存儲區”。

> **Note**
>
> AWS Glue 任務只會在執行期間與一個子網路關聯。這可能會影響您透過相同的任務連線至多個資料來源的能力。此行為不限於 JDBC 來源。

主題

JDBC 連線選項參考

如果您已經定義 JDBC AWS Glue 連線，則可重複使用其中定義的組態屬性，例如：URL、使用者和密碼；因此您不必在程式碼中將其指定為連線選項。這項功能僅能在 AWS Glue 3.0 及更新版本中使用。若要這樣做，請使用下列連線屬性：

- `"useConnectionProperties"`：將其設定為 "true"，以指示您想要使用連線中的組態。
- `"connectionName"`：輸入要從中擷取組態的連線名稱，連線必須在與任務相同的區域中定義。

將下列連線選項與 JDBC 連線搭配使用：

- `"url"`：(必要) 資料庫的 JDBC URL。
- `"dbtable"`：(必要) 要讀取的資料庫資料表。若是支援資料庫內結構描述的 JDBC 資料存放區，請指定 `schema.table-name`。如果未提供結構描述，則會使用預設的 "public" 結構描述。
- `"user"`：(必要) 連線時所要使用的使用者名稱。
- `"password"`：(必要) 連線時所要使用的密碼。
- (選用) 下列選項可讓您提供自訂 JDBC 驅動程式。如果您必須使用 AWS Glue 未原生支援的驅動程式，請使用這些選項。

  ETL 任務可以對資料來源和目標使用不同的 JDBC 驅動程式版本，即使來源和目標是相同的資料庫產品。這可讓您在不同版本的來源和目標資料庫之間移轉資料。若要使用這些選項，您必須先將 JDBC 驅動程式的 JAR 檔案上傳到 Amazon S3。

  - "customJdbcDriverS3Path"：自訂 JDBC 驅動程式的 Amazon S3 路徑。
  - "customJdbcDriverClassName"：JDBC 驅動程式的類別名稱。
- "bulkSize"：(選用) 用於設定平行插入，以加速 JDBC 目標的大量載入。指定寫入或插入資料時要使用之平行處理程度的整數值。此選項有助於改善寫入資料庫 (例如 Arch 使用者儲存庫 (AUR)) 的效能。
- "hashfield" (選用) 一個字串，用於指定 JDBC 資料表中的資料欄名稱，以便在從 JDBC 資料表平行讀取時將資料劃分為分割區。提供 "hashfield" 或 "hashexpression"。如需更多詳細資訊，請參閱 the section called “從 JDBC 中平行讀取”。
- "hashexpression" (選用) 傳回整數的 SQL 選取子句。用於在從 JDBC 資料表平行讀取時，將 JDBC 資料表中的資料劃分為分割區。提供 "hashfield" 或 "hashexpression"。如需更多詳細資訊，請參閱 the section called “從 JDBC 中平行讀取”。
- "hashpartitions" (選用) 正整數。在從 JDBC 資料表平行讀取時，用於指定 JDBC 資料表的平行讀取次數。預設：7。如需更多詳細資訊，請參閱 the section called “從 JDBC 中平行讀取”。
- "sampleQuery"：(選用) 自訂 SQL 查詢陳述式。用於指定資料表中的資訊子集，以擷取資料表內容範例。在不考慮資料的情況下進行設定時，效率可能比 DynamicFrame 方法低，從而會導致逾時或記憶體不足的錯誤。如需更多詳細資訊，請參閱 the section called “使用 sampleQuery”。
- "enablePartitioningForSampleQuery"：(選用) 布林值。預設：false。用於指定 sampleQuery 時啟用從 JDBC 資料表平行讀取。如果設定為 true，則 AWS Glue 的 **sampleQuery** 必須以 "where" 或 "and" 結尾，以附加分割條件。如需更多詳細資訊，請參閱 the section called “使用 sampleQuery”。
- "sampleSize"：(選用) 正整數。限制範例查詢傳回的資料列數。只有當 enablePartitioningForSampleQuery 為 true 時才適用。如果未啟用分割，則應直接在 sampleQuery 中新增 "limit x" 以限制大小。如需更多詳細資訊，請參閱 the section called “使用 sampleQuery”。

使用 sampleQuery

本節說明如何使用 sampleQuery、sampleSize 和 enablePartitioningForSampleQuery。

可以使用 sampleQuery 高效地對資料集的幾個資料列進行抽樣。依預設，由單一執行器執行查詢。在不考慮資料的情況下進行設定時，效率可能比 DynamicFrame 方法低，從而會導致逾時或記憶體不足的錯誤。作為 ETL 管道的一部分，在基礎資料庫上執行 SQL 通常只需用於效能目的。如果您嘗試預覽資料集的幾個資料列，請考慮使用 the section called “顯示”。如果您嘗試使用 SQL 轉換資料集，請考慮使用 the section called “toDF” 針對 DataFrame 表單中的資料定義 SparkSQL 轉換。

雖然您的查詢可能會操作各種資料表，但 dbtable 仍然是必需的。

使用 sampleQuery 擷取資料表範例

使用預設 sampleQuery 行為擷取資料範例時，AWS Glue 預期不會有大量的輸送量，因此它會在單一執行器上執行查詢。為了限制您提供的資料而不會導致效能問題，我們建議您向 SQL 提供一個 LIMIT 子句。

Example 使用 sampleQuery 而不進行分割

以下程式碼範例顯示如何使用 sampleQuery 而不進行分割。

```
//A full sql query statement.
val query = "select name from $tableName where age > 0 limit 1"
val connectionOptions = JsonOptions(Map(
    "url" -> url,
    "dbtable" -> tableName,
    "user" -> user,
    "password" -> password,
    "sampleQuery" -> query ))
val dyf = glueContext.getSource("mysql", connectionOptions)
          .getDynamicFrame()
```

針對較大的資料集使用 sampleQuery

如果您要讀取大型資料集，則可能需要啟用 JDBC 分割以並行查詢資料表。如需更多詳細資訊，請參閱 the section called “從 JDBC 中平行讀取”。若要搭配使用 sampleQuery 與 JDBC 分割，請將 enablePartitioningForSampleQuery 設定為 true。啟用此功能需要您對 sampleQuery 進行一些更改。

搭配使用 JDBC 分割區與 sampleQuery 時，您的查詢必須以 "where" 或 "and" 結尾，AWS Glue 才能附加分割條件。

如果您想要在從 JDBC 資料表平行讀取時限制 SampleQuery 的結果數量，請設定 "sampleSize" 參數，而不是指定 LIMIT 子句。

Example 搭配使用 sampleQuery 與 JDBC 分區

以下程式碼範例顯示如何搭配使用 sampleQuery 與 JDBC 分區。

```
//note that the query should end with "where" or "and" if use with JDBC partitioning.
val query = "select name from $tableName where age > 0 and"

//Enable JDBC partitioning by setting hashfield.
//to use sampleQuery with partitioning, set enablePartitioningForSampleQuery.
```

```
//use sampleSize to limit the size of returned data.
val connectionOptions = JsonOptions(Map(
    "url" -> url,
    "dbtable" -> tableName,
    "user" -> user,
    "password" -> password,
    "hashfield" -> primaryKey,
    "sampleQuery" -> query,
    "enablePartitioningForSampleQuery" -> true,
    "sampleSize" -> "1" ))
val dyf = glueContext.getSource("mysql", connectionOptions)
          .getDynamicFrame()
```

備註和限制：

範例查詢不可與任務書籤一起使用。在提供兩者的組態時，系統將忽略書籤狀態。

使用自訂 JDBC 驅動程式

下面的程式碼範例示範如何讀取和寫入使用自訂 JDBC 驅動程式的 JDBC 資料庫。這些範例示範讀取一個版本的資料庫產品並寫入相同產品的更高版本。

Python

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext, SparkConf
from awsglue.context import GlueContext
from awsglue.job import Job
import time
from pyspark.sql.types import StructType, StructField, IntegerType, StringType

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

# Construct JDBC connection options
connection_mysql5_options = {
    "url": "jdbc:mysql://<jdbc-host-name>:3306/db",
    "dbtable": "test",
    "user": "admin",
    "password": "pwd"}
```

```
connection_mysql8_options = {
    "url": "jdbc:mysql://<jdbc-host-name>:3306/db",
    "dbtable": "test",
    "user": "admin",
    "password": "pwd",
    "customJdbcDriverS3Path": "s3://DOC-EXAMPLE-BUCKET/mysql-connector-
java-8.0.17.jar",
    "customJdbcDriverClassName": "com.mysql.cj.jdbc.Driver"}

connection_oracle11_options = {
    "url": "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL",
    "dbtable": "test",
    "user": "admin",
    "password": "pwd"}

connection_oracle18_options = {
    "url": "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL",
    "dbtable": "test",
    "user": "admin",
    "password": "pwd",
    "customJdbcDriverS3Path": "s3://DOC-EXAMPLE-BUCKET/ojdbc10.jar",
    "customJdbcDriverClassName": "oracle.jdbc.OracleDriver"}

# Read from JDBC databases with custom driver
df_mysql8 = glueContext.create_dynamic_frame.from_options(connection_type="mysql",

 connection_options=connection_mysql8_options)

# Read DynamicFrame from MySQL 5 and write to MySQL 8
df_mysql5 = glueContext.create_dynamic_frame.from_options(connection_type="mysql",

 connection_options=connection_mysql5_options)
glueContext.write_from_options(frame_or_dfc=df_mysql5, connection_type="mysql",
                               connection_options=connection_mysql8_options)

# Read DynamicFrame from Oracle 11 and write to Oracle 18
df_oracle11 =
 glueContext.create_dynamic_frame.from_options(connection_type="oracle",

 connection_options=connection_oracle11_options)
glueContext.write_from_options(frame_or_dfc=df_oracle11, connection_type="oracle",
                               connection_options=connection_oracle18_options)
```

Scala

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamicFrame
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._


object GlueApp {
  val MYSQL_5_URI: String = "jdbc:mysql://<jdbc-host-name>:3306/db"
  val MYSQL_8_URI: String = "jdbc:mysql://<jdbc-host-name>:3306/db"
  val ORACLE_11_URI: String = "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL"
  val ORACLE_18_URI: String = "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL"

  // Construct JDBC connection options
  lazy val mysql5JsonOption = jsonOptions(MYSQL_5_URI)
  lazy val mysql8JsonOption = customJDBCDriverJsonOptions(MYSQL_8_URI, "s3://DOC-
EXAMPLE-BUCKET/mysql-connector-java-8.0.17.jar", "com.mysql.cj.jdbc.Driver")
  lazy val oracle11JsonOption = jsonOptions(ORACLE_11_URI)
  lazy val oracle18JsonOption = customJDBCDriverJsonOptions(ORACLE_18_URI, "s3://
DOC-EXAMPLE-BUCKET/ojdbc10.jar", "oracle.jdbc.OracleDriver")

  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    // Read from JDBC database with custom driver
    val df_mysql8: DynamicFrame = glueContext.getSource("mysql",
 mysql8JsonOption).getDynamicFrame()

    // Read DynamicFrame from MySQL 5 and write to MySQL 8
    val df_mysql5: DynamicFrame = glueContext.getSource("mysql",
 mysql5JsonOption).getDynamicFrame()
    glueContext.getSink("mysql", mysql8JsonOption).writeDynamicFrame(df_mysql5)

    // Read DynamicFrame from Oracle 11 and write to Oracle 18
```

```
    val df_oracle11: DynamicFrame = glueContext.getSource("oracle",
 oracle11JsonOption).getDynamicFrame()
    glueContext.getSink("oracle", oracle18JsonOption).writeDynamicFrame(df_oracle11)

    Job.commit()
  }

  private def jsonOptions(url: String): JsonOptions = {
    new JsonOptions(
      s"""{"url": "${url}",
         |"dbtable":"test",
         |"user": "admin",
         |"password": "pwd"}""".stripMargin)
  }

  private def customJDBCDriverJsonOptions(url: String, customJdbcDriverS3Path:
 String, customJdbcDriverClassName: String): JsonOptions = {
    new JsonOptions(
      s"""{"url": "${url}",
         |"dbtable":"test",
         |"user": "admin",
         |"password": "pwd",
         |"customJdbcDriverS3Path": "${customJdbcDriverS3Path}",
         |"customJdbcDriverClassName" :
 "${customJdbcDriverClassName}"}""".stripMargin)
  }
}
```

從 JDBC 資料表中平行讀取

您可以設定 JDBC 資料表的屬性，讓 AWS Glue 平行讀取資料。設定特定屬性時，您指示 AWS Glue 來對資料的邏輯分割區執行平行 SQL 查詢。您可以透過設定雜湊欄位或雜湊表達式來控制分割。您也可以控制平行讀取的數量，這些讀取會用來存取您的資料。

並行讀取 JDBC 資料表是一種可以提高性能的最佳化技術。如需有關識別此技術何時適用的程序的詳細資訊，請參閱《AWS 規定指南》中 AWS Glue for Apache Spark 任務效能調校最佳實務指引中的減少資料掃描數量 。

若要啟用平行讀取，您須在資料表結構的參數欄位中設定鍵值組。使用 JSON 符號來設定資料表的參數欄位值。關於編輯資料表屬性的詳細資訊，請參閱 檢視與編輯資料表的詳細資訊。您也可以在呼叫 ETL (擷取、轉換和載入) 方法 `create_dynamic_frame_from_options` 和

create_dynamic_frame_from_catalog 時啟用平行讀取。如需如何在這些方法中指定選項的詳細資訊，請參閱 from_options 和 from_catalog。

您可以對 JDBC 資料表 (也就是基礎資料為 JDBC 資料存放區的多數資料表) 使用此方法。讀取 Amazon Redshift 和 Amazon S3 資料表時，會略過這些屬性。

hashfield

將 hashfield 設為 JDBC 資料表中的欄位名稱，該 JDBC 資料表將用來將資料分配至分割區中。為獲得最佳結果，此欄位應擁有平均的分散值來將資料分散在分割區中。此欄位可以是任何資料類型。AWS Glue 會產生無重疊的查詢，這些查詢會平行執行，以讀取此欄位所分割的資料。例如，如果您的資料依月份平均分散，您可以使用 month 欄位來平行讀取每個月的資料：

```
'hashfield': 'month'
```

AWS Glue 建立查詢來將欄位值湊雜為分割區編號， 並針對所有分割區平行執行查詢。若要使用您自己的查詢來分割資料表讀取， 請提供 hashexpression 而非 hashfield。

hashexpression

將 hashexpression 設定為 SQL 表達式 (符合 JDBC 資料庫引擎文法)，此表達式會傳回一個整數。簡單表達式是資料表中任何數值欄位的名稱。AWS Glue 會產生 SQL 查詢來使用 hashexpression 子句中的 WHERE 來平行讀取 JDBC 資料以分割資料。

例如，使用數值欄位 customerID 來讀取依客戶編號分割的資料：

```
'hashexpression': 'customerID'
```

若要讓 AWS Glue 控制分割，請提供 hashfield 而非 hashexpression。

hashpartitions

將 hashpartitions 設定為 JDBC 資料表的平行讀取數。如果未設定該屬性，預設值為 7。

例如，將平行讀取數設定為 5，讓 AWS Glue 使用五個 (或更少) 查詢來讀取資料：

```
  'hashpartitions': '5'
```

為 Amazon RDS 資料存放區的 JDBC 連線設定 Amazon VPC AWS Glue

使用 JDBC 連線到 Amazon RDS 中的資料庫時，您將需要執行其他設定。若要讓AWS Glue元件能夠與 Amazon RDS 通訊，您必須在 Amazon VPC 中設定對 Amazon RDS 資料存放區的存取權。要讓 AWS Glue 在其元件之間通訊，請指定一個安全群組並為所有 TCP 連接埠建立自我參考的傳入規則。透過建立自我參考規則，您可以將來源限制為 VPC 中的相同安全群組。自我參考規則不會對所有網路開啟 VPC。VPC 的預設安全群組可能已經有了 ALL Traffic 的自我參考傳入規則。

若要設定 AWS Glue 和 Amazon RDS 資料存放區之間的存取權

1. 登入 AWS Management Console 並開啟 Amazon RDS 主控台，網址為 https://console.aws.amazon.com/rds/。
2. 在 Amazon RDS 主控台中，識別用來控制對 Amazon RDS 資料庫存取的安全群組。

   在左側導覽窗格中，選擇 [資料庫]，然後從主窗格的清單中選取您要連線的執行個體。

   在資料庫詳細資料頁面中，在 [連線與安全性] 索引標籤上找到 VPC 安全群組。
3. 根據您的網路架構，找出最適合修改的相關安全群組，以允許 AWS Glue 服務存取。保存其名稱，以*database-security-group*備 future 參考。如果沒有適當的安全群組，請遵循 Amazon RDS 文件中建立安全群組，以提供對 VPC 中資料庫執行個體的存取權的指示。
4. 登入 AWS Management Console 並開啟 Amazon VPC 主控台，網址為 https://console.aws.amazon.com/vpc/。
5. 在 Amazon VPC 主控台中，識別如何更新*database-security-group*。

   在左側導覽窗格中，選擇 [安全性群組]，然後*database-security-group*從主窗格的清單中選取。
6. 識別*database-security-group*、的安全性群組識別碼*database-sg-id*。保存它以備 future 參考。

   在安全性群組詳細資料頁面中，找到安全性群組 ID。
7. 變更的輸入規則 *database-security-group*，新增自我參照規則以允許AWS Glue元件進行通訊。具體來說，新增或確認有一個規則，其中「類型」為All TCP、「通訊協定是」TCP、「連接埠範圍」包含所有連接埠，而「來源」為*database-sg-id*。確認您為 S ource 輸入的安全性群組與您正在編輯的安全性群組相同。

在安全群組詳細資料頁面中，選取編輯輸入規則。

傳入規則類似：

Type	通訊協定	連接埠範圍	來源
所有 TCP	TCP	0–65535	*database-sg-id*

8. 新增輸出流量的規則。

在安全群組詳細資料頁面中，選取編輯輸出規則。

如果安全性群組允許所有輸出流量，則不需要個別的規則。例如：

Type	通訊協定	連接埠範圍	目的地
所有流量	ALL	ALL	0.0.0.0/0

如果您的網路架構是為限制輸出流量而設計的，請建立下列輸出規則：

建立自我參照規則All TCP，其中「類型」為「通訊協定」TCP，「連接埠範圍」包含所有連接埠，而「目的地」為 *database-sg-id* 確認您為「目的地」輸入的安全性群組與您正在編輯的安全性群組相同。

如果使用 Amazon S3 VPC 節點，請新增 HTTPS 規則以允許從 VPC 傳輸到 Amazon S3 的流量。建立規則HTTPS，其中「類型」為「通訊協定」TCP、「連接埠範圍是」，443而「目的地」是 Amazon S3 閘道端點 *s3-* 的受管前置詞清單 ID prefix-list-id。如需有關前置詞清單和 Amazon S3 閘道端點的詳細資訊，請參閱 Amazon VPC 說明文件中的 Amazon S3 閘道端點。

例如：

Type	通訊協定	連接埠範圍	目的地
所有 TCP	TCP	0–65535	*database-sg-id*
HTTPS	TCP	443	*S3-prefix-list-id*

MongoDB 連線

您可以使用 AWS Glue for Spark 從 AWS Glue 4.0 及更新版本中的 MongoDB 和 MongoDB Atlas 讀取和寫入資料表。您可以使用透過 AWS Glue 連線儲存於 AWS Secrets Manager 的使用者名稱和密碼憑證來連線至 MongoDB。

如需有關 MongoDB 的詳細資訊，請參閱 MongoDB 文件。

設定 MongoDB 連線

若要從 AWS 連線至 MongoDB，您將需要 MongoDB 憑證 *mongodbUser* 和 *mongodbPass*。

若要從 AWS Glue 連線至 MongoDB，您可能需要部分先決條件：

- 如果 Mongo DB 執行個體位於 Amazon VPC 中，請設定 Amazon VPC 以允許 AWS Glue 任務與 MongoDB 執行個體通訊，使流量不會周遊公有網際網路。

  在 Amazon VPC 中，識別或建立 AWS Glue 將在執行任務時使用的 VPC、子網路及安全群組。此外，您也需要確保 Amazon VPC 已完成設定，以允許 MongoDB 執行個體與此位置之間的網路流量。根據您的網路配置，這可能需要變更安全群組規則、網路 ACL、NAT 閘道及對等連線。

然後，您可以繼續設定 AWS Glue 以搭配 MongoDB 使用。

設定連至 MongoDB 的連線：

1. 或者，在 AWS Secrets Manager 中，使用 MongoDB 憑證建立密碼。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *secretName*，以便進行下一個步驟。

   - 在選取鍵/值組時，請使用 *mongodbUser* 值來建立 `username` 金鑰對。

     在選取鍵/值組時，請使用 *mongodbPass* 值來建立 `password` 金鑰對。

2. 在 AWS Glue 主控台中，依照 the section called “新增 AWS Glue 連線” 中的步驟建立連線。建立連線之後，請保留連線名稱 *connectionName*，以便未來在 AWS Glue 中使用。

   - 選取連線類型時，請選取 MongoDB 或 MongoDB Atlas。
   - 選取 MongoDB URL 或 MongoDB Atlas URL 時，請提供 MongoDB 執行個體的主機名稱。

     MongoDB URL 會以 `mongodb://`*mongoHost*`:`*mongoPort*`/`*mongoDBname* 格式提供。

MongoDB Atlas URL 會以 mongodb+srv://*mongoHost*:*mongoPort*/*mongoDBname* 格式提供。

您可選用 *mongoDBname* 針對連線提供預設資料庫。

- 如果您選擇建立 Secrets Manager 密碼，請選擇 AWS Secrets Manager 憑證類型。

  然後，在 AWS 密碼中提供 *secretName*。
- 如果您選擇提供使用者名稱和密碼，請提供 *mongodbUser* 和 *mongodbPass*。

3. 在下列情況中，您可能需要其他組態：

   - Amazon VPC 中託管於 AWS 的 MongoDB 執行個體
     - 您將需要向定義 MongoDB 安全憑證的 AWS Glue 連線提供 Amazon VPC 連線資訊。建立或更新連線時，請在網路選項中設定 VPC、子網路及安全群組。

建立 AWS Glue MongoDB 連線後，您將需要執行下列動作，才能呼叫連線方法：

- 如果您選擇建立 Secrets Manager 密碼，請授予與 AWS Glue 任務權限相關聯的 IAM 角色，以讀取 *secretName*。
- 在您的 AWS Glue 任務組態中，提供 *connectionName* 作為其他網路連線。

若要在 AWS Glue for Spark 中使用 AWS Glue MongoDB 連線，請在連線方法呼叫中提供 connectionName 選項。或者，您也可依照 the section called “與 MongoDB 整合” 中的步驟執行，以搭配 AWS Glue Data Catalog 使用連線。

使用 AWS Glue 連線從 MongoDB 讀取

先決條件：

- 您想要讀取的 MongoDB 集合。您將需要集合的識別資訊。

  MongoDB 集合由資料庫名稱與集合名稱 *mongodbName*、*mongodbCollection* 識別。
- 完成設定的 AWS Glue MongoDB 連線，可提供驗證資訊。完成上一個程序設定連至 MongoDB 的連線的步驟，以設定驗證資訊。您將會需要 AWS Glue 連線的名稱，*connectionName*。

例如：

```
mongodb_read = glueContext.create_dynamic_frame.from_options(
```

```
    connection_type="mongodb",
    connection_options={
        "connectionName": "connectionName",
        "database": "mongodbName",
        "collection": "mongodbCollection",
        "partitioner":
 "com.mongodb.spark.sql.connector.read.partitioner.SinglePartitionPartitioner",
        "partitionerOptions.partitionSizeMB": "10",
        "partitionerOptions.partitionKey": "_id",
        "disableUpdateUri": "false",
    }
)
```

寫入 MongoDB 資料表

此範例會從現有的 DynamicFrame *dynamicFrame* 將資訊寫入 MongoDB。

先決條件：

- 您想要寫入的 MongoDB 集合。您將需要集合的識別資訊。

  MongoDB 集合由資料庫名稱與集合名稱 *mongodbName*、*mongodbCollection* 識別。
- 完成設定的 AWS Glue MongoDB 連線，可提供驗證資訊。完成上一個程序設定連至 MongoDB 的連線的步驟，以設定驗證資訊。您將會需要 AWS Glue 連線的名稱，*connectionName*。

例如：

```
glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="mongodb",
    connection_options={
        "connectionName": "connectionName",
        "database": "mongodbName",
        "collection": "mongodbCollection",
        "disableUpdateUri": "false",
        "retryWrites": "false",
    },
)
```

讀取和寫入 MongoDB 資料表

此範例會從現有的 DynamicFrame *dynamicFrame* 將資訊寫入 MongoDB。

先決條件：

- 您想要讀取的 MongoDB 集合。您將需要集合的識別資訊。

  您想要寫入的 MongoDB 集合。您將需要集合的識別資訊。

  MongoDB 集合由資料庫名稱與集合名稱 *mongodbName*、*mongodbCollection* 識別。
- MongoDB 驗證資訊 *mongodbUser* 和 *mongodbPassword*。

例如：

Python

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext, SparkConf
from awsglue.context import GlueContext
from awsglue.job import Job
import time

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

job = Job(glueContext)
job.init(args['JOB_NAME'], args)

output_path = "s3://some_bucket/output/" + str(time.time()) + "/"
mongo_uri = "mongodb://<mongo-instanced-ip-address>:27017"
mongo_ssl_uri = "mongodb://<mongo-instanced-ip-address>:27017"
write_uri = "mongodb://<mongo-instanced-ip-address>:27017"

read_mongo_options = {
    "uri": mongo_uri,
    "database": "mongodbName",
    "collection": "mongodbCollection",
    "username": "mongodbUsername",
    "password": "mongodbPassword",
```

```
    "partitioner": "MongoSamplePartitioner",
    "partitionerOptions.partitionSizeMB": "10",
    "partitionerOptions.partitionKey": "_id"}

ssl_mongo_options = {
    "uri": mongo_ssl_uri,
    "database": "mongodbName",
    "collection": "mongodbCollection",
    "ssl": "true",
    "ssl.domain_match": "false"
}

write_mongo_options = {
    "uri": write_uri,
    "database": "mongodbName",
    "collection": "mongodbCollection",
    "username": "mongodbUsername",
    "password": "mongodbPassword",
}

# Get DynamicFrame from MongoDB
dynamic_frame =
 glueContext.create_dynamic_frame.from_options(connection_type="mongodb",

 connection_options=read_mongo_options)

# Write DynamicFrame to MongoDB
glueContext.write_dynamic_frame.from_options(dynamicFrame,
 connection_type="mongodb", connection_options=write_mongo_options)

job.commit()
```

Scala

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamicFrame
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
```

```
object GlueApp {
  val DEFAULT_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
  val WRITE_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
  lazy val defaultJsonOption = jsonOptions(DEFAULT_URI)
  lazy val writeJsonOption = jsonOptions(WRITE_URI)
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    // Get DynamicFrame from MongoDB
    val dynamicFrame: DynamicFrame = glueContext.getSource("mongodb",
 defaultJsonOption).getDynamicFrame()

    // Write DynamicFrame to MongoDB
    glueContext.getSink("mongodb", writeJsonOption).writeDynamicFrame(dynamicFrame)

    Job.commit()
  }

  private def jsonOptions(uri: String): JsonOptions = {
    new JsonOptions(
      s"""{"uri": "${uri}",
         |"database":"mongodbName",
         |"collection":"mongodbCollection",
         |"username": "mongodbUsername",
         |"password": "mongodbPassword",
         |"ssl":"true",
         |"ssl.domain_match":"false",
         |"partitioner": "MongoSamplePartitioner",
         |"partitionerOptions.partitionSizeMB": "10",
         |"partitionerOptions.partitionKey": "_id"}""".stripMargin)
  }
}
```

MongoDB 連線選項參考

指定與 MongoDB 的連線。來源連線和接收器連線的連線選項不同。

以下連線屬性可在來源連線和接收器連線之間共用：

- connectionName：用於讀取/寫入。完成設定之 AWS Glue MongoDB 連線的名稱，可向連線方法提供驗證和網路資訊。當 AWS Glue 連線完成設定 (如上一節 the section called “設定 MongoDB” 中所述) 後，提供 connectionName 將會取代提供 "uri"、"username" 及 "password" 連線選項的需要。
- "uri"：(必要) 讀取的 MongoDB 主機，格式為 mongodb://<host>:<port>。在 AWS Glue 4.0 之前的 AWS Glue 版本中使用。
- "connection.uri"：(必要) 讀取的 MongoDB 主機，格式為 mongodb://<host>:<port>。在 AWS Glue 4.0 及更新版本中使用。
- "username"：(必要) MongoDB 使用者名稱。
- "password"：(必要) MongoDB 密碼。
- "database"：(必要) 讀取的 MongoDB 資料庫。在任務指令碼中呼叫 glue_context.create_dynamic_frame_from_catalog 時，也可在 additional_options 中傳遞此選項。
- "collection"：(必要) 讀取的 MongoDB 集合。在任務指令碼中呼叫 glue_context.create_dynamic_frame_from_catalog 時，也可在 additional_options 中傳遞此選項。

"connectionType": "mongodb" as Source

使用下列有 "connectionType": "mongodb" 的連線選項作為來源：

- "ssl"：(選用) 如果 true，則啟用 SSL 連線。預設值為 false。
- "ssl.domain_match"：(選用) 如果 true 和 ssl 為 true，則執行網域符合檢查。預設值為 true。
- "batchSize"：(選用)：每個批次傳回的文件數目，於內部批次的游標內使用。
- "partitioner"：(選用)：從 MongoDB 讀取輸入資料的分割區類別名稱。連接器提供下列分割區：
  - MongoDefaultPartitioner (預設) (不支援 AWS Glue 4.0)
  - MongoSamplePartitioner (需要 MongoDB 3.2 或更新版本) (不支援 AWS Glue 4.0)
  - MongoShardedPartitioner (不支援 AWS Glue 4.0)
  - MongoSplitVectorPartitioner (不支援 AWS Glue 4.0)
  - MongoPaginateByCountPartitioner (不支援 AWS Glue 4.0)
  - MongoPaginateBySizePartitioner (不支援 AWS Glue 4.0)

  - com.mongodb.spark.sql.connector.read.partitioner.SinglePartitionPartitioner
  - com.mongodb.spark.sql.connector.read.partitioner.ShardedPartitioner
  - com.mongodb.spark.sql.connector.read.partitioner.PaginateIntoPartitionsPartit
- "partitionerOptions" (選用)：指定分割區的選項。每個分割區都支援下列選項：
  - MongoSamplePartitioner: partitionKey, partitionSizeMB, samplesPerPartition
  - MongoShardedPartitioner: shardkey
  - MongoSplitVectorPartitioner: partitionKey, partitionSizeMB
  - MongoPaginateByCountPartitioner: partitionKey, numberOfPartitions
  - MongoPaginateBySizePartitioner: partitionKey, partitionSizeMB

  如需有關這些選項的詳細資訊，請參閱 MongoDB 文件中的分割區組態。

"connectionType": "mongodb" as Sink

使用下列有 "connectionType": "mongodb" 的連線選項作為接收器：

- "ssl"：(選用) 如果 true，則啟用 SSL 連線。預設值為 false。
- "ssl.domain_match"：(選用) 如果 true 和 ssl 為 true，則執行網域符合檢查。預設值為 true。
- "extendedBsonTypes"：(選用) 如果 true，則在寫入資料至 MongoDB 時允許延伸的 BSON 類型。預設值為 true。
- "replaceDocument"：(選用) 如果 true，則在儲存包含 _id 欄位的資料集時取代整個文件。若為 false，則文件中僅有與資料集中欄位相符的欄位會更新。預設值為 true。
- "maxBatchSize"：(選用)：儲存資料時大量操作的批次大小上限。預設為 512。
- "retryWrites":(選用) 如果 AWS Glue 發生網路錯誤，系統會自動重試特定寫入操作一次。

SAP HANA 連線

您可以使用 AWS Glue for Spark 從 AWS Glue 4.0 及更新版本中的 SAP HANA 讀取和寫入資料表。您可以使用 SQL 查詢定義要從 SAP HANA 讀取的內容。您可以使用透過 AWS Glue SAP HANA 連線儲存於 AWS Secrets Manager 的 JDBC 憑證來連線至 SAP HANA。

如需有關 SAP HANA JDBC 的詳細資訊，請參閱 SAP HANA 文件。

設定 SAP HANA 連線

若要從 AWS Glue 連線至 SAP HANA，您將需要在 AWS Secrets Manager 密碼中建立並儲存 SAP HANA 憑證，然後將該密碼與 SAP HANA AWS Glue 連線建立關聯。您將需要設定 SAP HANA 服務與 AWS Glue 之間的網路連線。

若要連線至 SAP HANA，您可能需要部分先決條件：

- 如果 SAP HANA 服務位於 Amazon VPC 中，請設定 Amazon VPC 以允許 AWS Glue 工作與 SAP HANA 服務進行通訊，使流量不會周遊公有網際網路。

  在 Amazon VPC 中，識別或建立 AWS Glue 將在執行工作時使用的 VPC、子網路及安全群組。此外，您也需要確保 Amazon VPC 已完成設定，以允許 SAP HANA 端點與此位置之間的網路流量。您的工作將需要與 SAP HANA JDBC 連接埠建立 TCP 連線。如需有關 SAP HANA 連接埠的詳細資訊，請參閱 SAP HANA 文件。根據您的網路配置，這可能需要變更安全群組規則、網路 ACL、NAT 閘道及對等連線。
- 如果 SAP HANA 端點可存取網際網路，則無需其他先決條件。

設定連至 SAP HANA 的連線：

1. 在 AWS Secrets Manager 中，使用 SAP HANA 憑證建立密碼。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *secretName*，以便進行下一個步驟。

   - 在選取鍵/值組時，請使用 *saphanaUsername* 值來建立 `user` 金鑰對。
   - 在選取鍵/值組時，請使用 *saphanaPassword* 值來建立 `password` 金鑰對。
2. 在 AWS Glue 主控台中，依照 the section called “新增 AWS Glue 連線” 中的步驟建立連線。建立連線之後，請保留連線名稱 *connectionName*，以便未來在 AWS Glue 中使用。

   - 選取連線類型時，請選取 SAP HANA。
   - 提供 SAP HANA URL 時，請提供執行個體的 URL。

     SAP HANA JDBC URL 會採用的格式為 `jdbc:sap://`*saphanaHostname*`:`*saphanaPort*`/?`*databaseName*`=`*saphanaDBname*`,`*Paramete*

     AWS Glue 需要下列 JDBC URL 參數：

     - `databaseName`：要連線之 SAP HANA 的預設資料庫。
   - 選取 AWS 機密時，請提供 *secretName*。

建立 AWS Glue SAP HANA 連線之後，您將需要執行下列步驟，才能執行 AWS Glue 工作：

- 授予與您 AWS Glue 任務許可相關聯的 IAM 角色，以讀取 *secretName*。
- 在您的 AWS Glue 任務組態中，提供 *connectionName* 作為其他網路連線。

從 SAP HANA 資料表讀取

先決條件：

- 您想要讀取的 SAP HANA 資料表。您將需要資料表的識別資訊。

  您可以在表單 *schemaName*.*tableName* 中使用 SAP HANA 資料表名稱和結構描述名稱來指定資料表。如果資料表位於預設結構描述 "public" 中，則不需要結構描述名稱和 "." 分隔符號。呼叫此 *tableIdentifier*。請注意，在 connectionName 中，資料庫會以 JDBC URL 參數形式提供。
- 完成設定的 AWS Glue SAP HANA 連線，可提供驗證資訊。完成上一個程序中的步驟設定連至 SAP HANA 的連線，以設定驗證資訊。您將會需要 AWS Glue 連線的名稱，*connectionName*。

例如：

```
saphana_read_table = glueContext.create_dynamic_frame.from_options(
    connection_type="saphana",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableIdentifier",
    }
)
```

您也可提供 SELECT SQL 查詢，以篩選傳回 DynamicFrame 的結果。您將需要設定 query。

例如：

```
saphana_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="saphana",
    connection_options={
        "connectionName": "connectionName",
        "query": "query"
    }
)
```

寫入 SAP HANA 資料表

此範例會從現有的 DynamicFrame *dynamicFrame* 將資訊寫入 SAP HANA。如果資料表已具有資訊，則 AWS Glue 會發生錯誤。

先決條件：

- 您想要寫入的 SAP HANA 資料表。

  您可以在表單 *schemaName*.*tableName* 中使用 SAP HANA 資料表名稱和結構描述名稱來指定資料表。如果資料表位於預設結構描述 "public" 中，則不需要結構描述名稱和 "." 分隔符號。呼叫此 *tableIdentifier*。請注意，在 connectionName 中，資料庫會以 JDBC URL 參數形式提供。
- SAP HANA 驗證資訊。完成上一個程序中的步驟設定連至 SAP HANA 的連線，以設定驗證資訊。您將會需要 AWS Glue 連線的名稱，*connectionName*。

例如：

```
options = {
    "connectionName": "connectionName",
    "dbtable": 'tableIdentifier'
}

    saphana_write = glueContext.write_dynamic_frame.from_options(
        frame=dynamicFrame,
        connection_type="saphana",
        connection_options=options
)
```

SAP HANA 連線選項參考

- connectionName – 必要。用於讀取/寫入。完成設定之 AWS Glue SAP HANA 連線的名稱，可向連線方法提供驗證和網路資訊。
- databaseName：用於讀取/寫入。有效值：SAP HANA 中的資料庫名稱。要連線的資料庫名稱。
- dbtable：除非已提供 query，否則為寫入和讀取的必要項目。用於讀取/寫入。有效值：SAP HANA SQL FROM 子句的內容。識別要連線的 SAP HANA 資料表。您也可提供資料表名稱以外的其他 SQL (例如，子查詢)。如需詳細資訊，請參閱《SAP HANA 文件》中的 FROM 子句。
- query – 用於讀取。定義從 SAP HANA 讀取時應擷取之內容的 SAP HANA SQL SELECT 查詢。

Snowflake 連線

您可以使用 AWS Glue for Spark 從 AWS Glue 4.0 及更新版本中的 Snowflake 讀取和寫入資料表。您可以使用 SQL 查詢從 Snowflake 讀取。您可以使用使用者和密碼連線至 Snowflake。您可以透過 AWS Glue Data Catalog 參考存放在 AWS Secrets Manager 中的 Snowflake 憑證。AWS Glue for Spark 的 Data Catalog Snowflake 憑證會與爬蟲程式的 Data Catalog Snowflake 憑證分開存放。您必須選擇 `SNOWFLAKE` 類型連線，而不是設定為連線至 Snowflake 的 `JDBC` 類型連線。

如需有關 Snowflake 的詳細資訊，請參閱 Snowflake 網站。如需有關 AWS 上的 Snowflake 的詳細資訊，請參閱 Snowflake Data Warehouse on Amazon Web Services。

設定 Snowflake 連線

透過網際網路連線到可用的 Snowflake 資料庫沒有 AWS 先決條件。

或者，您可以執行下列組態，以使用 AWS Glue 管理您的連線憑證。

使用 AWS Glue 管理您的連線憑證

1. 在 Snowflake 中產生一個使用者、*`snowflakeUser`*和密碼 *`snowflakePassword`*。
2. 在 AWS Secrets Manager 中，使用您的 Snowflake 憑證建立機密。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *`secretName`*，以便進行下一個步驟。
   - 選取鍵值對時，使用索引鍵 `sfUser` 為 *`snowflakeUser`* 建立鍵值對。
   - 選取鍵值對時，使用索引鍵 `sfPassword` 為 *`snowflakePassword`* 建立鍵值對。
   - 選取鍵值對時，您可以為 Snowflake 倉儲提供索引鍵 `sfWarehouse`。
3. 在 AWS Glue Data Catalog 中，依照 the section called “新增 AWS Glue 連線” 中的步驟建立連線。建立連線之後，請保留連線名稱 *`connectionName`*，以便進行下一個步驟。
   - 選取連線類型時，請選取 Snowflake。
   - 選取 Snowflake URL 時，請提供 Snowflake 執行個體的 URL。URL 將在表單 *`account_identifier`*`.snowflakecomputing.com` 中使用 hostname。
   - 選取 AWS 機密時，請提供 *`secretName`*。
4. 在您的 AWS Glue 任務組態中，提供 *`connectionName`* 作為其他網路連線。

在下列情況下，您可能需要滿足以下條件：

- 對於 Amazon VPC 中在 AWS 上託管的 Snowflake

- 您需要為 Snowflake 進行適當的 Amazon VPC 設定。如需有關如何設定 Amazon VPC 的詳細資訊，請參閱 Snowflake 文件中的 AWS PrivateLink & Snowflake。
- 您需要為 AWS Glue 進行適當的 Amazon VPC 設定。the section called “VPC 端點 (AWS PrivateLink)”。
- 您需要建立 AWS Glue Data Catalog 連線，以提供 Amazon VPC 連線資訊 (除了定義 Snowflake 安全憑證的 AWS Secrets Manager 機密 ID 之外)。您的 URL 會在使用 AWS PrivateLink 時變更，如前一個項目中所連結的 Snowflake 文件所述。
- 您需要任務組態，才能將資料型錄連線納入為其他網路連線。

從 Snowflake 資料表讀取

先決條件：您想要讀取的 Snowflake 資料表。您需要 Snowflake 資料表名稱 *tableName*。您需要 Snowflake url *snowflakeUrl*、使用者名稱 *snowflakeUser* 和密碼 *snowflakePassword*。如果您的 Snowflake 使用者沒有預設的命名空間集，您將需要 Snowflake 資料庫名稱 *databaseName* 和結構描述名稱 *schemaName*。此外，如果您的 Snowflake 使用者沒有預設的倉儲集，您將需要倉儲名稱 *warehouseName*。

例如：

其他先決條件：完成步驟使用 AWS Glue 管理您的連線憑證，以設定 *SnowFlakeURL*、*snowflakeUsername* 和 *snowflakePassword*。若要檢閱這些步驟，請參閱上一節 the section called “設定 Snowflake”。若要選取要連線的其他網路連線，我們將使用 connectionName 參數。

```
snowflake_read = glueContext.create_dynamic_frame.from_options(
  connection_type="snowflake",
  connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName",
        "sfDatabase": "databaseName",
        "sfSchema": "schemaName",
        "sfWarehouse": "warehouseName",
    }
)
```

此外，您可以使用 autopushdown 和 query 參數來讀取 Snowflake 資料表的一部分。這可能比在將結果載入 Spark 後篩選結果的方式更高效。考量一個範例，其中所有銷售額都存放在同一個資料表

中，但您只需分析某個商店在假日的銷售額。如果該資訊存放在資料表中，則可以使用述詞下推來擷取結果，如下所示：

```
snowflake_node = glueContext.create_dynamic_frame.from_options(
    connection_type="snowflake",
    connection_options={
        "autopushdown": "on",
        "query": "select * from sales where store='1' and IsHoliday='TRUE'",
        "connectionName": "snowflake-glue-conn",
        "sfDatabase": "databaseName",
        "sfSchema": "schemaName",
        "sfWarehouse": "warehouseName",
    }
)
```

寫入 Snowflake 資料表

先決條件：您想要寫入的 Snowflake 資料庫。您需要最新的或所需的資料表名稱 *tableName*。您需要 Snowflake url *snowflakeUrl*、使用者名稱 *snowflakeUser* 和密碼 *snowflakePassword*。如果您的 Snowflake 使用者沒有預設的命名空間集，您將需要 Snowflake 資料庫名稱 *databaseName* 和結構描述名稱 *schemaName*。此外，如果您的 Snowflake 使用者沒有預設的倉儲集，您將需要倉儲名稱 *warehouseName*。

例如：

其他先決條件：完成步驟使用 AWS Glue 管理您的連線憑證，以設定 *SnowFlakeURL*、*snowflakeUsername* 和 *snowflakePassword*。若要檢閱這些步驟，請參閱上一節 the section called “設定 Snowflake”。若要選取要連線的其他網路連線，我們將使用 connectionName 參數。

```
glueContext.write_dynamic_frame.from_options(
    connection_type="snowflake",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName",
        "sfDatabase": "databaseName",
        "sfSchema": "schemaName",
        "sfWarehouse": "warehouseName",
    },
)
```

Snowflake 連線選項參考

Snowflake 連線類型採用下列連線選項：

您可以從資料型錄連線 (sfUrl、sfUser、sfPassword) 擷取本節中的某些參數，在這種情況下，您不需要提供它們。您可以提供參數 connectionName 來完成此操作。

您可以從 AWS Secrets Manager 機密 (sfUser、sfPassword) 擷取本節中的某些參數，在這種情況下，您不需要提供它們。機密必須提供 sfUser 和 sfPassword 索引鍵下的內容。您可以提供參數 secretId 來完成此操作。

連線到 Snowflake 時，通常使用以下參數。

- sfDatabase：未在 Snowflake 中設定使用者預設值時需要。用於讀取/寫入。連線後用於工作階段的資料庫。
- sfSchema：未在 Snowflake 中設定使用者預設值時需要。用於讀取/寫入。連線後用於工作階段的結構描述。
- sfWarehouse：未在 Snowflake 中設定使用者預設值時需要。用於讀取/寫入。連線後用於工作階段的預設虛擬倉儲。
- sfRole：未在 Snowflake 中設定使用者預設值時需要。用於讀取/寫入。連線後用於工作階段的預設安全角色。
- sfUrl：(必要) 用於讀取/寫入。以下列格式指定帳戶的主機名稱：*account_identifier*.snowflakecomputing.com。如需有關帳戶識別碼的詳細資訊，請參閱 Snowflake 文件中的 Account Identifiers。
- sfUser：(必要) 用於讀取/寫入。Snowflake 使用者的登入名稱。
- sfPassword — (除非 pem_private_key 提供，否則為必填項) 用於讀取/寫入。Snowflake 使用者的密碼。
- dbtable：使用完整資料表時需要。用於讀取/寫入。要讀取之資料表或向其寫入資料之資料表的名稱。讀取時，將擷取所有資料欄和記錄。
- pem_private_key：用於讀取/寫入。未加密的 b64 編碼私有金鑰字串。Snowflake 使用者的私有金鑰。通常將其從 PEM 檔案中複製出來。如需詳細資訊，請參閱 Snowflake 文件中的金鑰對驗證和金鑰對輪換。
- query：使用查詢讀取時需要。用於讀取。要執行的確切查詢 (SELECT 陳述式)

以下選項用於在連線到 Snowflake 過程中設定特定行為。

- `preactions`：用於讀取/寫入。有效值：以分號分隔的 SQL 陳述式清單作為字串。SQL 陳述式會在 AWS Glue 和 Snowflake 之間傳輸資料之前執行。如果陳述式包含 `%s`，則 `%s` 會以操作參考的資料表名稱取代。
- `postactions`：用於讀取/寫入。SQL 陳述式會在 AWS Glue 和 Snowflake 之間傳輸資料之後執行。如果陳述式包含 `%s`，則 `%s` 會以操作參考的資料表名稱取代。
- `autopushdown`：預設："on"。有效值："on"、"off"。此參數控制是否啟用自動查詢下推。如果下推已啟用，則在 Spark 上執行查詢時，若查詢的一部分可以「向下推」到 Snowflake 伺服器，它便會被下推。這可改善某些查詢的效能。如需有關是否可以下推查詢的資訊，請參閱 Snowflake 文件中的 Pushdown。

此外，AWS Glue 可支援 Snowflake Spark 連接器上的某些可用選項。如需有關 Snowflake Spark 連接器上可用選項的詳細資訊，請參閱 Snowflake 文件中的 Setting Configuration Options for the Connector。

Snowflake 連接器限制

透過 AWS Glue for Spark 連線到 Snowflake 會受到以下限制。

- 此連接器不支援任務書籤。如需有關任務書籤的詳細資訊，請參閱 the section called “使用任務書籤追蹤處理的資料”。
- 此連接器不支援使用 `create_dynamic_frame.from_catalog` 和 `write_dynamic_frame.from_catalog` 方法，透過 AWS Glue Data Catalog 中的資料表進行 Snowflake 讀取和寫入。
- 此連接器不支援透過使用者和密碼以外的憑證連線至 Snowflake。
- 串流任務不支援此連接器。
- 擷取資訊 (例如使用 `query` 參數擷取) 時，此連接器支援 `SELECT` 陳述式型查詢。不支援其他類型的查詢 (例如 `SHOW`、`DESC` 或 DML 陳述式)。
- Snowflake 會將透過 Snowflake 用戶端提交的查詢文字 (例如 SQL 陳述式) 的大小限制為每個陳述式 1 MB。如需詳細資訊，請參閱 Limits on Query Text Size。

Teradata Vantage 連線

您可以使用 Spark 的 AWS Glue 從 G AWS lue 4.0 及更高版本中的 Teradata 華帝奇中讀取和寫入現有的表。您可以使用 SQL 查詢定義要從 Teradata 讀取的內容。您可以使用 AWS Secrets Manager 透過 AWS Glue 連線儲存的使用者名稱和密碼憑證來連線到 Teradata。

如需有關 Teradata 的詳細資訊，請參閱 Teradata 文件。

設定 Teradata 連線

要從 AWS Glue 連接到 Teradata，您需要在秘密中創建和存儲您的 Teradata 憑據，然後將該 AWS Secrets Manager 秘密與 AWS Glue Teradata 連接相關聯。如果您的 Teradata 執行個體位於 Amazon VPC 中，您還需要為您的 AWS Glue Teradata 連線提供聯網選項。

要從 AWS Glue 連接到 Teradata，您可能需要一些先決條件：

- 如果您要透過 Amazon VPC 存取 Teradata 環境，請設定 Amazon VPC 以允許您的 AWS Glue 任務與 Teradata 環境進行通訊。我們不建議透過公有網際網路存取 Teradata 環境。

  在 Amazon VPC 中，識別或建立 AWS Glue 將在執行任務時使用的 VPC、子網路和安全群組。此外，您也需要確保 Amazon VPC 已完成設定，以允許 Teradata 執行個體與此位置之間的網路流量。您的任務將需要與 Teradata 用戶端連接埠建立 TCP 連線。如需有關 Teradata 連接埠的詳細資訊，請參閱 Teradata 文件。

  根據您的網路配置，安全 VPC 連線可能需要變更 Amazon VPC 和其他網路服務。如需有關 AWS 連線的詳細資訊，請參閱 Teradata 文件中的AWS 連線選項。

若要設定 AWS Glue 太資料連線：

1. *### Teradata ############### AWS Glue ## Teradata ########*如需詳細資訊，請參閱《Teradata 文件》中的 Vantage 安全概觀。
2. 在中 AWS Secrets Manager，使用您的 Teradata 認證建立密碼。若要在 Secrets Manager 中建立密碼，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 密碼中提供的教學課程。建立機密之後，請保留機密名稱 *secretName*，以便進行下一個步驟。

   - 在選取鍵/值組時，請使用 *teradataUsername* 值來建立 `user` 金鑰對。
   - 在選取鍵/值組時，請使用 *teradataPassword* 值來建立 `password` 金鑰對。
3. 在 AWS Glue 主控台中，依照中的步驟建立連線the section called “新增 AWS Glue 連線”。建立連線之後，請保留連線名稱 *connectionName*，以便進行下一個步驟。

   - 選取連線類型時，請選取 Teradata。
   - 提供 JDBC URL 時，請提供執行個體的 URL。您也可在 JDBC URL 中，針對特定逗號分隔的連線參數進行硬編碼。URL 必須符合下列格式：`jdbc:teradata://`*teradataHostname*`/`*ParameterName*`=`*ParameterValue*`,`*ParameterN*

支援的 URL 參數包括：

- DATABASE：依預設要存取之主機的資料庫名稱。
- DBS_PORT：在非標準連接埠上執行時所使用的資料庫連接埠。

- 選取憑證類型時，請選取 AWS Secrets Manager，然後將 AWS 密碼 設定為 *secretName*。

4. 在下列情況中，您可能需要其他組態：

   - 對於 AWS 在 Amazon VPC 上託管的 Teradata 執行個體
     - 您需要向定義 Teradata 安全登入資料的 AWS Glue 連線提供 Amazon VPC 連線資訊。建立或更新連線時，請在網路選項中設定 VPC、子網路及安全群組。

建立 AWS Glue Teradata 連線後，您必須先執行下列步驟，再呼叫連線方法。

- 授予與您的 AWS Glue 工作相關聯的 IAM 角色，以讀取###稱的權限。
- 在您的 AWS Glue 工作組態中，提供####做為其他網路連線。

從 Teradata 中讀取

先決條件：

- 您想要讀取的 Teradata 資料表。您將需要資料表名稱 *tableName*。
- 設定為提供驗證資訊的 AWS Glue Teradata 連線。完成步驟設定連至 Teradata 的連線，以設定身份驗證資訊。您將需要 AWS Glue 連接的名稱，##名稱。

例如：

```
teradata_read_table = glueContext.create_dynamic_frame.from_options(
    connection_type="teradata",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName"
    }
)
```

您也可以提供 SELECT SQL 查詢，以篩選傳回給您的 DynamicFrame. 您將需要設定 query。

例如：

```
teradata_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="teradata",
    connection_options={
        "connectionName": "connectionName",
        "query": "query"
    }
)
```

寫入 Teradata 資料表

先決條件：您想要寫入的 Teradata 資料表 *tableName*。您必須先建立資料表，再呼叫連線方法。

例如：

```
teradata_write = glueContext.write_dynamic_frame.from_options(
    connection_type="teradata",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName"
    }
)
```

Teradata 連線選項參考

- `connectionName` – 必要。用於讀取/寫入。 AWS Glue Teradata 連線的名稱，設定為為您的連線方法提供驗證和網路資訊。
- `dbtable`：除非已提供 `query`，否則為寫入和讀取的必要項目。用於讀取/寫入。您的連線方法將會互動的資料表名稱。
- `query` – 用於讀取。定義從 Teradata 讀取時應擷取之內容的 SELECT SQL 查詢。

Vertica 連線

您可以使用 AWS Glue for Spark 從 AWS Glue 4.0 及更新版本中的 Vertica 讀取和寫入資料表。您可以使用 SQL 查詢定義要從 Vertica 讀取的內容。您可以使用透過 AWS Glue 連線儲存於 AWS Secrets Manager 的使用者名稱和密碼憑證來連線至 Vertica。

如需有關 Vertica 的詳細資訊，請參閱 Vertica 文件。

設定 Vertica 連線

若要從 AWS Glue 連線至 Vertica，您將需要在 AWS Secrets Manager 密碼中建立並儲存 Vertica 憑證，然後將該密碼與 Vertica AWS Glue 連線建立關聯。如果 Vertica 執行個體位於 Amazon VPC 中，您也需要向 AWS Glue Vertica 連線提供網路選項。從資料庫讀取和寫入資料庫時，您將需要用於暫存空間的 Amazon S3 儲存貯體或資料夾。

若要從 AWS Glue 連線至 Vertica，您可能需要部分先決條件：

- 從資料庫讀取和寫入資料庫時用於暫存空間的 Amazon S3 儲存貯體或資料夾 (稱為 *tempS3Path*)。

> **Note**
>
> 在 AWS Glue 工作資料預覽中使用 Vertica 時，可能無法從 *tempS3Path* 中自動移除暫存檔案。若要確保移除暫存檔案，請選擇資料預覽窗格中的結束工作階段，直接結束資料預覽工作階段。
> 如果無法保證資料預覽工作階段會直接結束，請考慮設定 Amazon S3 生命週期組態以移除舊資料。根據最長任務執行期和餘裕，我們建議移除超過 49 小時的資料。如需有關設定 Amazon S3 生命週期的詳細資訊，請參閱《Amazon S3 文件》中的管理您的儲存空間生命週期。

- 您可以與 AWS Glue 任務角色建立關聯的 IAM 政策，其中具有 Amazon S3 的適當權限。
- 如果 Vertica 執行個體位於 Amazon VPC 中，請設定 Amazon VPC 以允許 AWS Glue 任務與 Vertica 執行個體通訊，使流量不會周遊公有網際網路。

  在 Amazon VPC 中，識別或建立 AWS Glue 將在執行任務時使用的 VPC、子網路及安全群組。此外，您也需要確保 Amazon VPC 已完成設定，以允許 Vertica 執行個體與此位置之間的網路流量。您的任務將需要與 Vertica 用戶端連接埠建立 TCP 連線 (預設 5433)。根據您的網路配置，這可能需要變更安全群組規則、網路 ACL、NAT 閘道及對等連線。

然後，您可以繼續設定 AWS Glue 以搭配 Vertica 使用。

設定連至 Vertica 的連線：

1. 在 AWS Secrets Manager 中，使用 Vertica 憑證 *verticaUsername* 和 *verticaPassword* 建立密碼。若要在 Secrets Manager 中建立機密，請遵循 AWS Secrets Manager 文件中建立 AWS Secrets Manager 機密中提供的教學課程。建立機密之後，請保留機密名稱 *secretName*，以便進行下一個步驟。

- 在選取鍵/值組時，請使用 *verticaUsername* 值來建立 user 金鑰對。
- 在選取鍵/值組時，請使用 *verticaPassword* 值來建立 password 金鑰對。

2. 在 AWS Glue 主控台中，依照 the section called “新增 AWS Glue 連線” 中的步驟建立連線。建立連線之後，請保留連線名稱 *connectionName*，以便進行下一個步驟。

   - 選取連線類型時，請選取 Vertica。
   - 選取 Vertica 主機後，請提供 Vertica 安裝的主機名稱。
   - 選取 Vertica 連接埠時，您可透過該連接埠安裝 Vertica。
   - 選取 AWS 機密時，請提供 *secretName*。

3. 在下列情況中，您可能需要其他組態：

   - Amazon VPC 中託管於 AWS 的 Vertica 執行個體
     - 向定義 Vertica 安全憑證的 AWS Glue 連線提供 Amazon VPC 連線資訊。建立或更新連線時，請在網路選項中設定 VPC、子網路及安全群組。

建立 AWS Glue Vertica 連線後，您將需要執行下列步驟，才能呼叫連線方法。

- 將與 AWS Glue 任務權限相關聯的 IAM 角色授予 *tempS3Path*。
- 授予與您 AWS Glue 任務許可相關聯的 IAM 角色，以讀取 *secretName*。
- 在您的 AWS Glue 任務組態中，提供 *connectionName* 作為其他網路連線。

從 Vertica 讀取

先決條件：

- 您想要讀取的 Vertica 資料表。您將需要 Vertica 資料庫名稱 *dbName* 和資料表名稱 *tableName*。
- 完成設定的 AWS Glue Vertica 連線，可提供驗證資訊。完成上一個程序設定連至 Vertica 的連線的步驟，以設定驗證資訊。您將會需要 AWS Glue 連線的名稱，*connectionName*。
- 用於暫存空間的 Amazon S3 儲存貯體或資料夾 (如前述)。您將需要名稱 *tempS3Path*。您將需要使用 s3a 通訊協定連線至此位置。

例如：

```
dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="vertica",
```

```
    connection_options={
        "connectionName": "connectionName",
        "staging_fs_url": "s3a://tempS3Path",
        "db": "dbName",
        "table": "tableName",
    }
)
```

您也可提供 SELECT SQL 查詢，以篩選傳回 DynamicFrame 的結果，或從多個資料表存取資料集。

例如：

```
dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="vertica",
    connection_options={
        "connectionName": "connectionName",
        "staging_fs_url": "s3a://tempS3Path",
        "db": "dbName",
        "query": "select * FROM tableName",
    },
)
```

寫入 Vertica 資料表

此範例會從現有的 DynamicFrame *dynamicFrame* 將資訊寫入 Vertica。如果資料表已具有資訊，則 AWS Glue 會從 DynamicFrame 附加資料。

先決條件：

- 您想要寫入的目前或所需資料表名稱 *tableName*。您也需要相應的 Vertica 資料庫名稱 *dbName*。
- 完成設定的 AWS Glue Vertica 連線，可提供驗證資訊。完成上一個程序設定連至 Vertica 的連線的步驟，以設定驗證資訊。您將會需要 AWS Glue 連線的名稱，*connectionName*。
- 用於暫存空間的 Amazon S3 儲存貯體或資料夾 (如前述)。您將需要名稱 *tempS3Path*。您將需要使用 `s3a` 通訊協定連線至此位置。

例如：

```
glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="vertica",
    connection_options={
```

```
        "connectionName": "connectionName",
        "staging_fs_url": "s3a://tempS3Path",
        "db": "dbName",
        "table": "tableName",
    }
)
```

Vertica 連線選項參考

- connectionName – 必要。用於讀取/寫入。完成設定之 AWS Glue Vertica 連線的名稱，可向連線方法提供驗證和網路資訊。
- db – 必要。用於讀取/寫入。您的連線方法將會互動的 Vertica 資料庫名稱。
- dbSchema：如需要識別資料表，則為必要項目。用於讀取/寫入。預設：public。您的連線方法將會互動的結構描述名稱。
- table：除非已提供 query，否則為寫入和讀取的必要項目。用於讀取/寫入。您的連線方法將會互動的資料表名稱。
- query – 用於讀取。定義從 Teradata 讀取時應擷取之內容的 SELECT SQL 查詢。
- staging_fs_url – 必要。用於讀取/寫入。有效值：s3a URL。用於暫存空間的 Amazon S3 儲存貯體或資料夾的 URL。

自訂和 AWS Marketplace connectionType 值

所需資訊包括下列項目：

- "connectionType": "marketplace.athena"：指定連至 Amazon Athena 資料存放區的連線。連線會使用來自 AWS Marketplace 的連接器。
- "connectionType": "marketplace.spark"：指定連至 Apache Spark 資料存放區的連線。連線會使用來自 AWS Marketplace 的連接器。
- "connectionType": "marketplace.jdbc"：指定連至 JDBC 資料存放區的連線。連線會使用來自 AWS Marketplace 的連接器。
- "connectionType": "custom.athena"：指定連至 Amazon Athena 資料存放區的連線。連線會使用您上傳至 AWS Glue Studio 的自訂連接器。
- "connectionType": "custom.spark"：指定連至 Apache Spark 資料存放區的連線。連線會使用您上傳至 AWS Glue Studio 的自訂連接器。
- "connectionType": "custom.jdbc"：指定連至 JDBC 資料存放區的連線。連線會使用您上傳至 AWS Glue Studio 的自訂連接器。

類型 custom.jdbc 或 marketplace.jdbc 的連線選項

- `className` – 字串、必要，驅動程式類別名稱。
- `connectionName` – 字串、必要，與連接器相關聯之連線的名稱。
- `url` – 字串、必要，具有預留位置 (`${}`) 的 JDBC URL，用來建立與資料來源的連線。預留位置 `${secretKey}` 會替換為 AWS Secrets Manager 中相同名稱的秘密。如需有關建構 URL 的更多資訊，請參閱資料存放區文件。
- `secretId` 或 `user/password` – 字串、必要，用於擷取 URL 憑證。
- `dbTable` 或 `query` – 字串、必要，要從中取得資料的資料表或 SQL 查詢。您可以指定 `dbTable` 或 `query`，但不能同時指定兩者。
- `partitionColumn` - 字串、選用，用於分割的整數欄名稱。此選項僅適用於包含在 `lowerBound`、`upperBound` 以及 `numPartitions` 中。此選項的運作方式與 Spark SQL JDBC 讀取器相同。如需詳細資訊，請參閱 Apache Spark SQL、DataFrames 和資料集指南中的 JDBC 至其他資料庫。

  `lowerBound` 和 `upperBound` 值用於決定分割區步幅，而不是用於篩選資料表中的列。資料表中的所有列都會進行分割並傳回。

  > **Note**
  >
  > 使用查詢而不是資料表名稱時，您應該驗證查詢是否適用於指定的分割條件。例如：
  >
  > - 如果您的查詢格式為 `"SELECT col1 FROM table1"`，則透過在使用分割區欄的查詢結尾附加 `WHERE` 子句來測試查詢。
  > - 如果您的查詢格式為 `"SELECT col1 FROM table1 WHERE col2=val"`，則透過使用 `AND` 和使用分割區欄的表達式擴展 `WHERE` 子句來測試查詢。

- `lowerBound` - 整數、選用，用來決定分割區步幅的 `partitionColumn` 最小值。
- `upperBound` - 整數、選用，用來決定分割區步幅的 `partitionColumn` 最大值。
- `numPartitions` - 整數、選用，分割區數目。這個值，搭配 `lowerBound` (包含) 及 `upperBound` (不含)，形成用於分割 `partitionColumn` 而產生之 `WHERE` 子句表達式的分割區步幅。

  > **Important**
  >
  > 請小心分割區的數目，因為太多的分割區可能會造成外部資料庫系統的問題。

- `filterPredicate` - 字串、選用，額外條件子句，用於篩選來源的資料。例如：

```
BillingCity='Mountain View'
```

當您使用查詢，而不是資料表名稱，您應該驗證查詢是否適用於指定的 `filterPredicate`。例如：

- 如果您的查詢格式為 `"SELECT col1 FROM table1"`，則透過在使用篩選述詞的查詢結尾附加 `WHERE` 子句來測試查詢。
- 如果您的查詢格式為 `"SELECT col1 FROM table1 WHERE col2=val"`，則透過使用 `AND` 和使用篩選述詞的表達式擴展 `WHERE` 子句來測試查詢。

- `dataTypeMapping` - 字典、選用、自訂資料類型映射，用於建構從 JDBC 資料類型到 Glue 資料類型的映射。例如，選項 `"dataTypeMapping":{"FLOAT":"STRING"}` 會將 JDBC 類型 `FLOAT` 的資料欄位映射至 Java `String` 類型中，方法是呼叫驅動程式的 `ResultSet.getString()` 方法，並使用它來建構 AWS Glue 記錄。`ResultSet` 物件是由每個驅動程式實作，因此行為是特定於您使用的驅動程式。請參閱 JDBC 驅動程式的文件，瞭解驅動程式如何執行轉換。
- 目前支援的 AWS Glue 資料類型為：
  - DATE
  - STRING
  - TIMESTAMP
  - INT
  - FLOAT
  - LONG
  - BIGDECIMAL
  - BYTE
  - SHORT
  - DOUBLE

  支援的 JDBC 資料類型為 Java8 java.sql.types。

  預設資料類型映射 (從 JDBC 到 AWS Glue) 是：

  - DATE -> DATE
  - VARCHAR -> STRING
  - CHAR -> STRING
  - LONGNVARCHAR -> STRING

- TIMESTAMP -> TIMESTAMP
- INTEGER -> INT
- FLOAT -> FLOAT
- REAL -> FLOAT
- BIT -> BOOLEAN
- BOOLEAN -> BOOLEAN
- BIGINT -> LONG
- DECIMAL -> BIGDECIMAL
- NUMERIC -> BIGDECIMAL
- TINYINT -> SHORT
- SMALLINT -> SHORT
- DOUBLE -> DOUBLE

如果您使用自訂資料類型映射搭配選項 `dataTypeMapping`，那麼您可以覆寫預設的資料類型映射。只有 `dataTypeMapping` 選項中的 JDBC 資料類型會受到影響；預設映射會用於所有其他 JDBC 資料類型。如果需要，您可以為其他 JDBC 資料類型新增映射。如果 JDBC 資料類型未包含在預設映射或自訂映射中，則資料類型預設會轉換為 AWS Glue `STRING` 資料類型。

下面的 Python 程式碼範例示範如何使用 AWS Marketplace JDBC 驅動程式從 JDBC 資料庫讀取。它演示了從資料庫讀取和寫入 S3 位置。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
## @type: DataSource
```

```
    ## @args: [connection_type = "marketplace.jdbc", connection_options =
     {"dataTypeMapping":{"INTEGER":"STRING"},"upperBound":"200","query":"select id,
       name, department from department where id < 200","numPartitions":"4",
       "partitionColumn":"id","lowerBound":"0","connectionName":"test-connection-
jdbc"},
        transformation_ctx = "DataSource0"]
    ## @return: DataSource0
    ## @inputs: []
    DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
      "marketplace.jdbc", connection_options = {"dataTypeMapping":{"INTEGER":"STRING"},
      "upperBound":"200","query":"select id, name, department from department where
       id < 200","numPartitions":"4","partitionColumn":"id","lowerBound":"0",
       "connectionName":"test-connection-jdbc"}, transformation_ctx = "DataSource0")
    ## @type: ApplyMapping
    ## @args: [mappings = [("department", "string", "department", "string"), ("name",
 "string",
     "name", "string"), ("id", "int", "id", "int")], transformation_ctx =
 "Transform0"]
    ## @return: Transform0
    ## @inputs: [frame = DataSource0]
    Transform0 = ApplyMapping.apply(frame = DataSource0, mappings = [("department",
 "string",
     "department", "string"), ("name", "string", "name", "string"), ("id", "int",
 "id", "int")],
      transformation_ctx = "Transform0")
    ## @type: DataSink
    ## @args: [connection_type = "s3", format = "json", connection_options = {"path":
     "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0"]
    ## @return: DataSink0
    ## @inputs: [frame = Transform0]
    DataSink0 = glueContext.write_dynamic_frame.from_options(frame = Transform0,
      connection_type = "s3", format = "json", connection_options = {"path":
      "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0")
    job.commit()
```

類型 custom.athena 或 marketplace.athena 的連線選項

- `className` – 字串、必要，驅動程式類別名稱。當您使用 Athena-CloudWatch 連接器時，此參數值就是類別名稱的字首 (例如 `"com.amazonaws.athena.connectors"`)。Athena-CloudWatch 連接器由兩個類別組成：中繼資料處理常式和記錄處理常式。如果您在此處提供通用字首，則 API 會根據該字首載入正確的類別。

- tableName – 字串、必要，要讀取的 CloudWatch 日誌串流名稱。此程式碼片段使用特殊檢視名稱 all_log_streams，這表示傳回的動態資料框架將包含來自日誌群組中所有日誌資料串流的資料。
- schemaName – 字串、必要，要讀取的 CloudWatch 日誌群組。例如：/aws-glue/jobs/output。
- connectionName – 字串、必要，與連接器相關聯之連線的名稱。

如需此連接器的其他選項，請參閱 Amazon Athena CloudWatch 連接器讀我檔案。

下列 Python 程式碼範例示範如何使用 AWS Marketplace 連接器從 Athena 資料存放區讀取。它展示了從 Athena 讀取和寫入 S3 位置。

```
    import sys
    from awsglue.transforms import *
    from awsglue.utils import getResolvedOptions
    from pyspark.context import SparkContext
    from awsglue.context import GlueContext
    from awsglue.job import Job

    ## @params: [JOB_NAME]
    args = getResolvedOptions(sys.argv, ['JOB_NAME'])

    sc = SparkContext()
    glueContext = GlueContext(sc)
    spark = glueContext.spark_session
    job = Job(glueContext)
    job.init(args['JOB_NAME'], args)
    ## @type: DataSource
    ## @args: [connection_type = "marketplace.athena", connection_options =
     {"tableName":"all_log_streams","schemaName":"/aws-glue/jobs/output",
      "connectionName":"test-connection-athena"}, transformation_ctx = "DataSource0"]
    ## @return: DataSource0
    ## @inputs: []
    DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
      "marketplace.athena", connection_options = {"tableName":"all_log_streams",,
      "schemaName":"/aws-glue/jobs/output","connectionName":
      "test-connection-athena"}, transformation_ctx = "DataSource0")
    ## @type: ApplyMapping
    ## @args: [mappings = [("department", "string", "department", "string"), ("name",
"string",
     "name", "string"), ("id", "int", "id", "int")], transformation_ctx =
"Transform0"]
```

```
    ## @return: Transform0
    ## @inputs: [frame = DataSource0]
    Transform0 = ApplyMapping.apply(frame = DataSource0, mappings = [("department",
"string",
      "department", "string"), ("name", "string", "name", "string"), ("id", "int",
"id", "int")],
       transformation_ctx = "Transform0")
    ## @type: DataSink
    ## @args: [connection_type = "s3", format = "json", connection_options = {"path":
     "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0"]
    ## @return: DataSink0
    ## @inputs: [frame = Transform0]
    DataSink0 = glueContext.write_dynamic_frame.from_options(frame = Transform0,
      connection_type = "s3", format = "json", connection_options = {"path":
      "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0")
    job.commit()
```

類型 custom.spark 或 marketplace.spark 的連接選項

- `className` – 字串、必要，連接器類別名稱。
- `secretId` – 字串、選用，用來擷取連接器連線的憑證。
- `connectionName` – 字串、必要，與連接器相關聯之連線的名稱。
- 其他選項視資料存放區而定。例如，OpenSearch 組態選項以字首 `es` 開頭，如 Elasticsearch for Apache Hadoop 文件中所述。Spark 連接到 Snowflake 使用選項，例如 `sfUser` 和 `sfPassword`，如連接到 Snowflake 指南中的使用 Spark Connector 所述。

下列 Python 程式碼範例示範如何使用 `marketplace.spark` 連線從 OpenSearch 資料存放區讀取。

```
    import sys
    from awsglue.transforms import *
    from awsglue.utils import getResolvedOptions
    from pyspark.context import SparkContext
    from awsglue.context import GlueContext
    from awsglue.job import Job

    ## @params: [JOB_NAME]
    args = getResolvedOptions(sys.argv, ['JOB_NAME'])

    sc = SparkContext()
    glueContext = GlueContext(sc)
    spark = glueContext.spark_session
```

```
    job = Job(glueContext)
    job.init(args['JOB_NAME'], args)
    ## @type: DataSource
    ## @args: [connection_type = "marketplace.spark", connection_options =
{"path":"test",
      "es.nodes.wan.only":"true","es.nodes":"https://<AWS endpoint>",
      "connectionName":"test-spark-es","es.port":"443"}, transformation_ctx =
"DataSource0"]
    ## @return: DataSource0
    ## @inputs: []
    DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
      "marketplace.spark", connection_options = {"path":"test","es.nodes.wan.only":
      "true","es.nodes":"https://<AWS endpoint>","connectionName":
      "test-spark-es","es.port":"443"}, transformation_ctx = "DataSource0")
    ## @type: DataSink
    ## @args: [connection_type = "s3", format = "json", connection_options = {"path":
         "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0"]
    ## @return: DataSink0
    ## @inputs: [frame = DataSource0]
    DataSink0 = glueContext.write_dynamic_frame.from_options(frame = DataSource0,
       connection_type = "s3", format = "json", connection_options = {"path":
       "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0")
    job.commit()
```

一般選項

本節中的選項作為 connection_options 提供，但並不特別套用於一個連接器。

設定書籤時，通常會使用下列參數。它們可能適用於 Amazon S3 或 JDBC 工作流程。如需更多詳細資訊，請參閱 the section called “使用任務書籤”。

- jobBookmarkKeys：資料欄名稱陣列。
- jobBookmarkKeysSortOrder：定義如何根據排序順序比較值的字串。有效值："asc"、"desc"。
- useS3ListImplementation：用於在列出 Amazon S3 儲存貯體內容時管理記憶體效能。如需詳細資訊，請參閱 Optimize memory management in AWS Glue。

AWS Glue for Spark 中的輸入與輸出的資料格式選項

這些頁面提供有關 AWS Glue for Spark 支援的資料格式之功能支援和組態參數的資訊。有關此信息的用法和適用性的說明，請參閱以下內容。

跨資料格式的功能支援AWSGlue

每種數據格式可能支持不同AWSGlue 功能。根據您的格式類型，可能會或可能不支持以下常見功能。請參閱資料格式的文件，瞭解如何運用我們的功能來滿足您的需求。

讀取	AWSGlue 可以識別和解釋這種數據格式，而無需額外的資源，例如連接器。
寫入	AWSGlue 可以在沒有額外資源的情況下以這種格式寫入數據。您可以在工作中包含協力廠商程式庫，並使用標準 Apache Spark 函數來寫入資料，就像在其他 Spark 環境中一樣。如需這些程式庫的詳細資訊，請參閱 the section called “Python 程式庫”。
串流讀取	AWSGlue 可以識別和解釋這種資料格式從 Apache Kafka、Amazon Managed Streaming for Apache Kafka 或 Amazon Kinesis 訊息流。我們希望流以一致的格式呈現數據，因此它們被讀為`DataFrames` 。
對小型檔案進行分組	AWSGlue 可以將文件組合在一起，以便在執行時批量發送到每個節點的工作AWSGlue 轉換。如此可大幅改善涉及大量小型檔案的工作負載的效能。如需詳細資訊，請參閱 the section called “群組輸入檔案”。
任務書籤	AWSGlue 可以透過工作書籤，追蹤在工作執行期間，在相同資料集上執行相同工作的轉換進度。這可以提高工作負載的效能，涉及自上次作業執行以來只需在新資料上完成工作的資料集。如需詳細資訊，請參閱 the section called “使用任務書籤追蹤處理的資料”。

用於與中的資料格式互動的參數AWSGlue

某些AWSGlue 連接類型支持多種format類型，要求您指定有關您的數據格式的信息format_options使用類似方法時的對象GlueContext.write_dynamic_frame.from_options。

- s3 – 如需詳細資訊，請參閱 AWS 中的 ETL 連線類型和選項：S3 連線參數您也可以檢視促進此連線類型之方法的文件：the section called “create_dynamic_frame_from_options”和the section called “write_dynamic_frame_from_options”在 Python 和相應的斯卡拉方法the section called “getSourceWith格式”和the section called “getSinkWith格式”。

- kinesis – 如需詳細資訊，請參閱 AWS 中的 ETL 連線類型和選項：Kinesis 連線參數您也可以檢視促進此連線類型的方法文件：the section called “create_data_frame_from_options”和相應的斯卡拉方法the section called “createDataFrameFromOptions”。
- kafka – 如需詳細資訊，請參閱 AWS 中的 ETL 連線類型和選項：Kafka 連線參數您也可以檢視促進此連線類型的方法文件：the section called “create_data_frame_from_options”和相應的斯卡拉方法the section called “createDataFrameFromOptions”。

某些連線類型不需要format_options。例如，在正常使用中，與關聯式資料庫的 JDBC 連線將以一致的表格式擷取資料。因此，從 JDBC 連接讀取不需要format_options。

在膠水中讀取和寫入數據的某些方法不需要format_options。例如，使用GlueContext.create_dynamic_frame.from_catalog取代為AWSGlue 爬蟲程式。爬蟲確定數據的形狀。使用爬蟲時，AWSGlue 分類器將檢查您的數據，以做出有關如何表示數據格式的明智決策。然後，它會將您的數據的表示存儲在AWSGlue 資料型錄，可在AWSGlue ETL 腳本來檢索您的數據GlueContext.create_dynamic_frame.from_catalog方法。檢索器無需手動指定有關數據格式的信息。

對於存取受 AWS Lake Formation 管控資料表的任務，AWS Glue 支援讀取和寫入受 Lake Formation 管控之資料表所支援的所有格式。如需受 AWS Lake Formation 管控之資料表支援的格式清單，請參閱《AWS Lake Formation 開發人員指南》中的受管控資料表的注意事項與限制。

> **Note**
>
> 對於寫入 Apache Parquet，AWS Glue ETL 只支援透過指定為動態框架優化的自訂 Parquet 寫入器類型的選項寫入受管控資料表。使用 parquet 格式寫入受管控資料表時，您應該在表參數中新增值為 true 的金鑰 useGlueParquetWriter。

主題

在 AWS Glue 中使用 CSV 格式

AWS Glue 會從各來源擷取資料，並將資料寫入以各種資料格式儲存和傳輸的目標。如果您的資料以 CSV 資料格式存放或傳輸，本文件將向您介紹在 AWS Glue 中使用資料的可用功能。

AWS Glue 支援使用逗號分隔值 (CSV) 格式。此格式是一種極小的、以列為基礎的資料格式。CSV 通常不會嚴格符合標準，但您可以參閱 RFC 4180 和 RFC 7111 來了解詳細資訊。

您可以使用 AWS Glue 從 Amazon S3 和串流來源讀取 CSV，並將 CSV 寫入 Amazon S3。您可以讀取和寫入來自 S3 的包含 CSV 檔案的 `bzip` 和 `gzip` 封存。您可以在 S3 連線參數 上設定壓縮行為，而不是在本頁討論的組態中設定。

下表顯示支援 CSV 格式選項的常見 AWS Glue 功能。

讀取	寫入	串流讀取	對小型檔案進行分組	任務書籤
支援	支援	支援	支援	支援

範例：從 S3 讀取 CSV 檔案或資料夾

先決條件：您需要指向希望讀取的 CSV 檔案或資料夾的 S3 路徑 (`s3path`)。

組態：在您的函數選項中，指定 format="csv"。在您的 connection_options 中，使用 paths 鍵來指定 s3path。您可以在 connection_options 中設定讀取器與 S3 互動的方式。如需詳細資訊，請參閱 AWS Glue 中 ETL 的連線類型和選項：S3 連線參數。您可以在 format_options 中設定讀取器解譯 CSV 的方式。如需詳細資訊，請參閱 CSV 組態參考。

以下 AWS Glue ETL 指令碼顯示從 S3 讀取 CSV 檔案或資料夾的流程。

我們透過 optimizePerformance 組態鍵為常見工作流提供經過效能最佳化的自訂 CSV 讀取器。若要判斷此讀取器是否適合您的工作負載，請參閱 the section called “使用優化的 CSV 讀取器”。

Python

在此範例中，使用 create_dynamic_frame.from_options 方法。

```
# Example: Read CSV from S3
# For show, we handle a CSV with a header row.  Set the withHeader option.
# Consider whether optimizePerformance is right for your workflow.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="csv",
    format_options={
        "withHeader": True,
        # "optimizePerformance": True,
    },
)
```

您也可以在指令碼中使用 DataFrames (pyspark.sql.DataFrame)。

```
dataFrame = spark.read\
    .format("csv")\
    .option("header", "true")\
    .load("s3://s3path")
```

Scala

在此範例中，使用 getSourceWithFormat 操作。

```
// Example: Read CSV from S3
// For show, we handle a CSV with a header row.  Set the withHeader option.
// Consider whether optimizePerformance is right for your workflow.

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      formatOptions=JsonOptions("""{"withHeader": true}"""),
      connectionType="s3",
      format="csv",
      options=JsonOptions("""{"paths": ["s3://s3path"], "recurse": true}""")
    ).getDynamicFrame()
  }
}
```

您也可以在指令碼中使用 DataFrames (`org.apache.spark.sql.DataFrame`)。

```
val dataFrame = spark.read
  .option("header","true")
  .format("csv")
  .load("s3://s3path“)
```

範例：將 CSV 檔案和資料夾寫入 S3

先決條件：您需要已初始化的 DataFrame (`dataFrame`) 或 DynamicFrame (`dynamicFrame`)。您還需要預期的 S3 輸出路徑 `s3path`。

組態：在您的函數選項中，指定 `format="csv"`。在您的 `connection_options` 中，使用 `paths` 鍵來指定 `s3path`。您可以在 `connection_options` 中設定寫入器與 S3 的互動方式。如需詳細資

訊，請參閱 AWS Glue 中 ETL 的連線類型和選項：S3 連線參數。您可以在 format_options 中設定操作如何寫入檔案內容。如需詳細資訊，請參閱 CSV 組態參考。以下 AWS Glue ETL 指令碼顯示將 CSV 檔案和資料夾寫入 S3 的流程。

Python

在此範例中，使用 write_dynamic_frame.from_options 方法。

```
# Example: Write CSV to S3
# For show, customize how we write string type values.  Set quoteChar to -1 so our
 values are not quoted.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    connection_options={"path": "s3://s3path"},
    format="csv",
    format_options={
        "quoteChar": -1,
    },
)
```

您也可以在指令碼中使用 DataFrames (pyspark.sql.DataFrame)。

```
dataFrame.write\
    .format("csv")\
    .option("quote", None)\
    .mode("append")\
    .save("s3://s3path")
```

Scala

在此範例中，使用 getSinkWithFormat 方法。

```
// Example: Write CSV to S3
```

```
// For show, customize how we write string type values. Set quoteChar to -1 so our
 values are not quoted.

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
        connectionType="s3",
        options=JsonOptions("""{"path": "s3://s3path"}"""),
        format="csv"
    ).writeDynamicFrame(dynamicFrame)
  }
}
```

您也可以在指令碼中使用 DataFrames (`org.apache.spark.sql.DataFrame`)。

```
dataFrame.write
    .format("csv")
    .option("quote", null)
    .mode("Append")
    .save("s3://s3path")
```

CSV 組態參考

您都可以使用下列 `format_options`，無論 AWS Glue 程式庫在何處指定 `format="csv"`：

- `separator` – 指定分隔符號字元。預設值為逗號，但您仍可指定任何其他字元。
  - 類型：文字，預設：`","`
- `escaper` – 指定用於逸出的字元。只有在讀取 (而非寫入) CSV 檔案時，才會使用此選項。若啟用，後面緊接的字元會維持現狀，一小組已知的逸出字元 (`\n`、`\r`、`\t` 與 `\0`) 除外。
  - 類型：文字，預設：無
- `quoteChar` – 指定用於引用的字元。預設為雙引號。將之設為 `-1` 可完全關閉引用功能。
  - 類型：文字，預設：`'"'`

- multiLine – 指定單項記錄是否可以跨越多行。當欄位內含引用的新行字元時，可能就會發生這種情況。若有任何記錄跨越多行，請務必將此選項設為 True。啟用 multiLine 可能會降低性能，因為在剖析時需要更加謹慎的檔案分割。
  - 類型：布林值，預設：false
- withHeader – 指定是否要將第一行做為標頭。DynamicFrameReader 類別可使用該選項。
  - 類型：布林值，預設：false
- writeHeader – 指定是否要將標頭寫入輸出之中。DynamicFrameWriter 類別可使用該選項。
  - 類型：布林值，預設：true
- skipFirst – 指定是否要略過第一個資料行。
  - 類型：布林值，預設：false
- optimizePerformance – 指定是否要使用進階 SIMD CSV 讀取器，以及以 Apache Arrow 為基礎的直欄式記憶體格式。僅適用於 AWS Glue 3.0+。
  - 類型：布林值，預設：false
- strictCheckForQuoting：寫入 CSV 時，Glue 可能會在解讀為字串的值中加上引號。這樣做是為了防止寫出的內容模糊。為了節省時間，決定要寫入的內容時，Glue 可能會在不需要引號的某些情況下進行引用。啟用嚴格檢查將執行更密集的運算，並且僅在必要時引用。僅適用於 AWS Glue 3.0+。
  - 類型：布林值，預設：false

使用向量化 SIMD CSV 讀取器最佳化讀取效能

AWS Glue 3.0 版本添加了一個經最佳化的 CSV 讀取器，與以資料行為基礎 CSV 讀取器相比，可以大幅加快整體任務效能。

經最佳化的閱讀器：

- 使用 CPU SIMD 指令從磁碟讀取
- 立即以直欄式格式 (Apache Arrow) 將記錄寫入記憶體
- 將記錄分成多個批次

這樣可以節省稍後將記錄分批或轉換為直欄式格式的處理時間。有些範例是變更結構描述或依資料行擷取資料時。

要使用經最佳化的讀取器，請在 format_options 或資料表屬性中將 "optimizePerformance" 設為 true。

```
glueContext.create_dynamic_frame.from_options(
    frame = datasource1,
    connection_type = "s3",
    connection_options = {"paths": ["s3://s3path"]},
    format = "csv",
    format_options={
        "optimizePerformance": True,
        "separator": ","
        },
    transformation_ctx = "datasink2")
```

向量化 CSV 讀取器的限制

請注意向量化 CSV 讀取器的下列限制：

- 它不支援 `multiLine` 和 `escaper` 格式選項。它使用雙引號字元 `'"'` 的預設 `escaper`。設定這些選項後，AWS Glue 會自動回退到使用以列為基礎的 CSV 讀取器。
- 它不支援建立具有 ChoiceType 的 DynamicFrame。
- 它不支援建立具有錯誤記錄的 DynamicFrame。
- 它不支援讀取含有多位元組字元 (例如日文或中文字元) 的 CSV 檔案。

在 AWS Glue 中使用 Parquet 格式

AWS Glue 會從各來源擷取資料，並將資料寫入以各種資料格式儲存和傳輸的目標。如果您的資料以 Parquet 資料格式存放或傳輸，本文件將向您介紹在 AWS Glue 中使用資料的可用功能。

AWS Glue 支援使用 Parquet 格式。此格式是效能導向、以資料行為基礎的資料格式。如需標準授權單位的格式簡介，請參閱 Apache Parquet Documentation Overview (Apache Parquet 文件概觀)。

您可以使用 AWS Glue 從 Amazon S3 和串流來源讀取 Parquet 檔案，並將 Parquet 檔案寫入 Amazon S3。您可以從 S3 讀取和寫入包含 Parquet 檔案的 `bzip` 和 `gzip` 封存檔。您可以在 S3 連線參數 上設定壓縮行為，而不是在本頁討論的組態中設定。

下表顯示支援 Parquet 格式選項的常見 AWS Glue 功能。

讀取	寫入	串流讀取	對小型檔案進行分組	任務書籤
支援	支援	支援	不支援	支援*

* 在 AWS Glue 1.0 或更新版本中支援

範例：從 S3 讀取 Parquet 檔案或資料夾

先決條件：您需要指向希望讀取的 Parquet 檔案或資料夾的 S3 路徑 (s3path)。

組態：在您的函數選項中，指定 format="parquet"。在您的 connection_options 中，使用 paths 索引鍵指定 s3path。

您可以在 connection_options 中設定讀取器與 S3 的互動方式。如需詳細資訊，請參閱 AWS Glue 中 ETL 的連線類型和選項：S3 連線參數。

您可以在 format_options 中設定讀取器解譯 Parquet 檔案的方式。如需詳細資訊，請參閱 Parquet 組態參考。

以下 AWS Glue ETL 指令碼顯示從 S3 讀取 Parquet 檔案或資料夾的流程：

Python

在此範例中，使用 create_dynamic_frame.from_options 方法。

```
# Example: Read Parquet from S3

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type = "s3",
    connection_options = {"paths": ["s3://s3path/"]},
    format = "parquet"
)
```

您也可以在指令碼中使用 DataFrames (pyspark.sql.DataFrame)。

```
dataFrame = spark.read.parquet("s3://s3path/")
```

Scala

在此範例中，使用 getSourceWithFormat 方法。

```
// Example: Read Parquet from S3

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType="s3",
      format="parquet",
      options=JsonOptions("""{"paths": ["s3://s3path"]}""")
    ).getDynamicFrame()
  }
}
```

您也可以在指令碼中使用 DataFrames (`org.apache.spark.sql.DataFrame`)。

```
spark.read.parquet("s3://s3path/")
```

範例：將 Parquet 檔案和資料夾寫入 S3

先決條件：您需要初始化 DataFrame (`dataFrame`) 或 DynamicFrame (`dynamicFrame`)。您還需要預期的 S3 輸出路徑 `s3path`。

組態：在您的函數選項中，指定 `format="parquet"`。在您的 `connection_options` 中，使用 `paths` 鍵來指定 `s3path`。

您可以在 `connection_options` 中進一步更改寫入器與 S3 的互動方式。如需詳細資訊，請參閱 AWS Glue 中 ETL 的連線類型和選項：S3 連線參數。您可以在 `format_options` 中設定操作如何寫入檔案內容。如需詳細資訊，請參閱 Parquet 組態參考。

以下 AWS Glue ETL 指令碼顯示將 Parquet 檔案和資料夾寫入 S3 的流程。

我們透過 `useGlueParquetWriter` 組態鍵為自訂 Parquet 寫入器提供 DynamicFrames 的效能最佳化。若要判斷此寫入器是否適合您的工作負載，請參閱 Glue Parquet 寫入器。

Python

在此範例中，使用 write_dynamic_frame.from_options 方法。

```
# Example: Write Parquet to S3
# Consider whether useGlueParquetWriter is right for your workflow.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    format="parquet",
    connection_options={
        "path": "s3://s3path",
    },
    format_options={
        # "useGlueParquetWriter": True,
    },
)
```

您也可以在指令碼中使用 DataFrames (`pyspark.sql.DataFrame`)。

```
df.write.parquet("s3://s3path/")
```

Scala

在此範例中，使用 getSinkWithFormat 方法。

```
// Example: Write Parquet to S3
// Consider whether useGlueParquetWriter is right for your workflow.

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext
```

```
object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
        connectionType="s3",
        options=JsonOptions("""{"path": "s3://s3path"}"""),
        format="parquet"
    ).writeDynamicFrame(dynamicFrame)
  }
}
```

您也可以在指令碼中使用 DataFrames (`org.apache.spark.sql.DataFrame`)。

```
df.write.parquet("s3://s3path/")
```

Parquet 組態參考

您都可以使用下列 `format_options`，無論 AWS Glue 程式庫在何處指定 `format="parquet"`：

- `useGlueParquetWriter` – 指定使用具有 DynamicFrame 工作流程效能最佳化的自訂 Parquet 寫入器。如需用量詳細資訊，請參閱 Glue Parquet 寫入器。
  - 類型：布林值，預設：`false`
- `compression` – 指定使用的壓縮轉碼器。值與 `org.apache.parquet.hadoop.metadata.CompressionCodecName` 完全相容。
  - 類型：列舉文字，預設：`"snappy"`
  - 值：`"uncompressed"`、`"snappy"`、`"gzip"` 和 `"lzo"`
- `blockSize` – 指定記憶體中將緩衝處理資料列群組的大小，以位元組為單位。您可以使用該值來調校效能。大小應完全分為多個 MB 單位。
  - 類型：數值，預設：`134217728`
  - 預設值等於 128 MB。
- `pageSize` – 指定分頁的大小，以位元組為單位。您可以使用該值來調校效能。分頁是必須完整讀取才能存取單一記錄之最小單位。
  - 類型：數值，預設：`1048576`
  - 預設值等於 1 MB。

> **Note**
>
> 此外，任何底層的 SparkSQL 程式碼所接受的選項均可透過 `connection_options` 對應參數傳送。例如，可為 AWS Glue Spark 讀取器設定 mergeSchema 等 Spark 組態以合併所有檔案的結構描述。

使用 AWS Glue Parquet 寫入器以最佳化寫入效能

> **Note**
>
> 過去，AWS Glue Parquet 寫入器都透過 `glueparquet` 格式類型存取。但現在不再提倡這種存取模式。請改用已啟用 `useGlueParquetWriter` 的 `parquet` 類型。

AWS Glue Parquet 寫入器具有效能增強功能，可加快 Parquet 檔案寫入速度。傳統的寫入器會在寫入之前計算結構描述。Parquet 格式不會以可供快速擷取的方式存放結構描述，因此這可能需要一些時間。使用 AWS Glue Parquet 寫入器，無需預先計算的結構描述。當資料送達時，寫入器會動態地計算並修改結構描述。

指定 `useGlueParquetWriter` 時，請注意以下限制：

- 寫入器僅支援結構描述演變 (例如新增或移除資料行)，但不會變更資料行類型，例如使用 `ResolveChoice`。
- 寫入器不支援寫入空白的 DataFrame，例如，寫入只含結構描述的檔案。透過設定 `enableUpdateCatalog=True` 與 AWS Glue 資料目錄整合時，嘗試寫入空白的 DataFrame 不會更新資料目錄，而會在資料目錄建立不含結構描述的資料表。

如果您的轉換不需要這些限制，則開啟 AWS Glue Parquet 寫入器會提升效能。

在 AWS Glue 中使用 XML 格式

AWS Glue 會從各來源擷取資料，並將資料寫入以各種資料格式儲存和傳輸的目標。如果您的資料以 XML 資料格式存放或傳輸，本文件將向您介紹在 AWS Glue 中使用資料的可用功能。

AWS Glue 支援使用 XML 格式。此格式表示並非基於資料列或資料行的、高度可設定且嚴格定義的資料結構。XML 是高度標準化的。如需標準授權單位的格式簡介，請參閱 XML Essentials (XML 基礎知識)。

您可以使用 AWS Glue 從 Amazon S3 讀取 XML 檔案以及包含 XML 檔案的 bzip 和 gzip 封存檔。您可以在 S3 連線參數 上設定壓縮行為，而不是在本頁討論的組態中設定。

下表顯示支援 XML 格式選項的常見 AWS Glue 功能。

讀取	寫入	串流讀取	對小型檔案進行分組	任務書籤
支援	不支援	不支援	支援	支援

範例：從 S3 讀取 XML

XML 讀取器接受 XML 標籤名稱。它會檢查其輸入中包含該標籤的元素，以推斷結構描述，並使用對應的值填入 DynamicFrame。AWS Glue XML 功能的行為與 XML Data Source for Apache Spark 類似。您可以透過比較此讀取器與該專案的文件，來獲得基本行為相關的洞見。

先決條件：您需要指向希望讀取的 XML 檔案或資料夾的 S3 路徑 (s3path) 以及有关 XML 檔案的部分資訊。您還需要希望讀取的 XML 元素的標籤，xmlTag。

組態：在您的函數選項中，指定 format="xml"。在您的 connection_options 中，使用 paths 鍵來指定 s3path。您可以在 connection_options 中进一步設定讀取器與 S3 的互動方式。如需詳細資訊，請參閱 AWS Glue 中 ETL 的連線類型和選項：S3 連線參數。在您的 format_options 中，使用 rowTag 鍵來指定 xmlTag。您可以在 format_options 中进一步設定讀取器解譯 XML 檔案的方式。如需詳細資訊，請參閱 XML 組態參考。

以下 AWS Glue ETL 指令碼顯示從 S3 讀取 XML 檔案或資料夾的流程。

Python

在此範例中，使用 create_dynamic_frame.from_options 方法。

```
# Example: Read XML from S3
# Set the rowTag option to configure the reader.

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

dynamicFrame = glueContext.create_dynamic_frame.from_options(
```

```
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="xml",
    format_options={"rowTag": "xmlTag"},
)
```

您也可以在指令碼中使用 DataFrames (pyspark.sql.DataFrame)。

```
dataFrame = spark.read\
    .format("xml")\
    .option("rowTag", "xmlTag")\
    .load("s3://s3path")
```

Scala

在此範例中，使用 getSourceWithFormat 操作。

```
// Example: Read XML from S3
// Set the rowTag option to configure the reader.

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.GlueContext
import org.apache.spark.sql.SparkSession

val glueContext = new GlueContext(SparkContext.getOrCreate())
val sparkSession: SparkSession = glueContext.getSparkSession

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val dynamicFrame = glueContext.getSourceWithFormat(
      formatOptions=JsonOptions("""{"rowTag": "xmlTag"}"""),
      connectionType="s3",
      format="xml",
      options=JsonOptions("""{"paths": ["s3://s3path"], "recurse": true}""")
    ).getDynamicFrame()
}
```

您也可以在指令碼中使用 DataFrames (org.apache.spark.sql.DataFrame)。

```
val dataFrame = spark.read
  .option("rowTag", "xmlTag")
  .format("xml")
```

```
    .load("s3://s3path“)
```

XML 組態參考

您都可以使用下列 format_options，無論 AWS Glue 程式庫在何處指定 format="xml"：

- rowTag – 指定要將其當做一列的檔案中的 XML 標籤。Row 標籤不能自動關閉。
  - 類型：文字，必要
- encoding – 指定字元編碼。它可以是我们的執行時間環境支援的 Charset 名稱或別名。我們不對編碼支援做出具體保證，但主要編碼應可適用。
  - 類型：文字，預設："UTF-8"
- excludeAttribute – 指定是否要排除元素中的屬性。
  - 類型：布林值，預設：false
- treatEmptyValuesAsNulls – 指定是否要將空格當做 null 值。
  - 類型：布林值，預設：false
- attributePrefix – 屬性的字首，用以和子元素文字作區別。此前綴用於欄位名稱。
  - 類型：文字，預設："_"
- valueTag – 當無子元素的元素有屬性時，此標籤會當做一個值。
  - 類型：文字，預設："_VALUE"
- ignoreSurroundingSpaces – 指定是否要忽略前後均有值的空格。
  - 類型：布林值，預設：false
- withSchema – 如果要覆寫推斷的結構描述，則包含預期的結構描述。若不使用此選項，AWS Glue 會從 XML 資料推斷出結構描述。
  - 類型：文字，預設：不適用
  - 此值應為代表 StructType 的 JSON 物件。

手動指定 XML 結構描述

手動 XML 結構描述範例

這是使用 withSchema 格式選項來指定 XML 資料結構描述的範例。

```
from awsglue.gluetypes import *
```

```
schema = StructType([
  Field("id", IntegerType()),
  Field("name", StringType()),
  Field("nested", StructType([
    Field("x", IntegerType()),
    Field("y", StringType()),
    Field("z", ChoiceType([IntegerType(), StringType()]))
  ]))
])

datasource0 = create_dynamic_frame_from_options(
    connection_type,
    connection_options={"paths": ["s3://xml_bucket/someprefix"]},
    format="xml",
    format_options={"withSchema": json.dumps(schema.jsonValue())},
    transformation_ctx = ""
)
```

在 AWS Glue 中使用 Avro 格式

AWS Glue 會從各來源擷取資料，並將資料寫入以各種資料格式儲存和傳輸的目標。如果您的資料以 Avro 資料格式存放或傳輸，本文件將向您介紹在 AWS Glue 中使用資料的可用功能。

AWS Glue 支援使用 Avro 格式。此格式是效能導向、以資料列為基礎的資料格式。如需標準授權單位的格式簡介，請參閱 Apache Avro 1.8.2 Documentation。

您可以使用 AWS Glue 從 Amazon S3 和串流來源讀取 Avro 檔案，並將 Avro 檔案寫入 Amazon S3。您可以讀取和寫入來自 S3 的包含 Avro 檔案的 `bzip2` 和 `gzip` 封存。此外，您可以寫入 `deflate`、`snappy` 和包含 Avro 檔案的 `xz` 封存。您可以在 S3 連線參數 上設定壓縮行為，而不是在本頁討論的組態中設定。

下表顯示支援 Avro 格式選項的常見 AWS Glue 操作。

讀取	寫入	串流讀取	對小型檔案進行分組	任務書籤
支援	支援	支援*	不支援	支援

* 表示支援，但有所限制。如需詳細資訊，請參閱 the section called “Avro 串流來源的注意事項與限制”。

範例：從 S3 讀取 Avro 檔案或資料夾

先決條件：您需要指向希望讀取的 Avro 檔案或資料夾的 S3 路徑 (s3path)。

組態：在您的函數選項中，指定 format="avro"。在您的 connection_options 中，使用 paths 鍵來指定 s3path。您可以在 connection_options 中設定讀取器與 S3 的互動方式。如需詳細資訊，請參閱 AWS Glue：the section called “S3 連線參數” 中 ETL 輸入與輸出的資料格式選項。您可以在 format_options 中設定讀取器解譯 Avro 的方式。如需詳細資訊，請參閱 Avro 組態參考。

以下 AWS Glue ETL 指令碼顯示從 S3 讀取 Avro 檔案或資料夾的流程：

Python

在此範例中，使用 create_dynamic_frame.from_options 方法。

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="avro"
)
```

Scala

在此範例中，使用 getSourceWithFormat 操作。

```
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.GlueContext
import org.apache.spark.sql.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType="s3",
      format="avro",
```

```
      options=JsonOptions("""{"paths": ["s3://s3path"]}""")
    ).getDynamicFrame()
  }
```

範例：將 Avro 檔案和資料夾寫入 S3

先決條件：您需要初始化 DataFrame (`dataFrame`) 或 DynamicFrame (`dynamicFrame`)。您還需要預期的 S3 輸出路徑 `s3path`。

組態：在您的函數選項中，指定 `format="avro"`。在您的 `connection_options` 中，使用 `paths` 索引鍵指定 `s3path`。您可以在 `connection_options` 中進一步更改寫入器與 S3 的互動方式。如需詳細資訊，請參閱 AWS Glue：the section called “S3 連線參數” 中 ETL 輸入與輸出的資料格式選項。您可以在 `format_options` 中更改寫入器解譯 Avro 檔案的方式。如需詳細資訊，請參閱 Avro 組態參考。

以下 AWS Glue ETL 指令碼顯示將 Avro 檔案或資料夾寫入 S3 的流程。

Python

在此範例中，使用 write_dynamic_frame.from_options 方法。

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    format="avro",
    connection_options={
        "path": "s3://s3path"
    }
)
```

Scala

在此範例中，使用 getSinkWithFormat 方法。

```
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
```

```
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
      connectionType="s3",
      options=JsonOptions("""{"path": "s3://s3path"}"""),
      format="avro"
    ).writeDynamicFrame(dynamicFrame)
  }
}
```

Avro 組態參考

您都可以使用下列 `format_options` 值，無論 AWS Glue 程式庫在何處指定 `format="avro"`：

- `version` — 指定 Apache Avro Reader/Writer 格式支援的版本。預設值為 "1.7"。您可以指定 `format_options={"version": “1.8”}`，以啟用 Avro 邏輯類型的讀取和寫入。如需詳細資訊，請參閱 Apache Avro 1.7.7 規格和 Apache Avro 1.8.2 規格。

  Apache Avro 1.8 連接器支援以下邏輯類型的轉換：

對於讀取器：此表顯示適用於 Avro Reader 1.7 和 1.8 的 Avro 資料類型 (邏輯類型與 Avro 基本類型) 與 AWS Glue `DynamicFrame` 資料類型之間的轉換。

Avro 資料類型： 邏輯類型	Avro 資料類型： Avro 基本類型	GlueDynamicFrame 資料類型： Avro Reader 1.7	GlueDynamicFrame 資料類型： Avro Reader 1.8
Decimal (小數)	位元組	BINARY	Decimal (小數)
Decimal (小數)	固定	BINARY	Decimal (小數)
Date	int	INT	Date
時間 (毫秒)	int	INT	INT

Avro 資料類型： 邏輯類型	Avro 資料類型： Avro 基本類型	GlueDynamicFrame 資料類型： Avro Reader 1.7	GlueDynamicFrame 資料類型： Avro Reader 1.8
時間 (微秒)	長整數	LONG	LONG
時間戳記 (毫秒)	長整數	LONG	時間戳記
時間戳記 (微秒)	長整數	LONG	LONG
持續時間 (不是邏輯類型)	12 (固定)	BINARY	BINARY

對於寫入器：此表顯示適用於 Avro Writer 1.7 和 1.8 的 AWS Glue `DynamicFrame` 資料類型與 Avro 資料類型之間的轉換。

AWS Glue **`DynamicFrame`** 資料類型	Avro 資料類型： Avro Writer 1.7	Avro 資料類型： Avro Writer 1.8
Decimal (小數)	字串	decimal
Date	字串	日期
時間戳記	字串	timestamp-micros

Avro Spark DataFrame 支援

為了從 Spark DataFrame API 使用Avro，您需要為相應的 Spark 版本安裝 Spark Avro 外掛程式。任務中可用的 Spark 版本取決於 AWS Glue 版本。如需 Spark 版本的詳細資訊，請參閱 the section called “AWS Glue 版本”。這個外掛程式是由 Apache 維護，我們不作出具體的支援保證。

在 AWS Glue 2.0 中：使用 Spark Avro 外掛程式 2.4.3 版。您可以在 Maven Central 找到此 JAR，請參閱 org.apache.spark:spark-avro_2.12:2.4.3。

在 AWS Glue 3.0 中：使用 Spark Avro 外掛程式 3.1.1 版。您可以在 Maven Central 找到此 JAR，請參閱 org.apache.spark:spark-avro_2.12:3.1.1。

若要在 AWSGlue ETL 任務中納入額外的 JAR，請使用 `--extra-jars` 任務參數。如需有關任務參數的詳細資訊，請參閱 the section called “任務參數”。您也可以在 AWS Management Console 中設定此參數。

在 AWS Glue 中使用 grokLog 格式

AWS Glue 會從各來源擷取資料，並將資料寫入以各種資料格式儲存和傳輸的目標。如果您的資料是以鬆散結構的純文字格式儲存或傳輸，本文件將向您介紹透過 Grok 模式在 AWS Glue 中使用資料的可用功能。

AWS Glue 支援使用 Grok 模式。Grok 模式類似於規則表達式擷取群組。它們辨識純文字檔案中字元序列的模式，並為其提供類型和用途。在 Glue AWS，其主要用途是讀取日誌。有關作者對 Grok 的介紹，請參閱 Logstash Reference: Grok filter plugin。

讀取	寫入	串流讀取	對小型檔案進行分組	任務書籤
支援	不適用	支援	支援	不支援

grokLog 組態參考

可以使用下列的 `format_options` 值搭配 `format="grokLog"`：

- `logFormat` — 指定符合記錄格式的 Grok 模式。
- `customPatterns` — 指定此處使用的其他 Grok 模式。
- `MISSING` — 指定用於識別遺漏值的訊號。預設值為 `'-'`。
- `LineCount` — 指定各個日誌記錄中的行數。預設為 `'1'`，目前也只支援單行記錄。
- `StrictMode` — 布林值，指定是否要開啟嚴格模式。在嚴格模式下，讀者不可自動轉換類型或復原。預設值為 `"false"`。

在 AWS Glue 中使用 Ion 格式

AWS Glue 會從各來源擷取資料，並將資料寫入以各種資料格式儲存和傳輸的目標。如果您的資料以 Ion 資料格式存放或傳輸，本文件將向您介紹在 AWS Glue 中使用資料的可用功能。

AWS Glue 支援使用 Ion 格式。此格式以可互換的二進位和純文字表示法表示資料結構（並非基於資料列或資料行的資料結構）。如需作者的格式簡介，請參閱 Amazon Ion。(如需詳細資訊，請參閱 Amazon Ion 規格。)

您可以使用 AWS Glue 從 Amazon S3 讀取 Ion 檔案。您可以讀取 bzip 和 gzip 來自 S3 的包含 Ion 檔案的封存。您可以在 S3 連線參數 上設定壓縮行為，而不是在本頁討論的組態中設定。

下表顯示支援 Ion 格式選項的常見 AWS Glue 操作。

讀取	寫入	串流讀取	對小型檔案進行分組	任務書籤
支援	不支援	不支援	支援	不支援

範例：從 S3 讀取 Ion 檔案或資料夾

先決條件：您需要指向希望讀取的 Ion 檔案或資料夾的 S3 路徑 (s3path)。

組態：在您的函數選項中，指定 format="json"。在您的 connection_options 中，使用 paths 索引鍵指定 s3path。您可以在 connection_options 中設定讀取器與 S3 的互動方式。如需詳細資訊，請參閱 AWS Glue 中 ETL 的連線類型和選項：the section called “S3 連線參數”。

以下 AWS Glue ETL 指令碼顯示從 S3 讀取 Ion 檔案或資料夾的流程：

Python

在此範例中，使用 create_dynamic_frame.from_options 方法。

```
# Example: Read ION from S3

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="ion"
)
```

Scala

在此範例中，使用 getSourceWithFormat 操作。

```
// Example: Read ION from S3

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.GlueContext
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType="s3",
      format="ion",
      options=JsonOptions("""{"paths": ["s3://s3path"], "recurse": true}""")
    ).getDynamicFrame()
  }
}
```

Ion 組態參考

無 `format="ion"` 的 `format_options` 值。

在 AWS Glue 中使用 JSON 格式

AWS Glue 會從來源擷取資料，並將資料寫入以各種資料格式儲存和傳輸的目標。如果您的資料是以 JSON 資料格式儲存或傳輸，本文件會向您介紹在 AWS Glue 中使用資料的可用功能。

AWS Glue 支持使用 JSON 格式。此格式表示具有一致形狀但內容靈活、且非基於資料列或資料行的資料結構。JSON 是由多個機構發佈的平行標準所定義的，其中之一是 ECMA-404。如需常用參考來源的格式簡介，請參閱 JSON 簡介。

您可以使用 AWS Glue 從 Amazon S3 讀取 JSON 文件，以bzip及gzip壓縮的 JSON 文件。您可以在 S3 連線參數 上設定壓縮行為，而不是在本頁討論的組態中設定。

讀取	寫入	串流讀取	對小型檔案進行分組	任務書籤	
支援	支援	支援	支援	支援	

範例：從 S3 讀取 JSON 檔案或資料夾

先決條件：您需要指向想要讀取的 JSON 檔案或資料夾的 S3 路徑 (s3path)。

組態：在您的函數選項中，指定 format="json"。在您的 connection_options 中，使用 paths 索引鍵指定 s3path。您可以在連線選項中進一步更改讀取操作將如何周遊 s3，如需詳細資訊，請諮詢 the section called “S3 連線參數”。您可以在 format_options 中設定讀取器解譯 JSON 的方式。如需詳細資訊，請參閱 JSON 組態參考。

下列 AWS Glue ETL 指令碼顯示從 S3 讀取 JSON 檔案或資料夾的程序：

Python

在此範例中，使用 create_dynamic_frame.from_options 方法。

```
# Example: Read JSON from S3
# For show, we handle a nested JSON file that we can limit with the JsonPath
 parameter
# For show, we also handle a JSON where a single entry spans multiple lines
# Consider whether optimizePerformance is right for your workflow.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="json",
    format_options={
        "jsonPath": "$.id",
        "multiline": True,
        # "optimizePerformance": True, -> not compatible with jsonPath, multiline
    }
)
```

您也可以 DataFrames 在腳本（pyspark.sql.DataFrame）中使用。

```
dataFrame = spark.read\
    .option("multiLine", "true")\
```

```
    .json("s3://s3path")
```

Scala

對於此範例，請使用「getSourceWith格式」作業。

```
// Example: Read JSON from S3
// For show, we handle a nested JSON file that we can limit with the JsonPath
 parameter
// For show, we also handle a JSON where a single entry spans multiple lines
// Consider whether optimizePerformance is right for your workflow.

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      formatOptions=JsonOptions("""{"jsonPath": "$.id", "multiline": true,
 "optimizePerformance":false}"""),
      connectionType="s3",
      format="json",
      options=JsonOptions("""{"paths": ["s3://s3path"], "recurse": true}""")
    ).getDynamicFrame()
  }
}
```

您也可以 DataFrames 在腳本（`pyspark.sql.DataFrame`）中使用。

```
val dataFrame = spark.read
    .option("multiLine", "true")
    .json("s3://s3path")
```

範例：將 JSON 檔案和資料夾寫入 S3

先決條件：您將需要初始化 DataFrame（`dataFrame`）或 DynamicFrame（`dynamicFrame`）。您還需要預期的 S3 輸出路徑 `s3path`。

組態：在您的函數選項中，指定 format="json"。在您的 connection_options 中，使用 paths 鍵來指定 s3path。您可以在 connection_options 中進一步更改寫入器與 S3 的互動方式。如需詳細資訊，請參閱 AWS Glue: the section called “S3 連線參數” 中 ETL 輸入和輸出的資料格式選項。您可以在 format_options 中設定寫入器解譯 JSON 的方式。如需詳細資訊，請參閱 JSON 組態參考。

下列 AWS Glue ETL 指令碼顯示從 S3 寫入 JSON 檔案或資料夾的程序：

Python

在此範例中，使用 write_dynamic_frame.from_options 方法。

```
# Example: Write JSON to S3

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    connection_options={"path": "s3://s3path"},
    format="json"
)
```

您也可以 DataFrames 在腳本（pyspark.sql.DataFrame）中使用。

```
df.write.json("s3://s3path/")
```

Scala

在此範例中，請使用「格getSinkWith式化」方法。

```
// Example: Write JSON to S3

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext
```

```
object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
        connectionType="s3",
        options=JsonOptions("""{"path": "s3://s3path"}"""),
        format="json"
    ).writeDynamicFrame(dynamicFrame)
  }
}
```

您也可以 DataFrames 在腳本（pyspark.sql.DataFrame）中使用。

```
df.write.json("s3://s3path")
```

Json 組態參考

可以使用下列的 format_options 值搭配 format="json"：

- jsonPath— 識別要讀入記錄之物件的JsonPath運算式。當檔案包含在外部陣列中的巢狀記錄時，這個方法特別實用。例如，下列 JsonPath 運算式會針對 JSON 物件的id欄位。

  ```
  format="json", format_options={"jsonPath": "$.id"}
  ```

- multiLine — 布林值，用以指定單項記錄是否可以跨越多行。當欄位內含引用的新行字元時，可能就會發生這種情況。若有任何記錄跨越多行，請務必將此選項設為 "true"。預設值為 "false"，如此在剖析時會更加積極地分割檔案。
- optimizePerformance — 布林值，指定是否要使用進階 SIMD JSON 讀取器，以及 Apache Arrow 為基礎的直欄式記憶體格式。僅適用於 AWS Glue 3.0。與 multiLine 或 jsonPath 不相容。提供這些選項中的任何一個將指示 AWS Glue 退回到標準讀卡機。
- withSchema — 字串值，以 the section called “指定 XML 結構描述” 中所述的格式指定資料表結構描述。僅在從非目錄連線讀取時搭配 optimizePerformance 使用。

使用向量化 SIMD JSON 讀取器搭配 Apache Arrow 直欄式格式

AWS Glue 3.0 版為 JSON 資料新增向量化讀取器。與標準讀取器相比，它在某些條件下的效能快 2 倍。此讀取器具有使用者在使用前應注意的某些限制，本區段中給出了這些限制。

要使用最佳化的讀取器，請在 format_options 或資料表屬性中將 "optimizePerformance" 設為 True。除非從目錄中讀取，否則還需要提供 withSchema。withSchema 需要輸入，如 the section called “指定 XML 結構描述” 中所述。

```
// Read from S3 data source
glueContext.create_dynamic_frame.from_options(
    connection_type = "s3",
    connection_options = {"paths": ["s3://s3path"]},
    format = "json",
    format_options={
        "optimizePerformance": True,
        "withSchema": SchemaString
        })

// Read from catalog table
glueContext.create_dynamic_frame.from_catalog(
    database = database,
    table_name = table,
    additional_options = {
    // The vectorized reader for JSON can read your schema from a catalog table
 property.
        "optimizePerformance": True,
        })
```

如需 AWS Glue 資源庫中建築物 a *SchemaString*的詳細資訊，請參閱the section called “類型”。

向量化 CSV 讀取器的限制

注意下列限制：

- 不支援具有巢狀物件或陣列值的 JSON 元素。如果提供， AWS Glue 將回退到標準讀卡機。
- 必須從目錄或使用 withSchema 提供結構描述。
- 與 multiLine 或 jsonPath 不相容。提供這些選項中的任何一個將指示 AWS Glue 退回到標準讀卡機。
- 提供不符合輸入結構描述的輸入記錄會導致讀取器失敗。
- 錯誤記錄將不會建立。
- 不支援含有多位元組字元 (例如日文或中文字元) 的 JSON 檔案。

在 AWS Glue 中使用 ORC 格式

AWS Glue 會從各來源擷取資料，並將資料寫入以各種資料格式儲存和傳輸的目標。如果您的資料以 ORC 資料格式存放或傳輸，本文件將向您介紹在 AWS Glue 中使用資料的可用功能。

AWS Glue 支援使用 ORC 格式。此格式是效能導向、以資料行為基礎的資料格式。如需標準授權單位的格式簡介，請參閱 Apache Orc。

您可以使用 AWS Glue 從 Amazon S3 和串流來源讀取 ORC 檔案，並將 ORC 檔案寫入 Amazon S3。您可以讀取和寫入來自 S3 的包含 ORC 檔案的 `bzip` 和 `gzip` 封存。您可以在 S3 連線參數 上設定壓縮行為，而不是在本頁討論的組態中設定。

下表顯示支援 ORC 格式選項的常見 AWS Glue 操作。

讀取	寫入	串流讀取	對小型檔案進行分組	任務書籤
支援	支援	支援	不支援	支援*

* 在 AWS Glue 1.0 或更新版本中支援

範例：從 S3 讀取 ORC 檔案或資料夾

先決條件：您需要指向希望讀取的 ORC 檔案或資料夾的 S3 路徑 (`s3path`)。

組態：在您的函數選項中，指定 `format="orc"`。在您的 `connection_options` 中，使用 `paths` 索引鍵指定 `s3path`。您可以在 `connection_options` 中設定讀取器與 S3 的互動方式。如需詳細資訊，請參閱 AWS Glue 中 ETL 的連線類型和選項：the section called “S3 連線參數”。

以下 AWS Glue ETL 指令碼顯示從 S3 讀取 ORC 檔案或資料夾的流程：

Python

在此範例中，使用 create_dynamic_frame.from_options 方法。

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
```

```
dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="orc"
)
```

您也可以在指令碼中使用 DataFrames (pyspark.sql.DataFrame)。

```
dataFrame = spark.read\
    .orc("s3://s3path")
```

Scala

在此範例中，使用 getSourceWithFormat 操作。

```
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.GlueContext
import org.apache.spark.sql.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType="s3",
      format="orc",
      options=JsonOptions("""{"paths": ["s3://s3path"]}""")
    ).getDynamicFrame()
  }
}
```

您也可以在指令碼中使用 DataFrames (pyspark.sql.DataFrame)。

```
val dataFrame = spark.read
    .orc("s3://s3path")
```

範例：將 ORC 檔案和資料夾寫入 S3

先決條件：您需要初始化 DataFrame (dataFrame) 或 DynamicFrame (dynamicFrame)。您還需要預期的 S3 輸出路徑 s3path。

組態：在您的函數選項中，指定 `format="orc"`。在您的連線選項中，使用 `paths` 鍵指定 `s3path`。您可以在 `connection_options` 中進一步更改寫入器與 S3 的互動方式。如需詳細資訊，請參閱 AWS Glue：the section called “S3 連線參數” 中 ETL 輸入與輸出的資料格式選項。下列程式碼範例顯示了此程序：

Python

在此範例中，使用 write_dynamic_frame.from_options 方法。

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    format="orc",
    connection_options={
        "path": "s3://s3path"
    }
)
```

您也可以在指令碼中使用 DataFrames (`pyspark.sql.DataFrame`)。

```
df.write.orc("s3://s3path/")
```

Scala

在此範例中，使用 getSinkWithFormat 方法。

```
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
```

```
      connectionType="s3",
      options=JsonOptions("""{"path": "s3://s3path"}"""),
      format="orc"
    ).writeDynamicFrame(dynamicFrame)
  }
}
```

您也可以在指令碼中使用 DataFrames (pyspark.sql.DataFrame)。

```
df.write.orc("s3://s3path/")
```

ORC 組態參考

無 format="orc" 的 format_options 值。不過任何底層的 SparkSQL 程式碼所接受的選項均可透過 connection_options 對應參數傳送。

將資料湖架構與 AWS Glue ETL 任務搭配使用

若檔案存放於在 Amazon S3 上建置的資料湖中，開放原始碼資料湖架構可以簡化這些檔案的增量資料處理。AWS Glue 3.0 及更高版本支援下列開放原始碼資料湖架構：

- Apache Hudi
- Linux Foundation Delta Lake
- Apache Iceberg

我們為這些架構提供原生支援，讓您能夠以交易一致的方式讀取和寫入存放在 Amazon S3 中的資料。您不需要安裝個別的連接器或完成額外的設定步驟，就能在 AWS Glue ETL 任務中使用這些架構。

透過 AWS Glue Data Catalog 管理資料集時，您可以使用 AWS Glue 方法來讀取和寫入具有 Spark DataFrame 的資料湖資料表。您也可以使用 Spark DataFrame API 讀取和寫入 Amazon S3 資料。

在本影片中，您可以了解有關 Apache Hudi、Apache Iceberg 和 Delta Lake 如何運作的基礎知識。您將了解如何插入、更新和刪除資料湖中的資料，以及這些架構的運作方式。

主題

- 限制
- 在 AWS Glue 中使用 Hudi 架構
- 在 AWS Glue 中使用 Delta Lake 架構

- 在 AWS Glue 中使用 Iceberg 架構

限制

在搭配使用資料湖架構之前，請考慮下列限制 AWS Glue。

- 下列 DynamicFrame 不支援讀取和寫入資料湖架構表格的 AWS Glue `GlueContext`方法。使用 DataFrame 或火花 DataFrame API 的`GlueContext`方法來代替。
  - 以下`GlueContext`方法 DynamicFrame 不支援 Lake Formation 權限控制：
    - `create_dynamic_frame.from_catalog`
    - `write_dynamic_frame.from_catalog`
    - `getDynamicFrame`
    - `writeDynamicFrame`
  - Lake Formation 權限控制支援下列`GlueContext`方法： DataFrame
    - `create_data_frame.from_catalog`
    - `write_data_frame.from_catalog`
    - `getDataFrame`
    - `writeDataFrame`
- 不支援將小型檔案分組。
- 不支援任務書籤。
- 阿帕奇胡迪 0.10.1 AWS Glue 3.0 不支持胡迪合併讀 (MOR) 表.
- `ALTER TABLE … RENAME TO`不適用於阿帕奇冰山 0.13.1 的 AWS Glue 3.0。

由 Lake Formation 權限管理的資料湖格式資料表的限制

資料湖格式透過 AWS Glue Lake Formation 權限與 ETL 整合。不支援建立 DynamicFrame 使用`create_dynamic_frame`。如需詳細資訊，請參閱下列範例：

- 範例：使用 Lake Formation 權限控制讀取和寫入 Iceberg 資料表
- 範例：使用 Lake Formation 權限控制讀取和寫入 Hudi 資料表
- 範例：使用 Lake Formation 權限控制讀取和寫入 Delta Lake 資料表

> **Note**
>
> 通過 Apache 胡迪，Apache 冰山和三角洲湖的 Lake Formation 權限與 AWS Glue ETL 集成僅在 4.0 版本中 AWS Glue 受支持。

阿帕奇冰山擁有通過 AWS Glue Lake Formation 權限與 ETL 的最佳集成。其支援幾乎所有操作 (包含 SQL 支援)。

Hudi 支援大多數基本操作 (除管理操作外)。這是因為這些選項通常會透過 DataFrame 寫入完成，並透過 `additional_options` 指定。您需要使用 AWS Glue API 來為您的操 DataFrames 作創建，因為 SparkSQL 不受支持。

Delta Lake 僅支援讀取、附加及覆寫資料表資料。Delta Lake 需要使用自有的程式庫才可執行不同任務 (例如，更新)。

下列功能不適用於 Lake Formation 權限管理的 Iceberg 資料表。

- 使用 ETL 進 AWS Glue 行壓實
- 通過 AWS Glue ETL 星火 SQL 支持

下列為 Lake Formation 權限管理的 Hudi 資料表限制：

- 移除遺棄的檔案

下列為 Lake Formation 權限管理的 Delta Lake 資料表限制：

- 插入 Delta Lake 資料表和從其中讀取以外的所有功能。

在 AWS Glue 中使用 Hudi 架構

AWS Glue 3.0 及更高版本支援適用於資料湖的 Apache Hudi 架構。Hudi 是開放原始碼資料湖儲存架構，可簡化增量資料處理與資料管道開發。本主題說明在 Hudi 資料表中傳輸或存放資料時，在AWS Glue 中使用資料的可用功能。若要進一步了解 Hudi，請參閱官方 Apache Hudi 文件。

您可以使用 AWS Glue，在 Amazon S3 中的 Hudi 資料表上執行讀取和寫入操作，或搭配 AWS Glue Data Catalog 使用 Hudi 資料表。還支援其他操作，包括插入、更新和所有 Apache Spark 操作。

> **Note**
>
> AWS Glue 3.0 的 Apache Hudi 0.10.1 不支援在 Hudi 讀取時合併 (MoR) 資料表。

下表列出每個 AWS Glue 版本中包含的 Hudi 版本。

AWS Glue 版本	支援的 Hudi 版本
4.0	0.12.1
3.0	0.10.1

若要進一步了解 AWS Glue 支援的資料湖架構，請參閱將資料湖架構與 AWS Glue ETL 任務搭配使用。

啟用 Hudi

若要為 AWS Glue 啟用 Hudi，請完成下列任務：

- 指定 `hudi` 作為 `--datalake-formats` 任務參數的值。如需詳細資訊，請參閱 AWS Glue 任務參數。
- 為 AWS Glue 任務建立名為 `--conf` 的索引鍵，並將其設定為下列值。您也可以選擇在指令碼中使用 `SparkConf` 設定以下組態。這些設定有助於 Apache Spark 正確處理 Hudi 資料表。

```
spark.serializer=org.apache.spark.serializer.KryoSerializer --conf
 spark.sql.hive.convertMetastoreParquet=false
```

- Hudi 的 Lake Formation 權限支援預設已針對 AWS Glue 4.0 啟用。讀取/寫入 Lake Formation 註冊的 Hudi 資料表時，不需要其他組態。若要讀取註冊的 Hudi 資料表，AWS Glue 工作 IAM 角色必須具有 SELECT 權限。若要寫入註冊的 Hudi 資料表，AWS Glue 工作 IAM 角色必須具有 SUPER 權限。若要進一步了解管理 Lake Formation 權限的資訊，請參閱授予和撤銷 Data Catalog 資源的權限。

使用不同的 Hudi 版本

若要使用 AWS Glue 不支援的 Hudi 版本，請使用 `--extra-jars` 任務參數指定您自己的 Hudi JAR 檔案。請勿包括 `hudi` 作為 `--datalake-formats` 任務參數的值。

範例：將 Hudi 資料表寫入 Amazon S3，並將其註冊到 AWS Glue Data Catalog

此範例指令碼示範如何將 Hudi 資料表寫入 Amazon S3，以及如何將該資料表註冊到 AWS Glue Data Catalog。此範例使用 Hudi Hive 同步工具來註冊該資料表。

> **Note**
>
> 此範例要求您設定 `--enable-glue-datacatalog` 任務參數，才能使用 AWS Glue Data Catalog 作為 Apache Spark Hive 中繼存放區。如需進一步了解，請參閱 AWS Glue 任務參數。

Python

```
# Example: Create a Hudi table from a DataFrame
# and register the table to Glue Data Catalog

additional_options={
    "hoodie.table.name": "<your_table_name>",
    "hoodie.datasource.write.storage.type": "COPY_ON_WRITE",
    "hoodie.datasource.write.operation": "upsert",
    "hoodie.datasource.write.recordkey.field": "<your_recordkey_field>",
    "hoodie.datasource.write.precombine.field": "<your_precombine_field>",
    "hoodie.datasource.write.partitionpath.field": "<your_partitionkey_field>",
    "hoodie.datasource.write.hive_style_partitioning": "true",
    "hoodie.datasource.hive_sync.enable": "true",
    "hoodie.datasource.hive_sync.database": "<your_database_name>",
    "hoodie.datasource.hive_sync.table": "<your_table_name>",
    "hoodie.datasource.hive_sync.partition_fields": "<your_partitionkey_field>",
    "hoodie.datasource.hive_sync.partition_extractor_class":
 "org.apache.hudi.hive.MultiPartKeysValueExtractor",
    "hoodie.datasource.hive_sync.use_jdbc": "false",
    "hoodie.datasource.hive_sync.mode": "hms",
    "path": "s3://<s3Path/>"
}

dataFrame.write.format("hudi") \
    .options(**additional_options) \
    .mode("overwrite") \
    .save()
```

Scala

```
// Example: Example: Create a Hudi table from a DataFrame
// and register the table to Glue Data Catalog

val additionalOptions = Map(
  "hoodie.table.name" -> "<your_table_name>",
  "hoodie.datasource.write.storage.type" -> "COPY_ON_WRITE",
  "hoodie.datasource.write.operation" -> "upsert",
  "hoodie.datasource.write.recordkey.field" -> "<your_recordkey_field>",
  "hoodie.datasource.write.precombine.field" -> "<your_precombine_field>",
  "hoodie.datasource.write.partitionpath.field" -> "<your_partitionkey_field>",
  "hoodie.datasource.write.hive_style_partitioning" -> "true",
  "hoodie.datasource.hive_sync.enable" -> "true",
  "hoodie.datasource.hive_sync.database" -> "<your_database_name>",
  "hoodie.datasource.hive_sync.table" -> "<your_table_name>",
  "hoodie.datasource.hive_sync.partition_fields" -> "<your_partitionkey_field>",
  "hoodie.datasource.hive_sync.partition_extractor_class" ->
 "org.apache.hudi.hive.MultiPartKeysValueExtractor",
  "hoodie.datasource.hive_sync.use_jdbc" -> "false",
  "hoodie.datasource.hive_sync.mode" -> "hms",
  "path" -> "s3://<s3Path/>")

dataFrame.write.format("hudi")
  .options(additionalOptions)
  .mode("append")
  .save()
```

範例：使用 AWS Glue Data Catalog 從 Amazon S3 讀取 Hudi 資料表

此範例會讀取您從 Amazon S3 在 範例：將 Hudi 資料表寫入 Amazon S3，並將其註冊到 AWS Glue Data Catalog 中建立的 Hudi 資料表。

> **Note**
>
> 此範例要求您設定 `--enable-glue-datacatalog` 任務參數，才能使用 AWS Glue Data Catalog 作為 Apache Spark Hive 中繼存放區。如需進一步了解，請參閱 AWS Glue 任務參數。

Python

在此範例中，使用 GlueContext.create_data_frame.from_catalog() 方法。

```
# Example: Read a Hudi table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

dataFrame = glueContext.create_data_frame.from_catalog(
    database = "<your_database_name>",
    table_name = "<your_table_name>"
)
```

Scala

在此範例中，使用 getCatalogSource 方法。

```
// Example: Read a Hudi table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dataFrame = glueContext.getCatalogSource(
      database = "<your_database_name>",
      tableName = "<your_table_name>"
    ).getDataFrame()
  }
}
```

範例：在 Amazon S3 中更新 **`DataFrame`** 並將其插入到 Hudi 資料表中

此範例使用 AWS Glue Data Catalog，將 DataFrame 插入您在 範例：將 Hudi 資料表寫入 Amazon S3，並將其註冊到 AWS Glue Data Catalog 中建立的 Hudi 資料表。

> **Note**
>
> 此範例要求您設定 `--enable-glue-datacatalog` 任務參數，才能使用 AWS Glue Data Catalog 作為 Apache Spark Hive 中繼存放區。如需進一步了解，請參閱 AWS Glue 任務參數。

Python

在此範例中，使用 `GlueContext.write_data_frame.from_catalog()` 方法。

```
# Example: Upsert a Hudi table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

glueContext.write_data_frame.from_catalog(
    frame = dataFrame,
    database = "<your_database_name>",
    table_name = "<your_table_name>",
    additional_options={
        "hoodie.table.name": "<your_table_name>",
        "hoodie.datasource.write.storage.type": "COPY_ON_WRITE",
        "hoodie.datasource.write.operation": "upsert",
        "hoodie.datasource.write.recordkey.field": "<your_recordkey_field>",
        "hoodie.datasource.write.precombine.field": "<your_precombine_field>",
        "hoodie.datasource.write.partitionpath.field": "<your_partitionkey_field>",
        "hoodie.datasource.write.hive_style_partitioning": "true",
        "hoodie.datasource.hive_sync.enable": "true",
        "hoodie.datasource.hive_sync.database": "<your_database_name>",
        "hoodie.datasource.hive_sync.table": "<your_table_name>",
        "hoodie.datasource.hive_sync.partition_fields": "<your_partitionkey_field>",
        "hoodie.datasource.hive_sync.partition_extractor_class":
 "org.apache.hudi.hive.MultiPartKeysValueExtractor",
        "hoodie.datasource.hive_sync.use_jdbc": "false",
        "hoodie.datasource.hive_sync.mode": "hms"
    }
)
```

Scala

在此範例中，使用 getCatalogSink 方法。

```
// Example: Upsert a Hudi table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.JsonOptions
import org.apacke.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    glueContext.getCatalogSink("<your_database_name>", "<your_table_name>",
      additionalOptions = JsonOptions(Map(
        "hoodie.table.name" -> "<your_table_name>",
        "hoodie.datasource.write.storage.type" -> "COPY_ON_WRITE",
        "hoodie.datasource.write.operation" -> "upsert",
        "hoodie.datasource.write.recordkey.field" -> "<your_recordkey_field>",
        "hoodie.datasource.write.precombine.field" -> "<your_precombine_field>",
        "hoodie.datasource.write.partitionpath.field" ->
 "<your_partitionkey_field>",
        "hoodie.datasource.write.hive_style_partitioning" -> "true",
        "hoodie.datasource.hive_sync.enable" -> "true",
        "hoodie.datasource.hive_sync.database" -> "<your_database_name>",
        "hoodie.datasource.hive_sync.table" -> "<your_table_name>",
        "hoodie.datasource.hive_sync.partition_fields" ->
 "<your_partitionkey_field>",
        "hoodie.datasource.hive_sync.partition_extractor_class" ->
 "org.apache.hudi.hive.MultiPartKeysValueExtractor",
        "hoodie.datasource.hive_sync.use_jdbc" -> "false",
        "hoodie.datasource.hive_sync.mode" -> "hms"
      )))
      .writeDataFrame(dataFrame, glueContext)
  }
}
```

範例：使用 Spark 從 Amazon S3 讀取 Hudi 資料表

此範例使用 Spark DataFrame API 從 Amazon S3 讀取 Hudi 資料表。

Python

```
# Example: Read a Hudi table from S3 using a Spark DataFrame

dataFrame = spark.read.format("hudi").load("s3://<s3path/>")
```

Scala

```
// Example: Read a Hudi table from S3 using a Spark DataFrame

val dataFrame = spark.read.format("hudi").load("s3://<s3path/>")
```

範例：使用 Spark 將 Hudi 資料表寫入 Amazon S3

此範例使用 Spark 將 Hudi 資料表寫入 Amazon S3。

Python

```
# Example: Write a Hudi table to S3 using a Spark DataFrame

dataFrame.write.format("hudi") \
    .options(**additional_options) \
    .mode("overwrite") \
    .save("s3://<s3Path/>")
```

Scala

```
// Example: Write a Hudi table to S3 using a Spark DataFrame

dataFrame.write.format("hudi")
  .options(additionalOptions)
  .mode("overwrite")
  .save("s3://<s3path/>")
```

範例：使用 Lake Formation 權限控制讀取和寫入 Hudi 資料表

此範例會使用 Lake Formation 權限控制讀取和寫入 Hudi 資料表。

1. 建立 Hudi 資料表，並在 Lake Formation 中進行註冊。

a. 若要啟用 Lake Formation 權限控制，您將需要先在 Lake Formation 中註冊資料表 Amazon S3 路徑。如需詳細資訊，請參閱 Registering an Amazon S3 location (註冊 Amazon S3 位置)。您可以從 Lake Formation 主控台或透過使用 AWS CLI 進行註冊：

```
aws lakeformation register-resource --resource-arn arn:aws:s3:::<s3-bucket>/<s3-
folder> --use-service-linked-role --region <REGION>
```

註冊 Amazon S3 位置後，任何指向該位置 (或其任何子位置) 的 AWS Glue 資料表將會在 `GetTable` 呼叫中傳回 `IsRegisteredWithLakeFormation` 參數值 true。

b. 建立透過 Spark DataFrame API 指向註冊之 Amazon S3 路徑的 Hudi 資料表：

```
hudi_options = {
    'hoodie.table.name': table_name,
    'hoodie.datasource.write.storage.type': 'COPY_ON_WRITE',
    'hoodie.datasource.write.recordkey.field': 'product_id',
    'hoodie.datasource.write.table.name': table_name,
    'hoodie.datasource.write.operation': 'upsert',
    'hoodie.datasource.write.precombine.field': 'updated_at',
    'hoodie.datasource.write.hive_style_partitioning': 'true',
    'hoodie.upsert.shuffle.parallelism': 2,
    'hoodie.insert.shuffle.parallelism': 2,
    'path': <S3_TABLE_LOCATION>,
    'hoodie.datasource.hive_sync.enable': 'true',
    'hoodie.datasource.hive_sync.database': database_name,
    'hoodie.datasource.hive_sync.table': table_name,
    'hoodie.datasource.hive_sync.use_jdbc': 'false',
    'hoodie.datasource.hive_sync.mode': 'hms'
}

df_products.write.format("hudi")  \
    .options(**hudi_options)  \
    .mode("overwrite")  \
    .save()
```

2. 將 Lake Formation 權限授予 AWS Glue 工作 IAM 角色。您可以從 Lake Formation 主控台或使用 AWS CLI 授予權限。如需詳細資訊，請參閱使用 Lake Formation 主控台和具名資源方法授予資料表權限。
3. 讀取在 Lake Formation 中註冊的 Hudi 資料表。該程式碼與讀取未註冊之 Hudi 資料表的程式碼相同。請注意，AWS Glue 工作 IAM 角色需要具有 SELECT 權限，才能成功讀取。

```
 val dataFrame = glueContext.getCatalogSource(
      database = "<your_database_name>",
      tableName = "<your_table_name>"
    ).getDataFrame()
```

4. 寫入在 Lake Formation 中註冊的 Hudi 資料表。該程式碼與寫入未註冊之 Hudi 資料表的程式碼相同。請注意，AWS Glue 工作 IAM 角色需要具有 SUPER 權限，才能成功寫入。

```
glueContext.getCatalogSink("<your_database_name>", "<your_table_name>",
      additionalOptions = JsonOptions(Map(
        "hoodie.table.name" -> "<your_table_name>",
        "hoodie.datasource.write.storage.type" -> "COPY_ON_WRITE",
        "hoodie.datasource.write.operation" -> "<write_operation>",
        "hoodie.datasource.write.recordkey.field" -> "<your_recordkey_field>",
        "hoodie.datasource.write.precombine.field" -> "<your_precombine_field>",
        "hoodie.datasource.write.partitionpath.field" -> "<your_partitionkey_field>",
        "hoodie.datasource.write.hive_style_partitioning" -> "true",
        "hoodie.datasource.hive_sync.enable" -> "true",
        "hoodie.datasource.hive_sync.database" -> "<your_database_name>",
        "hoodie.datasource.hive_sync.table" -> "<your_table_name>",
        "hoodie.datasource.hive_sync.partition_fields" ->
 "<your_partitionkey_field>",
        "hoodie.datasource.hive_sync.partition_extractor_class" ->
 "org.apache.hudi.hive.MultiPartKeysValueExtractor",
        "hoodie.datasource.hive_sync.use_jdbc" -> "false",
        "hoodie.datasource.hive_sync.mode" -> "hms"
      )))
      .writeDataFrame(dataFrame, glueContext)
```

在 AWS Glue 中使用 Delta Lake 架構

AWS Glue 3.0 及更高版本支援 Linux Foundation Delta Lake 架構。Delta Lake 是開放原始碼資料湖儲存架構，可協助您執行 ACID 交易、擴充中繼資料處理，以及統一串流與批次資料處理。本主題說明在 Delta Lake 資料表中傳輸或存放資料時，在 AWS Glue 中可用的資料使用功能。若要進一步了解 Delta Lake，請參閱官方 Delta Lake 文件。

您可以使用 AWS Glue，在 Amazon S3 中的 Delta Lake 資料表上執行讀取和寫入操作，或搭配 AWS Glue Data Catalog 使用 Delta Lake 資料表。插入、更新和資料表批次讀取和寫入等操作也受到支援。使用 Delta Lake 資料表時，您還可以選擇使用 Delta Lake Python 程式庫中的方法，例如

DeltaTable.forPath。如需三角洲湖 Python 程式庫的詳細資訊，請參閱三角洲湖的 Python 文件。

下表列出了每個 AWS Glue 版本中包含的 Delta Lake 版本。

AWS Glue 版本	支援的 Delta Lake 版本
4.0	2.1.0
3.0	1.0.0

若要進一步了解 AWS Glue 支援的資料湖架構，請參閱將資料湖架構與 AWS Glue ETL 任務搭配使用。

為 AWS Glue 啟用 Delta Lake

若要為 AWS Glue 啟用 Delta Lake，請完成下列任務：

- 指定 delta 作為 --datalake-formats 任務參數的值。如需詳細資訊，請參閱 AWS Glue 任務參數。
- 為 AWS Glue 任務建立名為 --conf 的索引鍵，並將其設定為下列值。您也可以選擇在指令碼中使用 SparkConf 設定以下組態。這些設定有助於 Apache Spark 正確處理 Delta Lake 資料表。

```
spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf
 spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog --
conf
 spark.delta.logStore.class=org.apache.spark.sql.delta.storage.S3SingleDriverLogStore
```

- Delta 資料表的 Lake Formation 權限支援預設已針對 AWS Glue 4.0 啟用。讀取/寫入 Lake Formation 註冊的 Delta 資料表時，不需要其他組態。若要讀取註冊的 Delta 資料表，AWS Glue 任務 IAM 角色必須具有 SELECT 權限。若要讀取註冊的 Delta 資料表，AWS Glue 任務 IAM 角色必須具有 SUPER 權限。若要進一步了解管理 Lake Formation 權限的資訊，請參閱授予和撤銷 Data Catalog 資源的權限。

使用不同的 Delta Lake 版本

若要使用 AWS Glue 不支援的 Delta Lake 版本，請使用 --extra-jars 任務參數指定您自己的 Delta Lake JAR 檔案。請勿包括 delta 作為 --datalake-formats 任務參數的值。若要在此案例

中使用 Delta Lake Python 程式庫，您必須使用 `--extra-py-files` 任務參數指定程式庫 JAR 檔案。Python 程式庫封裝在 Delta Lake JAR 檔案中。

範例：將 Delta Lake 資料表寫入 Amazon S3，並將其註冊到 AWS Glue Data Catalog

下列 AWS Glue ETL 指令碼示範如何將 Delta Lake 資料表寫入 Amazon S3，以及如何將該資料表註冊到 AWS Glue Data Catalog。

Python

```
# Example: Create a Delta Lake table from a DataFrame
# and register the table to Glue Data Catalog

additional_options = {
    "path": "s3://<s3Path>"
}
dataFrame.write \
    .format("delta") \
    .options(**additional_options) \
    .mode("append") \
    .partitionBy("<your_partitionkey_field>") \
    .saveAsTable("<your_database_name>.<your_table_name>")
```

Scala

```
// Example: Example: Create a Delta Lake table from a DataFrame
// and register the table to Glue Data Catalog

val additional_options = Map(
  "path" -> "s3://<s3Path>"
)
dataFrame.write.format("delta")
  .options(additional_options)
  .mode("append")
  .partitionBy("<your_partitionkey_field>")
  .saveAsTable("<your_database_name>.<your_table_name>")
```

範例：使用 AWS Glue Data Catalog 從 Amazon S3 讀取 Delta Lake 資料表

下列 AWS Glue ETL 指令碼會讀取您在 範例：將 Delta Lake 資料表寫入 Amazon S3，並將其註冊到 AWS Glue Data Catalog 中建立的 Delta Lake 資料表。

Python

在此範例中，使用 create_data_frame.from_catalog 方法。

```
# Example: Read a Delta Lake table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

df = glueContext.create_data_frame.from_catalog(
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

Scala

在此範例中，使用 getCatalogSource 方法。

```
// Example: Read a Delta Lake table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apacke.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val df = glueContext.getCatalogSource("<your_database_name>",
 "<your_table_name>",
      additionalOptions = additionalOptions)
      .getDataFrame()
  }
}
```

範例：使用 AWS Glue Data Catalog 將 **`DataFrame`** 插入 Amazon S3 中的 Delta Lake 資料表

此範例會將資料插入您在 範例：將 Delta Lake 資料表寫入 Amazon S3，並將其註冊到 AWS Glue Data Catalog 中建立的 Delta Lake 資料表中。

> **Note**
>
> 此範例要求您設定 `--enable-glue-datacatalog` 任務參數，才能使用 AWS Glue Data Catalog 作為 Apache Spark Hive 中繼存放區。如需進一步了解，請參閱 AWS Glue 任務參數。

Python

在此範例中，使用 write_data_frame.from_catalog 方法。

```
# Example: Insert into a Delta Lake table in S3 using Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

glueContext.write_data_frame.from_catalog(
    frame=dataFrame,
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

Scala

在此範例中，使用 getCatalogSink 方法。

```
// Example: Insert into a Delta Lake table in S3 using Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apacke.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    glueContext.getCatalogSink("<your_database_name>", "<your_table_name>",
      additionalOptions = additionalOptions)
      .writeDataFrame(dataFrame, glueContext)
```

```
  }
}
```

範例：使用 Spark API 從 Amazon S3 讀取 Delta Lake 資料表

此範例使用 Spark API 從 Amazon S3 讀取 Delta Lake 資料表。

Python

```
# Example: Read a Delta Lake table from S3 using a Spark DataFrame

dataFrame = spark.read.format("delta").load("s3://<s3path/>")
```

Scala

```
// Example: Read a Delta Lake table from S3 using a Spark DataFrame

val dataFrame = spark.read.format("delta").load("s3://<s3path/>")
```

範例：使用 Spark 將 Delta Lake 資料表寫入 Amazon S3

此範例會使用 Spark 將 Delta Lake 資料表寫入 Amazon S3。

Python

```
# Example: Write a Delta Lake table to S3 using a Spark DataFrame

dataFrame.write.format("delta") \
    .options(**additional_options) \
    .mode("overwrite") \
    .partitionBy("<your_partitionkey_field>") \
    .save("s3://<s3Path>")
```

Scala

```
// Example: Write a Delta Lake table to S3 using a Spark DataFrame

dataFrame.write.format("delta")
  .options(additionalOptions)
  .mode("overwrite")
```

```
    .partitionBy("<your_partitionkey_field>")
    .save("s3://<s3path/>")
```

範例：使用 Lake Formation 權限控制讀取和寫入 Delta Lake 資料表

此範例會讀取和寫入具有 Lake Formation 權限控制的 Delta Lake 資料表。

1. 建立 Delta 資料表，並在 Lake Formation 中進行註冊

   a. 若要啟用 Lake Formation 權限控制，您將需要先在 Lake Formation 中註冊資料表 Amazon S3 路徑。如需詳細資訊，請參閱 Registering an Amazon S3 location (註冊 Amazon S3 位置)。您可以從 Lake Formation 主控台或透過使用 AWS CLI 進行註冊：

   ```
   aws lakeformation register-resource --resource-arn arn:aws:s3:::<s3-bucket>/<s3-folder> --use-service-linked-role --region <REGION>
   ```

   註冊 Amazon S3 位置後，任何指向該位置 (或其任何子位置) 的 AWS Glue 資料表將會在 `GetTable` 呼叫中傳回 `IsRegisteredWithLakeFormation` 參數值 true。

   b. 建立透過 Spark 指向註冊之 Amazon S3 路徑的 Delta 資料表：

   > **Note**
   >
   > 下列為 Python 範例。

   ```
   dataFrame.write \
    .format("delta") \
    .mode("overwrite") \
    .partitionBy("<your_partitionkey_field>") \
    .save("s3://<the_s3_path>")
   ```

   將資料寫入 Amazon S3 後，請使用 AWS Glue 爬蟲程式建立新的 Delta 目錄資料表。如需詳細資訊，請參閱使用 AWS Glue 爬蟲程式的原生 Delta Lake 資料表支援簡介。

   您也可透過 AWS Glue `CreateTable` API 手動建立資料表。

2. 將 Lake Formation 權限授予 AWS Glue 任務 IAM 角色。您可以從 Lake Formation 主控台或使用 AWS CLI 授予權限。如需詳細資訊，請參閱使用 Lake Formation 主控台和具名資源方法授予資料表權限。

3. 讀取在 Lake Formation 中註冊的 Delta 資料表。該程式碼與讀取未註冊之 Delta 資料表的程式碼相同。請注意，AWS Glue 任務 IAM 角色需要具有 SELECT 權限，才能成功讀取。

```
# Example: Read a Delta Lake table from Glue Data Catalog

df = glueContext.create_data_frame.from_catalog(
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

4. 寫入在 Lake Formation 中註冊的 Delta 資料表。該程式碼與寫入未註冊之 Delta 資料表的程式碼相同。請注意，AWS Glue 任務 IAM 角色需要具有 SUPER 權限，才能成功寫入。

   依預設，AWS Glue 會使用 Append 作為 saveMode。您可以透過設定 additional_options 中的 saveMode 選項進行變更。如需有關 Delta 資料表之 saveMode 支援的資訊，請參閱寫入資料表。

```
glueContext.write_data_frame.from_catalog(
    frame=dataFrame,
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

在 AWS Glue 中使用 Iceberg 架構

AWS Glue 3.0 及更高版本支援適用於資料湖的 Apache Iceberg 架構。Iceberg 提供高效能的資料表格式，其運作方式就像 SQL 資料表一樣。本主題說明在 Iceberg 資料表中傳輸或存放資料時，在 AWS Glue 中使用資料的可用功能。若要進一步了解 Iceberg，請參閱官方 Apache Iceberg 文件。

您可以使用 AWS Glue，在 Amazon S3 中的 Iceberg 資料表上執行讀取和寫入操作，或搭配 AWS Glue Data Catalog 使用 Iceberg 資料表。插入、更新和所有 Spark 查詢 Spark 寫入等操作也受到支援。

> **Note**
>
> ALTER TABLE … RENAME TO 不適用於 AWS Glue 3.0 的 Apache Iceberg 0.13.1。

下表列出了每個 AWS Glue 版本中包含的 Iceberg 版本。

AWS Glue 版本	支援的 Iceberg 版本
4.0	1.0.0
3.0	0.13.1

若要進一步了解 AWS Glue 支援的資料湖架構，請參閱[將資料湖架構與 AWS Glue ETL 任務搭配使用]。

啟用 Iceberg 架構

若要為 AWS Glue 啟用 Iceberg，請完成下列任務：

- 指定 `iceberg` 作為 `--datalake-formats` 任務參數的值。如需詳細資訊，請參閱 [AWS Glue 任務參數]。
- 為 AWS Glue 任務建立名為 `--conf` 的索引鍵，並將其設定為下列值。您也可以選擇在指令碼中使用 `SparkConf` 設定以下組態。這些設定有助於 Apache Spark 正確處理 Iceberg 資料表。

```
spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
--conf spark.sql.catalog.glue_catalog=org.apache.iceberg.spark.SparkCatalog
--conf spark.sql.catalog.glue_catalog.warehouse=s3://<your-warehouse-dir>/
--conf spark.sql.catalog.glue_catalog.catalog-
impl=org.apache.iceberg.aws.glue.GlueCatalog
--conf spark.sql.catalog.glue_catalog.io-impl=org.apache.iceberg.aws.s3.S3FileIO
```

  如果您正在讀取或寫入向 Lake Formation 註冊的 Iceberg 資料表，請新增下列組態以啟用 Lake Formation 支援。請注意，僅有 AWS Glue 4.0 支援向 Lake Formation 註冊的 Iceberg 資料表：

```
--conf spark.sql.catalog.glue_catalog.glue.lakeformation-enabled=true
--conf spark.sql.catalog.glue_catalog.glue.id=<table-catalog-id>
```

  如果您將 AWS Glue 3.0 與 Iceberg 0.13.1 搭配使用，則必須設定下列其他組態，才能使用 Amazon DynamoDB 鎖定管理程式來確保不可部分完成的交易。AWSGlue 4.0 依預設使用樂觀鎖定。如需詳細資訊，請參閱官方 Apache Iceberg 文件中的 Iceberg AWS 整合。

```
--conf spark.sql.catalog.glue_catalog.lock-
impl=org.apache.iceberg.aws.glue.DynamoLockManager
```

```
--conf spark.sql.catalog.glue_catalog.lock.table=<your-dynamodb-table-name>
```

使用不同的 Iceberg 版本

若要使用 AWS Glue 不支援的 Iceberg 版本，請使用 `--extra-jars` 任務參數指定您自己的 Iceberg JAR 檔案。請勿包括 `iceberg` 作為 `--datalake-formats` 參數的值。

啟用 Iceberg 資料表的加密

> **Note**
>
> Iceberg 資料表具有自己的機制來啟用伺服器端加密。除了 AWS Glue 的安全組態之外，您還應啟用此組態。

若要在 Iceberg 資料表上啟用伺服器端加密，請檢閱 Iceberg 文件中的指引。

範例：將 Iceberg 資料表寫入 Amazon S3，並將其註冊到 AWS Glue Data Catalog

此範例指令碼示範如何將 Iceberg 資料表寫入 Amazon S3。此範例使用 Iceberg AWS 整合，將資料表註冊到 AWS Glue Data Catalog。

Python

```
# Example: Create an Iceberg table from a DataFrame
# and register the table to Glue Data Catalog

dataFrame.createOrReplaceTempView("tmp_<your_table_name>")

query = f"""
CREATE TABLE glue_catalog.<your_database_name>.<your_table_name>
USING iceberg
TBLPROPERTIES ("format-version"="2")
AS SELECT * FROM tmp_<your_table_name>
"""
spark.sql(query)
```

Scala

```
// Example: Example: Create an Iceberg table from a DataFrame
```

```
// and register the table to Glue Data Catalog

dataFrame.createOrReplaceTempView("tmp_<your_table_name>")

val query = """CREATE TABLE glue_catalog.<your_database_name>.<your_table_name>
USING iceberg
TBLPROPERTIES ("format-version"="2")
AS SELECT * FROM tmp_<your_table_name>
"""
spark.sql(query)
```

或者，您可使用 Spark 方法將 Iceberg 資料表寫入 Amazon S3 和 Data Catalog。

先決條件：您需要佈建型錄才能使用 Iceberg 程式庫。使用 AWS Glue Data Catalog 時，AWS Glue 會簡化此操作。AWS Glue Data Catalog 已預先設定，以供 Spark 程式庫作為 `glue_catalog` 使用。資料型錄資料表由 *`databaseName`* 和 *`tableName`* 識別。如需有關 AWS Glue Data Catalog 的詳細資訊，請參閱 資料探索與編目。

如果您未使用 AWS Glue Data Catalog，則需要透過 Spark API 佈建型錄。如需詳細資訊，請參閱 Iceberg 文件中的 Spark 組態。

此範例使用 Spark 將 Iceberg 資料表寫入至 Amazon S3和 Data Catalog。

Python

```
# Example: Write an Iceberg table to S3 on the Glue Data Catalog

# Create (equivalent to CREATE TABLE AS SELECT)
dataFrame.writeTo("glue_catalog.databaseName.tableName") \
    .tableProperty("format-version", "2") \
    .create()

# Append (equivalent to INSERT INTO)
dataFrame.writeTo("glue_catalog.databaseName.tableName") \
    .tableProperty("format-version", "2") \
    .append()
```

Scala

```
// Example: Write an Iceberg table to S3 on the Glue Data Catalog
```

```
// Create (equivalent to CREATE TABLE AS SELECT)
dataFrame.writeTo("glue_catalog.databaseName.tableName")
    .tableProperty("format-version", "2")
    .create()

// Append (equivalent to INSERT INTO)
dataFrame.writeTo("glue_catalog.databaseName.tableName")
    .tableProperty("format-version", "2")
    .append()
```

範例：使用 AWS Glue Data Catalog 從 Amazon S3 讀取 Iceberg 資料表

此範例會讀取您在 範例：將 Iceberg 資料表寫入 Amazon S3，並將其註冊到 AWS Glue Data Catalog 中建立的 Iceberg 資料表。

Python

在此範例中，使用 GlueContext.create_data_frame.from_catalog() 方法。

```
# Example: Read an Iceberg table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

df = glueContext.create_data_frame.from_catalog(
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

Scala

在此範例中，使用 getCatalogSource 方法。

```
// Example: Read an Iceberg table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apacke.spark.SparkContext
```

```
object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val df = glueContext.getCatalogSource("<your_database_name>",
 "<your_table_name>",
      additionalOptions = additionalOptions)
      .getDataFrame()
  }
}
```

範例：使用 AWS Glue Data Catalog 將 **DataFrame** 插入 Amazon S3 中的 Iceberg 資料表

此範例會將資料插入您在 範例：將 Iceberg 資料表寫入 Amazon S3，並將其註冊到 AWS Glue Data Catalog 中建立的 Iceberg 資料表中。

> **Note**
>
> 此範例要求您設定 `--enable-glue-datacatalog` 任務參數，才能使用 AWS Glue Data Catalog 作為 Apache Spark Hive 中繼存放區。如需進一步了解，請參閱 AWS Glue 任務參數。

Python

在此範例中，使用 `GlueContext.write_data_frame.from_catalog()` 方法。

```
# Example: Insert into an Iceberg table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

glueContext.write_data_frame.from_catalog(
    frame=dataFrame,
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

Scala

在此範例中，使用 getCatalogSink 方法。

```
// Example: Insert into an Iceberg table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apacke.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    glueContext.getCatalogSink("<your_database_name>", "<your_table_name>",
      additionalOptions = additionalOptions)
      .writeDataFrame(dataFrame, glueContext)
  }
}
```

範例：使用 Spark 從 Amazon S3 讀取 Iceberg 資料表

先決條件：您需要佈建型錄才能使用 Iceberg 程式庫。使用 AWS Glue Data Catalog 時，AWS Glue 會簡化此操作。AWS Glue Data Catalog 已預先設定，以供 Spark 程式庫作為 `glue_catalog` 使用。資料型錄資料表由 *`databaseName`* 和 *`tableName`* 識別。如需有關 AWS Glue Data Catalog 的詳細資訊，請參閱 資料探索與編目。

如果您未使用 AWS Glue Data Catalog，則需要透過 Spark API 佈建型錄。如需詳細資訊，請參閱 Iceberg 文件中的 Spark 組態。

此範例使用 Spark 從資料型錄讀取 Amazon S3 中的 Iceberg 資料表。

Python

```
# Example: Read an Iceberg table on S3 as a DataFrame from the Glue Data Catalog

dataFrame = spark.read.format("iceberg").load("glue_catalog.databaseName.tableName")
```

Scala

```
// Example: Read an Iceberg table on S3 as a DataFrame from the Glue Data Catalog
```

```
val dataFrame =
 spark.read.format("iceberg").load("glue_catalog.databaseName.tableName")
```

範例：使用 Lake Formation 權限控制讀取和寫入 Iceberg 資料表

此範例會使用 Lake Formation 權限控制讀取和寫入 Iceberg 資料表。

1. 建立 Iceberg 資料表，並在 Lake Formation 進行註冊：

   a. 若要啟用 Lake Formation 權限控制，您將需要先在 Lake Formation 中註冊資料表 Amazon S3 路徑。如需詳細資訊，請參閱 Registering an Amazon S3 location (註冊 Amazon S3 位置)。您可以從 Lake Formation 主控台或透過使用 AWS CLI 進行註冊：

   ```
   aws lakeformation register-resource --resource-arn arn:aws:s3:::<s3-bucket>/<s3-folder> --use-service-linked-role --region <REGION>
   ```

   註冊 Amazon S3 位置後，任何指向該位置 (或其任何子位置) 的 AWS Glue 資料表將會在 `GetTable` 呼叫中傳回 `IsRegisteredWithLakeFormation` 參數值 true。

   b. 建立透過 Spark SQL 指向註冊之路徑的 Iceberg 資料表：

   > **Note**
   >
   > 下列為 Python 範例。

   ```
   dataFrame.createOrReplaceTempView("tmp_<your_table_name>")

   query = f"""
   CREATE TABLE glue_catalog.<your_database_name>.<your_table_name>
   USING iceberg
   AS SELECT * FROM tmp_<your_table_name>
   """
   spark.sql(query)
   ```

   您也可以透過 AWS Glue `CreateTable` API 手動建立資料表。如需詳細資訊，請參閱建立 Apache Iceberg 資料表。

2. 將 Lake Formation 權限授予工作 IAM 角色。您可以從 Lake Formation 主控台或使用 AWS CLI 授予權限。如需詳細資訊，請參閱：https://docs.aws.amazon.com/lake-formation/latest/dg/granting-table-permissions.html
3. 讀取向 Lake Formation 註冊的 Iceberg 資料表。該程式碼與讀取未註冊之 Iceberg 資料表的程式碼相同。請注意，AWS Glue 工作 IAM 角色需要具有 SELECT 權限，才能成功讀取。

```
# Example: Read an Iceberg table from the AWS Glue Data Catalog
from awsglue.context import GlueContextfrom pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

df = glueContext.create_data_frame.from_catalog(
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

4. 寫入向 Lake Formation 註冊的 Iceberg 資料表。該程式碼與寫入未註冊之 Iceberg 資料表的程式碼相同。請注意，AWS Glue 工作 IAM 角色需要具有 SUPER 權限，才能成功寫入。

```
glueContext.write_data_frame.from_catalog(
    frame=dataFrame,
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

共用的組態參考

您可以將以下 `format_options` 值與任何格式類型搭配使用。

- `attachFilename`：以適當格式作為資料欄名稱使用的字串。如果您提供此選項，記錄的來源檔案名稱會附加到記錄中。參數值將用作資料欄名稱。
- `attachTimestamp`：以適當格式作為資料欄名稱使用的字串。如果您提供此選項，記錄的來源檔案修改時間會附加到記錄中。參數值將用作資料欄名稱。

AWS Glue 資料目錄支援 Spark SQL 任務

AWS Glue Data Catalog 是 Apache Hive 中繼存放區相容的目錄。您可以設定您的 AWS Glue 任務和開發端點，使其使用 Data Catalog 作為外部 Apache Hive 中繼存放區。然後，您就可以直接對存放於 Data Catalog 的資料表執行 Apache Spark SQL 查詢。依預設，AWS Glue 動態框架會與 Data Catalog 進行整合。不過，搭配這項功能時，Spark SQL 任務就能使用 Data Catalog 做為外部 Hive 中繼存放區。

此功能需要網路存取 AWS Glue API 端點。適用於帶有私有子網路中連線的 AWS Glue 任務，您必須設定 VPC 端點或 NAT 閘道以提供網路存取。如需 VPC 端點組態的詳細資訊，請參閱 設定對資料存放區的網路存取。若要建立 NAT 閘道，請參閱 Amazon VPC 使用者指南中的 NAT 閘道。

您可以在任務引數和開發端點引數中分別新增 `"--enable-glue-datacatalog": ""` 引數，執行 AWS Glue 任務和開發端點的設定。透過這個引數傳遞，系統就會在 Spark 中設定特定的組態，進而可使用 Data Catalog 做為外部 Hive 中繼存放區。它也會在 AWS Glue 任務或開發端點中建立的 `SparkSession` 物件中啟用 Hive 支援。

若要啟用 Data Catalog 存取權，請在主控台新增任務或新增端點頁面上的 Catalog 選項群組中，勾選使用 AWS Glue Data Catalog 做為 Hive 中繼存放區核取方塊。請注意，用於此任務或開發端點的 IAM 角色應該擁有 `glue:CreateDatabase` 許可。稱為 "`default`" 的資料庫隨即在 Data Catalog 中建立 (如當時該資料庫尚未存在)。

讓我們從下面範例，了解您可以如何在 Spark SQL 任務中應用這項功能。下面範例是假設您已分類過下列網址提供的美國國會議員資料集，`s3://awsglue-datasets/examples/us-legislators`。

為了要對 AWS Glue Data Catalog 中的定義資料表資料進行序列化/取消序列化，Spark SQL 必須具備 Hive SerDe 類別，才能在其 Spark 任務的類別路徑中採用 AWS Glue Data Catalog 的定義格式。

特定通用格式的 SerDes 則由 AWS Glue 發佈。下面是這類格式的 Amazon S3 連結：

- JSON
- XML
- Grok

將 JSON SerDe 當做 額外的 JAR 新增至開發端點。處理任務時，您可以在引數欄位中使用 `--extra-jars` 引數來新增 SerDe 。如需更多詳細資訊，請參閱 AWS Glue 任務參數。

下面範例是輸入 JSON，建立啟用 Data Catalog 進行 Spark SQL 的開發端點。

```
{
    "EndpointName": "Name",
    "RoleArn": "role_ARN",
    "PublicKey": "public_key_contents",
    "NumberOfNodes": 2,
    "Arguments": {
      "--enable-glue-datacatalog": ""
    },
    "ExtraJarsS3Path": "s3://crawler-public/json/serde/json-serde.jar"
}
```

現在查詢使用 Spark SQL 建立自美國國會議員資料集的資料表。

```
>>> spark.sql("use legislators")
DataFrame[]
>>> spark.sql("show tables").show()
+-----------+-----------------+-----------+
|   database|        tableName|isTemporary|
+-----------+-----------------+-----------+
|legislators|       areas_json|      false|
|legislators|   countries_json|      false|
|legislators|      events_json|      false|
|legislators| memberships_json|      false|
|legislators|organizations_json|      false|
|legislators|     persons_json|      false|
+-----------+-----------------+-----------+
>>> spark.sql("describe memberships_json").show()
+--------------------+---------+-----------------+
|            col_name|data_type|          comment|
+--------------------+---------+-----------------+
|             area_id|   string|from deserializer|
|     on_behalf_of_id|   string|from deserializer|
|     organization_id|   string|from deserializer|
|                role|   string|from deserializer|
|           person_id|   string|from deserializer|
|legislative_perio...|   string|from deserializer|
|          start_date|   string|from deserializer|
|            end_date|   string|from deserializer|
+--------------------+---------+-----------------+
```

如果此格式的 SerDe 類別不能用於任務的類別路徑，則您會收到類似下列的錯誤訊息。

```
>>> spark.sql("describe memberships_json").show()

Caused by: MetaException(message:java.lang.ClassNotFoundException Class
 org.openx.data.jsonserde.JsonSerDe not found)
    at
 org.apache.hadoop.hive.metastore.MetaStoreUtils.getDeserializer(MetaStoreUtils.java:399)
    at
 org.apache.hadoop.hive.ql.metadata.Table.getDeserializerFromMetaStore(Table.java:276)
    ... 64 more
```

若只要檢視源自 memberships 資料表的唯一 organization_id，請執行下列 SQL 查詢。

```
>>> spark.sql("select distinct organization_id from memberships_json").show()
+--------------------+
|     organization_id|
+--------------------+
|d56acebe-8fdc-47b...|
|8fa6c3d2-71dc-478...|
+--------------------+
```

如果您需要搭配動態框架執行相同的作業，請執行下列步驟。

```
>>> memberships = glueContext.create_dynamic_frame.from_catalog(database="legislators",
 table_name="memberships_json")
>>> memberships.toDF().createOrReplaceTempView("memberships")
>>> spark.sql("select distinct organization_id from memberships").show()
+--------------------+
|     organization_id|
+--------------------+
|d56acebe-8fdc-47b...|
|8fa6c3d2-71dc-478...|
+--------------------+
```

雖然 DynamicFrames 已針對 ETL 操作經過最佳化處理，但在執行複雜 SQL 陳述式或轉運現有應用程式方面，啟用 Spark SQL 來直接存取 Data Catalog 卻是最簡潔的方法。

使用任務書籤

AWS Glue for Spark 使用任務書籤追蹤已處理的資料。如需任務書籤功能及其支援內容的摘要，請參閱 the section called “使用任務書籤追蹤處理的資料”。使用書籤對 AWS Glue 任務進行程式設計時，您可以取得視覺化任務中無法使用的彈性。

- 從 JDBC 讀取時，您可以在 AWS Glue 指令碼中指定要作為書籤索引鍵使用的資料欄。
- 您可以選擇要將哪個 `transformation_ctx` 套用至每個方法呼叫。

始終*job.init*在腳本的開頭調用，並使用適當配置的參數*job.commit*在腳本的末尾調用。這兩個函數會初始化書籤服務並更新服務的狀態變更。如果沒有呼叫書籤，書籤將不會運作。

指定書籤索引鍵

對於 JDBC 工作流程，書籤會透過比較索引鍵欄位的值與書籤值，追蹤任務已讀取的資料列。對於 Amazon S3 工作流程而言，這不是必需的程序，也不適用。在不使用視覺化編輯器的情況下撰寫 AWS Glue 指令碼時，您可以指定要使用書籤追蹤的資料欄。您也可以指定多個資料欄。指定使用者定義的書籤索引鍵時，允許值序列中存在間隙。

> **Warning**
>
> 如果使用的是使用者定義書籤索引鍵，則均須嚴格單調增加或減少。為複合索引鍵選取其他欄位時，「次要版本」或「修訂編號」等概念的欄位不符合此條件，因為這些欄位的值會在整個資料集中重複使用。

您可採用以下方式來指定 `jobBookmarkKeys` 和 `jobBookmarkKeysSortOrder`：

- `create_dynamic_frame.from_catalog` – 請使用 `additional_options`。
- `create_dynamic_frame.from_options` – 請使用 `connection_options`。

轉換內容

許多AWS Glue PySpark 動態影格方法都包含名為的選用參數`transformation_ctx`，這是 ETL 運算子實體的唯一識別碼。`transformation_ctx` 參數用來識別指定運算子之任務書籤內的狀態資訊。具體而言，AWS Glue 使用 `transformation_ctx` 來編製書籤狀態之索引鍵的索引。

> **Warning**
>
> `transformation_ctx` 作為在書籤狀態中搜索指令碼中特定來源的鍵。為了讓書籤正常運作，您應始終確保來源和相關聯 `transformation_ctx` 的一致性。變更來源屬性或重新命名 `transformation_ctx` 可能會使之前的書籤無效，且基於時間戳記的篩選條件可能無法產生正確的結果。

為了讓任務書籤正常運作，請啟用任務書籤參數，並設定 transformation_ctx 參數。如果您未傳入 transformation_ctx 參數，則不會針對方法中使用的動態框架或資料表啟用任務書籤。例如，如果您有一個讀取和加入兩個 Amazon S3 來源的 ETL 任務，您可以選擇將 transformation_ctx 參數僅傳入您想要啟用書籤的這些方法。如果您重設任務的任務書籤，則不論所使用的 transformation_ctx 為何，系統都會重設所有與任務建立關聯的轉換。

如需 DynamicFrameReader 類別的詳細資訊，請參閱DynamicFrameReader 類。如需副檔名的詳細資 PySpark 訊，請參閱AWS Glue PySpark 延伸模組參考。

範例

Example

以下是針對 Amazon S3 資料來源產生指令碼的範例。使用任務書籤所需的指令碼部分會以斜體顯示。如需這些元素的詳細資訊，請參閱 GlueContext 類 API 和 DynamicFrameWriter 類別 API。

```
# Sample Script
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

datasource0 = glueContext.create_dynamic_frame.from_catalog(
    database = "database",
    table_name = "relatedqueries_csv",
    transformation_ctx = "datasource0"
)

applymapping1 = ApplyMapping.apply(
    frame = datasource0,
    mappings = [("col0", "string", "name", "string"), ("col1", "string", "number",
 "string")],
    transformation_ctx = "applymapping1"
)
```

```
datasink2 = glueContext.write_dynamic_frame.from_options(
    frame = applymapping1,
    connection_type = "s3",
    connection_options = {"path": "s3://input_path"},
    format = "json",
    transformation_ctx = "datasink2"
)


job.commit()
```

Example

以下是針對 JDBC 來源產生指令碼的範例。來源資料表是 empno 欄為主要金鑰的員工資料表。雖然根據預設，如果有指定書籤金鑰，則任務會使用序列主要金鑰做為書籤金鑰，因為 empno 不一定是按序排列值之間可能有間隙不符合預設書籤金鑰的資格。因此，指令碼會明確地指定 empno 為書籤金鑰。程式碼的該部分會以斜體顯示。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

datasource0 = glueContext.create_dynamic_frame.from_catalog(
    database = "hr",
    table_name = "emp",
    transformation_ctx = "datasource0",
    additional_options = {"jobBookmarkKeys":["empno"],"jobBookmarkKeysSortOrder":"asc"}
)

applymapping1 = ApplyMapping.apply(
    frame = datasource0,
```

```
    mappings = [("ename", "string", "ename", "string"), ("hrly_rate", "decimal(38,0)",
 "hrly_rate", "decimal(38,0)"), ("comm", "decimal(7,2)", "comm", "decimal(7,2)"),
 ("hiredate", "timestamp", "hiredate", "timestamp"), ("empno", "decimal(5,0)", "empno",
 "decimal(5,0)"), ("mgr", "decimal(5,0)", "mgr", "decimal(5,0)"), ("photo", "string",
 "photo", "string"), ("job", "string", "job", "string"), ("deptno", "decimal(3,0)",
 "deptno", "decimal(3,0)"), ("ssn", "decimal(9,0)", "ssn", "decimal(9,0)"), ("sal",
 "decimal(7,2)", "sal", "decimal(7,2)")],
    transformation_ctx = "applymapping1"
)

datasink2 = glueContext.write_dynamic_frame.from_options(
    frame = applymapping1,
    connection_type = "s3",
    connection_options = {"path": "s3://hr/employees"},
    format = "csv",
    transformation_ctx = "datasink2"
)

job.commit()
```

在 AWS Glue Studio 外部使用敏感數據檢測

AWS Glue Studio 允許您檢測敏感數據，但是，您也可以使用 AWS Glue Studio 之外的敏感數據檢測功能。

如需受管敏感資料類型的完整清單，請參閱 Managed data types。

使用受管理的 PII 類 AWS 型偵測敏感性資料

AWS Glue 在 AWS Glue ETL 工作中提供兩個 API。它們是 `detect()` 和 `classifyColumns()`：

```
 detect(frame: DynamicFrame,
     entityTypesToDetect: Seq[String],
     outputColumnName: String = "DetectedEntities",
     detectionSensitivity: String = "LOW"): DynamicFrame

detect(frame: DynamicFrame,
     detectionParameters: JsonOptions,
     outputColumnName: String = "DetectedEntities",
     detectionSensitivity: String = "LOW"): DynamicFrame

 classifyColumns(frame: DynamicFrame,
     entityTypesToDetect: Seq[String],
```

```
      sampleFraction: Double = 0.1,
      thresholdFraction: Double = 0.1,
      detectionSensitivity: String = "LOW")
```

您可以使用 detect() API 來識別 AWS 受管理的 PII 類型和自訂實體類型。系統會自動建立包含偵測結果的新欄。該 classifyColumns() API 傳回一個映射，其中索引鍵是資料欄名稱，值是偵測到的實體類型的清單。SampleFraction 指示掃描 PII 實體時要採樣的資料部分，而 ThresholdFraction 指示為了將資料欄標識為 PII 資料而必須滿足的資料部分。

資料列層級偵測

在此範例中，任務正在使用 detect() 和 classifyColumns() API 執行下列動作：

- 從 Amazon S3 存儲桶中讀取數據並將其轉換為動態框架
- 偵測 dynamicFrame 中的 "Email" 和 "Credit Card" 執行個體
- 傳回具有原始值的 dynamicFrame 外加一個資料欄，其中包含每列的偵測結果
- 將返回的動態幀寫入另一個路徑 Amazon S3

```
 import com.amazonaws.services.glue.GlueContext
 import com.amazonaws.services.glue.MappingSpec
 import com.amazonaws.services.glue.errors.CallSite
 import com.amazonaws.services.glue.util.GlueArgParser
 import com.amazonaws.services.glue.util.Job
 import com.amazonaws.services.glue.util.JsonOptions
 import org.apache.spark.SparkContext
 import scala.collection.JavaConverters._
 import com.amazonaws.services.glue.ml.EntityDetector

 object GlueApp {
   def main(sysArgs: Array[String]) {
     val spark: SparkContext = new SparkContext()
     val glueContext: GlueContext = new GlueContext(spark)
     val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
     Job.init(args("JOB_NAME"), glueContext, args.asJava)
     val frame=
glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar": "\"",
"withHeader": true, "separator": ","}"""), connectionType="s3", format="csv",
options=JsonOptions("""{"paths": ["s3://pathToSource"], "recurse": true}"""),
transformationContext="AmazonS3_node1650160158526").getDynamicFrame()
```

```
    val frameWithDetectedPII = EntityDetector.detect(frame, Seq("EMAIL",
"CREDIT_CARD"))

    glueContext.getSinkWithFormat(connectionType="s3",
options=JsonOptions("""{"path": "s3://pathToOutput/", "partitionKeys": []}"""),
transformationContext="someCtx",
format="json").writeDynamicFrame(frameWithDetectedPII)

    Job.commit()
  }
 }
```

具有微調動作的資料列層級偵測

在此範例中，任務正在使用 `detect()` API 執行下列動作：

- 從 Amazon S3 儲存貯體中讀取資料，並將其轉換為 DynamicFrame
- 偵測 DynamicFrame 中的敏感資料類型 “USA_PTIN”、“BANK_ACCOUNT”、“USA_SSN”、“USA_PASSPORT_NUMBER” 及 “PHONE_NUMBER”
- 傳回具有修改遮罩值的 dynamicFrame 與一個資料欄，其中包含每列的偵測結果
- 在其他 Amazon S3 路徑中寫入傳回的 DynamicFrame

與上述 `detect()` API 相反，這會針對要偵測的實體類型使用微調動作。如需詳細資訊，請參閱 使用 detect() 的偵測參數。

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
import com.amazonaws.services.glue.ml.EntityDetector

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
```

```
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)
    val frame =
glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar": "\"",
"withHeader": true, "separator": ","}"""), connectionType="s3", format="csv",
options=JsonOptions("""{"paths": ["s3://pathToSource"], "recurse": true}"""),
transformationContext="AmazonS3_node_source").getDynamicFrame()

    val detectionParameters = JsonOptions(
      """
        {
          "USA_DRIVING_LICENSE": [{
            "action": "PARTIAL_REDACT",
            "sourceColumns": ["Driving License"],
            "actionOptions": {
              "matchPattern": "[0-9]",
              "redactChar": "*"
            }
          }],
          "BANK_ACCOUNT": [{
            "action": "DETECT",
            "sourceColumns": ["*"]
          }],
          "USA_SSN": [{
            "action": "SHA256_HASH",
            "sourceColumns": ["SSN"]
          }],
          "IP_ADDRESS": [{
            "action": "REDACT",
            "sourceColumns": ["IP Address"],
            "actionOptions": {"redactText": "*****"}
          }],
          "PHONE_NUMBER": [{
            "action": "PARTIAL_REDACT",
            "sourceColumns": ["Phone Number"],
            "actionOptions": {
              "numLeftCharsToExclude": 1,
              "numRightCharsToExclude": 0,
              "redactChar": "*"
            }
          }]
        }
      """
    )
```

```
    val frameWithDetectedPII = EntityDetector.detect(frame, detectionParameters,
 "DetectedEntities", "HIGH")

    glueContext.getSinkWithFormat(connectionType="s3", options=JsonOptions("""{"path":
 "s3://pathToOutput/", "partitionKeys": []}"""),
 transformationContext="AmazonS3_node_target",
 format="json").writeDynamicFrame(frameWithDetectedPII)

    Job.commit()
  }
}
```

資料欄層級偵測

在此範例中，任務正在使用 `classifyColumns()` API 執行下列動作：

- 從 Amazon S3 儲存貯體中讀取資料，並將其轉換為 DynamicFrame
- 偵測 dynamicFrame 中的 "Email" 和 "Credit Card" 執行個體
- 設定參數以對資料欄進行 100% 取樣，且如果實體位於 10% 的儲存格中，並設定為 "LOW" 敏感度，則標示為已偵測。
- 傳回一個對應，其中金鑰為資料欄名稱，而值為偵測到的實體類型清單
- 在其他 Amazon S3 路徑中寫入傳回的 DynamicFrame

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
import com.amazonaws.services.glue.DynamicFrame
import com.amazonaws.services.glue.ml.EntityDetector

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
```

```
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)
    val frame =
 glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar":
 "\"", "withHeader": true, "separator": ",", "optimizePerformance": false}"""),
 connectionType="s3", format="csv", options=JsonOptions("""{"paths": ["s3://
pathToSource"], "recurse": true}"""), transformationContext="frame").getDynamicFrame()

    import glueContext.sparkSession.implicits._

    val detectedDataFrame = EntityDetector.classifyColumns(
        frame,
        entityTypesToDetect = Seq("CREDIT_CARD", "PHONE_NUMBER"),
        sampleFraction = 1.0,
        thresholdFraction = 0.1,
        detectionSensitivity = "LOW"
    )
    val detectedDF = (detectedDataFrame).toSeq.toDF("columnName", "entityTypes")
    val DetectSensitiveData_node = DynamicFrame(detectedDF, glueContext)

    glueContext.getSinkWithFormat(connectionType="s3", options=JsonOptions("""{"path":
 "s3://pathToOutput", "partitionKeys": []}"""), transformationContext="someCtx",
 format="json").writeDynamicFrame(DetectSensitiveData_node)

    Job.commit()
  }
}
```

使用 AWS CustomEntityType PII 類型偵測敏感資料偵測

您可以通過 AWS 工作室定義自定義實體。但是，若要在 AWS Studio 中使用此功能，您必須先定義自訂實體類型，然後將已定義的自訂實體類型新增至的清單`entityTypesToDetect`。

如果您的資料中有特定敏感資料類型 (例如「員工識別碼」)，您可以呼叫 `CreateCustomEntityType()` API 來建立自訂實體。下列範例使用請求參數為 `CreateCustomEntityType()` API 定義自訂實體類型 EMPLOYEE_ID：

```
  {
      "name": "EMPLOYEE_ID",
```

```
    "regexString": "\d{4}-\d{3}",
    "contextWords": ["employee"]
}
```

然後，將自訂實體類型 (EMPLOYE_ID) 新增至 EntityDetector( ) API，來修改任務以使用新的自訂敏感資料類型：

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
import com.amazonaws.services.glue.ml.EntityDetector

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)
    val frame=
glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar": "\"",
"withHeader": true, "separator": ","}"""), connectionType="s3", format="csv",
options=JsonOptions("""{"paths": ["s3://pathToSource"], "recurse": true}"""),
transformationContext="AmazonS3_node1650160158526").getDynamicFrame()

    val frameWithDetectedPII = EntityDetector.detect(frame, Seq("EMAIL",
"CREDIT_CARD", "EMPLOYEE_ID"))

    glueContext.getSinkWithFormat(connectionType="s3",
options=JsonOptions("""{"path": "s3://pathToOutput/", "partitionKeys": []}"""),
transformationContext="someCtx",
format="json").writeDynamicFrame(frameWithDetectedPII)

    Job.commit()
  }
}
```

> **Note**
>
> 如果使用與現有受管實體類型相同的名稱來定義自訂敏感資料類型，則自訂敏感資料類型將優先，並覆寫受管實體類型的邏輯。

使用 `detect()` 的偵測參數

此方法用於檢測中的實體 DynamicFrame。它返回一個新的原 DataFrame 始值和一個具有 PII 檢測元數據的附加列 outputColumnName 。自定義屏蔽可以在AWS Glue腳本中返回後完成，或者可以使用帶有細粒度操作 API 的 detect ( ) 。 DynamicFrame

```
detect(frame: DynamicFrame,
       entityTypesToDetect: Seq[String],
       outputColumnName: String = "DetectedEntities",
       detectionSensitivity: String = "LOW"): DynamicFrame
```

參數：

- frame- (類型：`DynamicFrame`) DynamicFrame 包含要處理的數據的輸入。
- entityTypesTo偵測 — (類型:`[Seq[String]`) 要偵測的實體類型清單。可以是受管實體類型或自訂實體類型。
- outputColumnName— (類型：`String`，預設值：DetectedEntities"「) 將儲存偵測到的實體的欄名稱。如果未提供，則預設欄名稱為 "DetectedEntities"。
- detectionSensitivity：(類型：`String`，選項："LOW" 或 "HIGH"，預設："LOW") 指定偵測程序的敏感度。有效選項為 "LOW" 或 "HIGH"。如果未提供，則預設敏感度會設定為 "LOW"。

`outputColumnName` 設定：

資料欄的名稱，其中偵測到的實體將會受到儲存。如果未提供，則預設欄名稱為 "DetectedEntities"。針對輸出資料欄中的每一列，補充資料欄會包含資料欄名稱至偵測到之實體中繼資料的對應，其中包含下列鍵值組：

- entityType：偵測到的實體類型。
- start：原始資料中偵測到的實體起始位置。
- start：原始資料中偵測到的實體結束位置。

- actionUsed：執行於偵測到之實體的動作 (例如，"DETECT"、"REDACT"、"PARTIAL_REDACT"、"SHA256_HASH")。

範例：

```
{
   "DetectedEntities":{
      "SSN Col":[
         {
            "entityType":"USA_SSN",
            "actionUsed":"DETECT",
            "start":4,
            "end":15
         }
      ],
      "Random Data col":[
         {
            "entityType":"BANK_ACCOUNT",
            "actionUsed":"PARTIAL_REDACT",
            "start":4,
            "end":13
         },
         {
            "entityType":"IP_ADDRESS",
            "actionUsed":"REDACT",
            "start":4,
            "end":13
         }
      ]
   }
}
```

具有微調動作之 **detect()** 的偵測參數

此方法用於檢測實體中 DynamicFrame 使用指定的參數。它返回一個新的原 DataFrame 始值替換為被屏蔽的敏感數據和一個具有 PII 檢測元數據`outputColumnName`的附加列。

```
detect(frame: DynamicFrame,
       detectionParameters: JsonOptions,
       outputColumnName: String = "DetectedEntities",
       detectionSensitivity: String = "LOW"): DynamicFrame
```

參數：

- frame- (類型：DynamicFrame) ： DynamicFrame 包含要處理的數據的輸入。
- detectionParameters：(類型 Json0ptions) 針對偵測程序指定參數的 JSON 選項。
- outputColumnName— (類型：String，預設值：DetectedEntities"「)：將儲存偵測到的實體的欄名稱。如果未提供，則預設欄名稱為 "DetectedEntities"。
- detectionSensitivity：(類型：String，選項："LOW" 或 "HIGH"，預設："LOW") 指定偵測程序的敏感度。有效選項為 "LOW" 或 "HIGH"。如果未提供，則預設敏感度會設定為 "LOW"。

detectionParameters 設定

如果未包含任何設定，則會使用預設值。

- action：(類型：String，選項："DETECT"、"REDACT"、"PARTIAL_REDACT"、"SHA256_HASH") 指定要在實體中執行的動作。必要。請注意，執行遮罩的動作 (除了 "DETECT" 以外的所有動作) 僅可在每個資料欄執行一個動作。此為遮罩合併實體的預防措施。
- sourceColumns：(類型：List[String]，預設：[“*”]) 針對實體要執行偵測的來源資料欄名稱清單。如果不存在，則會預設為 [“*”]。如果使用無效的資料欄名稱，則會引發 IllegalArgumentException。
- sourceColumnsToExclude — (類型:List[String]) 要對實體執行偵測的來源欄名稱清單。使用 sourceColumns 或 sourceColumnsToExclude。如果使用無效的資料欄名稱，則會引發 IllegalArgumentException。
- actionOptions：根據指定動作的其他選項：
  - 如為 "DETECT" 和 "SHA256_HASH"，則不允許任何選項。
  - 如為 "REDACT"：
    - redactText：(類型：String，預設值："*****") 用來取代偵測到之實體的文字。
  - 如為 "PARTIAL_REDACT"：
    - redactChar：(類型：String，預設："*") 用於取代實體中每個偵測到之字元的字元。
    - matchPattern：(類型：String) 適用於部分遮蔽的 Regex 模式。不能與 numLeftCharsToExclude 或結合使用numRightCharsToExclude。
    - numLeftCharsToExclude— (類型:String, integer) 要排除的左邊字元數。無法與 matchPattern 結合，但可搭配 numRightCharsToExclude 使用。

- numRightCharsToExclude— (類型:`String, integer`) 要排除的右邊字元數。無法與 matchPattern 結合，但可搭配 `numRightCharsToExclude` 使用。

`outputColumnName` 設定

查看 outputColumnName 設定

適用於 **`classifyColumns()`** 的偵測參數

此方法用於檢測中的實體 DynamicFrame。此方法會傳回一個對應，其中金鑰為資料欄名稱，而值為偵測到的實體類型清單。當其在 AWS Glue 指令碼內傳回後，即可進行自訂遮罩。

```
classifyColumns(frame: DynamicFrame,
                entityTypesToDetect: Seq[String],
                sampleFraction: Double = 0.1,
                thresholdFraction: Double = 0.1,
                detectionSensitivity: String = "LOW")
```

參數：

- frame- (類型：`DynamicFrame`) DynamicFrame 包含要處理的數據的輸入。
- entityTypesTo偵測 — (類型:`Seq[String]`) 要偵測的實體類型清單。可以是受管實體類型或自訂實體類型。
- sampleFraction：(類型：`Double`，預設：10%) 掃描 PII 實體時，要取樣的資料部分。
- thresholdFraction：(類型：`Double`，預設：10%) 為使資料欄經識別為 PII 資料，必須符合的資料部分。
- detectionSensitivity：(類型：`String`，選項："LOW" 或 "HIGH"，預設："LOW") 指定偵測程序的敏感度。有效選項為 "LOW" 或 "HIGH"。如果未提供，則預設敏感度會設定為 "LOW"。

受管敏感資料類型

全域實體

資料類型	類別	描述
PERSON_NAME	Universal	人員姓名。

資料類型	類別	描述
EMAIL	個人	電子郵件地址。
IP_ADDRESS	Computer	IP 地址
MAC_ADDRESS	個人	MAC 地址。

美國資料類型

資料類型	描述
BANK_ACCOUNT	銀行帳戶號碼。不是特定於某個國家或區域，但目前僅偵測到美國和加拿大的帳戶格式。
CREDIT_CARD	信用卡號碼。
PHONE_NUMBER	電話號碼。不是特定於某個國家或區域，但目前僅偵測到美國和加拿大的電話號碼。
USA_ATIN	美國國稅局核發的美國領養納稅人識別號碼。
USA_CPT_CODE	CPT 編碼 (美國特定)。
USA_DEA_NUMBER	DEA 號碼 (美國特定)。
USA_DRIVING_LICENSE	駕照號碼 (美國特定)。
USA_HCPCS_CODE	HCPCS 編碼 (美國特定)。
USA_HEALTH_INSURANCE_CLAIM_NUMBER	健康保險索償編碼 (美國特定)。
USA_ITIN	ITIN (適用於美國人或實體)。
USA_MEDICARE_BENEFICIARY_IDENTIFIER	聯邦醫療保險受益人識別碼 (美國特定)。
USA_NATIONAL_DRUG_CODE	NDC 編碼 (美國特定)。

資料類型	描述
USA_NATIONAL_PROVIDER_IDENTIFIER	國家提供者識別號碼 (美國特定)。
USA_PASSPORT_NUMBER	護照號碼 (適用於美國人)。
USA_PTIN	美國國稅局核發的美國報稅人稅務識別號碼。
USA_SSN	社會安全號碼 (適用於美國人)。

阿根廷資料類型

資料類型	描述
ARGENTINA_TAX_IDENTIFICATIO N_NUMBER	阿根廷稅務識別號碼。也稱為 CUIT 或 CUIL。

澳洲資料類型

資料類型	描述
AUSTRALIA_BUSINESS_NUMBER	澳洲企業編號 (ABN)。由澳洲商業登記處 (ABR) 核發的唯一識別碼，用於向政府和社群識別企業。
AUSTRALIA_COMPANY_NUMBER	澳洲公司編號 (ACN)。由澳洲證券與投資委員會核發的唯一識別碼。
AUSTRALIA_DRIVING_LICENSE	澳洲駕照號碼。
AUSTRALIA_MEDICARE_NUMBER	澳洲醫療保險號碼。由澳洲健康保險委員會核發的個人識別碼。
AUSTRALIA_PASSPORT_NUMBER	澳洲護照號碼。
AUSTRALIA_TAX_FILE_NUMBER	澳洲稅務檔案號碼 (TFN)。由澳洲稅務局 (ATO) 核發給納稅人 (個人、公司等) 進行稅務交易。

奧地利資料類型

資料類型	描述
AUSTRIA_DRIVING_LICENSE	駕照號碼 (奧地利特定)。
AUSTRIA_PASSPORT_NUMBER	護照號碼 (奧地利特定)。
AUSTRIA_SSN	社會安全號碼 (適用於奧地利人)。
AUSTRIA_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (奧地利特定)。
AUSTRIA_VALUE_ADDED_TAX	增值稅 (奧地利特定)。

巴爾幹資料類型

資料類型	描述
BOSNIA_UNIQUE_MASTER_CITIZEN_NUMBER	波士尼亞與赫塞哥維納公民的唯一主公民號碼 (JMBG)。
KOSOVO_UNIQUE_MASTER_CITIZEN_NUMBER	科索沃的唯一主公民號碼 (JMBG)。
MACEDONIA_UNIQUE_MASTER_CITIZEN_NUMBER	馬其頓的唯一主公民號碼。
MONTENEGRO_UNIQUE_MASTER_CITIZEN_NUMBER	蒙特內哥羅的唯一主公民號碼 (JMBG)。
SERBIA_UNIQUE_MASTER_CITIZEN_NUMBER	塞爾維亞的唯一主公民號碼 (JMBG)。
SERBIA_VALUE_ADDED_TAX	增值稅 (塞爾維亞特定)。
VOJVODINA_UNIQUE_MASTER_CITIZEN_NUMBER	佛伊弗迪納自治省的唯一主公民號碼 (JMBG)。

比利時資料類型

資料類型	描述
BELGIUM_DRIVING_LICENSE	駕照號碼 (比利時特定)。
BELGIUM_NATIONAL_IDENTIFICATION_NUMBER	比利時國家號碼 (BNN)。
BELGIUM_PASSPORT_NUMBER	護照號碼 (比利時特定)。
BELGIUM_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (比利時特定)。
BELGIUM_VALUE_ADDED_TAX	增值稅 (比利時特定)。

巴西資料類型

資料類型	描述
BRAZIL_BANK_ACCOUNT	銀行帳號 (巴西特定)。
BRAZIL_NATIONAL_IDENTIFICATION_NUMBER	國家識別碼 (巴西特定)。
BRAZIL_NATIONAL_REGISTRY_OF_LEGAL_ENTITIES_NUMBER	核發給公司 (巴西特定) 的識別號碼，也稱為 CNPJ。
BRAZIL_NATURAL_PERSON_REGISTRY_NUMBER	自然人登記號碼，也稱為 CPF。

保加利亞資料類型

資料類型	描述
BULGARIA_DRIVING_LICENSE	駕照號碼 (保加利亞特定)。
BULGARIA_UNIFORM_CIVIL_NUMBER	統一公民號碼 (EGN)，作為國家識別號碼。

資料類型	描述
BULGARIA_VALUE_ADDED_TAX	增值稅 (保加利亞特定)。

加拿大資料類型

資料類型	描述
CANADA_DRIVING_LICENSE	駕照號碼 (加拿大特定)。
CANADA_GOVERNMENT_IDENTIFICATION_CARD_NUMBER	國家識別碼 (加拿大特定)。
CANADA_PASSPORT_NUMBER	護照號碼 (加拿大特定)。
CANADA_PERMANENT_RESIDENCE_NUMBER	永久居留號碼 (PR 卡號)。
CANADA_PERSONAL_HEALTH_NUMBER	醫療保健唯一識別碼 (PHN 號碼)。
CANADA_SOCIAL_INSURANCE_NUMBER	加拿大社會保險號碼 (SIN)。

智利資料類型

資料類型	描述
CHILE_DRIVING_LICENSE	駕照號碼 (智利特定)。
CHILE_NATIONAL_IDENTIFICATION_NUMBER	智利國家識別碼，也稱為 RUT 或 RUN。

中國、香港、澳門和臺灣資料類型

資料類型	描述
CHINA_IDENTIFICATION	中國識別碼。

資料類型	描述
CHINA_LICENSE_PLATE_NUMBER	駕照號碼 (中國特定)。
CHINA_MAINLAND_TRAVEL_PERMIT_ID_HONG_KONG_MACAU	港澳居民來往內地通行證。
CHINA_MAINLAND_TRAVEL_PERMIT_ID_TAIWAN	中華人民共和國 (PRC) 政府核發的臺灣居民來往內地通行證。
CHINA_PASSPORT_NUMBER	護照號碼 (中國特定)。
CHINA_PHONE_NUMBER	電話號碼 (中國特定)。
HONG_KONG_IDENTITY_CARD	香港入境事務處核發的正式身分證明文件。
MACAU_RESIDENT_IDENTITY_CARD	澳門居民身分證 (BIR) 是由澳門身分證明局核發的正式身分證。
TAIWAN_NATIONAL_IDENTIFICATION_NUMBER	國家識別碼 (臺灣特定)
TAIWAN_PASSPORT_NUMBER	護照號碼 (臺灣特定)。

哥倫比亞資料類型

資料類型	描述
COLOMBIA_PERSONAL_IDENTIFICATION_NUMBER	哥倫比亞人出生時指派到的唯一識別碼。
COLOMBIA_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (哥倫比亞特定)。

克羅埃西亞資料類型

資料類型	描述
CROATIA_DRIVING_LICENSE	駕照號碼 (克羅埃西亞特定)。

資料類型	描述
CROATIA_IDENTITY_NUMBER	國家識別碼 (克羅埃西亞特定)。
CROATIA_PASSPORT_NUMBER	護照號碼 (克羅埃西亞特定)。
CROATIA_PERSONAL_IDENTIFICATION_NUMBER	個人識別號碼 (OIB)。

賽普勒斯資料類型

資料類型	描述
CYPRUS_DRIVING_LICENSE	駕照號碼 (賽普勒斯特定)。
CYPRUS_NATIONAL_IDENTIFICATION_NUMBER	賽普勒斯身分證。
CYPRUS_PASSPORT_NUMBER	護照號碼 (賽普勒斯特定)。
CYPRUS_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (賽普勒斯特定)。
CYPRUS_VALUE_ADDED_TAX	增值稅 (賽普勒斯特定)。

捷克資料類型

資料類型	描述
CZECHIA_DRIVING_LICENSE	駕照號碼 (捷克特定)。
CZECHIA_PERSONAL_IDENTIFICATION_NUMBER	個人識別號碼 (捷克特定)。
CZECHIA_VALUE_ADDED_TAX	增值稅 (捷克特定)。

丹麥資料類型

資料類型	描述
DENMARK_DRIVING_LICENSE	駕照號碼 (丹麥特定)。
DENMARK_PERSONAL_IDENTIFICATION_NUMBER	個人識別號碼 (丹麥特定)。
DENMARK_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (丹麥特定)。
DENMARK_VALUE_ADDED_TAX	增值稅 (丹麥特定)。

愛沙尼亞資料類型

資料類型	描述
ESTONIA_DRIVING_LICENSE	駕照號碼 (愛沙尼亞特定)。
ESTONIA_PASSPORT_NUMBER	護照號碼 (愛沙尼亞特定)。
ESTONIA_PERSONAL_IDENTIFICATION_CODE	個人識別號碼 (愛沙尼亞特定)。
ESTONIA_VALUE_ADDED_TAX	增值稅 (愛沙尼亞特定)。

芬蘭資料類型

資料類型	描述
FINLAND_DRIVING_LICENSE	駕照號碼 (芬蘭特定)。
FINLAND_HEALTH_INSURANCE_NUMBER	健康保險號碼 (芬蘭特定)。
FINLAND_NATIONAL_IDENTIFICATION_NUMBER	國家識別號碼 (芬蘭特定)。
FINLAND_PASSPORT_NUMBER	護照號碼 (芬蘭特定)。
FINLAND_VALUE_ADDED_TAX	增值稅 (芬蘭特定)。

法國資料類型

資料類型	描述
FRANCE_BANK_ACCOUNT	銀行帳號 (法國特定)。
FRANCE_DRIVING_LICENSE	駕照號碼 (法國特定)。
FRANCE_HEALTH_INSURANCE_NUMBER	法國健康保險號碼。
FRANCE_INSEE_CODE	法國社會安全號碼，SSN 或 NIR 號碼。
FRANCE_NATIONAL_IDENTIFICATION_NUMBER	法國國家識別碼 (CNI)。
FRANCE_PASSPORT_NUMBER	護照號碼 (法國特定)。
FRANCE_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (法國特定)。
FRANCE_VALUE_ADDED_TAX	增值稅 (法國特定)。

德國資料類型

資料類型	描述
GERMANY_BANK_ACCOUNT	銀行帳號 (德國特定)。
GERMANY_DRIVING_LICENSE	駕照號碼 (德國特定)。
GERMANY_PASSPORT_NUMBER	護照號碼 (德國特定)。
GERMANY_PERSONAL_IDENTIFICATION_NUMBER	個人識別號碼 (德國特定)。
GERMANY_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (德國特定)。
GERMANY_VALUE_ADDED_TAX	增值稅 (德國特定)。

希臘資料類型

資料類型	描述
GREECE_DRIVING_LICENSE	駕照號碼 (希臘特定)。
GREECE_PASSPORT_NUMBER	護照號碼 (希臘特定)。
GREECE_SSN	社會安全號碼 (適用於希臘人)。
GREECE_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (希臘特定)。
GREECE_VALUE_ADDED_TAX	增值稅 (希臘特定)。

匈牙利資料類型

資料類型	描述
HUNGARY_DRIVING_LICENSE	駕照號碼 (匈牙利特定)。
HUNGARY_PASSPORT_NUMBER	護照號碼 (匈牙利特定)。
HUNGARY_SSN	社會安全號碼 (適用於匈牙利人)。
HUNGARY_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (匈牙利特定)。
HUNGARY_VALUE_ADDED_TAX	增值稅 (匈牙利特定)。

冰島資料類型

資料類型	描述
ICELAND_NATIONAL_IDENTIFICATION_NUMBER	國家識別碼 (冰島特定)。
ICELAND_PASSPORT_NUMBER	護照號碼 (冰島特定)。
ICELAND_VALUE_ADDED_TAX	增值稅 (冰島特定)。

印度資料類型

資料類型	描述
INDIA_AADHAAR_NUMBER	印度唯一身分識別機構核發的數位身份證識別號碼。
INDIA_PERMANENT_ACCOUNT_NUMBER	印度永久帳號 (PAN)。

印尼資料類型

資料類型	描述
INDONESIA_IDENTITY_CARD_NUMBER	國家識別碼 (印尼特定)。

愛爾蘭資料類型

資料類型	描述
IRELAND_DRIVING_LICENSE	駕照號碼 (愛爾蘭特定)。
IRELAND_PASSPORT_NUMBER	護照號碼 (愛爾蘭特定)。
IRELAND_PERSONAL_PUBLIC_SERVICE_NUMBER	愛爾蘭個人公共服務號碼 (PPS)。
IRELAND_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (愛爾蘭特定)。
IRELAND_VALUE_ADDED_TAX	增值稅 (愛爾蘭特定)。

以色列資料類型

資料類型	描述
ISRAEL_IDENTIFICATION_NUMBER	國家識別碼 (以色列特定)。

義大利資料類型

資料類型	描述
ITALY_BANK_ACCOUNT	銀行帳號 (義大利特定)。
ITALY_DRIVING_LICENSE	駕照號碼 (義大利特定)。
ITALY_FISCAL_CODE	識別號碼，也稱為義大利稅務號碼 (Italian Codice Fiscale)。
ITALY_PASSPORT_NUMBER	護照號碼 (義大利特定)。
ITALY_VALUE_ADDED_TAX	增值稅 (義大利特定)。

日本資料類型

資料類型	描述
JAPAN_BANK_ACCOUNT	日本銀行帳戶。
JAPAN_DRIVING_LICENSE	日本駕照號碼。
JAPAN_MY_NUMBER	用於稅務管理、社會保障管理和災害響應的日本公民或法人的唯一識別碼
JAPAN_PASSPORT_NUMBER	日本護照號碼。

韓國資料類型

資料類型	描述
KOREA_PASSPORT_NUMBER	護照號碼 (韓國特定)。
KOREA_RESIDENCE_REGISTRATION_NUMBER_FOR_CITIZENS	韓國居民的居住登記號碼。

資料類型	描述
KOREA_RESIDENCE_REGISTRATION_NUMBER_FOR_FOREIGNERS	韓國外國人的居住登記號碼。

拉脫維亞資料類型

資料類型	描述
LATVIA_DRIVING_LICENSE	駕照號碼 (拉脫維亞特定)。
LATVIA_PASSPORT_NUMBER	護照號碼 (拉脫維亞特定)。
LATVIA_PERSONAL_IDENTIFICATION_NUMBER	個人識別號碼 (拉脫維亞特定)。
LATVIA_VALUE_ADDED_TAX	增值稅 (拉脫維亞特定)。

列支敦斯登資料類型

資料類型	描述
LIECHTENSTEIN_NATIONAL_IDENTIFICATION_NUMBER	國家識別碼 (列支敦斯登特定)。
LIECHTENSTEIN_PASSPORT_NUMBER	護照號碼 (列支敦斯登特定)。
LIECHTENSTEIN_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (列支敦斯登特定)。

立陶宛資料類型

資料類型	描述
LITHUANIA_DRIVING_LICENSE	駕照號碼 (立陶宛特定)。

資料類型	描述
LITHUANIA_PERSONAL_IDENTIFICATION_NUMBER	個人識別號碼 (立陶宛特定)。
LITHUANIA_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (立陶宛特定)。
LITHUANIA_VALUE_ADDED_TAX	增值稅 (立陶宛特定)。

盧森堡資料類型

資料類型	描述
LUXEMBOURG_DRIVING_LICENSE	駕照號碼 (盧森堡特定)。
LUXEMBOURG_NATIONAL_INDIVIDUAL_NUMBER	國家識別碼 (盧森堡特定)。
LUXEMBOURG_PASSPORT_NUMBER	護照號碼 (盧森堡特定)。
LUXEMBOURG_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (盧森堡特定)。
LUXEMBOURG_VALUE_ADDED_TAX	增值稅 (盧森堡特定)。

馬來西亞資料類型

資料類型	描述
MALAYSIA_MYKAD_NUMBER	國家識別碼 (馬來西亞特定)。
MALAYSIA_PASSPORT_NUMBER	護照號碼 (馬來西亞特定)。

馬爾他資料類型

資料類型	描述
MALTA_DRIVING_LICENSE	駕照號碼 (馬爾他特定)。
MALTA_NATIONAL_IDENTIFICATION_NUMBER	國家識別碼 (馬爾他特定)。
MALTA_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (馬爾他特定)。
MALTA_VALUE_ADDED_TAX	增值稅 (馬爾他特定)。

墨西哥資料類型

資料類型	描述
MEXICO_CLABE_NUMBER	墨西哥 CLABE (Clave Bancaria Estandarizada) 銀行號碼。
MEXICO_DRIVING_LICENSE	駕照號碼 (墨西哥特定)。
MEXICO_PASSPORT_NUMBER	護照號碼 (墨西哥特定)。
MEXICO_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (墨西哥特定)。
MEXICO_UNIQUE_POPULATION_REGISTRY_CODE	Clave Única de Registro de Población (CURP) 墨西哥的唯一識別碼。

荷蘭資料類型

資料類型	描述
NETHERLANDS_CITIZEN_SERVICE_NUMBER	荷蘭公民號碼 (BSN，burgerservicenummer)。
NETHERLANDS_DRIVING_LICENSE	駕照號碼 (荷蘭特定)。
NETHERLANDS_PASSPORT_NUMBER	護照號碼 (荷蘭特定)。

資料類型	描述
NETHERLANDS_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (荷蘭特定)。
NETHERLANDS_VALUE_ADDED_TAX	增值稅 (荷蘭特定)。
NETHERLANDS_BANK_ACCOUNT	銀行帳號 (荷蘭特定)。

紐西蘭資料類型

資料類型	描述
NEW_ZEALAND_DRIVING_LICENSE	駕照號碼 (紐西蘭特定)。
NEW_ZEALAND_NATIONAL_HEALTH_INDEX_NUMBER	紐西蘭國家健康指數號碼。
NEW_ZEALAND_TAX_IDENTIFICATION_NUMBER	稅務識別號碼，也稱為稅務號碼 (紐西蘭特定)。

挪威資料類型

資料類型	描述
NORWAY_BIRTH_NUMBER	挪威國民身分證號碼。
NORWAY_DRIVING_LICENSE	駕照號碼 (挪威特定)。
NORWAY_HEALTH_INSURANCE_NUMBER	挪威健康保險號碼。
NORWAY_NATIONAL_IDENTIFICATION_NUMBER	國家識別號碼 (挪威特定)。
NORWAY_VALUE_ADDED_TAX	增值稅 (挪威特定)。

菲律賓資料類型

資料類型	描述
PHILIPPINES_DRIVING_LICENSE	駕照號碼 (菲律賓特定)。
PHILIPPINES_PASSPORT_NUMBER	護照號碼 (菲律賓特定)。

波蘭資料類型

資料類型	描述
POLAND_DRIVING_LICENSE	駕照號碼 (波蘭特定)。
POLAND_IDENTIFICATION_NUMBER	波蘭識別碼。
POLAND_PASSPORT_NUMBER	護照號碼 (波蘭特定)。
POLAND_REGON_NUMBER	REGON 識別號碼，也稱為統計識別碼。
POLAND_SSN	社會安全號碼 (適用於波蘭人)。
POLAND_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (波蘭特定)。
POLAND_VALUE_ADDED_TAX	增值稅 (波蘭特定)。

葡萄牙資料類型

資料類型	描述
PORTUGAL_DRIVING_LICENSE	駕照號碼 (葡萄牙特定)。
PORTUGAL_NATIONAL_IDENTIFICATION_NUMBER	國家識別號碼 (葡萄牙特定)。
PORTUGAL_PASSPORT_NUMBER	護照號碼 (葡萄牙特定)。
PORTUGAL_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (葡萄牙特定)。
PORTUGAL_VALUE_ADDED_TAX	增值稅 (葡萄牙特定)。

羅馬尼亞資料類型

資料類型	描述
ROMANIA_DRIVING_LICENSE	駕照號碼 (羅馬尼亞特定)。
ROMANIA_NUMERICAL_PERSONAL_CODE	個人識別號碼 (羅馬尼亞特定)。
ROMANIA_PASSPORT_NUMBER	護照號碼 (羅馬尼亞特定)。
ROMANIA_VALUE_ADDED_TAX	增值稅 (羅馬尼亞特定)。

新加坡資料類型

資料類型	描述
SINGAPORE_DRIVING_LICENSE	駕照號碼 (新加坡特定)。
SINGAPORE_NATIONAL_REGISTRY_IDENTIFICATION_NUMBER	新加坡國民登記身分證。
SINGAPORE_PASSPORT_NUMBER	護照號碼 (新加坡特定)。
SINGAPORE_UNIQUE_ENTITY_NUMBER	新加坡唯一實體編號。

斯洛伐克資料類型

資料類型	描述
SLOVAKIA_DRIVING_LICENSE	駕照號碼 (斯洛伐克特定)。
SLOVAKIA_NATIONAL_IDENTIFICATION_NUMBER	國家識別號碼 (斯洛伐克特定)。
SLOVAKIA_PASSPORT_NUMBER	護照號碼 (斯洛伐克特定)。
SLOVAKIA_VALUE_ADDED_TAX	增值稅 (斯洛伐克特定)。

斯洛維尼亞資料類型

資料類型	描述
SLOVENIA_DRIVING_LICENSE	駕照號碼 (斯洛維尼亞特定)。
SLOVENIA_PASSPORT_NUMBER	護照號碼 (斯洛維尼亞特定)。
SLOVENIA_TAX_IDENTIFICATION_NUMBER	稅務識別號碼 (斯洛維尼亞特定)。
SLOVENIA_UNIQUE_MASTER_CITIZEN_NUMBER	斯洛維尼亞公民的唯一主公民號碼 (JMBG)。
SLOVENIA_VALUE_ADDED_TAX	增值稅 (斯洛維尼亞特定)。

南非資料類型

資料類型	描述
SOUTH_AFRICA_PERSONAL_IDENTIFICATION_NUMBER	個人識別號碼 (南非特定)。

西班牙資料類型

資料類型	描述
SPAIN_BANK_ACCOUNT	銀行帳號 (西班牙特定)。
SPAIN_DNI	西班牙國民身分證 (身分證明文件)。
SPAIN_DRIVING_LICENSE	駕照號碼 (西班牙特定)。
SPAIN_NIE	外國人身分證號碼 (西班牙特定)，也稱為 NIE。
SPAIN_NIF	稅務識別號碼 (西班牙特定)，也稱為 NIF。
SPAIN_PASSPORT_NUMBER	護照號碼 (西班牙特定)。

資料類型	描述
SPAIN_SSN	社會安全號碼 (適用於西班牙人)。
SPAIN_VALUE_ADDED_TAX	增值稅 (西班牙特定)。

斯里蘭卡資料類型

資料類型	描述
SRI_LANKA_NATIONAL_IDENTIFICATION_NUMBER	國家識別碼編號 (斯里蘭卡特定)。

瑞典資料類型

資料類型	描述
SWEDEN_DRIVING_LICENSE	駕照號碼 (瑞典特定)。
SWEDEN_PASSPORT_NUMBER	護照號碼 (瑞典特定)。
SWEDEN_PERSONAL_IDENTIFICATION_NUMBER	國家識別號碼 (瑞典特定)。
SWEDEN_TAX_IDENTIFICATION_NUMBER	瑞典稅務識別號碼 (personnummer)。
SWEDEN_VALUE_ADDED_TAX	增值稅 (瑞典特定)。

瑞士資料類型

資料類型	描述
SWITZERLAND_AHV	瑞士人的社會安全號碼 (AHV)。
SWITZERLAND_HEALTH_INSURANCE_NUMBER	瑞士健康保險號碼。

資料類型	描述
SWITZERLAND_PASSPORT_NUMBER	護照號碼 (瑞士特定)。
SWITZERLAND_VALUE_ADDED_TAX	增值稅 (瑞士特定)。

泰國資料類型

資料類型	描述
THAILAND_PASSPORT_NUMBER	護照號碼 (泰國特定)。
THAILAND_PERSONAL_IDENTIFICATION_NUMBER	個人識別號碼 (泰國特定)。

土耳其資料類型

資料類型	描述
TURKEY_NATIONAL_IDENTIFICATION_NUMBER	國家識別號碼 (土耳其特定)。
TURKEY_PASSPORT_NUMBER	護照號碼 (土耳其特定)。
TURKEY_VALUE_ADDED_TAX	增值稅 (土耳其特定)。

烏克蘭資料類型

資料類型	描述
UKRAINE_INDIVIDUAL_IDENTIFICATION_NUMBER	唯一識別碼 (烏克蘭特定)。
UKRAINE_PASSPORT_NUMBER_DOMESTIC	國內護照號碼 (烏克蘭特定)。

資料類型	描述
UKRAINE_PASSPORT_NUMBER_INTERNATIONAL	國際護照號碼 (烏克蘭特定)。

阿拉伯聯合大公國 (UAE) 資料類型

資料類型	描述
UNITED_ARAB_EMIRATES_PERSONAL_NUMBER	個人識別號碼 (阿拉伯聯合大公國特定)。

英國資料類型

資料類型	描述
UK_BANK_ACCOUNT	英國 (UK) 銀行帳戶。
UK_BANK_SORT_CODE	英國 (UK) 銀行分類代碼。分類代碼是銀行代碼，用於透過各自的清算組織在各自國家/地區的銀行之間進行資金轉帳。
UK_DRIVING_LICENSE	大不列顛和北愛爾蘭聯合王國的駕駛執照號碼 (英國特有)
UK_ELECTORAL_ROLL_NUMBER	選舉名冊號碼 (ERN) 是為英國選舉登記而簽發給個人的識別號碼。此號碼的格式由英國內閣辦公室的英國政府標準規定。
UK_NATIONAL_HEALTH_SERVICE_NUMBER	國家衛生服務 (NHS) 號碼是分配給英國公共衛生服務註冊使用者的唯一號碼。
UK_NATIONAL_INSURANCE_NUMBER	國民保險號碼 (NINO) 是英國 (UK) 用於識別國家保險計畫或社會保障系統之個人的號碼。該號碼有時稱為 NI No 或 NINO。
UK_PASSPORT_NUMBER	英國 (UK) 護照號碼。

資料類型	描述
UK_UNIQUE_TAXPAYER_REFERENCE_NUMBER	英國 (UK) 唯一納稅人參考 (UTR) 號碼。英國政府用來管理稅務系統的識別碼。
UK_VALUE_ADDED_TAX	增值稅是由最終消費者承擔的消費稅。製造和分銷過程中的每筆交易都要支付增值稅。對於英國，增值稅號碼由開展業務所在區域的增值稅辦公室核發。
UK_PHONE_NUMBER	英國 (UK) 電話號碼。

委內瑞拉資料類型

資料類型	描述
VENEZUELA_DRIVING_LICENSE	駕照號碼 (委內瑞拉特定)。
VENEZUELA_NATIONAL_IDENTIFICATION_NUMBER	國家識別號碼 (委內瑞拉特定)。
VENEZUELA_VALUE_ADDED_TAX	增值稅 (委內瑞拉特定)。

使用微調敏感資料偵測

> **Note**
>
> 微調動作僅適用於 AWS Glue 3.0 和 4.0。其中包含 AWS Glue Studio 體驗。持續稽核日誌變更亦不適用於 2.0。
>
> 所有 AWS Glue Studio 3.0 和 4.0 視覺化工作皆具有建立的指令碼，該指令碼會自動使用微調動作 API。

Detect Sensitive Data 轉換可偵測、遮罩或移除您定義或 AWS Glue 預先定義的實體。微調動作可讓您進一步針對每個實體套用特定動作。其他優點包括：

- 系統偵測到資料後會隨即套用動作，使效能獲得改善。

- 可選擇包含或排除特定資料欄。
- 可使用部分遮罩。這可讓您部分遮罩偵測到的敏感資料實體，而非遮罩整個字串。支援具有位移和 regex 的兩種簡易參數。

以下為用於下一節引用之範例工作的敏感資料偵測 API 和微調動作的程式碼片段。

Detect API – 微調動作會使用新的 `detectionParameters` 參數：

```
def detect(
    frame: DynamicFrame,
    detectionParameters: JsonOptions,
    outputColumnName: String = "DetectedEntities",
    detectionSensitivity: String = "LOW"
): DynamicFrame = {}
```

使用具有微調動作的敏感資料偵測 API

使用 detect 的敏感資料偵測 API 會分析特定資料、判斷資料列或資料欄是否為敏感資料實體類型，以及執行使用者針對每個實體類型指定的動作。

使用具有微調動作的 detect API

使用 detect API 並指定 `outputColumnName` 和 `detectionParameters`。

```
    object GlueApp {
      def main(sysArgs: Array[String]) {

        val spark: SparkContext = new SparkContext()
        val glueContext: GlueContext = new GlueContext(spark)

        // @params: [JOB_NAME]
        val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
        Job.init(args("JOB_NAME"), glueContext, args.asJava)

        // Script generated for node S3 bucket. Creates DataFrame from data stored in
 S3.
        val S3bucket_node1 =
 glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar":
 "\"", "withHeader": true, "separator": ",", "optimizePerformance": false}"""),
 connectionType="s3", format="csv", options=JsonOptions("""{"paths":
```

```
["s3://189657479688-ddevansh-pii-test-bucket/tiny_pii.csv"], "recurse": true}"""),
transformationContext="S3bucket_node1").getDynamicFrame()

       // Script generated for node Detect Sensitive Data. Will run detect API for the
DataFrame
       // detectionParameter contains information on which EntityType are being
detected
       // and what actions are being applied to them when detected.
       val DetectSensitiveData_node2 = EntityDetector.detect(
           frame = S3bucket_node1,
           detectionParameters = JsonOptions(
            """
               {
                   "PHONE_NUMBER": [
                       {
                           "action": "PARTIAL_REDACT",
                           "actionOptions": {
                               "numLeftCharsToExclude": "3",
                               "numRightCharsToExclude": "4",
                               "redactChar": "#"
                           },
                           "sourceColumnsToExclude": [ "Passport No", "DL NO#" ]
                       }
                   ],
                   "USA_PASSPORT_NUMBER": [
                       {
                           "action": "SHA256_HASH",
                           "sourceColumns": [ "Passport No" ]
                       }
                   ],
                   "USA_DRIVING_LICENSE": [
                       {
                           "action": "REDACT",
                           "actionOptions": {
                               "redactText": "USA_DL"
                           },
                           "sourceColumns": [ "DL NO#" ]
                       }
                   ]

               }
           """
           ),
           outputColumnName = "DetectedEntities"
```

```
        )

        // Script generated for node S3 bucket. Store Results of detect to S3 location
        val S3bucket_node3 = glueContext.getSinkWithFormat(connectionType="s3",
 options=JsonOptions("""{"path": "s3://189657479688-ddevansh-pii-test-bucket/
test-output/", "partitionKeys": []}"""), transformationContext="S3bucket_node3",
 format="json").writeDynamicFrame(DetectSensitiveData_node2)

        Job.commit()
      }
```

以上程式碼將會從 Amazon S3 的位置建立 DataFrame 並且執行 `detect` API。由於 `detect` API 會要求欄位 `detectionParameters` (實體名稱至要用於該實體之所有動作設定清單的對應) 透過 AWS Glue 的 `JsonOptions` 物件表示，因此也會讓我們擴展 API 的功能。

針對每個實體指定的每個動作，輸入要套用實體/動作組合的所有資料欄名稱清單。這可讓您自訂要針對資料集中每個資料欄偵測的實體，並略過已知不在特定資料欄中的實體。這也可讓您透過省略執行非必要的偵測呼叫這些實體，以提高工作的效能，並可讓您執行每個欄位和實體組合專屬的動作。

仔細查看 `detectionParameters`，會發現範例工作中具有三種實體類型。這三種類型為 Phone Number、USA_PASSPORT_NUMBER 及 USA_DRIVING_LICENSE。針對每個實體類型，AWS Glue 將會執行不同的動作，其中分別為 PARTIAL_REDACT、SHA256_HASH、REDACT 及 DETECT。每個實體類型也會具有要套用的 `sourceColumns` 和/或 `sourceColumnsToExclude` (如果偵測到)。

> **Note**
>
> 每資料欄僅能使用一個就地編輯動作 (PARTIAL_REDACT、SHA256_HASH、或 REDACT)，但 DETECT 動作可與任何這些動作搭配使用。

`detectionParameters` 欄位具有以下配置：

```
    ENTITY_NAME -> List[Actions]
    {
     "ENTITY_NAME": [{
      Action, // required
      ColumnSpecs,
      ActionOptionsMap
```

```
        }],
        "ENTITY_NAME2": [{
      ...
        }]
    }
```

下列為 actions 和 actionOptions 類型：

```
DETECT
{
    # Required
    "action": "DETECT",
    # Optional, depending on action chosen
    "actionOptions": {
        // There are no actionOptions for DETECT
    },
    # 1 of below required, both can also used
    "sourceColumns": [
        "COL_1", "COL_2", ..., "COL_N"
    ],
    "sourceColumnsToExclude": [
        "COL_5"
    ]
}

SHA256_HASH
{
    # Required
    "action": "SHA256_HASH",
    # Required or optional, depending on action chosen
    "actionOptions": {
        // There are no actionOptions for SHA256_HASH
    },

    # 1 of below required, both can also used
    "sourceColumns": [
        "COL_1", "COL_2", ..., "COL_N"
    ],
    "sourceColumnsToExclude": [
        "COL_5"
    ]
}
```

```
REDACT
{
    # Required
    "action": "REDACT",
    # Required or optional, depending on action chosen
    "actionOptions": {
        // The text that is being replaced
        "redactText": "USA_DL"
    },

    # 1 of below required, both can also used
    "sourceColumns": [
        "COL_1", "COL_2", ..., "COL_N"
    ],
    "sourceColumnsToExclude": [
        "COL_5"
    ]
}

PARTIAL_REDACT
{
    # Required
    "action": "PARTIAL_REDACT",
    # Required or optional, depending on action chosen
    "actionOptions": {
        // number of characters to not redact from the left side
        "numLeftCharsToExclude": "3",
        // number of characters to not redact from the right side
        "numRightCharsToExclude": "4",
        // the partial redact will be made with this redacted character
        "redactChar": "#",
        // regex pattern for partial redaction
        "matchPattern": "[0-9]"
    },

    # 1 of below required, both can also used
    "sourceColumns": [
        "COL_1", "COL_2", ..., "COL_N"
    ],
    "sourceColumnsToExclude": [
        "COL_5"
    ]
}
```

當指令碼執行後，結果會輸出至指定的 Amazon S3 位置。您可以在 Amazon S3 中檢視資料，但其中特定的實體類型會根據選取的動作受到敏感處理。在此情況下，我們會得到的資料列結果如下所示：


```
{
    "Name": "Colby Schuster",
    "Address": "39041 Antonietta Vista, South Rodgerside, Nebraska 24151",
    "Car Owned": "Fiat",
    "Email": "Kitty46@gmail.com",
    "Company": "O'Reilly Group",
    "Job Title": "Dynamic Functionality Facilitator",
    "ITIN": "991-22-2906",
    "Username": "Cassandre.Kub43",
    "SSN": "914-22-2906",
    "DOB": "2020-08-27",
    "Phone Number": "1-2#######1718",
    "Bank Account No": "69741187",
    "Credit Card Number": "6441-6289-6867-2162-2711",
    "Passport No": "94f311e93a623c72ccb6fc46cf5f5b0265ccb42c517498a0f27fd4c43b47111e",
    "DL NO#": "USA_DL"
}
```


在以上指令碼中，Phone Number 受到 # 部分遮蔽。Passport No 變更為 SHA256 雜湊。DL NO# 經偵測為美國駕照號碼，因此遭遮蔽為 “USA_DL” (如 detectionParameters 中所述)。

> **Note**
>
> 由於 API 的性質，因此 classifyColumns API 無法與微調動作搭配使用。此 API 會執行資料欄取樣 (可依使用者調整，但具有預設值)，以加快執行偵測速度。由於此原因，微調動作將需要針對每個值進行反覆運算。

持續稽核日誌

全新引進具有微調動作的功能 (但在使用一般 API 時亦適用) 為產生持續稽核日誌。目前執行 detect API 將會新增其他資料欄 (預設為 DetectedEntities，但可透過 outputColumnName 自訂) 參數，其中包含 PII 偵測中繼資料。現在這樣會產生 "actionUsed" 中繼資料索引鍵，可能會是 DETECT、PARTIAL_REDACT、SHA256_HASH、REDACT 其中一個。

```
"DetectedEntities": {
    "Credit Card Number": [
        {
            "entityType": "CREDIT_CARD",
            "actionUsed": "DETECT",
            "start": 0,
            "end": 19
        }
    ],
    "Phone Number": [
        {
            "entityType": "PHONE_NUMBER",
            "actionUsed": "REDACT",
            "start": 0,
            "end": 14
        }
    ]
}
```

即使客戶使用不具有微調動作的 API (例如，`detect(entityTypesToDetect, outputColumnName)`)，仍會在產生的 DataFrame 中看見這項持續稽核日誌。

使用具有微調動作之 API 的客戶將會看到所有動作，無論其是否遭到遮蔽。範例：

```
+---------------------+---------------
+-------------------------------------------------------------------------------------------------------
+
| Credit Card Number  |  Phone Number  |
                                              DetectedEntities
                                                                  |
+---------------------+---------------
+-------------------------------------------------------------------------------------------------------
+
| 622126741306XXXX    | +12#####7890   | {"Credit Card Number":
[{"entityType":"CREDIT_CARD","actionUsed":"PARTIAL_REDACT","start":0,"end":16}],"Phone
 Number":
[{"entityType":"PHONE_NUMBER","actionUsed":"PARTIAL_REDACT","start":0,"end":12}]}} |
| 6221 2674 1306 XXXX | +12#######7890 | {"Credit Card Number":
[{"entityType":"CREDIT_CARD","actionUsed":"PARTIAL_REDACT","start":0,"end":19}],"Phone
 Number":
[{"entityType":"PHONE_NUMBER","actionUsed":"PARTIAL_REDACT","start":0,"end":14}]}} |
```

```
| 6221-2674-1306-XXXX | 22#######7890  | {"Credit Card Number":
[{"entityType":"CREDIT_CARD","actionUsed":"PARTIAL_REDACT","start":0,"end":19}],"Phone
 Number":
[{"entityType":"PHONE_NUMBER","actionUsed":"PARTIAL_REDACT","start":0,"end":14}]}} |
+--------------------+---------------
+-------------------------------------------------------------------------------------------------------
+
```

如果您不想要看到 DetectedEntities 資料欄，可以在自訂程式碼中直接捨棄該額外資料欄。

AWS Glue 視覺化任務 API

AWS Glue 提供一個 API，允許客戶使用 AWS Glue API 從 JSON 物件建立資料整合，而該使用來自代表視覺化步驟工作流程之 的 。然後，客戶可以使用 AWS Glue Studio 中的視覺化編輯器處理這些任務。

如需有關視覺化任務 API 資料類型的詳細資訊，請參閱視覺化任務 API。

主題

- API 設計和 CRUD API
- 入門
- 視覺化任務限制

API 設計和 CRUD API

CreateJob 和 UpdateJob API 現已支援一個額外的選用參數，即 codeGenConfigurationNodes。為此欄位提供非空白的 JSON 結構將導致在 AWS Glue Studio 中為建立的任務和正在產生的相關程式碼註冊 DAG。系統將忽略任務建立時此欄位的空值或空字串。

透過 UpdateJob AWS Glue API，會以與 CreateJob 類似的方式更新 codeGenConfigurationNodes 欄位。應該在 UpdateJob 中指定整個欄位，其中 DAG 已視需要變更。系統會忽略所提供的空值，且不會對 DAG 執行任何更新。空的結構或字串 codeGenConfigurationNodes 設定為空白，並移除任何先前的 DAG。GetJob API 將會傳回 DAG (若有)。DeleteJob API 也會刪除任何相關聯的 DAG。

入門

若要建立任務，請使用 CreateJob 動作。`CreateJob` 請求輸入會有一個額外欄位 "codeGenConfigurationNodes"，讓您可在其中指定 JSON 中的 DAG 物件。

需要謹記的事項：

- ‘codeGenConfigurationNodes’ 是節點的 nodeId 映射。
- 每個節點都以識別其類型的索引鍵開頭。
- 節點只能是一種類型，因此只能指定一個索引鍵。
- 輸入欄位包含當前節點的父節點。

以下為 createJob 輸入的 JSON 表示法。

```
{
  "node-1": {
    "S3CatalogSource": {
      "Table": "csvFormattedTable",
      "PartitionPredicate": "",
      "Name": "S3 bucket",
      "AdditionalOptions": {},
      "Database": "myDatabase"
    }
  },
  "node-3": {
    "S3DirectTarget": {
      "Inputs": ["node-2"],
      "PartitionKeys": [],
      "Compression": "none",
      "Format": "json",
      "SchemaChangePolicy": { "EnableUpdateCatalog": false },
      "Path": "",
      "Name": "S3 bucket"
    }
  },
  "node-2": {
    "ApplyMapping": {
      "Inputs": ["node-1"],
      "Name": "ApplyMapping",
      "Mapping": [
        {
          "FromType": "long",
          "ToType": "long",
          "Dropped": false,
          "ToKey": "myheader1",
          "FromPath": ["myheader1"]
```

```
        },
        {
          "FromType": "long",
          "ToType": "long",
          "Dropped": false,
          "ToKey": "myheader2",
          "FromPath": ["myheader2"]
        },
        {
          "FromType": "long",
          "ToType": "long",
          "Dropped": false,
          "ToKey": "myheader3",
          "FromPath": ["myheader3"]
        }
      ]
    }
  }
}
```

更新及取得任務

由於 UpdateJob 也將會有 "codeGenConfigurationNodes" 欄位，因此輸入格式將相同。請參閱 UpdateJob 動作。

GetJob 動作也會以相同的格式傳回 "codeGenConfigurationNodes" 欄位。請參閱 GetJob 動作。

視覺化任務限制

由於 'codeGenConfigurationNodes' 參數已新增至現有的 API 中，這些 API 中的任何限制將得到繼承。此外，codeGenConfigurationNodes 和某些節點的大小會受到限制。請參閱任務結構以了解詳細資訊。

Ray 指令碼程式設計

AWS Glue 使 Ray 指令碼的撰寫和執行更容易。本節介紹 AWS Glue for Ray 中可用的支援的 Ray 功能。以 Python 進行 Ray 指令碼的程式設計。

自訂指令碼必須與任務定義中 `Runtime` 欄位定義的 Ray 版本相容。如需有關任務 API 中 `Runtime` 的詳細資訊，請參閱 the section called “任務”。如需有關每個執行期環境的資訊，請參閱 the section called “支援的 Ray 執行期環境”。

主題

教學課程：在 AWS Glue for Ray 中撰寫 ETL 指令碼

Ray 可讓您在 Python 中以原生方式撰寫和擴展分散式任務。AWS Glue for Ray 提供無伺服器 Ray 環境，可讓您同時存取任務與互動式工作階段 (Ray 互動式工作階段為預覽版)。AWS Glue 任務系統可讓您運用一致的方式來管理和執行任務，包括按排程、透過觸發或從 AWS Glue 主控台進行。

結合這些 AWS Glue 工具可建立強大的工具鏈，讓您用來擷取、轉換和載入 (ETL) 工作負載，此為最受歡迎的 AWS Glue 使用案例。在本教學課程中，您將了解整合使用此解決方案的基礎概念。

我們也支援使用適用於 Spark 的 AWS Glue 來處理 ETL 工作負載。如需有關撰寫適用於 Spark 的 AWS Glue 指令碼的教學課程，請參閱 the section called “教學課程：撰寫 Spark 指令碼”。如需有關可用引擎的詳細資訊，請參閱 the section called “AWS Glue for Spark 與 AWS Glue for Ray”。Ray 能夠在分析、機器學習 (ML) 和應用程式開發過程中處理眾多不同類型的任務。

在本教學課程中，您將擷取、轉換和載入在 Amazon Simple Storage Service (Amazon S3) 中託管的 CSV 資料集。您將從存放在公有 Amazon S3 儲存貯體中的紐約市計程車禮車委員會 (TLC) 行程記錄資料集開始。如需有關此資料集的詳細資訊，請參閱 AWS 上的開放資料登錄檔。

您將使用 Ray Data 程式庫中可用的預先定義轉換來轉換資料。Ray Data 是由 Ray 設計的資料集準備程式庫，預設包含在 AWS Glue for Ray 環境中。如需有關這些預設內含程式庫的詳細資訊，請參閱 the section called “Ray 任務隨附的模組”。接著，您可以將轉換後的資料寫入您控制的 Amazon S3 儲存貯體。

先決條件：在本教學課程中，您需要一個可存取 AWS Glue 和 Amazon S3 的 AWS 帳戶。

步驟 1：在 Amazon S3 中建立儲存貯體以保存您的輸出資料

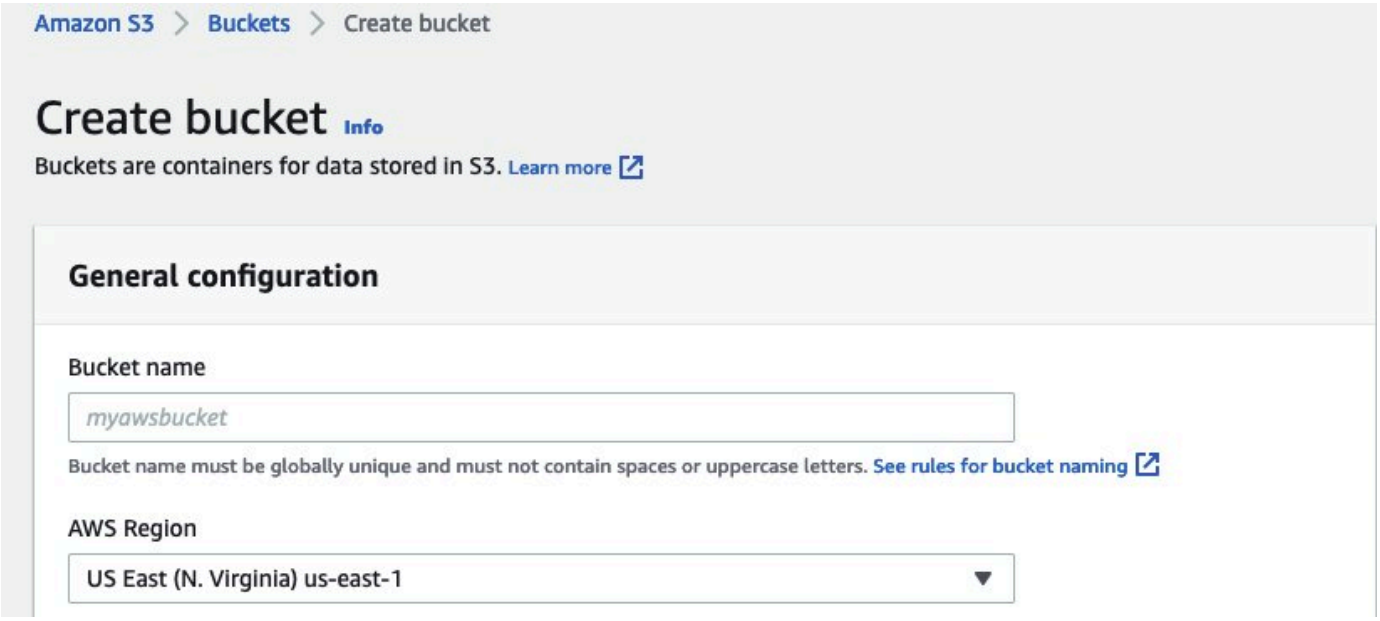
您將需要一個由您控制的 Amazon S3 儲存貯體，作為本教學課程中所建立資料的接收器。您可以使用下列程序建立此儲存貯體。

> **Note**
>
> 如果您想要將資料寫入您控制的現有儲存貯體，則可略過此步驟。記下 *yourBucketName* (現有儲存貯體的名稱)，以便在後續步驟中使用。

為 Ray 任務輸出建立儲存貯體

- 按照《Amazon S3 使用者指南》的建立儲存貯體中所述的步驟，建立儲存貯體。
  - 選擇儲存貯體名稱時，請記下 *yourBucketName*，您將在後續步驟中參考該名稱。
  - 對於其他組態，Amazon S3 主控台中提供的建議設定應可在本教學課程中正常運作。

  舉例來說，儲存貯體建立對話方塊在 Amazon S3 主控台中可能看起來會像這樣。



步驟 2：為 Ray 任務建立 IAM 角色與政策

您的任務將需要具有以下內容的 AWS Identity and Access Management (IAM) 角色：

- 由 `AWSGlueServiceRole` 受管政策授予的許可。這些是執行 AWS Glue 任務所需的基本許可。
- 適用於 `nyc-tlc/*` Amazon S3 資源的 `Read` 存取層級許可。
- 適用於 *yourBucketName*`/*` Amazon S3 資源的 `Write` 存取層級許可。
- 可讓 `glue.amazonaws.com` 主體擔任角色的信任關係。

您可以使用下列程序建立此角色。

為 AWS Glue for Ray 任務建立 IAM 角色

> **Note**
>
> 您可以按照許多不同的程序建立 IAM 角色。如需有關如何佈建 IAM 資源的詳細資訊或選項，請參閱 AWS Identity and Access Management 文件。

1. 依照《IAM 使用者指南》中使用視覺化編輯器來建立 IAM 政策 (主控台) 所述的下列步驟，建立政策以定義先前所述的 Amazon S3 許可。
   - 選取服務時，請選擇 Amazon S3。
   - 選取政策的許可時，請針對下列資源 (先前所述) 附加下列動作集：
     - 適用於 `nyc-tlc/*` Amazon S3 資源的讀取存取層級許可。
     - 適用於 *`yourBucketName`*`/*` Amazon S3 資源的寫入存取層級許可。
   - 選取政策名稱時，請記下 *`YourPolicyName`*，您將在後續步驟中參考該名稱。
2. 依照《IAM 使用者指南》中為 AWS 服務建立角色 (主控台) 所述的步驟，為 AWS Glue for Ray 任務建立角色。
   - 選取信任的 AWS 服務實體時，請選擇 `Glue`。這會自動為您的任務填入必要的信任關係。
   - 選取許可政策的政策時，請附加下列政策：
     - `AWSGlueServiceRole`
     - *`YourPolicyName`*
   - 選取角色名稱時，請記下 *`YourRoleName`*，您將在後續步驟中參考該名稱。

步驟 3：建立並執行 AWS Glue for Ray 任務。

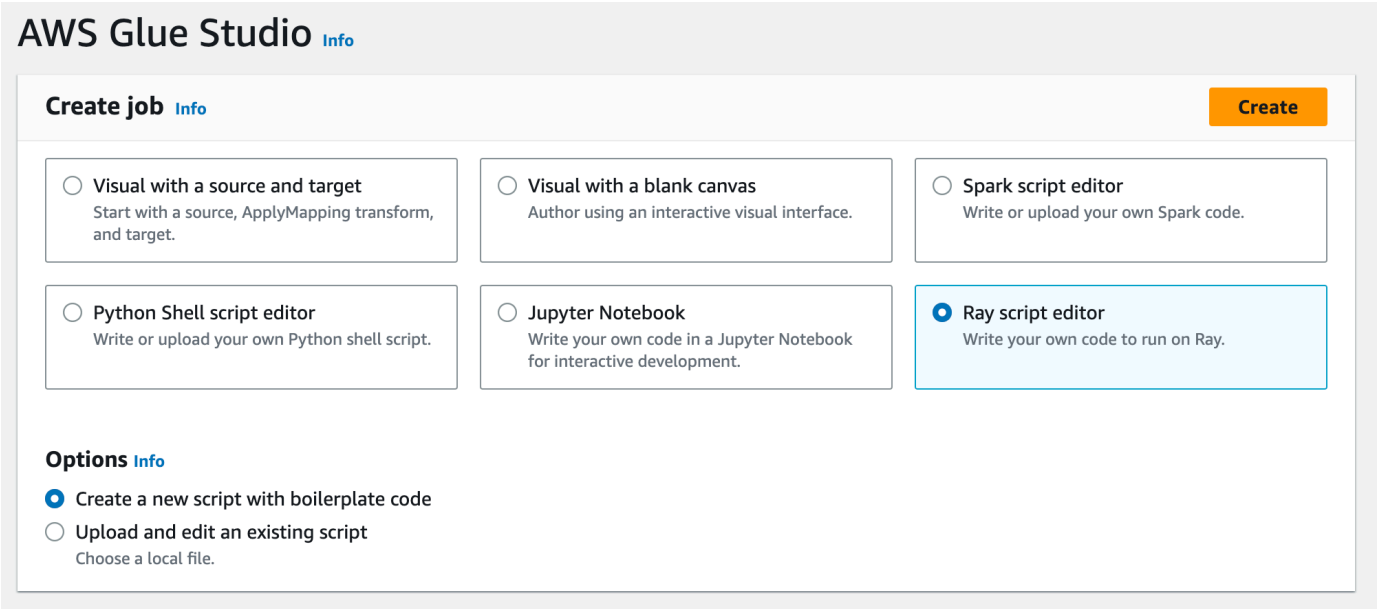
在此步驟中，您需使用 AWS Management Console 建立 AWS Glue 任務、為其提供範例指令碼以及執行任務。當您建立任務時，系統會在主控台中建立一個位置，供您存放、設定和編輯 Ray 指令碼。如需有關建立任務的詳細資訊，請參閱the section called “登入主控台”。

在本教學課程中，我們會討論以下 ETL 案例：您想要從 紐約市計程車禮車委員會 (TLC) 行程記錄資料集中讀取 2022 年 1 月的記錄，透過合併現有資料欄中的資料，在資料集中新增資料欄 (`tip_rate`)，然後移除一些與目前分析無關的資料欄，接著再將結果寫入 *`yourBucketName`*。下列 Ray 指令碼會執行這些步驟：

```
import ray
import pandas
from ray import data

ray.init('auto')

ds = ray.data.read_csv("s3://nyc-tlc/opendata_repo/opendata_webconvert/yellow/
yellow_tripdata_2022-01.csv")

# Add the given new column to the dataset and show the sample record after adding a new
 column
ds = ds.add_column( "tip_rate", lambda df: df["tip_amount"] / df["total_amount"])

# Dropping few columns from the underlying Dataset
ds = ds.drop_columns(["payment_type", "fare_amount", "extra", "tolls_amount",
 "improvement_surcharge"])

ds.write_parquet("s3://yourBucketName/ray/tutorial/output/")
```

建立並執行 AWS Glue for Ray 任務

1. 在 AWS Management Console 中，導覽至 AWS Glue 登陸頁面。
2. 在側邊的導覽窗格中，選擇 ETL 任務。
3. 在建立任務中，選擇 Ray 指令碼編輯器，然後選擇建立，如下圖所示。



4. 將指令碼的全文貼至指令碼窗格中，並替換任何現有的文字。

5. 導覽至任務詳細資訊，並將 IAM 角色屬性設定為 *YourRoleName*。
6. 選擇儲存，然後選擇執行。

步驟 4：檢查輸出

執行 AWS Glue 任務之後，您應驗證輸出符合此案例的期望值。若要執行此作業，請依照下列程序操作。

驗證您的 Ray 任務是否順利執行

1. 在任務詳細資訊頁面上，導覽至執行。
2. 幾分鐘後，您應會看到執行狀態為成功的執行。
3. 在 https://console.aws.amazon.com/s3/ 上導覽至 Amazon S3 主控台，並檢查 *yourBucketName*。您應會看到寫入輸出儲存貯體的檔案。
4. 讀取 Parquet 檔案並驗證其內容。您可以使用現有工具執行此操作。如果您沒有驗證 Parquet 檔案的程序，可透過 AWS Glue 互動式工作階段，在 AWS Glue 主控台中使用 Spark 或 Ray (預覽版) 執行此操作。

   在互動式工作階段中，您可以存取預設提供的 Ray Data、Spark 或 pandas 程式庫 (根據您選擇的引擎)。若要驗證您的檔案內容，您可使用適用於這些程式庫的常用檢查方法，例如 `count`、`schema` 和 `show` 等。如需有關主控台中互動式工作階段的詳細資訊，請參閱 Using notebooks with AWS Glue Studio and AWS Glue。

   由於您已確認檔案寫入儲存貯體，您可以相對地確認若輸出有任何問題，則與 IAM 組態無關。使用 *yourRoleName* 設定工作階段，以存取相關檔案。

若結果不如預期，請檢查本指南中的疑難排解內容，以識別並修復錯誤的來源。若要解譯任務執行錯誤狀態，請參閱 the section called “任務執行狀態”。您可以在 AWS Glue 疑難排解 一章中找到疑難排解內容。如需了解與 Ray 任務相關的特定錯誤，請參閱疑難排解一章中的 the section called “對 Ray 錯誤進行疑難排解”。

後續步驟

您現已充分了解並使用 AWS Glue for Ray 執行 ETL 程序。您可使用下列資源了解 AWS Glue for Ray 提供哪些工具來大規模轉換和解譯資料。

- 如需有關 Ray 任務模型的詳細資訊，請參閱 the section called “在 AWS Glue for Ray 中使用 Ray Core 和 Ray Data”。如需有關使用 Ray 任務的更多經驗，請參考 Ray Core 文件中的範例。參閱 Ray 文件中的 Ray Core：Ray 教學課程與範例 (2.4.0)。
- 如需有關 AWS Glue for Ray 的可用資料管理程式庫指引，請參閱 the section called “連線至資料”。如需有關使用 Ray Data 轉換和寫入資料集的更多經驗，請參考 Ray Data 文件中的範例。參閱 Ray Data：範例 (2.4.0)。
- 如需有關設定 AWS Glue for Ray 任務的詳細資訊，請參閱 the section called “使用 Ray 任務”。
- 如需有關撰寫 AWS Glue for Ray 指令碼的詳細資訊，請繼續閱讀本節中的文件。

在 AWS Glue for Ray 中使用 Ray Core 和 Ray Data

Ray 是透過在叢集中分發任務來縱向擴展 Python 指令碼的架構。您可以使用 Ray 作為各種問題的解決方案，因此 Ray 提供程式庫以最佳化特定任務。在 AWS Glue 中，我們著重於使用 Ray 轉換大型資料集。AWS Glue 提供 Ray Data 和部分 Ray Core 的支援，以協助執行此任務。

什麼是 Ray Core？

建立分發應用程式的第一步，即是識別與定義可同時執行的任務。Ray Core 包含部分的 Ray，可讓您用來定義能夠同時執行的任務。Ray 提供了參考與快速入門資訊，您可以使用這些資訊來學習其提供的工具。如需詳細資訊，請參閱 What is Ray Core? 和 Ray Core Quick Start。如需有關有效定義可在 Ray 中同時執行之任務的詳細資訊，請參閱 Tips for first-time users。

> **Ray 任務與執行者**
>
> 在 AWS Glue for Ray 文件中，我們可參考任務與執行者，其為 Ray 中的核心概念。
> Ray 使用 Python 函數和類別作為分散式運算系統的建置區塊。就像 Python 函數和變數在類別中使用時會變為「方法」和「屬性」一樣，若在 Ray 中用於向工作者傳送程式碼，則函數會變為「任務」，且類別會變為「執行者」。您可依 `@ray.remote` 注釋，識別可能由 Ray 使用的函數和類別。
> 任務與執行者可供設定，其具有生命週期，且會在其生命週期中佔用運算資源。當您找到問題的根本原因時，引發錯誤的程式碼可以追溯至任務或執行者。因此，當您了解如何設定、監控或偵錯 AWS Glue for Ray 任務時，可能會出現這些術語。
> 若要開始了解如何有效使用任務和執行者來建置分發應用程式，請參閱 Ray 文件中的 Key Concepts。

AWS Glue for Ray 中的 Ray Core

AWS Glue for Ray 環境可管理叢集的形成與擴展，以及收集和視覺化日誌。我們管理這些問題，因此會限制存取和支援在 Ray Core 中用來解決這些開放原始碼叢集問題的 API。

在受管 `Ray2.4` 執行期環境中，我們不支援以下項目：

- Ray Core CLI
- Ray State CLI
- `ray.util.metrics` Prometheus 指標公用程式方法：
  - 計數器
  - 量測計
  - 直方圖
- 其他偵錯工具：
  - ray.util.pdb.set_trace
  - ray.util.inspect_serializability
  - ray.timeline

什麼是 Ray Data？

當您連線至資料來源和目的地、處理資料集以及啟動常見轉換時，Ray Data 是使用 Ray 解決 Ray 資料集轉換問題的簡單方法。如需有關使用 Ray Data 的詳細資訊，請參閱 Ray 資料集：分散式資料預先處理。

您可以使用 Ray Data 或其他工具來存取資料。如需有關在 Ray 中存取資料的詳細資訊，請參閱 the section called “連線至資料”。

AWS Glue for Ray 中的 Ray Data

在受管 `Ray2.4` 執行期環境中，預設會支援並提供 Ray Data。如需有關所提供模組的詳細資訊，請參閱 the section called “Ray 任務隨附的模組”。

為 Ray 任務提供檔案和 Python 程式庫

本節提供的資訊有助於您搭配 AWS Glue Ray 任務使用 Python 程式庫。您可以使用預設包含於所有 Ray 任務的某些常見程式庫。您還可以為自己的 Ray 任務提供專屬 Python 程式庫。

Ray 任務隨附的模組

您可以使用下列提供的套件，在 Ray 任務中執行資料整合工作流程。這些套件預設可在 Ray 任務中使用。

AWS Glue version 4.0

在 AWS Glue 4.0 中，Ray (`Ray2.4` 執行期) 環境提供下列套件：

- boto3 == 1.26.133
- ray == 2.4.0
- pyarrow == 11.0.0
- pandas == 1.5.3
- numpy == 1.24.3
- fsspec == 2023.4.0

此清單包含將隨 `ray[data] == 2.4.0` 一併安裝的所有套件。Ray Data 支援開箱即用。

為 Ray 任務提供檔案

您可以使用 `--working-dir` 參數為 Ray 任務提供檔案。為此參數提供指向 Amazon S3 上託管之 .zip 檔案的路徑。在 .zip 檔案中，您的檔案必須包含在單一頂層目錄中。頂層不應具有其他任何檔案。

在指令碼開始執行之前，您的檔案將會分發至每個 Ray 節點。請考慮這可能會如何影響每個 Ray 節點的可用磁碟空間。可用磁碟空間是由任務組態中設定的 WorkerType 決定。如果您想大規模提供任務資料，此機制並非適合的解決方案。如需有關為任務提供資料的詳細資訊，請參閱 the section called “連線至資料”。

您的檔案將可供存取，如同目錄透過 `working_dir` 參數提供給 Ray 一樣。例如，若要讀取 .zip 檔案頂層目錄中名為 `sample.txt` 的檔案，則可呼叫：

```
@ray.remote
def do_work():
    f = open("sample.txt", "r")
    print(f.read())
```

如需有關 `working_dir` 的詳細資訊，請參閱 Ray 文件。此功能的運作方式與 Ray 的原生功能類似。

Ray 任務的其他 Python 模組

PyPI 中的其他模組

Ray 任務會使用 Python Package Installer (pip3)，安裝 Ray 指令碼使用的其他模組。您可以使用 `--pip-install` 參數與逗號分隔的 Python 模組清單來新增新模組或變更現有模組的版本。

例如，若要更新或新增 `scikit-learn` 模組，請使用以下鍵值對：

```
"--pip-install", "scikit-learn==0.21.3"
```

如果您有自訂模組或自訂修補程式，則可使用 `--s3-py-modules` 參數從 Amazon S3 分發專屬程式庫。您的分發內容可能需要重新封裝並重建才能上傳。遵循 the section called “在 Ray 任務中包含 Python 程式碼” 中的指引操作。

來自 Amazon S3 的自訂分發

自訂發行版本應遵守 Ray 封裝相依性的準則。您可在下一節中找到關於如何建立這些發行版本的資訊。如需有關 Ray 如何設定相依性的詳細資訊，請參閱 Ray 文件中的 Environment Dependencies (環境相依性)。

若要在評估其內容後加入自訂分發套件，請將您的分發套件上傳至可供任務 IAM 角色使用的儲存貯體。在您的參數組態中，指定 Python zip 封存的 Amazon S3 路徑。如果您提供多個分發套件，請用逗號加以分隔。例如：

```
"--s3-py-modules", "s3://s3bucket/pythonPackage.zip"
```

限制

Ray 任務不支援在任務環境中編譯原生程式碼。如果您的 Python 相依性遞移地依賴於原生的編譯程式碼，則可能會受此限制。Ray 任務可執行提供的二進位檔案，但必須在 ARM64 上針對 Linux 進行編譯。這表示您可使用 `aarch64manylinux` wheel 的內容。您可將 wheel 重新封裝為 Ray 標準，以編譯形式提供原生相依性。這通常表示會移除 `dist-info` 資料夾，讓封存的根僅具有一個資料夾。

您無法使用此參數升級 `ray` 或 `ray[data]` 的版本。若要使用新版本的 Ray，您需要在我們發行相關支援後變更任務上的執行期欄位。如需有關支援的 Ray 版本的詳細資訊，請參閱 the section called “AWS Glue 版本”。

在 Ray 任務中包含 Python 程式碼

Python 軟體基金會提供封裝 Python 檔案的標準化行為，以便在不同的執行階段使用。Ray 引入了您應留意的封裝標準限制。AWS Glue 不會指定超出 Ray 規定的封裝標準。下列說明提供關於封裝簡易 Python 套件的標準指引。

將檔案封裝於 `.zip` 封存。目錄應位於封存的根目錄中。封存的根層級不應具有其他檔案，否則會導致發生非預期的行為。根目錄為套件，其名稱用於在匯入時參照您的 Python 程式碼。

若您透過 `--s3-py-modules` 使用此表單提供分發套件給 Ray 任務，將可從 Ray 指令碼中的套件匯入 Python 程式碼。

您的套件可提供包含一些 Python 檔案的單一 Python 模組，或者您可以將眾多模組整併封裝。重新封裝諸如 PyPI 的程式庫等相依性時，請在這些套件中檢查是否有隱藏的檔案和中繼資料目錄。

> **Warning**
>
> 某些作業系統行為會讓您難以正確遵循這些封裝指示。
>
> - OSX 可能會將 `__MACOSX` 等隱藏檔案新增至頂層的 zip 檔案。
> - Windows 可能會自動將您的檔案新增至 zip 內的資料夾中，而在無意間建立巢狀資料夾。

下列程序假設您正在與 Amazon Linux 2，或是提供 Info-Zip `zip` 和 `zipinfo` 公用程式分發套件之類似作業系統中的檔案互動。我們建議您使用這些工具來防止意外行為。

封裝 Python 檔案以供在 Ray 中使用

1. 使用您的套件名稱建立臨時目錄，然後確認您的工作目錄是其父目錄。您可使用下列命令來執行此作業：

   ```
   cd parent_directory
   mkdir temp_dir
   ```

2. 將檔案複製到臨時目錄中，然後確認您的目錄結構。此目錄的內容將作為您的 Python 模組直接存取。您可使用下列命令來執行此作業：

   ```
   ls -AR temp_dir
   # my_file_1.py
   # my_file_2.py
   ```

3. 使用 zip 壓縮您的臨時資料夾。您可使用下列命令來執行此作業：

   ```
   zip -r zip_file.zip temp_dir
   ```

4. 確認檔案已正確封裝。*zip_file*.zip 現應可在您的工作目錄中找到。您可使用下列命令來檢查：

```
zipinfo -1 zip_file.zip
# temp_dir/
# temp_dir/my_file_1.py
# temp_dir/my_file_2.py
```

重新封裝 Python 套件以供在 Ray 中使用。

1. 使用您的套件名稱建立臨時目錄，然後確認您的工作目錄是其父目錄。您可使用下列命令來執行此作業：

```
cd parent_directory
mkdir temp_dir
```

2. 解壓縮套件並將內容複製到臨時目錄中。移除與先前封裝標準相關的檔案，只留下模組內容。使用下列命令確認檔案結構看起來正確無誤：

```
ls -AR temp_dir
# my_module
# my_module/__init__.py
# my_module/my_file_1.py
# my_module/my_submodule/__init__.py
# my_module/my_submodule/my_file_2.py
# my_module/my_submodule/my_file_3.py
```

3. 使用 zip 壓縮您的臨時資料夾。您可使用下列命令來執行此作業：

```
zip -r zip_file.zip temp_dir
```

4. 確認檔案已正確封裝。*zip_file*.zip 現應可在您的工作目錄中找到。您可使用下列命令來檢查：

```
zipinfo -1 zip_file.zip
# temp_dir/my_module/
# temp_dir/my_module/__init__.py
# temp_dir/my_module/my_file_1.py
# temp_dir/my_module/my_submodule/
```

```
# temp_dir/my_module/my_submodule/__init__.py
# temp_dir/my_module/my_submodule/my_file_2.py
# temp_dir/my_module/my_submodule/my_file_3.py
```

連線至 Ray 任務中的資料

AWS Glue Ray 任務可以使用各種專為您快速整合資料而設計的 Python 套件。我們提供了一組最小的相依性，以免造成您的環境混亂。如需有關這些預設內含項目的詳細資訊，請參閱 the section called "Ray 任務隨附的模組"。

> **Note**
>
> AWS Glue擷取、轉換和載入 (ETL) 提供了 DynamicFrame 抽象化，以簡化 ETL 工作流程，您可以在其中解決資料集中資料列之間的結構定義差異。 AWS GlueETL 提供其他功能：任務書籤和分組輸入檔案。我們目前不在 Ray 任務中提供對應的功能。
> 適用於 Spark 的 AWS Glue 為連線至特定資料格式、來源和接收器提供直接支援。在 Ray 中，適用於 pandas 和當前第三方程式庫的 AWS SDK 已實質涵蓋該需求。您將需要查閱這些程式庫，以了解可用的功能。

目前無法與 Amazon VPC 進行 AWS Glue for Ray 整合。若無公有路由，將無法存取 Amazon VPC 中的資源。如需有關使用 AWS Glue 搭配 Amazon VPC 的詳細資訊，請參閱 the section called "VPC 端點 (AWS PrivateLink)"。

用於在 Ray 中處理資料的常用程式庫

Ray Data：Ray Data 提供了處理常用資料格式、來源和接收器的方法。如需有關 Ray Data 中支援之格式和來源的詳細資訊，請參閱 Ray Data 文件中的輸入/輸出。Ray Data 是一個固定程式庫而非通用程式庫，用於處理資料集。

Ray 針對 Ray Data 可能是您任務最佳解決方案的使用案例，提供某些指引。如需詳細資訊，請參閱 Ray 文件中的 Ray 使用案例。

AWS熊貓 SDK (awswrangler) — 熊貓的 AWS SDK 是一種AWS產品，當您的轉換管理數據與熊貓管理數據時，可提供乾淨，經過測試的解決方案，用於讀取和寫入AWS服務。 DataFrames如需有關適用於 pandas 的 AWS SDK 支援的格式與來源詳細資訊，請參閱適用於 pandas 的 AWS SDK 文件中的 API 參考。

如需如何使用適用於 pandas 的 AWS SDK 讀取和寫入資料的範例，請參閱適用於 pandas 的 AWS SDK 文件中的快速入門。適用於 pandas 的 AWS SDK 不提供資料轉換功能。其僅提供對於讀取與寫入來源的支援。

Modin：Modin 是一個 Python 程式庫，可採用分發套件方式實作常用的 pandas 操作。如需有關 Modin 的詳細資訊，請參閱 Modin 文件。Modin 本身不提供對於讀取與寫入來源的支援。其提供常用轉換的分發式實作。Modin 由適用於 pandas 的 AWS SDK 提供支援。

當您在 Ray 環境中一併執行 Modin 和適用於 pandas 的 AWS SDK 時，可執行具有高效能結果的常見 ETL 任務。如需有關將 Modin 與適用於 pandas 的 AWS SDK 搭配使用的詳細資訊，請參閱適用於 pandas 的 AWS SDK 文件中的大規模。

其他架構 — 如需 Ray 支援之架構的詳細資訊，請參閱 Ray 文件中的 Ray 生態系統。我們不為 AWS Glue for Ray 提供其他架構支援。

透過資料型錄連線至資料

支援使用適用於 pandas 的 AWS SDK，透過資料型錄與 Ray 任務管理資料。如需詳細資訊，請參閱適用於 pandas 的 AWS SDK 網站上的 Glue Catalog。

搭配 AWS SDK 使用此服務

AWS 軟件開發套件（SDK）可用於許多流行的編程語言。每個 SDK 都提供 API、程式碼範例和說明文件，讓開發人員能夠更輕鬆地以偏好的語言建置應用程式。

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

如需此服務的特定範例，請參閱 AWS Glue 使用 AWS SDK 的 API 程式碼範例。

可用性範例

找不到所需的內容嗎？ 請使用本頁面底部的提供意見回饋連結申請程式碼範例。

AWS Glue API

本節介紹了 AWS Glue SDK 和工具使用的數據類型和原語。有三種一般的方式可以在以程式設計方 AWS Glue 式之外進行互動 AWS Management Console，每種方式都有自己的文件：

- 語言開發套件程式庫可讓您存取常見程式設計語言的 AWS 資源。如需詳細資訊，請參閱在 AWS上建立的工具。
- 可 AWS CLI 讓您從命令列存取 AWS 資源。如需詳細資訊，請參閱 AWS CLI 命令參考。
- AWS CloudFormation 可讓您定義一組要一致佈建的 AWS 資源。如需詳細資訊，請參閱 AWS CloudFormation： AWS Glue 資源類型參考。

本節介紹獨立於這些開發套件和工具的共享基本元素。工具會使用 AWS Glue Web API 參考來進行通訊 AWS。

內容

- SkewedInfo 結構
- TableVersion 結構
- TableError 結構
- TableVersionError 結構
- SortCriterion 結構
- TableIdentifier 結構
- KeySchemaElement 結構
- PartitionIndex 結構
- PartitionIndexDescriptor 結構
- BackfillError 結構
- IcebergInput 結構
- OpenTableFormatInput 結構
- ViewDefinition 結構
- ViewDefinitionInput 結構
- ViewRepresentation 結構
- ViewRepresentationInput 結構
- 作業
- CreateTable 動作（Python：創建表）
- UpdateTable 行動（Python：更新表）
- DeleteTable 行動（Python：刪除表）
- BatchDeleteTable 行動（Python：批處理刪除表）
- GetTable 行動（Python：獲取表）
- GetTables 行動（Python：獲取表）
- GetTableVersion 行動（Python：獲取表格版本）
- GetTableVersions 行動（Python：獲取表格版本）
- DeleteTableVersion 行動（Python：刪除表格版本）
- BatchDeleteTableVersion 行動（Python：批處理刪除表格版本）
- SearchTables 行動（Python：搜索表）
- GetPartitionIndexes 操作（Python：獲取分區索引）
- CreatePartitionIndex 動作（Python：創建分區索引）

- HudiTarget 結構
- CatalogTarget 結構
- CrawlerMetrics 結構
- CrawlerHistory 結構
- CrawlsFilter 結構
- SchemaChangePolicy 結構
- LastCrawlInfo 結構
- RecrawlPolicy 結構
- LineageConfiguration 結構
- LakeFormationConfiguration 結構
- 作業
- CreateCrawler 動作（Python：創建履帶）
- DeleteCrawler 行動（Python：刪除履帶）
- GetCrawler 行動（Python：獲取履帶）
- GetCrawlers 行動（Python：獲取爬蟲）
- GetCrawlerMetrics 動作（Python：獲取履帶程序度量）
- UpdateCrawler 行動（Python：更新 _ 爬蟲）
- StartCrawler 行動（Python：開始履帶）
- StopCrawler 行動（Python：停止履帶）
- BatchGetCrawlers 動作（Python：批處理抓取器）
- ListCrawlers 動作（Python：列表爬蟲）
- ListCrawls 動作（Python：列表爬行）

- 資料欄統計資料 API
  - 資料類型
  - ColumnStatisticsTaskRun 結構
  - ColumnStatisticsTaskRunningException 結構
  - ColumnStatisticsTaskNotRunningException 結構
  - ColumnStatisticsTaskStoppingException 結構
  - 操作
  - StartColumnStatisticsTaskRun 動作 (Python：start_column_statistics_task_run)

- JDBC ConnectorSource 結構
- SparkConnectorSource 結構
- CatalogSource 結構
- MySQL CatalogSource 結構
- PostgreSQL CatalogSource 結構
- OracleSQL CatalogSource 結構
- 微软结构 ServerCatalogSource
- CatalogKinesisSource 結構
- DirectKinesisSource 結構
- KinesisStreamingSourceOptions 結構
- CatalogKafkaSource 結構
- DirectKafkaSource 結構
- KafkaStreamingSourceOptions 結構
- RedshiftSource 結構
- AmazonRedshiftSource 結構
- AmazonRedshiftNodeData 結構
- AmazonRedshiftAdvancedOption 結構
- 選項結構
- S3 CatalogSource 結構
- S3 SourceAdditionalOptions 結構
- S3 CsvSource 結構
- DirectJDBCSource 結構
- S3 DirectSourceAdditionalOptions 結構
- S3 JsonSource 結構
- S3 ParquetSource 結構
- S3 DeltaSource 結構
- S3 CatalogDeltaSource 結構
- CatalogDeltaSource 結構
- S3 HudiSource 結構
- S3 CatalogHudiSource 結構

- CatalogHudiSource 結構
- DynamoDB CatalogSource 結構
- RelationalCatalogSource 結構
- JDBC ConnectorTarget 結構
- SparkConnectorTarget 結構
- BasicCatalogTarget 結構
- MySQL CatalogTarget 結構
- PostgreSQL CatalogTarget 結構
- OracleSQL CatalogTarget 結構
- 微软结构 ServerCatalogTarget
- RedshiftTarget 結構
- AmazonRedshiftTarget 結構
- UpsertRedshiftTargetOptions 結構
- S3 CatalogTarget 結構
- S3 GlueParquetTarget 結構
- CatalogSchemaChangePolicy 結構
- S3 DirectTarget 結構
- S3 HudiCatalogTarget 結構
- S3 HudiDirectTarget 結構
- S3 DeltaCatalogTarget 結構
- S3 DeltaDirectTarget 結構
- DirectSchemaChangePolicy 結構
- ApplyMapping 結構
- Mapping 結構
- SelectFields 結構
- DropFields 結構
- RenameField 結構
- Spigot 結構
- Join 結構
- JoinColumn 結構

- SplitFields 結構
- SelectFromCollection 結構
- FillMissingValues 結構
- Filter 結構
- FilterExpression 結構
- FilterValue 結構
- CustomCode 結構
- SparkSQL 結構
- SqlAlias 結構
- DropNullFields 結構
- NullCheckBoxList 結構
- NullValueField 結構
- Datatype 結構
- Merge 結構
- Union 結構
- PIIDetection 結構
- Aggregate 結構
- DropDuplicates 結構
- GovernedCatalogTarget 結構
- GovernedCatalogSource 結構
- AggregateOperation 結構
- GlueSchema 結構
- GlueStudioSchemaColumn 結構
- GlueStudioColumn 結構
- DynamicTransform 結構
- TransformConfigParameter 結構
- EvaluateDataQuality 結構
- DQ ResultsPublishingOptions 結構
- DQ StopJobOnFailureOptions 結構
- EvaluateDataQualityMultiFrame 結構

- BatchStopJobRunSuccessfulSubmission 結構
- BatchStopJobRunError 結構
- NotificationProperty 結構
- 作業
- StartJobRun 行動（Python：開始工作運行）
- BatchStopJobRun 行動（Python：批處理停工運行）
- GetJobRun 行動（Python：獲取工作運行）
- GetJobRuns 行動（Python：獲取工作運行）
- GetJobBookmark 行動（Python：獲取工作書籤）
- GetJobBookmarks 行動（Python：獲取工作書籤）
- ResetJobBookmark 行動（Python：重置工作書籤）

- 觸發
  - 資料類型
  - Trigger 結構
  - TriggerUpdate 結構
  - Predicate 結構
  - Condition 結構
  - Action 結構
  - EventBatchingCondition 結構
  - 作業
  - CreateTrigger 動作（Python：創建觸發器）
  - StartTrigger 行動（Python：啟動觸發器）
  - GetTrigger 行動（Python：獲取觸發器）
  - GetTriggers 行動（Python：獲取觸發器）
  - UpdateTrigger 行動（Python：更新觸發器）
  - StopTrigger 行動（Python：停止觸發器）
  - DeleteTrigger 動作（Python：刪除觸發器）
  - ListTriggers 行動（Python：列表觸發器）
  - BatchGetTriggers 行動（Python：批處理觸發器）

- 互動式工作階段 API

- 資料類型
- Session 結構
- SessionCommand 結構
- Statement 結構
- StatementOutput 結構
- StatementOutputData 結構
- ConnectionsList 結構
- 作業
- CreateSession 行動（Python：創建會話）
- StopSession 行動（Python：停止會話）
- DeleteSession 行動（Python：刪除會話）
- GetSession 行動（Python：獲取會話）
- ListSessions 行動（Python：列表會話）
- RunStatement 行動（Python：運行語句）
- CancelStatement 行動（Python：取消語句）
- GetStatement 行動（Python：獲取語句）
- ListStatements 行動（Python：列表語句）

- 開發端點 API
  - 資料類型
  - DevEndpoint 結構
  - DevEndpointCustomLibraries 結構
  - 作業
  - CreateDevEndpoint 行動（Python：創建開發端點）
  - UpdateDevEndpoint 行動（Python：更新_開發端點）
  - DeleteDevEndpoint 行動（Python：刪除開發端點）
  - GetDevEndpoint 行動（Python：獲取_開發端點）
  - GetDevEndpoints 行動（Python：獲取開發端點）
  - BatchGetDevEndpoints 行動（Python：批處理開發端點）
  - ListDevEndpoints 行動（Python：列表開發端點）

- 結構描述登錄檔

- 資料類型
- RegistryId 結構
- RegistryListItem 結構
- MetadataInfo 結構
- OtherMetadataValueListItem 結構
- SchemaListItem 結構
- SchemaVersionListItem 結構
- MetadataKeyValuePair 結構
- SchemaVersionErrorItem 結構
- ErrorDetails 結構
- SchemaVersionNumber 結構
- SchemaId 結構
- 作業
- CreateRegistry 動作（Python：創建註冊表）
- CreateSchema 動作 (Python: 建立結構描述)
- GetSchema 行動（Python：獲取模式）
- ListSchemaVersions 行動（Python：列表模式版本）
- GetSchemaVersion 行動（Python：獲取模式版本）
- GetSchemaVersionsDiff 行動（Python：獲取模式版本差異）
- ListRegistries 動作（Python：列表註冊表）
- ListSchemas 動作（Python：列表模式）
- RegisterSchemaVersion 行動（Python：註冊模式版本）
- UpdateSchema 行動（Python：更新模式）
- CheckSchemaVersionValidity 操作（Python：檢查模式版本有效性）
- UpdateRegistry 行動（Python：更新註冊表）
- GetSchemaByDefinition 行動（Python：獲取模式定義）
- GetRegistry 行動（Python：獲取註冊表）
- PutSchemaVersionMetadata 行動（Python：放置版本的元數據）
- QuerySchemaVersionMetadata 行動（Python：查詢模式版本的元數據）
- RemoveSchemaVersionMetadata 行動（Python：刪除模式版本的元數據）

- DeleteRegistry 行動（Python：刪除註冊表）
- DeleteSchema 操作（Python：刪除模式）
- DeleteSchemaVersions 行動（Python：刪除模式版本）
- 工作流程
  - 資料類型
  - JobNodeDetails 結構
  - CrawlerNodeDetails 結構
  - TriggerNodeDetails 結構
  - Crawl 結構
  - Node 結構
  - Edge 結構
  - Workflow 結構
  - WorkflowGraph 結構
  - WorkflowRun 結構
  - WorkflowRunStatistics 結構
  - StartingEventBatchCondition 結構
  - Blueprint 結構
  - BlueprintDetails 結構
  - LastActiveDefinition 結構
  - BlueprintRun 結構
  - 作業
  - CreateWorkflow 動作 (Python: 建立工作流程)
  - UpdateWorkflow 動作 (Python: 更新工作流程)
  - DeleteWorkflow 動作 (Python: 刪除工作流程)
  - GetWorkflow 行動（Python：獲取工作流程）
  - ListWorkflows 動作 (Python: 清單工作流程)
  - BatchGetWorkflows 操作（Python：批處理工作流程）
  - GetWorkflowRun 行動（Python：工作流程運行）
  - GetWorkflowRuns 行動（Python：獲取工作流程運行）
  - GetWorkflowRunProperties 行動（Python：獲取工作流程_運行屬性）

- PutWorkflowRunProperties 行動（Python：把工作流程 _ 運行屬性）
- CreateBlueprint 行動（Python：創建藍圖）
- UpdateBlueprint 行動（Python：更新藍圖）
- DeleteBlueprint 行動（Python：刪除藍圖）
- ListBlueprints 行動（Python：列表藍圖）
- BatchGetBlueprints 行動（Python：批處理藍圖）
- StartBlueprintRun 行動（Python：開始 _ 藍印運行）
- GetBlueprintRun 行動（Python：獲取藍印運行）
- GetBlueprintRuns 行動（Python：獲取藍印運行）
- StartWorkflowRun 行動（Python：開始工作流程運行）
- StopWorkflowRun 行動（Python：停止工作流程 _ 運行）
- ResumeWorkflowRun 行動（Python：恢復工作流程運行）

- 使用設定檔
  - 資料類型
  - ProfileConfiguration 結構
  - ConfigurationObject 結構
  - UsageProfileDefinition 結構
  - 作業
  - CreateUsageProfile 行動（Python：創建 _ 使用 _ 配置文件）
  - GetUsageProfile 行動（Python：獲取使用 _ 配置文件）
  - UpdateUsageProfile 行動（Python：更新 _ 使用 _ 配置文件）
  - DeleteUsageProfile 行動（Python：刪除使用 _ 配置文件）
  - ListUsageProfiles 行動（Python：列表使用 _ 配置文件）
- 機器學習 API
  - 資料類型
  - TransformParameters 結構
  - EvaluationMetrics 結構
  - MLTransform 結構
  - FindMatchesParameters 結構
  - FindMatchesMetrics 結構

- ConfusionMatrix 結構
- GlueTable 結構
- TaskRun 結構
- TransformFilterCriteria 結構
- TransformSortCriteria 結構
- TaskRunFilterCriteria 結構
- TaskRunSortCriteria 結構
- TaskRunProperties 結構
- FindMatchesTaskRunProperties 結構
- ImportLabelsTaskRunProperties 結構
- ExportLabelsTaskRunProperties 結構
- LabelingSetGenerationTaskRunProperties 結構
- SchemaColumn 結構
- TransformEncryption 結構
- 毫升UserDataEncryption 結構
- ColumnImportance 結構
- 作業
- CreateMLTransform 動作 (Python: create_ml_transform)
- UpdateMLTransform 動作 (Python: update_ml_transform)
- DeleteMLTransform 動作 (Python: delete_ml_transform)
- GetMLTransform 動作 (Python: get_ml_transform)
- GetMLTransforms 動作 (Python: get_ml_transforms)
- ListMLTransforms 動作 (Python: list_ml_transforms)
- 啟EvaluationTaskRun 動 ML 操作 (Python：開始計算任務運行)
- 啟LabelingSetGenerationTaskRun 動 ML 動作 (Python: 啟動 _ 標籤化 _ 設定產生 _ 任務執行)
- 獲取 ML TaskRun 操作 (Python：運行)
- 獲取 ML TaskRuns 操作 (Python：運行)
- 取消毫升TaskRun 操作 (Python：取消任務運行)
- StartExportLabelsTaskRun 操作 (Python：開始 _ 導出 _ 標籤 _ 任務 _ 運行)
- StartImportLabelsTaskRun 操作 (Python：開始導入標籤任務運行)

- ColumnError 結構
- ColumnStatisticsData 結構
- BooleanColumnStatisticsData 結構
- DateColumnStatisticsData 結構
- DecimalColumnStatisticsData 結構
- DoubleColumnStatisticsData 結構
- LongColumnStatisticsData 結構
- StringColumnStatisticsData 結構
- BinaryColumnStatisticsData 結構
- 字串模式
- 例外狀況
  - AccessDeniedException 結構
  - AlreadyExistsException 結構
  - ConcurrentModificationException 結構
  - ConcurrentRunsExceededException 結構
  - CrawlerNotRunningException 結構
  - CrawlerRunningException 結構
  - CrawlerStoppingException 結構
  - EntityNotFoundException 結構
  - FederationSourceException 結構
  - FederationSourceRetryableException 結構
  - GlueEncryptionException 結構
  - IdempotentParameterMismatchException 結構
  - IllegalWorkflowStateException 結構
  - InternalServiceException 結構
  - InvalidExecutionEngineException 結構
  - InvalidInputException 結構
  - InvalidStateException 結構
  - InvalidTaskStatusTransitionException 結構
  - JobDefinitionErrorException 結構

- JobRunInTerminalStateException 結構
- JobRunInvalidStateTransitionException 結構
- JobRunNotInTerminalStateException 結構
- LateRunnerException 結構
- NoScheduleException 結構
- OperationTimeoutException 結構
- ResourceNotReadyException 結構
- ResourceNumberLimitExceededException 結構
- SchedulerNotRunningException 結構
- SchedulerRunningException 結構
- SchedulerTransitioningException 結構
- UnrecognizedRunnerException 結構
- ValidationException 結構
- VersionMismatchException 結構

中的安全性 API AWS Glue

安全性 API 說明安全性資料類型，以及中與安全性相關的 API AWS Glue。

資料類型

- DataCatalogEncryptionSettings 結構
- EncryptionAtRest 結構
- ConnectionPasswordEncryption 結構
- EncryptionConfiguration 結構
- S3Encryption 結構
- CloudWatchEncryption 結構
- JobBookmarksEncryption 結構
- SecurityConfiguration 結構
- GluePolicy 結構

DataCatalogEncryptionSettings 結構

包含了維護資料型錄安全性的組態資訊。

欄位

- `EncryptionAtRest` – EncryptionAt休息 物件。

  指 encryption-at-rest 定「資料目錄」的規劃。

- `ConnectionPasswordEncryption` – ConnectionPassword加密 物件。

  啟用連線密碼保護時，資料型錄使用客戶提供的金鑰加密密碼，做為 `CreateConnection` 或 `UpdateConnection` 的一部分，並將密碼存放在連線屬性中的 `ENCRYPTED_PASSWORD` 欄位。 您可以啟用目錄加密，或僅使用密碼加密。

EncryptionAtRest 結構

指 encryption-at-rest 定「資料目錄」的規劃。

欄位

- `CatalogEncryptionMode` – 必要：UTF-8 字串 (有效值：`DISABLED` | `SSE-KMS="SSEKMS"` | `SSE-KMS-WITH-SERVICE-ROLE="SSEKMSWITHSERVICEROLE"`)。

  加密「資料目錄」資料的 encryption-at-rest 模式。

- `SseAwsKmsKeyId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於靜態加密的 AWS KMS 金鑰識別碼。

- `CatalogEncryptionServiceRole` – UTF-8 字串，需符合Custom string pattern #24。

  AWS Glue 假設代表呼叫者加密和解密資料目錄物件的角色。

ConnectionPasswordEncryption 結構

資料型錄所使用的資料結構，用於加密密碼，做為 `CreateConnection` 或 `UpdateConnection` 的一部分，並將密碼存放在連線屬性中的 `ENCRYPTED_PASSWORD` 欄位。 您可以啟用目錄加密，或僅使用密碼加密。

當到達包含密碼的CreationConnection請求時，資料目錄會先使用您的 AWS KMS 金鑰加密密碼。如果啟用了目錄加密，則會再次加密整個連線物件。

此加密要求您設定 AWS KMS 金鑰權限，以根據您的安全性需求來啟用或限制密碼金鑰的存取。例如，您可能希望只有管理員能擁有密碼金鑰的解密許可。

欄位

- ReturnConnectionPasswordEncrypted – 必要：布林值。

  如果 ReturnConnectionPasswordEncrypted 旗標設為「true」，則在 GetConnection 與 GetConnections 回應中，密碼仍維持加密。 此加密獨立生效，不受目錄加密影響。

- AwsKmsKeyId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用來加密連線密碼的 AWS KMS 金鑰。

  如果啟用連線密碼保護，則呼叫者CreateConnection和至少UpdateConnection需要指定 AWS KMS 金鑰的kms:Encrypt權限，才能在將密碼儲存在資料目錄中之前加密密碼。

  您可以設定解密許可，以根據您的安全要求啟用或限制密碼金鑰的存取。

EncryptionConfiguration 結構

指定加密組態。

欄位

- S3Encryption – 一個 S3Encryption 物件陣列。

  Amazon Simple Storage Service (Amazon S3) 的資料的加密設定。

- CloudWatchEncryption – CloudWatch加密 物件。

  Amazon 的加密配置 CloudWatch。

- JobBookmarksEncryption – JobBookmarks加密 物件。

  適用於任務書籤的加密組態。

S3Encryption 結構

指定 Amazon Simple Storage Service (Amazon S3) 資料應如何進行加密。

欄位

- `S3EncryptionMode` – UTF-8 字串 (有效值：`DISABLED` | `SSE-KMS="SSEKMS"` | `SSE-S3="SSES3"`)。

  Amazon S3 資料使用的加密模式。
- `KmsKeyArn` – UTF-8 字串，需符合Custom string pattern #25。

  用來加密資料的 KMS 金鑰 Amazon Resource Name (ARN)。

CloudWatchEncryption 結構

指定應如何加密 Amazon CloudWatch 資料。

欄位

- `CloudWatchEncryptionMode` – UTF-8 字串 (有效值：`DISABLED` | `SSE-KMS="SSEKMS"`)。

  用於資料的加密模 CloudWatch 式。
- `KmsKeyArn` – UTF-8 字串，需符合Custom string pattern #25。

  用來加密資料的 KMS 金鑰 Amazon Resource Name (ARN)。

JobBookmarksEncryption 結構

指定任務書籤資料的加密方式。

欄位

- `JobBookmarksEncryptionMode` – UTF-8 字串 (有效值：`DISABLED` | `CSE-KMS="CSEKMS"`)。

  任務書籤資料使用的加密模式。
- `KmsKeyArn` – UTF-8 字串，需符合Custom string pattern #25。

  用來加密資料的 KMS 金鑰 Amazon Resource Name (ARN)。

SecurityConfiguration 結構

指定安全組態。

欄位

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  安全組態的名稱。

- CreatedTimeStamp – 時間戳記。

  此安全組態建立的時間點。

- EncryptionConfiguration – EncryptionConfiguration 物件。

  與此安全組態關聯的加密組態。

GluePolicy 結構

傳回資源政策的結構。

欄位

- PolicyInJson – UTF-8 字串，長度至少為 2 個位元組。

  包含所要求的 JSON 格式政策文件。

- PolicyHash – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  包含與此政策相關聯的雜湊值。

- CreateTime – 時間戳記。

  建立政策的日期和時間。

- UpdateTime – 時間戳記。

  上次更新政策的日期和時間。

作業

- GetDataCatalogEncryptionSettings 操作（Python：獲取數據目錄加密設置）
- PutDataCatalogEncryptionSettings 操作（Python：輸入數據目錄加密設置）
- PutResourcePolicy 行動（Python：輸入資源策略）
- GetResourcePolicy 行動（Python：獲取資源策略）
- DeleteResourcePolicy 操作（Python：刪除資源策略）
- CreateSecurityConfiguration 動作 (Python: 建立安全性組態)
- DeleteSecurityConfiguration 操作（Python：刪除安全配置）
- GetSecurityConfiguration 行動（Python：獲取安全配置）
- GetSecurityConfigurations 行動（Python：獲取安全配置）
- GetResourcePolicies 動作（Python：獲取資源策略）

GetDataCatalogEncryptionSettings 操作（Python：獲取數據目錄加密設置）

為指定目錄擷取安全組態。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於擷取安全組態的 Data Catalog（Data Catalog）ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。

回應

- `DataCatalogEncryptionSettings` – DataCatalogEncryptionSettings 物件。

  要求的安全組態。

錯誤

- `InternalServiceException`

- InvalidInputException
- OperationTimeoutException

PutDataCatalogEncryptionSettings 操作（Python：輸入數據目錄加密設置）

為指定目錄設定安全組態。設定組態後，指定的加密將套用到每個目錄寫入。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於設定安全組態的 Data Catalog（Data Catalog）ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。

- DataCatalogEncryptionSettings – 必要：DataCatalogEncryptionSettings 物件。

  要設定的安全組態。

回應

- 無回應參數。

錯誤

- InternalServiceException
- InvalidInputException
- OperationTimeoutException

PutResourcePolicy 行動（Python：輸入資源策略）

設定 Data Catalog 資源政策以用於存取控制。

請求

- PolicyInJson – 必要：UTF-8 字串，長度至少為 2 個位元組。

包含要設定的 JSON 格式政策文件。

- ResourceArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  請勿使用。僅供內部使用。

- PolicyHashCondition – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  在使用 PutResourcePolicy 設定前一個政策時傳回的雜湊值。它的目的是為了防止並行修改政策。如果尚未設定前一個政策，請不要使用此參數。

- PolicyExistsCondition – UTF-8 字串 (有效值：MUST_EXIST | NOT_EXIST | NONE)。

  MUST_EXIST 值是用來更新政策。NOT_EXIST 值是用來建立新政策。如果使用 NONE 值或 null 值，則呼叫不需依賴政策的存在。

- EnableHybrid – UTF-8 字串 (有效值：TRUE | FALSE)。

  如果 'TRUE'，表示您正在使用這兩種方式授予 Data Catalog 資源的跨帳戶存取權：

  - 透過 PutResourePolicy 直接更新資源政策
  - 藉由在 AWS Management Console上使用授予許可命令。

  如果您已經使用管理主控台授與跨帳戶存取權則必須設定為 'TRUE'，否則呼叫會失敗。預設為 'FALSE'。

回應

- PolicyHash – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  剛設定的政策雜湊。這必須包含在可覆寫或更新此政策的後續呼叫中。

錯誤

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException

- ConditionCheckFailureException

GetResourcePolicy 行動（Python：獲取資源策略）

擷取指定的資源政策。

請求

- ResourceArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  要擷取其資 AWS Glue 源策略之資源的 ARN。如果未提供，則會傳回 Data Catalog 資源政策。使用 GetResourcePolicies 檢視所有現有的資源政策。如需詳細資訊，請參閱指定 AWS Glue 資源 ARN。

回應

- PolicyInJson – UTF-8 字串，長度至少為 2 個位元組。

  包含所要求的 JSON 格式政策文件。
- PolicyHash – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  包含與此政策相關聯的雜湊值。
- CreateTime – 時間戳記。

  建立政策的日期和時間。
- UpdateTime – 時間戳記。

  上次更新政策的日期和時間。

錯誤

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException

DeleteResourcePolicy 操作（Python：刪除資源策略）

刪除指定的政策。

請求

- PolicyHashCondition – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  此政策設定時傳回的雜湊值。
- ResourceArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  要刪除之資 AWS Glue 源策略之資源的 ARN。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ConditionCheckFailureException

CreateSecurityConfiguration 動作 (Python: 建立安全性組態)

建立新的安全組態。安全組態是一組安全屬性， AWS Glue會使用這組屬性。您可以使用安全組態來加密靜態資料。如需有關在中使用安全性設定的詳細資訊 AWS Glue，請參閱加密編目器、工作和開發端點所寫入的資料。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

新安全組態的名稱。

- `EncryptionConfiguration` – 必要：EncryptionConfiguration 物件。

  新安全組態的加密組態。

回應

- `Name` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  指派給新安全組態的名稱。

- `CreatedTimestamp` – 時間戳記。

  該新安全組態建立的時間點。

錯誤

- `AlreadyExistsException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`

DeleteSecurityConfiguration 操作（Python：刪除安全配置）

刪除指定的安全組態。

請求

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欲刪除的安全組態名稱。

回應

- 無回應參數。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetSecurityConfiguration 行動（Python：獲取安全配置）

擷取指定的安全組態。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欲擷取的安全組態名稱。

回應

- `SecurityConfiguration` – SecurityConfiguration 物件。

  要求的安全組態。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetSecurityConfigurations 行動（Python：獲取安全配置）

擷取所有安全組態的清單。

請求

- `MaxResults` – 數字 (整數)，不可小於 1，也不可以大於 1000。

  回傳結果的數量上限。
- `NextToken` – UTF-8 字串。

  接續符記，如果這是接續呼叫。

回應

- `SecurityConfigurations` – 一個 SecurityConfiguration 物件陣列。

  安全組態清單。
- `NextToken` – UTF-8 字串。

  接續符記，若有更多要傳回的安全組態。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetResourcePolicies 動作（Python：獲取資源策略）

在跨帳號權限授與 AWS Resource Access Manager 期間擷取在個別資源上設定的資源策略。同時擷取 Data Catalog 資源政策。

如果您在資料目錄設定中啟用中繼資料加密，且您沒有 AWS KMS 金鑰的權限，則作業無法傳回資料目錄資源原則。

請求

- NextToken – UTF-8 字串。

  接續符記，如果這是接續要求。
- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  所要回傳清單的大小上限。

回應

- GetResourcePoliciesResponseList – 一個 GluePolicy 物件陣列。

  個別資源政策和帳號層級資源政策的清單。
- NextToken – UTF-8 字串。

  接續字元，如果傳回的清單未包含最後一個可用資源政策。

錯誤

- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- GlueEncryptionException

目錄 API

Catalog API 說明 AWS Glue 中的資料類型以及與目錄作業相關的 API。

主題

- 資料庫 API
- 資料表 API
- 分區 API
- 連線 API
- 使用者定義的函數 API

- 正在將 Athena 目錄匯入 AWS Glue

資料庫 API

Database API 說明資料庫資料類型，以及用於建立、刪除、尋找、更新和列出資料庫的 API。

資料類型

- Database 結構
- DatabaseInput 結構
- PrincipalPermissions 結構
- DataLakePrincipal 結構
- DatabaseIdentifier 結構
- FederatedDatabase 結構

Database 結構

`Database` 物件代表了資料表的邏輯分組，這可能存放於 Hive 中繼存放區或 RDBMS。

欄位

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料庫的名稱。為了相容於 Hive，它在存放時會折疊為小寫。
- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  資料庫的描述。
- `LocationUri` – 統一資源識別符 (uri)，長度不可小於 1 個位元組，也不可以超過 1024 個位元組，需符合URI address multi-line string pattern。

  資料庫的位置 (例如 HDFS 路徑)。
- `Parameters` – 金鑰值對的對應陣列。

  每個金鑰都是金鑰字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

每個值都是 UTF-8 字串，長度不可超過 512000 個位元組。

這些金鑰值對會定義資料庫的參數和屬性。

- `CreateTime` – 時間戳記。

  中繼資料資料庫在目錄中建立的時間。

- `CreateTableDefaultPermissions` – 一個 PrincipalPermissions 物件陣列。

  為主體在資料表上建立一組預設許可。使用者：AWS Lake Formation。在 AWS Glue 正常操作過程中不使用。

- `TargetDatabase` – DatabaseIdentifier 物件。

  描述資源連結的目標資料庫的 `DatabaseIdentifier` 結構。

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料庫存放所在 Data Catalog 的 ID。

- `FederatedDatabase` – FederatedDatabase 物件。

  參照 AWS Glue Data Catalog 外部實體的 `FederatedDatabase` 結構。

DatabaseInput 結構

用於建立或更新資料庫的結構。

欄位

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料庫的名稱。為了相容於 Hive，它在存放時會折疊為小寫。

- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  資料庫的描述。

- `LocationUri` – 統一資源識別符 (uri)，長度不可小於 1 個位元組，也不可以超過 1024 個位元組，需符合URI address multi-line string pattern。

資料庫的位置 (例如 HDFS 路徑)。

- `Parameters` – 金鑰值對的對應陣列。

  每個金鑰都是金鑰字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串，長度不可超過 512000 個位元組。

  這些金鑰值對會定義資料庫的參數和屬性。

  這些金鑰值對會定義資料庫的參數和屬性。

- `CreateTableDefaultPermissions` – 一個 PrincipalPermissions 物件陣列。

  為主體在資料表上建立一組預設許可。使用者：AWS Lake Formation。在 AWS Glue 正常操作過程中不使用。

- `TargetDatabase` – DatabaseIdentifier 物件。

  描述資源連結的目標資料庫的 `DatabaseIdentifier` 結構。

- `FederatedDatabase` – FederatedDatabase 物件。

  參照 AWS Glue Data Catalog 外部實體的 `FederatedDatabase` 結構。

PrincipalPermissions 結構

授予委託人的許可。

欄位

- `Principal` – DataLakePrincipal 物件。

  獲授予許可的主體。

- `Permissions` – UTF-8 字串陣列。

  授予主體的許可。

DataLakePrincipal 結構

AWS Lake Formation 委託人。

欄位

- `DataLakePrincipalIdentifier` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組。

  AWS Lake Formation 委託人的識別符。

DatabaseIdentifier 結構

描述資源連結的目標資料庫的結構。

欄位

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料庫存放所在 Data Catalog 的 ID。
- `DatabaseName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  型錄資料庫的名稱。
- `Region` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  目標資料庫的區域。

FederatedDatabase 結構

指向 AWS Glue Data Catalog 外部實體的資料庫。

欄位

- `Identifier` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組，且需符合 Single-line string pattern。

  聯合資料庫的唯一識別碼。
- `ConnectionName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  連線到外部中繼存放區的名稱。

操作

- CreateDatabase 動作 (Python: create_database)
- UpdateDatabase 動作 (Python: update_database)
- DeleteDatabase 動作 (Python: delete_database)
- GetDatabase 動作 (Python: get_database)
- GetDatabases 動作 (Python: get_databases)

CreateDatabase 動作 (Python: create_database)

在 Data Catalog 建立新的資料庫。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於建立資料庫的 Data Catalog 之 ID。如果未提供，預設會使用 AWS 帳戶 ID。
- `DatabaseInput` – 必要：DatabaseInput 物件。

  資料庫的中繼資料。
- `Tags` – 金鑰值對的映射陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  指派給資料庫的標籤。

回應

- 無回應參數。

錯誤

- `InvalidInputException`
- `AlreadyExistsException`

- ResourceNumberLimitExceededException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ConcurrentModificationException
- FederatedResourceAlreadyExistsException

UpdateDatabase 動作 (Python: update_database)

更新 Data Catalog 中現有的資料庫定義。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  存放中繼資料資料庫的 Data Catalog 之 ID。如果未提供，預設會使用 AWS 帳戶 ID。
- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  目錄中要更新的資料庫之名稱。為了相容於 Hive，名稱必須轉換為小寫。
- DatabaseInput – 必要：DatabaseInput 物件。

  DatabaseInput 物件，可為目錄裡的中繼資料資料庫指定新定義。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

- ConcurrentModificationException

DeleteDatabase 動作 (Python: delete_database)

從 Data Catalog 移除指定的資料庫。

> **Note**
>
> 完成此操作後，您就無法再存取已刪除資料庫中的資料表 (和可能屬於資料表的所有資料表版本和分割區) 和使用者定義的函數。AWS Glue 服務會自行決定以非同步方式即時刪除這些「孤立」資源。
> 若要確保能夠立即刪除所有相關資源，請在呼叫 DeleteDatabase 之前，先使用 DeleteTableVersion 或 BatchDeleteTableVersion、DeletePartition 或 BatchDeletePartition、DeleteUserDefinedFunction，以及 DeleteTable 或 BatchDeleteTable，來刪除任何屬於資料庫的資源。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料庫存放所在 Data Catalog 的 ID。如果未提供，預設會使用 AWS 帳戶 ID。
- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要刪除的資料庫之名稱。為了相容於 Hive，此項目必須完全使用小寫。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException

- OperationTimeoutException
- ConcurrentModificationException

GetDatabase 動作 (Python: get_database)

擷取指定資料庫的定義。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料庫存放所在 Data Catalog 的 ID。如果未提供，預設會使用 AWS 帳戶 ID。
- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要擷取的資料庫之名稱。為了相容於 Hive，名稱應完全小寫。

回應

- Database – 資料庫 物件。

  Data Catalog 中所指定資料庫的定義。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- FederationSourceException

GetDatabases 動作 (Python: get_databases)

擷取特定 Data Catalog 中所有已定義的資料庫。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於擷取 Databases 的 Data Catalog 之 ID。如果未提供，預設會使用 AWS 帳戶 ID。

- NextToken – UTF-8 字串。

  接續符記，如果這是接續呼叫。

- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 100。

  一次回應傳回資料庫的最大數量。

- ResourceShareType – UTF-8 字串 (有效值：FOREIGN | ALL | FEDERATED)。

  可讓您指定要列出與帳戶共用的資料庫。允許的值為 FEDERATED、FOREIGN 或 ALL。

  - 如果設定為 FEDERATED，會列出與您的帳戶共用的聯合資料庫 (參照外部實體)。
  - 如果設定為 FOREIGN，會列出與您的帳戶共用的資料庫。
  - 如果設定為 ALL，將列出與您的帳戶共享的資料庫，以及您本機帳戶中的資料庫。

回應

- DatabaseList – 必要：一個 資料庫 物件。

  指定目錄中的 Database 物件清單。

- NextToken – UTF-8 字串。

  為一種接續符記，用於將傳回的符記清單分頁，而如果清單目前的區段不是最後區段就會傳回。

錯誤

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

資料表 API

Table API 說明與資料表相關的資料類型和操作。

資料類型

- Table 結構
- TableInput 結構
- FederatedTable 結構
- 欄結構
- StorageDescriptor 結構
- SchemaReference 結構
- SerDeInfo 結構
- Order 結構
- SkewedInfo 結構
- TableVersion 結構
- TableError 結構
- TableVersionError 結構
- SortCriterion 結構
- TableIdentifier 結構
- KeySchemaElement 結構
- PartitionIndex 結構
- PartitionIndexDescriptor 結構
- BackfillError 結構
- IcebergInput 結構
- OpenTableFormatInput 結構
- ViewDefinition 結構
- ViewDefinitionInput 結構
- ViewRepresentation 結構
- ViewRepresentationInput 結構

Table 結構

表示整理為欄和列的相關資料的集合。

欄位

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表名稱。為了相容於 Hive，這必須完全小寫。

- DatabaseName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表中繼資料所在的資料庫名稱。為了相容於 Hive，此項目必須完全使用小寫。

- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  資料表的說明。

- Owner – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表的擁有者。

- CreateTime – 時間戳記。

  在 Data Catalog 中建立資料表定義的時間。

- UpdateTime – 時間戳記。

  資料表上次更新的時間。

- LastAccessTime – 時間戳記。

  資料表上次存取的時間。這通常取自 HDFS，而且可能不可靠。

- LastAnalyzedTime – 時間戳記。

  此資料表上次運算欄位統計的時間。

- Retention – 數字 (整數)，不可大於 None (無)。

  此資料表的保留時間。

- StorageDescriptor – StorageDescriptor 物件。

儲存描述項包含有關此資料表實體儲存的資訊。

- PartitionKeys – 一個 資料行 物件陣列。

  資料表進行分區的欄位清單。僅支援基本類型做為分割區索引鍵。

  在建立 Amazon Athena 使用的資料表，且您未指定任何 partitionKeys 時，您必須在空白清單設定 partitionKeys 值。例如：

  "PartitionKeys": []

- ViewOriginalText – UTF-8 字串，長度不可超過 409600 個位元組。

  包括以取得 Apache Hive 相容性。在正常 AWS Glue 操作過程中未使用。如果表是一個VIRTUAL_VIEW，某些 Athena 配置以 base64 編碼。

- ViewExpandedText – UTF-8 字串，長度不可超過 409600 個位元組。

  包括以取得 Apache Hive 相容性。在正常 AWS Glue 操作過程中未使用。

- TableType – UTF-8 字串，長度不可超過 255 個位元組。

  此表格的類型。 AWS Glue 將建立具有該EXTERNAL_TABLE類型的表格。其他服務 (例如 Athena) 可能會建立具有其他表格類型的資料表。

  AWS Glue 相關表格類型：

  EXTERNAL_TABLE

  Hive 相容屬性 – 表示非 Hive 受管的資料表。

  GOVERNED

  由使用 AWS Lake Formation。資 AWS Glue 料目錄瞭解GOVERNED。

- Parameters – 金鑰值對的對應陣列。

  每個金鑰都是金鑰字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串，長度不可超過 512000 個位元組。

  這些金鑰值對會定義與此資料表相關聯的屬性。

- CreatedBy – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

建立此資料表的人員或實體。

- `IsRegisteredWithLakeFormation` – 布林值。

  指出資料表是否已註冊於 AWS Lake Formation。

- `TargetTable` – TableIdentifier 物件。

  描述資源連結的目標資料表的 `TableIdentifier` 結構。

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表存放所在 Data Catalog 的 ID。

- `VersionId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表版本的 ID。

- `FederatedTable` – FederatedTable 物件。

  參照 AWS Glue Data Catalog外部實體的 `FederatedTable` 結構。

- `ViewDefinition` – ViewDefinition 物件。

  一種結構，其中包含定義檢視的所有資訊，包括檢視表的方言或方言以及查詢。

- `IsMultiDialectView` – 布林值。

  指定檢視是否支援一或多個不同查詢引擎的 SQL 方言，因此這些引擎可以讀取。

TableInput 結構

用於定義資料表的結構。

欄位

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表名稱。為了相容於 Hive，它在存放時會折疊為小寫。

- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

資料表的說明。

- Owner – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表的擁有者。包括以取得 Apache Hive 相容性。在正常 AWS Glue 操作過程中未使用。

- LastAccessTime – 時間戳記。

  資料表上次存取的時間。

- LastAnalyzedTime – 時間戳記。

  此資料表上次運算欄位統計的時間。

- Retention – 數字 (整數)，不可大於 None (無)。

  此資料表的保留時間。

- StorageDescriptor – StorageDescriptor 物件。

  儲存描述項包含有關此資料表實體儲存的資訊。

- PartitionKeys – 一個 資料行 物件陣列。

  資料表進行分區的欄位清單。僅支援基本類型做為分割區索引鍵。

  在建立 Amazon Athena 使用的資料表，且您未指定任何 partitionKeys 時，您必須在空白清單設定 partitionKeys 值。例如：

  ```
  "PartitionKeys": []
  ```

- ViewOriginalText – UTF-8 字串，長度不可超過 409600 個位元組。

  包括以取得 Apache Hive 相容性。在正常 AWS Glue 操作過程中未使用。如果表是一個VIRTUAL_VIEW，某些 Athena 配置以 base64 編碼。

- ViewExpandedText – UTF-8 字串，長度不可超過 409600 個位元組。

  包括以取得 Apache Hive 相容性。在正常 AWS Glue 操作過程中未使用。

- TableType – UTF-8 字串，長度不可超過 255 個位元組。

  此表格的類型。 AWS Glue 將建立具有該EXTERNAL_TABLE類型的表格。其他服務 (例如 Athena) 可能會建立具有其他表格類型的資料表。

  AWS Glue 相關表格類型：

EXTERNAL_TABLE

Hive 相容屬性 – 表示非 Hive 受管的資料表。

GOVERNED

由使用 AWS Lake Formation。資 AWS Glue 料目錄瞭解GOVERNED。

- `Parameters` – 金鑰值對的對應陣列。

  每個金鑰都是金鑰字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串，長度不可超過 512000 個位元組。

  這些金鑰值對會定義與此資料表相關聯的屬性。

- `TargetTable` – TableIdentifier 物件。

  描述資源連結的目標資料表的 `TableIdentifier` 結構。

- `ViewDefinition` – ViewDefinitionInput 物件。

  一種結構，其中包含定義檢視的所有資訊，包括檢視表的方言或方言以及查詢。

FederatedTable 結構

指向 AWS Glue Data Catalog外部實體的資料表。

欄位

- `Identifier` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組，且需符合 Single-line string pattern。

  聯合資料表的唯一識別碼。

- `DatabaseIdentifier` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組，且需符合 Single-line string pattern。

  聯合資料庫的唯一識別碼。

- `ConnectionName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  連線到外部中繼存放區的名稱。

欄結構

Table 中的欄位。

欄位

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  Column 的名稱。
- Type – UTF-8 字串，長度不可超過 131072 個位元組，需符合Single-line string pattern。

  Column 的資料類型。
- Comment – 註解字串，長度不可超過 255 個位元組，需符合Single-line string pattern。

  自由格式的文字註解。
- Parameters – 金鑰值對的對應陣列。

  每個金鑰都是金鑰字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串，長度不可超過 512000 個位元組。

  這些金鑰值對會定義與此資料行相關聯的屬性。

StorageDescriptor 結構

描述資料表資料的實體儲存。

欄位

- Columns – 一個 資料行 物件陣列。

  資料表中的 Columns 的清單。
- Location – 位置字串，長度不可超過 2056 個位元組，需符合URI address multi-line string pattern。

  資料表的實體位置。依預設，它採用倉儲位置的形式，後面接著是資料庫在倉儲中的位置，最後是資料表名稱。
- AdditionalLocations – UTF-8 字串陣列。

指向 Delta 資料表所在路徑的位置清單。

- `InputFormat` – 格式字串，長度不可超過 128 個位元組，需符合Single-line string pattern。

  輸入格式：`SequenceFileInputFormat` (二進位)，或者 `TextInputFormat`，或自訂格式。

- `OutputFormat` – 格式字串，長度不可超過 128 個位元組，需符合Single-line string pattern。

  輸出格式：`SequenceFileOutputFormat` (二進位)，或者 `IgnoreKeyTextOutputFormat`，或自訂格式。

- `Compressed` – 布林值。

  如果資料表中的資料都經過壓縮則為 `True`，否則為 `False`

- `NumberOfBuckets` – 數字 (整數)。

  如果資料表包含任何維度欄位，將必須指定。

- `SerdeInfo` – SerDe信息 物件。

  序列化/反序列化 ( ) 信息。SerDe

- `BucketColumns` – UTF-8 字串陣列。

  資料表中的縮減器分組欄位、叢集欄位及值區欄位的清單。

- `SortColumns` – 一個 順序 物件陣列。

  指定資料表中各個儲存貯體排序順序的清單。

- `Parameters` – 金鑰值對的對應陣列。

  每個金鑰都是金鑰字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串，長度不可超過 512000 個位元組。

  使用者提供的屬性，格式為金鑰/值。

- `SkewedInfo` – SkewedInfo 物件。

  關於欄位中頻繁出現值的資訊 (偏斜值)。

- `StoredAsSubDirectories` – 布林值。

  如果資料表資料存放於子目錄則為 `True`，否則為 `False`

- SchemaReference – SchemaReference 物件。

  參考儲存在結構描述登錄中之結 AWS Glue 構描述的物件。

  建立資料表時，您可以傳遞結構描述欄的空白清單，並改用結構描述參考。

SchemaReference 結構

參考儲存在結構描述登錄中之結 AWS Glue 構描述的物件。

欄位

- SchemaId – SchemaId 物件。

  包含結構描述身分欄位的結構。必須提供此結構或 SchemaVersionId。

- SchemaVersionId – UTF-8 字串，長度不可小於 36 個位元組，也不可以超過 36 個位元組，且需符合 Custom string pattern #17。

  指派給結構描述版本的唯一 ID。必須提供此結構或 SchemaId。

- SchemaVersionNumber – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  結構描述的版本編號。

SerDeInfo 結構

有關作為提取器和加載器的序列化/反序列化程序（SerDe）的信息。

欄位

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  的名稱 SerDe。

- SerializationLibrary – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  通常是實作 SerDe. 例如，org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe。

- Parameters – 金鑰值對的對應陣列。

每個金鑰都是金鑰字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

每個值都是 UTF-8 字串，長度不可超過 512000 個位元組。

這些鍵值配對定義的初始化參數。 SerDe

Order 結構

指定已排序欄位的排序順序。

欄位

- `Column` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欄位的名稱。
- `SortOrder` – 必要：數字 (整數)，不可大於 1。

  指示欄位以遞增順序 (== 1) 或以遞減順序 (==0) 排序。

SkewedInfo 結構

指定資料表中的偏斜值。偏斜值是指頻率發生非常高的值。

欄位

- `SkewedColumnNames` – UTF-8 字串陣列。

  包含偏斜值的欄位名稱清單。
- `SkewedColumnValues` – UTF-8 字串陣列。

  頻繁出現而被視為偏斜的值的清單。
- `SkewedColumnValueLocationMaps` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  偏斜值與包含這些值的欄位的映射。

TableVersion 結構

指定資料表的版本。

欄位

- `Table` – 資料表 物件。

  有問題的資料表。
- `VersionId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於識別此資料表版本的 ID 值。`VersionId` 是一個整數的表示字串。每個版本會增加 1。

TableError 結構

資料表操作的錯誤記錄。

欄位

- `TableName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表的名稱。為了相容於 Hive，這必須完全小寫。
- `ErrorDetail` – ErrorDetail 物件。

  關於錯誤的詳細資訊。

TableVersionError 結構

資料表版本操作的錯誤記錄。

欄位

- `TableName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  有問題的資料表的名稱。
- `VersionId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

有問題的版本的 ID 值。VersionID 是一個整數的表示字串。每個版本會增加 1。

- ErrorDetail – ErrorDetail 物件。

  關於錯誤的詳細資訊。

SortCriterion 結構

指定欄位排序的依據和排序順序。

欄位

- FieldName – 值字串，長度不可超過 1024 個位元組。

  要排序的欄位名稱。
- Sort – UTF-8 字串 (有效值：ASC="ASCENDING" |DESC="DESCENDING")。

  遞增或遞減排序。

TableIdentifier 結構

描述資源連結的目標資料表的結構。

欄位

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表存放所在 Data Catalog 的 ID。
- DatabaseName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  包含目標資料表的型錄資料庫名稱。
- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  目標資料表的名稱。
- Region – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

目標資料表的區域。

KeySchemaElement 結構

由名稱和類型組成的分割區索引鍵對。

欄位

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分割區索引鍵的名稱。
- Type – 必要：UTF-8 字串，長度不可超過 131072 個位元組，且需符合Single-line string pattern。

  分割區索引鍵的類型。

PartitionIndex 結構

分割區索引的結構。

欄位

- Keys – 必要：UTF-8 字串的陣列，至少要有 1 個字串。

  分割區索引的索引鍵。
- IndexName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分割區索引的名稱。

PartitionIndexDescriptor 結構

資料表中分割區索引的描述元。

欄位

- IndexName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

分割區索引的名稱。

- Keys–必要：KeySchemaElement 物件陣列，至少有 1 個結構。

  分割區索引的一或多個索引鍵的清單，作為 KeySchemaElement 結構。

- IndexStatus – 必要：UTF-8 字串 (有效值：CREATING | ACTIVE | DELETING | FAILED)。

  分割區索引的狀態。

  可能的狀態如下：

  - CREATING：正在建立索引。當索引處於 CREATING (正在建立) 狀態時，索引或其資料表無法刪除。
  - ACTIVE：索引建立成功。
  - FAILED：索引建立失敗。
  - DELETING：索引會從索引清單中刪除。

- BackfillErrors – 一個 BackfillError 物件陣列。

  註冊現有資料表的分割區索引時可能發生的錯誤清單。

BackfillError 結構

註冊現有資料表的分割區索引時可能發生的錯誤清單。

這些錯誤提供索引註冊失敗原因的詳細資訊，並在回應中提供有限數目的分割區，讓您可以修正錯誤的分割區，然後再次嘗試註冊索引。可能發生的最常見的錯誤集分類如下：

- EncryptedPartitionError：分區已加密。
- InvalidPartitionTypeDataError: 分區值與該分區列的數據類型不匹配。
- MissingPartitionValueError：分區已加密。
- UnsupportedPartitionCharacterError: 不支援分割區值內的字元。例如：U+0000、U+0001、U+0002。
- InternalError：任何不屬於其他錯誤代碼的錯誤。

欄位

- Code – UTF-8 字串 (有效值：ENCRYPTED_PARTITION_ERROR | INTERNAL_ERROR | INVALID_PARTITION_TYPE_DATA_ERROR | MISSING_PARTITION_VALUE_ERROR | UNSUPPORTED_PARTITION_CHARACTER_ERROR)。

  註冊現有資料表的分割區索引時發生的錯誤碼。

- Partitions – 一個 PartitionValue清單 物件陣列。

  回應中有限數量的分割區索引鍵清單。

IcebergInput 結構

定義要在目錄中建立的 Apache Iceberg 中繼資料資料表的結構。

欄位

- MetadataOperation – 必要：UTF-8 字串 (有效值：CREATE)。

  所需的中繼資料操作。只能設定為 CREATE。

- Version – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  Iceberg 資料表的資料表版本。預設值為 2。

OpenTableFormatInput 結構

表示開放格式資料表的結構。

欄位

- IcebergInput – IcebergInput 物件。

  指定定義 Apache Iceberg 中繼資料資料表的 IcebergInput 結構。

ViewDefinition 結構

包含表現詳細資訊的結構。

欄位

- `IsProtected` – 布林值。

  您可以將此旗標設定為 true，以指示引擎在查詢計劃期間不要將使用者提供的作業推入檢視表的邏輯計劃。但是，設定此旗標並不能保證引擎符合規定。請參閱引擎的文件以瞭解提供的保證 (如果有的話)。
- `Definer` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組，且需符合 Single-line string pattern。

  SQL 中視圖的定義者。
- `SubObjects` – UTF-8 字串的陣列，不可超過 10 個字串。

  表 Amazon 資源名稱（ARN）的列表。
- `Representations` – ViewRepresentation 物件陣列，不小於 1 個結構，也不大於 1000 個結構。

  表示清單。

ViewDefinitionInput 結構

包含建立或更新 AWS Glue 檢視表之詳細資訊的結構。

欄位

- `IsProtected` – 布林值。

  您可以將此旗標設定為 true，以指示引擎在查詢計劃期間不要將使用者提供的作業推入檢視表的邏輯計劃。但是，設定此旗標並不能保證引擎符合規定。請參閱引擎的文件以瞭解提供的保證 (如果有的話)。
- `Definer` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組，且需符合 Single-line string pattern。

  SQL 中視圖的定義者。
- `Representations`— ViewRepresentationInput 物件陣列，不小於 1 或多於 10 個結構。

  包含檢視方言的結構清單，以及定義檢視的查詢。
- `SubObjects` – UTF-8 字串的陣列，不可超過 10 個字串。

  構成檢視的基底資料表 ARN 清單。

ViewRepresentation 結構

一種結構，其中包含檢視的方言，以及定義檢視的查詢。

欄位

- Dialect – UTF-8 字串 (有效值：REDSHIFT | ATHENA | SPARK)。

  查詢引擎的方言。

- DialectVersion – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組。

  查詢引擎的方言版本。例如，3.0.0。

- ViewOriginalText – UTF-8 字串，長度不可超過 409600 個位元組。

  客戶在期間提供的SELECT查詢CREATE VIEW DDL。在檢視上查詢期間不會使用此 SQL (而ViewExpandedText是使用)。 ViewOriginalText用於像SHOW CREATE VIEW用戶希望看到創建視圖的原始 DDL 命令的情況。

- ViewExpandedText – UTF-8 字串，長度不可超過 409600 個位元組。

  檢視表的展開 SQL。在處理視圖上的查詢時，引擎使用此 SQL。引擎可能會在視圖創建過程中執行操作以轉ViewOriginalText換為ViewExpandedText。例如：

  - 完全限定的標識符：SELECT * from table1 -> SELECT * from db1.table1

- ValidationConnection – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於驗證視圖的特定表現法的連接對名稱。

- IsStale – 布林值。

  標記為過時的方言已不再有效，必須先更新，才能在各自的查詢引擎中查詢。

ViewRepresentationInput 結構

一種結構，其中包含要更新或建立 Lake Formation 視圖的表現法詳細資訊。

欄位

- Dialect – UTF-8 字串 (有效值：REDSHIFT | ATHENA | SPARK)。

  指定特定表示之引擎類型的參數。

- `DialectVersion` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組。

  指定特定表示之引擎版本的參數。

- `ViewOriginalText` – UTF-8 字串，長度不可超過 409600 個位元組。

  字串；代表描述檢視表的原始 SQL 查詢。

- `ValidationConnection` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於驗證視圖的特定表現法的連接對名稱。

- `ViewExpandedText` – UTF-8 字串，長度不可超過 409600 個位元組。

  字串，代表描述具有展開資源 ARN 之檢視的 SQL 查詢

作業

- CreateTable 動作（Python：創建表）
- UpdateTable 行動（Python：更新表）
- DeleteTable 行動（Python：刪除表）
- BatchDeleteTable 行動（Python：批處理刪除表）
- GetTable 行動（Python：獲取表）
- GetTables 行動（Python：獲取表）
- GetTableVersion 行動（Python：獲取表格版本）
- GetTableVersions 行動（Python：獲取表格版本）
- DeleteTableVersion 行動（Python：刪除表格版本）
- BatchDeleteTableVersion 行動（Python：批處理刪除表格版本）
- SearchTables 行動（Python：搜索表）
- GetPartitionIndexes 操作（Python：獲取分區索引）
- CreatePartitionIndex 動作（Python：創建分區索引）
- DeletePartitionIndex 動作（Python：刪除分區索引）
- GetColumnStatisticsForTable 動作（Python：獲取列統計表）
- UpdateColumnStatisticsForTable 行動（Python：更新列統計表）
- DeleteColumnStatisticsForTable 操作（Python：刪除列統計表）

CreateTable 動作（Python：創建表）

在 Data Catalog 建立新的資料表定義。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於建立 `Table` 的 Data Catalog 之 ID。如果沒有提供，則依預設會使用 AWS 帳號 ID。
- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  用於建立新資料表的目錄資料庫。為了相容於 Hive，此名稱必須完全小寫。
- `TableInput` – 必要：TableInput 物件。

  用於定義中繼資料資料表以建立目錄的 `TableInput` 物件。
- `PartitionIndexes` – PartitionIndex 物件陣列，不可超過 3 個結構。

  要在資料表中建立的分割區索引的清單，`PartitionIndex` 結構。
- `TransactionId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #16。

  交易的 ID。
- `OpenTableFormatInput` – OpenTableFormatInput 物件。

  建立開放格式資料表時指定 `OpenTableFormatInput` 結構。

回應

- 無回應參數。

錯誤

- `AlreadyExistsException`
- `InvalidInputException`
- `EntityNotFoundException`
- `ResourceNumberLimitExceededException`

- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ConcurrentModificationException
- ResourceNotReadyException

UpdateTable 行動（Python：更新表）

在 Data Catalog 更新中繼資料資料表。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在的 Data Catalog 的 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。
- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表所在的目錄資料庫的名稱。為了相容於 Hive，此名稱必須完全小寫。
- TableInput – 必要：TableInput 物件。

  用於定義目錄中的中繼資料資料表的已更新 TableInput 物件。
- SkipArchive – 布林值。

  依預設，UpdateTable 在更新資料表之前，一律會建立資料表的封存版本。但是，如果 skipArchive 設為 true，UpdateTable 將不會建立封存版本。
- TransactionId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #16。

  藉助其更新資料表內容的交易 ID。
- VersionId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  藉助其更新資料表內容的版本 ID。
- ViewUpdateAction – UTF-8 字串 (有效值：ADD | REPLACE | ADD_OR_REPLACE | DROP)。

更新視圖時要執行的操作。

- Force – 布林值。

  可設定為 true 以忽略相符儲存區描述元和子物件相符需求的旗標。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException
- ResourceNumberLimitExceededException
- GlueEncryptionException
- ResourceNotReadyException

DeleteTable 行動（Python：刪除表）

從 Data Catalog 移除資料表定義。

> **Note**
>
> 完成此操作之後，您就不能再存取屬於已刪除資料表的資料表版本和分割區。 AWS Glue 會根據服務的判斷，以非同步方式即時刪除這些「孤立」資源。
> 若要確保能夠立即刪除所有相關資源，請在呼叫 DeleteTable 之前，先使用 DeleteTableVersion 或 BatchDeleteTableVersion，以及 DeletePartition 或 BatchDeletePartition，來刪除任何屬於資料表的資源。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在的 Data Catalog 的 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表所在的目錄資料庫的名稱。為了相容於 Hive，此名稱必須完全小寫。

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要刪除的資料表的名稱。為了相容於 Hive，此名稱必須完全小寫。

- `TransactionId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #16。

  藉助其刪除資料表內容的交易 ID。

回應

- 無回應參數。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentModificationException`
- `ResourceNotReadyException`

BatchDeleteTable 行動（Python：批處理刪除表）

一次刪除多個資料表。

> **Note**
>
> 完成此操作之後，您就不能再存取屬於已刪除資料表的資料表版本和分割區。 AWS Glue 會根據服務的判斷，以非同步方式即時刪除這些「孤立」資源。
> 若要確保能夠立即刪除所有相關資源，請在呼叫 `BatchDeleteTable` 之前，先使用 `DeleteTableVersion` 或 `BatchDeleteTableVersion`，以及 `DeletePartition` 或 `BatchDeletePartition`，來刪除任何屬於資料表的資源。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在的 Data Catalog 的 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。
- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要刪除的資料表所在目錄資料庫的名稱。為了相容於 Hive，此名稱必須完全小寫。
- `TablesToDelete` – 必要：UTF-8 字串的陣列，不可超過 100 個字串。

  要刪除的資料表的清單。
- `TransactionId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #16。

  藉助其刪除資料表內容的交易 ID。

回應

- `Errors` – 一個 TableError 物件陣列。

  嘗試刪除指定的資料表時發生的錯誤清單。

錯誤

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`

- OperationTimeoutException
- GlueEncryptionException
- ResourceNotReadyException

GetTable 行動（Python：獲取表）

擷取 Data Catalog 中指定資料表的 Table 定義。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在的 Data Catalog 的 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。
- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表所在目錄的資料庫的名稱。為了相容於 Hive，此名稱必須完全小寫。
- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要擷取其定義的資料表的名稱。為了相容於 Hive，此名稱必須完全小寫。
- TransactionId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #16。

  藉助其讀取資料表內容的交易 ID。
- QueryAsOfTime – 時間戳記。

  讀取資料表內容的時間。如果沒有設定，將使用最近的交易遞交時間。無法連同 TransactionId 一起指定。

回應

- Table – 資料表 物件。

  用於定義指定資料表的 Table 物件。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ResourceNotReadyException
- FederationSourceException
- FederationSourceRetryableException

GetTables 行動（Python：獲取表）

擷取指定的 Database 中的部分或所有資料表的定義。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在的 Data Catalog 的 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。
- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  目錄的資料表要列出的資料庫。為了相容於 Hive，此名稱必須完全小寫。
- Expression – UTF-8 字串，長度不可超過 2048 個位元組，且需符合 Single-line string pattern。

  規則表達式模式。如果存在，只會傳回名稱符合模式的資料表。
- NextToken – UTF-8 字串。

  接續符記，如果這是接續呼叫，將會包含在內。
- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 100。

  在單一回應中可傳回的最大資料表數量。
- TransactionId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #16。

藉助其讀取資料表內容的交易 ID。

- QueryAsOfTime – 時間戳記。

  讀取資料表內容的時間。如果沒有設定，將使用最近的交易遞交時間。無法連同 TransactionId 一起指定。

回應

- TableList – 一個 資料表 物件陣列。

  要求的 Table 物件的清單。

- NextToken – UTF-8 字串。

  接續字元，如果目前清單區段不是最後一個，將會出現此接續字元。

錯誤

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- GlueEncryptionException
- FederationSourceException
- FederationSourceRetryableException

GetTableVersion 行動（Python：獲取表格版本）

擷取指定的資料表版本。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在的 Data Catalog 的 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。

- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表所在的目錄中的資料庫。為了相容於 Hive，此名稱必須完全小寫。

- TableName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表的名稱。為了相容於 Hive，此名稱必須完全小寫。

- VersionId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要擷取的資料表版本的 ID 值。VersionID 是一個整數的表示字串。每個版本會增加 1。

回應

- TableVersion – TableVersion 物件。

  要求的資料表版本。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

GetTableVersions 行動（Python：獲取表格版本）

擷取用於識別指定的資料表可用版本的字串清單。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在的 Data Catalog 的 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。

- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表所在的目錄中的資料庫。為了相容於 Hive，此名稱必須完全小寫。

- TableName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表的名稱。為了相容於 Hive，此名稱必須完全小寫。

- NextToken – UTF-8 字串。

  接續符記，如果這不是第一個呼叫。

- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 100。

  一次回應傳回的最大資料表版本數量。

回應

- TableVersions – 一個 TableVersion 物件陣列。

  用於識別指定的資料表可用版本的字串清單。

- NextToken – UTF-8 字串。

  接續字元，如果可用版本的清單不包含最後一個可用版本。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

DeleteTableVersion 行動（Python：刪除表格版本）

刪除指定的資料表版本。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在的 Data Catalog 的 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。
- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表所在的目錄中的資料庫。為了相容於 Hive，此名稱必須完全小寫。
- TableName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表的名稱。為了相容於 Hive，此名稱必須完全小寫。
- VersionId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要刪除的資料表版本的 ID。VersionID 是一個整數的表示字串。每個版本會增加 1。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchDeleteTableVersion 行動（Python：批處理刪除表格版本）

刪除指定的資料表版本批次。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

資料表所在的 Data Catalog 的 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表所在的目錄中的資料庫。為了相容於 Hive，此名稱必須完全小寫。

- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表的名稱。為了相容於 Hive，此名稱必須完全小寫。

- `VersionIds` – 必要：UTF-8 字串的陣列，不可超過 100 個字串。

  要刪除的版本的 ID 清單。`VersionId` 是一個整數的表示字串。每個版本會增加 1。

回應

- `Errors` – 一個 TableVersionError 物件陣列。

  嘗試刪除指定的資料表版本時發生的錯誤清單。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

SearchTables 行動（Python：搜索表）

根據資料表中繼資料和父資料庫中的屬性搜尋一組資料表。您可以搜尋文字或篩選條件。

您只能根據在 Lake Formation 中定義的安全政策，取得您有權存取的資料表。您至少需要資料表的唯讀存取權，才能傳回該資料表。如果您無法存取資料表中的所有資料欄，在將資料表清單傳回給您時，將不會搜尋這些資料欄。如果您有權存取資料欄，但無法存取資料欄中的資料，這些資料欄以及其相關中繼資料將包含在搜尋中。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  唯一識別符，由 *account_id* 組成。

- NextToken – UTF-8 字串。

  接續符記，如果這是接續呼叫，將會包含在內。

- Filters – 一個 PropertyPredicate 物件陣列。

  索引鍵/值組清單，以及用來篩選搜尋結果的比較器。傳回符合述詞的所有實體。

  PropertyPredicate 結構的 Comparator 成員僅用於時間欄位，並且在其他欄位類型可以省略。此外，在比較字串值時，例如 Key=Name，會使用模糊比對演算法。Key 欄位 (例如，Name 欄位的值) 會將某些標點符號 (例如 -、:、# 等) 分割成標記。然後，每個標記都會與 PropertyPredicate 的 Value 成員進行完全相符比較。例如，如果 Key=Name 和 Value=link，會傳回命名為 customer-link 和 xx-link-yy 的資料表，但不會傳回 xxlinkyy。

- SearchText – 值字串，長度不可超過 1024 個位元組。

  用於文字搜尋的字串。

  根據與值的完全相符，在引號篩選條件中指定值。

- SortCriteria – 一個 SortCriterion 物件陣列，不可超過 1 個結構。

  用於依資料欄位名稱排序結果的條件清單，按遞增或遞減順序。

- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  在單一回應中可傳回的最大資料表數量。

- ResourceShareType – UTF-8 字串 (有效值：FOREIGN | ALL | FEDERATED)。

  可讓您指定要搜尋與您帳戶共用的資料表。允許的值為 FOREIGN 或 ALL。

  - 如果設定為 FOREIGN，會搜尋與您帳戶共用的資料表。
  - 如果設定為 ALL，會搜尋與您帳戶共用的資料表，以及您本機帳戶中的資料表。

回應

- NextToken – UTF-8 字串。

  接續字元，如果目前清單區段不是最後一個，將會出現此接續字元。
- TableList – 一個 資料表 物件陣列。

  要求的 Table 物件的清單。SearchTables 回應只會傳回您有權存取的資料表。

錯誤

- InternalServiceException
- InvalidInputException
- OperationTimeoutException

GetPartitionIndexes 操作（Python：獲取分區索引）

擷取與資料表相關聯的分割區索引。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在的目錄的 ID。
- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  指定您要從中擷取分割區索引的資料庫名稱。
- TableName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  指定您要擷取分割區索引的資料表名稱。
- NextToken – UTF-8 字串。

  接續符記，如果這是接續呼叫，將會包含在內。

回應

- PartitionIndexDescriptorList – 一個 PartitionIndexDescriptor 物件陣列。

  索引描述元的清單。
- NextToken – UTF-8 字串。

  接續字元，如果目前清單區段不是最後一個，將會出現此接續字元。

錯誤

- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- EntityNotFoundException
- ConflictException

CreatePartitionIndex 動作（Python：創建分區索引）

在現有資料表中建立指定的分割區索引。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在的目錄的 ID。
- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  指定您要建立分割區索引的資料庫名稱。
- TableName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  指定您要建立分割區索引的資料表名稱。
- PartitionIndex – 必要：PartitionIndex 物件。

  指定 PartitionIndex 結構，在現有資料表中建立分割區索引。

回應

- 無回應參數。

錯誤

- `AlreadyExistsException`
- `InvalidInputException`
- `EntityNotFoundException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeletePartitionIndex 動作（Python：刪除分區索引）

從現有的資料表刪除指定的分割區索引。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在的目錄的 ID。

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  指定您要從中刪除分割區索引的資料庫名稱。

- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  指定您要從中刪除分割區索引的資料表名稱。

- `IndexName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要刪除的分割區索引的名稱。

回應

- 無回應參數。

錯誤

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `EntityNotFoundException`
- `ConflictException`
- `GlueEncryptionException`

GetColumnStatisticsForTable 動作（Python：獲取列統計表）

擷取欄的資料表統計數字。

此作業所需的 Identity and Access Management (IAM) 許可為 `GetTable`。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  有問題分區所在 Data Catalog 的 ID。如果沒有提供，則依預設會使用 AWS 帳號 ID。
- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區所在的目錄資料庫的名稱。
- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區的資料表的名稱。
- `ColumnNames` – 必要：UTF-8 字串的陣列，不可超過 100 個字串。

  欄名稱之清單。

回應

- `ColumnStatisticsList` – 一個 ColumnStatistics 物件陣列。

  的清單 ColumnStatistics。

- `Errors` – 一個 ColumnError 物件陣列。

  無法 ColumnStatistics 擷取的清單。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

UpdateColumnStatisticsForTable 行動（Python：更新列統計表）

建立或更新欄的資料表統計數字。

此作業所需的 Identity and Access Management (IAM) 許可為 `UpdateTable`。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  有問題分區所在 Data Catalog 的 ID。如果沒有提供，則依預設會使用 AWS 帳號 ID。

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區所在的目錄資料庫的名稱。

- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區的資料表的名稱。

- `ColumnStatisticsList` – 必要：ColumnStatistics 物件的陣列，不可超過 25 個結構。

  欄統計數字的清單。

回應

- `Errors` – 一個 ColumnStatisticsError 物件陣列。

  的清單 ColumnStatisticsErrors。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeleteColumnStatisticsForTable 操作（Python：刪除列統計表）

擷取欄的資料表統計數字。

此作業所需的 Identity and Access Management (IAM) 許可為 `DeleteTable`。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  有問題分區所在 Data Catalog 的 ID。如果沒有提供，則依預設會使用 AWS 帳號 ID。
- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區所在的目錄資料庫的名稱。
- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區的資料表的名稱。

- ColumnName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欄位的名稱。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

分區 API

Partition API 說明用於分區的資料類型和操作。

資料類型

- Partition 結構
- PartitionInput 結構
- PartitionSpecWithSharedStorageDescriptor 結構
- PartitionListComposingSpec 結構
- PartitionSpecProxy 結構
- PartitionValueList 結構
- Segment 結構
- PartitionError 結構
- BatchUpdatePartitionFailureEntry 結構
- BatchUpdatePartitionRequestEntry 結構

- StorageDescriptor 結構
- SchemaReference 結構
- SerDeInfo 結構
- SkewedInfo 結構

Partition 結構

代表資料表資料的切片。

欄位

- `Values` – UTF-8 字串陣列。

  分區的值。

- `DatabaseName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要在其中建立分割區的目錄資料庫名稱。

- `TableName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於建立分區的資料庫資料表之名稱。

- `CreationTime` – 時間戳記。

  建立分區的時間。

- `LastAccessTime` – 時間戳記。

  上次存取分區的時間。

- `StorageDescriptor` – StorageDescriptor 物件。

  提供有關分區實體存放位置的資訊。

- `Parameters` – 金鑰值對的對應陣列。

  每個金鑰都是金鑰字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串，長度不可超過 512000 個位元組。

這些金鑰值對會定義分割區參數。

- `LastAnalyzedTime` – 時間戳記。

  此分區上一次運算欄位統計的時間。

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  分割區所在 Data Catalog 的 ID。

PartitionInput 結構

用於建立和更新分割區的結構。

欄位

- `Values` – UTF-8 字串陣列。

  分區的值。雖然軟體開發套件不需要此參數，您必須為此參數指定一個有效的輸入。

  新分割區的索引鍵值必須以字串物件陣列傳遞，且順序必須與出現在 Amazon S3 前綴中的分割區索引鍵順序相同。否則， AWS Glue 將值添加到錯誤的鍵。

- `LastAccessTime` – 時間戳記。

  上次存取分區的時間。

- `StorageDescriptor` – StorageDescriptor 物件。

  提供有關分區實體存放位置的資訊。

- `Parameters` – 金鑰值對的對應陣列。

  每個金鑰都是金鑰字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串，長度不可超過 512000 個位元組。

  這些金鑰值對會定義分割區參數。

- `LastAnalyzedTime` – 時間戳記。

  此分區上一次運算欄位統計的時間。

PartitionSpecWithSharedStorageDescriptor 結構

適用於共用實體位置的分區的分區規格。

欄位

- `StorageDescriptor` – StorageDescriptor 物件。

  共用實體儲存資訊。
- `Partitions` – 一個 分區 物件陣列。

  共用此實體位置的分區的清單。

PartitionListComposingSpec 結構

列出相關的分區。

欄位

- `Partitions` – 一個 分區 物件陣列。

  符合組成規格的分區的清單。

PartitionSpecProxy 結構

提供指定分區的根路徑。

欄位

- `DatabaseName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  分區所在的目錄資料庫。
- `TableName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  包含資分區之資料表的名稱。
- `RootPath` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

定址分區的 Proxy 的根路徑。

- PartitionSpecWithSharedSD – PartitionSpecWithSharedStorageDescriptor 物件。

  共用實體儲存位置的分區的規格。

- PartitionListComposingSpec – PartitionListComposingSpec 物件。

  指定分區的清單。

PartitionValueList 結構

包含定義分區的值的清單。

欄位

- Values – 必要：UTF-8 字串陣列。

  值的清單。

Segment 結構

定義資料表分割區的非重疊區域，讓多個要求以平行方式執行。

欄位

- SegmentNumber – 必要：數字 (整數)，不可大於 None (無)。

  區段的零基索引號碼。例如，如果區段總數為 4，則 SegmentNumber 的值將從 0 到 3。

- TotalSegments – 必要：數字 (整數)，不可小於 1，也不可以大於 10。

  區段的總數。

PartitionError 結構

包含有關分區錯誤的資訊。

欄位

- PartitionValues – UTF-8 字串陣列。

定義分區的值。

- `ErrorDetail` – ErrorDetail 物件。

  關於分區錯誤的詳細資訊。

BatchUpdatePartitionFailureEntry 結構

包含有關批次更新分割區錯誤的資訊。

欄位

- `PartitionValueList` – UTF-8 字串的陣列，不可超過 100 個字串。

  定義分割區的值的清單。

- `ErrorDetail` – ErrorDetail 物件。

  關於批次更新分割區錯誤的詳細資訊。

BatchUpdatePartitionRequestEntry 結構

包含用於更新分割區的值和結構的結構。

欄位

- `PartitionValueList` – 必要：UTF-8 字串的陣列，不可超過 100 個字串。

  定義分割區的值的清單。

- `PartitionInput` – 必要：PartitionInput 物件。

  用於更新分割區的結構。

StorageDescriptor 結構

描述資料表資料的實體儲存。

欄位

- `Columns` – 一個 資料行 物件陣列。

資料表中的 Columns 的清單。

- Location – 位置字串，長度不可超過 2056 個位元組，需符合URI address multi-line string pattern。

  資料表的實體位置。依預設，它採用倉儲位置的形式，後面接著是資料庫在倉儲中的位置，最後是資料表名稱。

- AdditionalLocations – UTF-8 字串陣列。

  指向 Delta 資料表所在路徑的位置清單。

- InputFormat – 格式字串，長度不可超過 128 個位元組，需符合Single-line string pattern。

  輸入格式：SequenceFileInputFormat (二進位)，或者 TextInputFormat，或自訂格式。

- OutputFormat – 格式字串，長度不可超過 128 個位元組，需符合Single-line string pattern。

  輸出格式：SequenceFileOutputFormat (二進位)，或者 IgnoreKeyTextOutputFormat，或自訂格式。

- Compressed – 布林值。

  如果資料表中的資料都經過壓縮則為 True，否則為 False

- NumberOfBuckets – 數字 (整數)。

  如果資料表包含任何維度欄位，將必須指定。

- SerdeInfo – SerDe信息 物件。

  序列化/反序列化（ ）信息。SerDe

- BucketColumns – UTF-8 字串陣列。

  資料表中的縮減器分組欄位、叢集欄位及值區欄位的清單。

- SortColumns – 一個 順序 物件陣列。

  指定資料表中各個儲存貯體排序順序的清單。

- Parameters – 金鑰值對的對應陣列。

  每個金鑰都是金鑰字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串，長度不可超過 512000 個位元組。

使用者提供的屬性，格式為金鑰/值。

- SkewedInfo – SkewedInfo 物件。

  關於欄位中頻繁出現值的資訊 (偏斜值)。

- StoredAsSubDirectories – 布林值。

  如果資料表資料存放於子目錄則為 True，否則為 False

- SchemaReference – SchemaReference 物件。

  參考儲存在結構描述登錄中之結 AWS Glue 構描述的物件。

  建立資料表時，您可以傳遞結構描述欄的空白清單，並改用結構描述參考。

SchemaReference 結構

參考儲存在結構描述登錄中之結 AWS Glue 構描述的物件。

欄位

- SchemaId – SchemaId 物件。

  包含結構描述身分欄位的結構。必須提供此結構或 SchemaVersionId。

- SchemaVersionId – UTF-8 字串，長度不可小於 36 個位元組，也不可以超過 36 個位元組，且需符合 Custom string pattern #17。

  指派給結構描述版本的唯一 ID。必須提供此結構或 SchemaId。

- SchemaVersionNumber – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  結構描述的版本編號。

SerDeInfo 結構

有關作為提取器和加載器的序列化/反序列化程序（SerDe）的信息。

欄位

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

的名稱 SerDe。

- `SerializationLibrary` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  通常是實作 SerDe. 例如，`org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe`。
- `Parameters` – 金鑰值對的對應陣列。

  每個金鑰都是金鑰字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串，長度不可超過 512000 個位元組。

  這些鍵值配對定義的初始化參數。 SerDe

SkewedInfo 結構

指定資料表中的偏斜值。偏斜值是指頻率發生非常高的值。

欄位

- `SkewedColumnNames` – UTF-8 字串陣列。

  包含偏斜值的欄位名稱清單。
- `SkewedColumnValues` – UTF-8 字串陣列。

  頻繁出現而被視為偏斜的值的清單。
- `SkewedColumnValueLocationMaps` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  偏斜值與包含這些值的欄位的映射。

作業

- CreatePartition 動作（Python：創建分區）
- BatchCreatePartition 操作（Python：批處理創建分區）

- UpdatePartition 行動（Python：更新分區）
- DeletePartition 動作（Python：刪除分區）
- BatchDeletePartition 操作（Python：批處理刪除分區）
- GetPartition 動作（Python：獲取分區）
- GetPartitions 動作（Python：獲取分區）
- BatchGetPartition 操作（Python：批處理分區）
- BatchUpdatePartition 操作（Python：批處理更新分區）
- GetColumnStatisticsForPartition 操作（Python：獲取列統計 _ 分區）
- UpdateColumnStatisticsForPartition 操作（Python：更新列統計 _ 分區）
- DeleteColumnStatisticsForPartition 操作（Python：刪除列統計 _ 分區）

CreatePartition 動作（Python：創建分區）

建立新的分區。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要在其中建立磁碟分割之目錄的 AWS 帳戶識別碼。
- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  在其中建立分區的中繼資料資料庫的名稱。
- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要在其中建立分割區的中繼資料表名稱。
- `PartitionInput` – 必要：PartitionInput 物件。

  用於定義要建立的分區的 `PartitionInput` 結構。

回應

- 無回應參數。

錯誤

- `InvalidInputException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `GlueEncryptionException`

BatchCreatePartition 操作（Python：批處理創建分區）

在批次處理中建立一或多個分區。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  分區建立所在目錄的 ID。目前，這應該是 AWS 帳戶 ID。
- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  在其中建立分區的中繼資料資料庫的名稱。
- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要在其中建立分割區的中繼資料表名稱。
- `PartitionInputList` – 必要：PartitionInput 物件的陣列，不可超過 100 個結構。

  用於定義要建立分區的 `PartitionInput` 結構清單。

回應

- `Errors` – 一個 PartitionError 物件陣列。

  在嘗試建立所要求分區時發生的錯誤。

錯誤

- `InvalidInputException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `GlueEncryptionException`

UpdatePartition 行動（Python：更新分區）

更新分區。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要更新之分區所在 Data Catalog 的 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。
- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  有問題資料表所在目錄資料庫的名稱。
- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  更新的分區所在資料表的名稱。
- `PartitionValueList` – 必要：UTF-8 字串的陣列，不可超過 100 個字串。

  定義要更新之分割區的分割區索引鍵值的清單。
- `PartitionInput` – 必要：PartitionInput 物件。

  要更新分區所在的新分區物件。

  `Values` 屬性無法變更。如果您想要變更分割區的分割區索引鍵值，請刪除並重新建立分割區。

回應

- 無回應參數。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeletePartition 動作（Python：刪除分區）

刪除指定的分區。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要刪除分區所在 Data Catalog 的 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。
- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  有問題資料表所在目錄資料庫的名稱。
- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表的名稱，資料表內包含將要刪除的分區。
- `PartitionValues` – 必要：UTF-8 字串陣列。

  定義分區的值。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchDeletePartition 操作（Python：批處理刪除分區）

在批次處理中刪除一或多個分區。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要刪除分區所在 Data Catalog 的 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。
- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  有問題資料表所在目錄資料庫的名稱。
- TableName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表的名稱，資料表內包含將要刪除的分區。
- PartitionsToDelete – 必要：PartitionValue清單 物件的陣列，不可超過 25 個結構。

  用於定義要刪除分區的 PartitionInput 結構清單。

回應

- Errors – 一個 PartitionError 物件陣列。

  在嘗試建立所要求分區時發生的錯誤。

錯誤

- InvalidInputException

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetPartition 動作（Python：獲取分區）

擷取有關指定的分區的資訊。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  有問題分區所在 Data Catalog 的 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。
- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區所在的目錄資料庫的名稱。
- TableName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區的資料表的名稱。
- PartitionValues – 必要：UTF-8 字串陣列。

  定義分區的值。

回應

- Partition – 分區 物件。

  要求的資訊，格式為 Partition 物件。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException

- OperationTimeoutException
- GlueEncryptionException
- FederationSourceException
- FederationSourceRetryableException

GetPartitions 動作（Python：獲取分區）

擷取資料表中有關分區的資訊。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  有問題分區所在 Data Catalog 的 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。
- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區所在的目錄資料庫的名稱。
- TableName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區的資料表的名稱。
- Expression – 述詞字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  用於篩選要傳回之分區的表達式。

  該表達式使用類似於 SQL WHERE 篩選條件子句的 SQL 語法。SQL 陳述式剖析器 JSQLParser 剖析該表達式。

  Operators (運算子)：以下是您可以在 Expression API 呼叫中使用的運算子：

  =

  檢查兩個運算元的值是否相等；如果是，則條件成立。

  範例：假設「variable a」(變數 a) 保持為 10，而「variable b」(變數 b) 保持為 20。

  (a= b) 不為真。

< >

檢查兩個運算元的值是否相等；如果值並不相等，則條件成立。

範例：(a < > b) 為真。

>

檢查左運算元的值是否大於右運算元的值；如果是，則條件成立。

範例：(a > b) 不為真。

<

檢查左運算元的值是否小於右運算元的值；如果是，則條件成立。

範例：(a < b) 為真。

>=

檢查左運算元的值是否大於或等於右運算元的值；如果是，則條件成立。

範例：(a > = b) 不為真。

<=

檢查左運算元的值是否小於或等於右運算元的值；如果是，則條件成立。

範例：(a <= b) 為真。

AND、OR、IN、BETWEEN、LIKE、NOT、IS NULL

邏輯運算子。

支援的分割區索引鍵類型：以下是支援的分割區索引鍵。

- `string`
- `date`
- `timestamp`
- `int`
- `bigint`
- `long`
- `tinyint`
- `smallint`

- decimal

如果遇到無效的類型，則擲出例外狀況。

下表顯示每個類型的有效運算子。當您定義一個爬蟲程式時，partitionKey 類型將以 STRING 建立，並與目錄分區。

範例 API 呼叫：

Example

該資料表 twitter_partition 有三個分區：

```
year = 2015
        year = 2016
        year = 2017
```

Example

取得 year 等於 2015 的分區

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
        --expression "year*=*'2015'"
```

Example

取得 year 介於 2016-2018 之間 (不包含) 的分區

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
        --expression "year>'2016' AND year<'2018'"
```

Example

取得 year 介於 2015-2018之間 (不包含) 的分區 以下 API 呼叫彼此相同：

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
        --expression "year>='2015' AND year<='2018'"

        aws glue get-partitions --database-name dbname --table-name
 twitter_partition
        --expression "year BETWEEN 2015 AND 2018"
```

```
       aws glue get-partitions --database-name dbname --table-name
twitter_partition
       --expression "year IN (2015,2016,2017,2018)"
```

Example

萬用字元分區篩選條件，其中以下呼叫輸出將為分區年份 = 2017。LIKE 不支援規則表達式。

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
       --expression "year LIKE '%7'"
```

- NextToken – UTF-8 字串。

  接續字元，如果這不是第一個用於擷取這些分區的呼叫。

- Segment – 區段 物件。

  在此要求中要掃描的資料表的分區的區段。

- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  在單一回應中可傳回的最大分區數量。

- ExcludeColumnSchema – 布林值。

  如果為 True，指定不返回分割區資料行結構描述。僅對其他分割區屬性 (例如分割區值或位置) 感興趣時會很實用。這種方法不會傳回重複的資料，從而避免了大型響應的問題。

- TransactionId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #16。

  藉助其讀取分割區內容的交易 ID。

- QueryAsOfTime – 時間戳記。

  讀取分割區內容的時間。如果沒有設定，將使用最近的交易遞交時間。無法連同 TransactionId 一起指定。

回應

- Partitions – 一個 分區 物件陣列。

  要求的分區的清單。

- NextToken – UTF-8 字串。

  接續字元，如果傳回的分區清單不包含最後一個分區。

錯誤

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- GlueEncryptionException
- InvalidStateException
- ResourceNotReadyException
- FederationSourceException
- FederationSourceRetryableException

BatchGetPartition 操作（Python：批處理分區）

在批次要求中擷取分區。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  有問題分區所在 Data Catalog 的 ID。如果沒有提供，則依預設會使用 AWS 帳號 ID。
- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區所在的目錄資料庫的名稱。
- TableName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區的資料表的名稱。
- PartitionsToGet – 必要：一個 PartitionValue清單 物件陣列，不可超過 1000 個結構。

  用於識別要擷取的分區的分區值清單。

回應

- `Partitions` – 一個 分區 物件陣列。

  要求的分區的清單。

- `UnprocessedKeys` – 一個 PartitionValue清單 物件陣列，不可超過 1000 個結構。

  在未傳回分區之要求中的分區值清單。

錯誤

- `InvalidInputException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `InternalServiceException`
- `GlueEncryptionException`
- `InvalidStateException`
- `FederationSourceException`
- `FederationSourceRetryableException`

BatchUpdatePartition 操作（Python：批處理更新分區）

在批次處理中更新一或多個分割區。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  分割區更新所在目錄的 ID。目前，這應該是 AWS 帳戶 ID。

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  在其中更新分割區的中繼資料資料庫的名稱。

- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

要在其中更新分割區的中繼資料資料表名稱。

- `Entries` – 必要：BatchUpdatePartitionRequest入境 物件陣列，不小於 1 個結構，也不大於 100 個結構。

  最多 100 個要更新的 `BatchUpdatePartitionRequestEntry` 物件清單。

回應

- `Errors` – 一個 BatchUpdatePartitionFailure入境 物件陣列。

  在嘗試更新所要求分區時發生的錯誤。`BatchUpdatePartitionFailureEntry` 物件的清單。

錯誤

- `InvalidInputException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `InternalServiceException`
- `GlueEncryptionException`

GetColumnStatisticsForPartition 操作（Python：獲取列統計 _ 分區）

擷取欄的分割區統計數字。

此作業所需的 Identity and Access Management (IAM) 許可為 `GetPartition`。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  有問題分區所在 Data Catalog 的 ID。如果沒有提供，則依預設會使用 AWS 帳號 ID。

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區所在的目錄資料庫的名稱。

- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區的資料表的名稱。
- `PartitionValues` – 必要：UTF-8 字串陣列。

  用於識別分割區的分割區值清單。
- `ColumnNames` – 必要：UTF-8 字串的陣列，不可超過 100 個字串。

  欄名稱之清單。

回應

- `ColumnStatisticsList` – 一個 ColumnStatistics 物件陣列。

  無法 ColumnStatistics 擷取的清單。
- `Errors` – 一個 ColumnError 物件陣列。

  擷取欄統計數字時發生錯誤。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

UpdateColumnStatisticsForPartition 操作（Python：更新列統計 _ 分區）

建立或更新欄的分割區統計數字。

此作業所需的 Identity and Access Management (IAM) 許可為 `UpdatePartition`。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

有問題分區所在 Data Catalog 的 ID。如果沒有提供，則依預設會使用 AWS 帳號 ID。

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區所在的目錄資料庫的名稱。

- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區的資料表的名稱。

- `PartitionValues` – 必要：UTF-8 字串陣列。

  用於識別分割區的分割區值清單。

- `ColumnStatisticsList` – 必要：ColumnStatistics 物件的陣列，不可超過 25 個結構。

  欄統計數字的清單。

回應

- `Errors` – 一個 ColumnStatisticsError 物件陣列。

  更新欄統計數字時發生錯誤。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeleteColumnStatisticsForPartition 操作（Python：刪除列統計 _ 分區）

刪除欄的分割區欄統計數字。

此作業所需的 Identity and Access Management (IAM) 許可為 `DeletePartition`。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  有問題分區所在 Data Catalog 的 ID。如果沒有提供，則依預設會使用 AWS 帳號 ID。
- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區所在的目錄資料庫的名稱。
- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分區的資料表的名稱。
- `PartitionValues` – 必要：UTF-8 字串陣列。

  用於識別分割區的分割區值清單。
- `ColumnName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欄的名稱。

回應

- 無回應參數。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

連線 API

連線 API 說明 AWS Glue 連線資料類型，以及建立、刪除、更新和列出連線的 API。

資料類型

- Connection 結構
- ConnectionInput 結構
- PhysicalConnectionRequirements 結構
- GetConnectionsFilter 結構

Connection 結構

定義連接至資料來源的連線

欄位

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  連線定義的名稱。
- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  此連接的描述。
- ConnectionType – UTF-8 字串 (有效值：JDBC | SFTP | MONGODB | KAFKA | NETWORK | MARKETPLACE | CUSTOM | SALESFORCE)。

  連線的類型。目前不支援 SFTP。
- MatchCriteria – UTF-8 字串的陣列，不可超過 10 個字串。

  用於選擇此連線的條件清單。
- ConnectionProperties – 金鑰值對的對應陣列，不超過 100 對

  每個關鍵字都是一個 UTF-8 字串 (有效值：HOSTPORTUSERNAME="USER_NAME"PASSWORDENCRYPTED_PASSWORDJDBC_DRIVER_JAR_URIJDBC_DRIVER_CLASS_NAMEJDBC_ENGINE | JDBC_ENGINE_VERSION | CONFIG_FILES | INSTANCE_ID | JDBC_CONNECTION_URL | JDBC_ENFORCE_SSL | CUSTOM_JDBC_CERT | SKIP_CUSTOM_JDBC_CERT_VALIDATION | CUSTOM_JDBC_CERT_STRING | CONNECTION_URL | KAFKA_BOOTSTRAP_SERVERS | KAFKA_SSL_ENABLED | KAFKA_CUSTOM_CERT | KAFKA_SKIP_CUSTOM_CERT_VALIDATION | KAFKA_CLIENT_KEYSTORE | KAFKA_CLIENT_KEYSTORE_PASSWORD |

KAFKA_CLIENT_KEY_PASSWORD | | ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD | SECRET_ID | CONNECTOR_URL | CONNECTOR_TYPE | CONNECTOR_CLASS_NAME | KAFKA_SASL_MECHANISM | KAFKA_SASL_PLAIN_USERNAME | KAFKA_SASL_PLAIN_PASSWORD | ENCRYPTED_KAFKA_SASL_PLAIN_PASSWORD | KAFKA_SASL_SCRAM_USERNAME | KAFKA_SASL_SCRAM_PASSWORD| KAFKA_SASL_SCRAM_SECRETS_ARN | ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD | KAFKA_SASL_GSSAPI_KEYTAB KAFKA_SASL_GSSAPI_KRB5_CONF | KAFKA_SASL_GSSAPI_SERVICE | KAFKA_SASL_GSSAPI_PRINCIPAL |ROLE_ARN) 。

每個值都是 Value 字串，長度不可超過 1024 個位元組。

這些金鑰值對會定義連線的參數：

- HOST- 主機 URI：完整網域名稱 (FQDN) 或資料庫主機 IPv4 地址。
- PORT- 資料庫主機用來監聽資料庫連線之連接埠的連接埠號碼 (介於 1024 和 65535 之間)。
- USER_NAME- 用來登入資料庫的名稱。USER_NAME 的值字串為 "USERNAME"。
- PASSWORD- 使用者名稱的密碼 (如果有使用)。
- ENCRYPTED_PASSWORD - 當您在 Data Catalog 加密設定中的 ConnectionPasswordEncryption 啟用連線密碼保護時，此欄位存放加密密碼。
- JDBC_DRIVER_JAR_URI - 包含要使用之 JDBC 驅動程式 JAR 檔案的 Amazon Simple Storage Service (Amazon S3) 路徑。
- JDBC_DRIVER_CLASS_NAME - 要使用之 JDBC 驅動程式的類別名稱。
- JDBC_ENGINE - 要使用的 JDBC 引擎名稱。
- JDBC_ENGINE_VERSION - 要使用的 JDBC 引擎版本。
- CONFIG_FILES - (保留以供日後使用)。
- INSTANCE_ID- 要使用的執行個體 ID。
- JDBC_CONNECTION_URL - 連接至 JDBC 資料來源的 URL。
- JDBC_ENFORCE_SSL - 布林值字串 (true、false)，會指定是否對用戶端上的 JDBC 連線強制使用主機名稱相符的 Secure Sockets Layer (SSL)。預設值為 false。
- CUSTOM_JDBC_CERT-指定客戶根憑證的 Amazon S3 位置。 AWS Glue 連線至客戶資料庫時，會使用此根憑證來驗證客戶的憑證。 AWS Glue 僅處理 X.509 憑證。提供的憑證必須為 DER 編碼，並以 Base64 編碼 PEM 格式提供。
- SKIP_CUSTOM_JDBC_CERT_VALIDATION-默認情況下，這是false。 AWS Glue 驗證客戶憑證的簽名演算法和主體公開金鑰演算法。簽章演算法允許的演算法僅限

SHA256withRSA、SHA384withRSA 或 SHA512withRSA。針對主體公開金鑰演算法，金鑰長度必須至少為 2048。您可以將此屬性的值設定為 `true` 以略過客戶憑證的 AWS Glue驗證。

- `CUSTOM_JDBC_CERT_STRING`-用於網域比對或辨別名稱比對以防止 man-in-the-middle 攻擊的自訂 JDBC 憑證字串。在 Oracle 資料庫中，這會做為 `SSL_SERVER_CERT_DN` 使用；在 Microsoft SQL Server 中，這會做為 `hostNameInCertificate` 使用。
- `CONNECTION_URL` - 連接至一般 (非 JDBC) 資料來源的 URL。
- `SECRET_ID` - 用於憑證的秘密管理器的秘密 ID。
- `CONNECTOR_URL` - MARKETPLACE 或 CUSTOM 連線的連接器 URL。
- `CONNECTOR_TYPE` - MARKETPLACE 或 CUSTOM 連線的連接器類型。
- `CONNECTOR_CLASS_NAME` - MARKETPLACE 或 CUSTOM 連線的連接器類別名稱。
- `KAFKA_BOOTSTRAP_SERVERS` - 主機和連接埠對的逗號分隔清單，列出 Kafka 叢集中 Kafka 用戶端即將連線以引導自身的 Apache Kafka 中介裝置地址。
- `KAFKA_SSL_ENABLED` - 在 Apache Kafka 連線上啟用或是停用 SSL。預設值為 "true"。
- `KAFKA_CUSTOM_CERT` - 私有 CA 憑證檔的 Amazon S3 URL (.pem 格式)。預設為空字串。
- `KAFKA_SKIP_CUSTOM_CERT_VALIDATION`-是否跳過 CA 憑證檔案的驗證。 AWS Glue 驗證三種演算法：具有 RSA 的 SHA256、與 RSA 和 RSA 的 SHA512。預設值為 "false"。
- `KAFKA_CLIENT_KEYSTORE` - 用於 Kafka 用戶端身分驗證的用戶端金鑰存放區檔案的 Amazon S3 位置 (選用)。
- `KAFKA_CLIENT_KEYSTORE_PASSWORD` - 存取所提供金鑰存放區的密碼 (選用)。
- `KAFKA_CLIENT_KEY_PASSWORD` - 金鑰存放區可以由多個金鑰組成，所以這是用來存取用於 Kafka 伺服器端金鑰之用戶端金鑰的密碼 (選用)。
- `ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD`-Kafka 用戶端金鑰儲存庫密碼的加密版本 (如果使用者已選取 AWS Glue 加密密碼設定)。
- `ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD`-Kafka 用戶端金鑰密碼的加密版本 (如果使用者已選取 AWS Glue 加密密碼設定)。
- `KAFKA_SASL_MECHANISM`-`"SCRAM-SHA-512"` `"GSSAPI"`、`"AWS_MSK_IAM"`、或`"PLAIN"`。這些是受支援的 SASL 機制。
- `KAFKA_SASL_PLAIN_USERNAME`-用於透過「PLAIN」機制進行驗證的純文字使用者名稱。
- `KAFKA_SASL_PLAIN_PASSWORD`-用於透過「PLAIN」機制進行驗證的純文字密碼。
- `ENCRYPTED_KAFKA_SASL_PLAIN_PASSWORD`-卡夫卡 SASL 普通密碼的加密版本 (如果使用者已選取 AWS Glue 加密密碼設定)。

- KAFKA_SASL_SCRAM_USERNAME - 用於藉助 "SCRAM-SHA-512" 機制進行身分驗證的純文字使用者名稱。
- KAFKA_SASL_SCRAM_PASSWORD - 用於藉助 "SCRAM-SHA-512" 機制進行身分驗證的純文字密碼。
- ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD-卡夫卡 SASL SCRAM 密碼的加密版本（如果使用者已選取 AWS Glue 加密密碼設定）。
- KAFKA_SASL_SCRAM_SECRETS_ARN-秘 AWS 密管理器中秘密的 Amazon 資源名稱。
- KAFKA_SASL_GSSAPI_KEYTAB - Kerberos `keytab` 檔案的 S3 位置。Keytab 存放了一個或多個主體的長期金鑰。如需詳細資訊，請參閱 MIT Kerberos 文件：Keytab。
- KAFKA_SASL_GSSAPI_KRB5_CONF - Kerberos `krb5.conf` 檔案的 S3 位置。krb5.conf 存放了 Kerberos 組態資訊，例如 KDC 伺服器的位置。如需詳細資訊，請參閱 MIT Kerberos 文件：krb5.conf。
- KAFKA_SASL_GSSAPI_SERVICE - Kerberos 服務名稱，如在 Kafka 配置使用 `sasl.kerberos.service.name` 進行設定。
- KAFKA_SASL_GSSAPI_PRINCIPAL-使用的 Kerberos 印刷品的名稱。 AWS Glue如需詳細資訊，請參閱 Kafka 文件：設定 Kafka 代理程式。

- `PhysicalConnectionRequirements` – PhysicalConnection要求 物件。

  成功建立此連線所需的實體連線需求`SecurityGroup`，例如虛擬私有雲 (VPC) 和。

- `CreationTime` – 時間戳記。

  建立此連線定義的時間戳記。

- `LastUpdatedTime` – 時間戳記。

  上次更新連線定義的時間戳記。

- `LastUpdatedBy` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  上一次更新此連線定義的使用者、群組或角色。

- `Status` – UTF-8 字串 (有效值：`READY` | `IN_PROGRESS` | `FAILED`)。

  連線狀態。可以是下列其中一個：`READY`、`IN_PROGRESS` 或 `FAILED`。

- `StatusReason`— UTF-8 字串，長度不小於 1 個或超過 16384 個位元組。

  連線狀態的原因。

- `LastConnectionValidationTime` – 時間戳記。

  上次驗證此連線的時間戳記。

- `AuthenticationConfiguration` – AuthenticationConfiguration 物件。

  連線的驗證內容。

ConnectionInput 結構

結構用於指定要建立或更新的連線。

欄位

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  連線的名稱。

- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  此連接的描述。

- `ConnectionType`— 必要項目：UTF-8 字串 (有效值：`JDBCSFTPMONGODBKAFKA| NETWORK | MARKETPLACE | | CUSTOM |SALESFORCE`)。

  連線的類型。目前支援這些類型：

  - `JDBC` - 指定透過 Java 資料庫連接 (JDBC) 與資料庫的連線。

    `JDBC`連接使用以下內容 ConnectionParameters。

    - 必要：所有的 (`HOST`、`PORT`、`JDBC_ENGINE`) 或 `JDBC_CONNECTION_URL`。
    - 必要：所有的 (`USERNAME`、`PASSWORD`) 或 `SECRET_ID`。
    - 選用：`JDBC_ENFORCE_SSL`、`CUSTOM_JDBC_CERT`、`CUSTOM_JDBC_CERT_STRING`、`SKIP_CUSTOM_` 這些參數用於使用 JDBC 設定 SSL。

  - `KAFKA` - 指定連至 Apache Kafka 串流平台的連線。

    `KAFKA`連接使用以下內容 ConnectionParameters。

    - 必要：`KAFKA_BOOTSTRAP_SERVERS`。

- 選
  用：KAFKA_SSL_ENABLED、KAFKA_CUSTOM_CERT、KAFKA_SKIP_CUSTOM_CERT_VALIDATION。
  這些參數用於使用 KAFKA 設定 SSL。
- 選
  用：KAFKA_CLIENT_KEYSTORE、KAFKA_CLIENT_KEYSTORE_PASSWORD、KAFKA_CLIENT_KEY_P
  這些參數用於在 KAFKA 中設定使用 SSL 的 TLS 用戶端組態。
- 選用：KAFKA_SASL_MECHANISM。可指定為 SCRAM-SHA-512、GSSAPI 或 AWS_MSK_IAM。
- 選
  用：KAFKA_SASL_SCRAM_USERNAME、KAFKA_SASL_SCRAM_PASSWORD、ENCRYPTED_KAFKA_SAS
  這些參數用於使用 KAFKA 設定 SASL/SCRAM-SHA-512 驗證。
- 選
  用：KAFKA_SASL_GSSAPI_KEYTAB、KAFKA_SASL_GSSAPI_KRB5_CONF、KAFKA_SASL_GSSAPI_
  這些參數用於使用 KAFKA 設定 SASL/GSSAPI 驗證。

- MONGODB - 指定與 MongoDB 文件資料庫的連接。

  MONGODB連接使用以下內容 ConnectionParameters。

  - 必要：CONNECTION_URL。
  - 必要：所有的 (USERNAME、PASSWORD) 或 SECRET_ID。
- SALESFORCE-使用 OAuth 驗證指定與銷售力量的連線。
  - 需要設定AuthenticationConfiguration成員。
- NETWORK - 將網路連線指定至 Amazon Virtual Private Cloud 環境 (Amazon VPC) 內的資料來源。

  NETWORK連線不需要 ConnectionParameters。相反地，請提供 PhysicalConnectionRequirements.
- MARKETPLACE-使用從中購買的連接器中包含的組態設定， AWS Marketplace 以讀取和寫入資料存放區原生不支援的資料倉庫。 AWS Glue

  MARKETPLACE連接使用以下內容 ConnectionParameters。

  - 必
    要：CONNECTOR_TYPE、CONNECTOR_URL、CONNECTOR_CLASS_NAME、CONNECTION_URL。
  - JDBC CONNECTOR_TYPE 連線的必要項目：所有的 (USERNAME、PASSWORD) 或 SECRET_ID。
- CUSTOM - 使用自訂連接器中包含的組態設定來讀取和寫入資料存放區，這些資料存放區不受 AWS Glue原生支援。

連線不支援 SFTP。

如需 ConnectionProperties 有關如何使用選用性來配置功能的詳細資訊 AWS Glue，請參閱AWS Glue 連線內容。

如需 ConnectionProperties 有關如何使用選用性來設定 AWS Glue Studio 中功能的詳細資訊，請參閱使用連接器和連線。

- `MatchCriteria` – UTF-8 字串的陣列，不可超過 10 個字串。

  用於選擇此連線的條件清單。

- `ConnectionProperties` – 必要：索引鍵/值對的映射陣列，不可超過 100 對。

  每個關鍵字都是一個 UTF-8 字串 (有效值：HOSTPORTUSERNAME="USER_NAME"PASSWORDENCRYPTED_PASSWORDJDBC_DRIVER_JAR_URIJDBC_ENGINE | JDBC_ENGINE_VERSION | CONFIG_FILES | INSTANCE_ID | JDBC_CONNECTION_URL | JDBC_ENFORCE_SSL | CUSTOM_JDBC_CERT | SKIP_CUSTOM_JDBC_CERT_VALIDATION | CUSTOM_JDBC_CERT_STRING | CONNECTION_URL | KAFKA_BOOTSTRAP_SERVERS | KAFKA_SSL_ENABLED | KAFKA_CUSTOM_CERT | KAFKA_SKIP_CUSTOM_CERT_VALIDATION | KAFKA_CLIENT_KEYSTORE | KAFKA_CLIENT_KEYSTORE_PASSWORD | KAFKA_CLIENT_KEY_PASSWORD | | ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD | SECRET_ID | CONNECTOR_URL | CONNECTOR_TYPE | CONNECTOR_CLASS_NAME | KAFKA_SASL_MECHANISM | KAFKA_SASL_PLAIN_USERNAME | KAFKA_SASL_PLAIN_PASSWORD | ENCRYPTED_KAFKA_SASL_PLAIN_PASSWORD | KAFKA_SASL_SCRAM_USERNAME | KAFKA_SASL_SCRAM_PASSWORD| KAFKA_SASL_SCRAM_SECRETS_ARN | ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD | KAFKA_SASL_GSSAPI_KEYTAB KAFKA_SASL_GSSAPI_KRB5_CONF | KAFKA_SASL_GSSAPI_SERVICE | KAFKA_SASL_GSSAPI_PRINCIPAL |ROLE_ARN) 。

  每個值都是 Value 字串，長度不可超過 1024 個位元組。

  這些金鑰值對會定義連線的參數。

- `PhysicalConnectionRequirements` – PhysicalConnection要求 物件。

  成功建立此連線所需的實體連線需求`SecurityGroup`，例如虛擬私有雲 (VPC) 和。

- `AuthenticationConfiguration` – AuthenticationConfiguration輸入 物件。

  連線的驗證內容。用於銷售力量連線。

- `ValidateCredentials` – 布林值。

  在建立連線期間驗證認證的旗標。用於銷售力量連線。預設為 true。

PhysicalConnectionRequirements 結構

作為回應的 OAuth 用戶端 GetConnection 應用程式。

欄位

- `SubnetId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  連線使用的子網路 ID。
- `SecurityGroupIdList` – UTF-8 字串的陣列，不可超過 50 個字串。

  連線使用的安全群組 ID 清單。
- `AvailabilityZone` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  連線的可用區域。

GetConnectionsFilter 結構

篩選 `GetConnections` API 操作傳回的連線定義。

欄位

- `MatchCriteria` – UTF-8 字串的陣列，不可超過 10 個字串。

  條件字串必須符合連線定義中記錄的條件，才能傳回該連線定義。
- `ConnectionType` – UTF-8 字串 (有效值：`JDBC` | `SFTP` | `MONGODB` | `KAFKA` | `NETWORK` | `MARKETPLACE` | `CUSTOM` | `SALESFORCE`)。

  傳回的連線類型。目前不支援 SFTP。

作業

- CreateConnection 動作（Python：創建連接）

- DeleteConnection 行動（Python：刪除連接）
- GetConnection 行動（Python：獲取連接）
- GetConnections 行動（Python：獲取連接）
- UpdateConnection 行動（Python：更新連接）
- BatchDeleteConnection 行動（Python：批處理刪除連接）

CreateConnection 動作（Python：創建連接）

在 Data Catalog 中建立連線定義。

用於建立聯合資源的連線需要 IAM `glue:PassConnection` 許可。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於建立連線的 Data Catalog 之 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。
- `ConnectionInput` – 必要：ConnectionInput 物件。

  用於定義所要建立的連線的 `ConnectionInput` 物件。
- `Tags` – 金鑰值對的映射陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  指派給連線的標籤。

回應

- `CreateConnectionStatus` – UTF-8 字串 (有效值：`READY` | `IN_PROGRESS` | `FAILED`)。

  連線建立要求的狀態。對於某些身份驗證類型，請求可能需要一些時間，例如在通過 VPC 上使用令牌交換創建 OAuth 連接時。

錯誤

- `AlreadyExistsException`
- `InvalidInputException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `GlueEncryptionException`

DeleteConnection 行動（Python：刪除連接）

從 Data Catalog 刪除連線。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  連線所在的 Data Catalog 之 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。
- `ConnectionName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欲刪除的連線的名稱。

回應

- 無回應參數。

錯誤

- `EntityNotFoundException`
- `OperationTimeoutException`

GetConnection 行動（Python：獲取連接）

從 Data Catalog 擷取連線定義

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  連線所在的 Data Catalog 之 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欲擷取的連線定義的名稱。

- HidePassword – 布林值。

  可讓您擷取連線中繼資料，無需傳回密碼。例如， AWS Glue 控制台使用此標誌來檢索連接，並且不顯示密碼。當呼叫者可能沒有使用密 AWS KMS 鑰解密密碼的權限時，請設置此參數，但它確實具有訪問其餘連接屬性的權限。

回應

- Connection – 連線 物件。

  要求的連線定義。

錯誤

- EntityNotFoundException
- OperationTimeoutException
- InvalidInputException
- GlueEncryptionException

GetConnections 行動（Python：獲取連接）

從 Data Catalog 擷取連線定義清單。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

連線所在的 Data Catalog 之 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。

- `Filter` – GetConnections過濾器 物件。

  用於控制將傳回哪些連線的篩選條件。

- `HidePassword` – 布林值。

  可讓您擷取連線中繼資料，無需傳回密碼。例如， AWS Glue 控制台使用此標誌來檢索連接，並且不顯示密碼。當呼叫者可能沒有使用密 AWS KMS 鑰解密密碼的權限時，請設置此參數，但它確實具有訪問其餘連接屬性的權限。

- `NextToken` – UTF-8 字串。

  接續符記，如果這是接續呼叫。

- `MaxResults` – 數字 (整數)，不可小於 1，也不可以大於 1000。

  一次回應傳回的最大連線數量。

回應

- `ConnectionList` – 一個 連線 物件陣列。

  要求的連線定義的清單。

- `NextToken` – UTF-8 字串。

  接續符記 (如果傳回的連線清單不包括最後篩選的連線)。

錯誤

- `EntityNotFoundException`
- `OperationTimeoutException`
- `InvalidInputException`
- `GlueEncryptionException`

UpdateConnection 行動（Python：更新連接）

更新 Data Catalog 中的連線定義。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  連線所在的 Data Catalog 之 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。
- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欲更新的連線定義的名稱。
- `ConnectionInput` – 必要：ConnectionInput 物件。

  用於重新定義有問題的連線的 `ConnectionInput` 物件。

回應

- 無回應參數。

錯誤

- `InvalidInputException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `InvalidInputException`
- `GlueEncryptionException`

BatchDeleteConnection 行動（Python：批處理刪除連接）

從 Data Catalog 刪除連線定義清單。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  連線所在的 Data Catalog 之 ID。如果未提供任何內容，則預設會使用 AWS 帳號 ID。
- `ConnectionNameList` – 必要：UTF-8 字串的陣列，不可超過 25 個字串。

欲刪除的連線名稱的清單。

回應

- `Succeeded` – UTF-8 字串陣列。

  已成功刪除的連線定義名稱清單。

- `Errors` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 ErrorDetail 物件。

  未成功刪除的連線名稱與錯誤詳細資訊的映射。

錯誤

- `InternalServiceException`
- `OperationTimeoutException`

驗證組態

- AuthenticationConfiguration 結構
- AuthenticationConfigurationInput 結構
- 物業結構
- OAuth2 PropertiesInput 結構
- OAuth2 ClientApplication 結構
- AuthorizationCodeProperties 結構

AuthenticationConfiguration 結構

包含驗證組態的結構。

欄位

- AuthenticationType – UTF-8 字串 (有效值：BASIC | OAUTH2 | CUSTOM)。

  包含驗證組態的結構。
- SecretArn – UTF-8 字串，需符合Custom string pattern #11。

  秘密管理器 ARN 存儲憑據。
- OAuth2Properties – OAuth2Properties 物件。

  OAuth2 驗證的內容。

AuthenticationConfigurationInput 結構

包含 CreateConnection要求中驗證組態的結構。

欄位

- AuthenticationType – UTF-8 字串 (有效值：BASIC | OAUTH2 | CUSTOM)。

  包含 CreateConnection要求中驗證組態的結構。
- SecretArn – UTF-8 字串，需符合Custom string pattern #11。

  秘密管理器 ARN 在 CreateConnection 請求中存儲憑據。
- OAuth2Properties – OAuth2 PropertiesInput 物件。

  要求中 OAuth2 驗證的 CreateConnection 內容。

物業結構

包含 OAuth2 驗證之屬性的結構。

欄位

- OAuth2GrantType – UTF-8 字串 (有效值：AUTHORIZATION_CODE | CLIENT_CREDENTIALS | JWT_BEARER)。

  OAuth2 授權類型。例如，AUTHORIZATION_CODE、JWT_BEARER 或 CLIENT_CREDENTIALS。
- OAuth2ClientApplication – OAuth2 ClientApplication 物件。

用戶端應用程式類型。例如，AW_ 受管或使用者管理。

- TokenUrl – UTF-8 字串，長度不可超過 256 個位元組，且需符合 Custom string pattern #12。

  提供程序的身份驗證服務器的 URL，用於交換訪問令牌的授權代碼。

- TokenUrlParametersMap – 金鑰值對的映射陣列。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是一個 UTF-8 字符串，長度不小於 1 個或大於 512 字節。

  新增至權杖GET要求的參數對映。

OAuth2 PropertiesInput 結構

包含要 CreateConnection求中 OAuth2 屬性的結構。

欄位

- OAuth2GrantType – UTF-8 字串 (有效值：AUTHORIZATION_CODE | CLIENT_CREDENTIALS | JWT_BEARER)。

  要求中的 OAuth2 授與類型。 CreateConnection 例如，AUTHORIZATION_CODE、JWT_BEARER 或 CLIENT_CREDENTIALS。

- OAuth2ClientApplication – OAuth2 ClientApplication 物件。

  CreateConnection 要求中的用戶端應用程式類型。例如 AWS_MANAGED 或 USER_MANAGED。

- TokenUrl – UTF-8 字串，長度不可超過 256 個位元組，且需符合 Custom string pattern #12。

  提供程序的身份驗證服務器的 URL，用於交換訪問令牌的授權代碼。

- TokenUrlParametersMap – 金鑰值對的映射陣列。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是一個 UTF-8 字符串，長度不小於 1 個或大於 512 字節。

  新增至權杖GET要求的參數對映。

- AuthorizationCodeProperties – AuthorizationCode性質 物件。

  OAuth2 AUTHORIZATION_CODE 授權類型所需的屬性集。

OAuth2 ClientApplication 結構

用於連接的 OAuth2 客戶端應用程序。

欄位

- `UserManagedClientApplicationClientId` – UTF-8 字串，長度不可超過 2048 個位元組，且需符合 Custom string pattern #13。

  用戶端應用程式 clientID 如果是 ClientAppType)。`USER_MANAGED`
- `AWSManagedClientApplicationReference` – UTF-8 字串，長度不可超過 2048 個位元組，且需符合 Custom string pattern #13。

  受管理的 SaaS 端用戶端應用程式的參考。 AWS

AuthorizationCodeProperties 結構

OAuth2 `AUTHORIZATION_CODE` 授與類型工作流程所需的屬性集。

欄位

- `AuthorizationCode`— UTF-8 字串，長度不小於 1 或超過 4096 個位元組，符合Custom string pattern #13.

  在授權工作流程的第三階段使用的`AUTHORIZATION_CODE`授權碼。這是一個單次使用的代碼，一旦交換了訪問令牌，就會變為無效，因此將此值作為請求參數是可以接受的。
- `RedirectUri`— UTF-8 字串，長度不超過 512 個位元組，符合Custom string pattern #14.

  發出授權代碼時，用戶被授權服務器重定向到的重定向 URI。當授權代碼交換為訪問令牌時，隨後使用 URI。

使用者定義的函數 API

使用者定義的函數 API 說明用於函數的 AWS Glue 資料類型和操作。

資料類型

- UserDefinedFunction 結構
- UserDefinedFunctionInput 結構

UserDefinedFunction 結構

此結構相當於 Hive 使用者定義函數 (UDF) 定義。

欄位

- FunctionName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  函數的名稱。
- DatabaseName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  包含函數的目錄資料庫名稱。
- ClassName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  包含函數程式碼的 Java 類別。
- OwnerName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  函數的擁有者。
- OwnerType – UTF-8 字串 (有效值：USER | ROLE | GROUP)。

  擁有者類型。
- CreateTime – 時間戳記。

  函數建立的時間。
- ResourceUris – 一個 ResourceUri 物件陣列，不可超過 1000 個結構。

  函數的資源 URI。
- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  函數所在的 Data Catalog 之 ID。

UserDefinedFunctionInput 結構

用於建立或更新使用者定義函數的結構。

欄位

- FunctionName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  函數的名稱。

- ClassName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  包含函數程式碼的 Java 類別。

- OwnerName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  函數的擁有者。

- OwnerType – UTF-8 字串 (有效值：USER | ROLE | GROUP)。

  擁有者類型。

- ResourceUris – 一個 ResourceUri 物件陣列，不可超過 1000 個結構。

  函數的資源 URI。

操作

- CreateUserDefinedFunction 動作 (Python: create_user_defined_function)
- UpdateUserDefinedFunction 動作 (Python: update_user_defined_function)
- DeleteUserDefinedFunction 動作 (Python: delete_user_defined_function)
- GetUserDefinedFunction 動作 (Python: get_user_defined_function)
- GetUserDefinedFunctions 動作 (Python: get_user_defined_functions)

CreateUserDefinedFunction 動作 (Python: create_user_defined_function)

在 Data Catalog 建立新的函數定義。

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

用於建立函數的 Data Catalog 之 ID。如果未提供，預設會使用 AWS 帳戶 ID。

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  用於建立函數的目錄資料庫之名稱。

- `FunctionInput` – 必要：UserDefinedFunctionInput 物件。

  定義在 Data Catalog 中所建立函數的 `FunctionInput` 物件。

回應

- 無回應參數。

錯誤

- `AlreadyExistsException`
- `InvalidInputException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `GlueEncryptionException`

UpdateUserDefinedFunction 動作 (Python: update_user_defined_function)

更新 Data Catalog 中現有的函數定義

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要更新函數所在 Data Catalog 的 ID。如果未提供，預設會使用 AWS 帳戶 ID。

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

欲更新函數所在的目錄資料庫之名稱。

- `FunctionName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  函數的名稱。

- `FunctionInput` – 必要：UserDefinedFunctionInput 物件。

  一個 `FunctionInput` 物件，可在 Data Catalog 內重新定義函數。

回應

- 無回應參數。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeleteUserDefinedFunction 動作 (Python: delete_user_defined_function)

從 Data Catalog 刪除現有的函數定義

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  欲刪除函數所在的 Data Catalog 之 ID。若沒有提供，則根據預設會使用 AWS 帳戶 ID。

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  函數所在的目錄資料庫之名稱。

- FunctionName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欲刪除的函數定義之名稱。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetUserDefinedFunction 動作 (Python: get_user_defined_function)

從 Data Catalog 擷取指定的函數定義

請求

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要擷取函數所在 Data Catalog 的 ID。如果未提供，預設會使用 AWS 帳戶 ID。

- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  函數所在的目錄資料庫之名稱。

- FunctionName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  函數的名稱。

回應

- `UserDefinedFunction` – UserDefinedFunction 物件。

  要求的函數定義。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

GetUserDefinedFunctions 動作 (Python: get_user_defined_functions)

從 Data Catalog 擷取多函數定義。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要擷取函數所在 Data Catalog 的 ID。如果未提供，預設會使用 AWS 帳戶 ID。
- `DatabaseName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  函數所在的目錄資料庫之名稱。如果沒有提供，則會傳回來自目錄中所有資料庫的函數。
- `Pattern` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  一種選用的函數名稱模式字串，可篩選傳回的函數定義。
- `NextToken` – UTF-8 字串。

  接續符記，如果這是接續呼叫。
- `MaxResults` – 數字 (整數)，不可小於 1，也不可以大於 100。

  一次回應傳回函數的最大數量。

回應

- UserDefinedFunctions – 一個 UserDefinedFunction 物件陣列。

  要求的函數定義之清單。
- NextToken – UTF-8 字串。

  接續符記 (如果傳回的函數清單不包括最後要求的函數)。

錯誤

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- GlueEncryptionException

正在將 Athena 目錄匯入 AWS Glue

遷移 API 說明 AWS Glue 資料類型和操作與將 Athena 資料型錄遷移到 AWS Glue 的關係。

資料類型

- CatalogImportStatus 結構

CatalogImportStatus 結構

結構包含遷移狀態資訊。

欄位

- ImportCompleted – 布林值。

  如果遷移完成為 True，否則為 False。
- ImportTime – 時間戳記。

  遷移開始進行的時間。

- `ImportedBy` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  啟動遷移的人員名稱。

操作

- ImportCatalogToGlue 動作 (Python: import_catalog_to_glue)
- GetCatalogImportStatus 動作 (Python: get_catalog_import_status)

ImportCatalogToGlue 動作 (Python: import_catalog_to_glue)

將現有的 Amazon Athena Data Catalog 匯入 AWS Glue。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  欲匯入的目錄的 ID。目前，此項目應為 AWS 帳戶 ID。

回應

- 無回應參數。

錯誤

- `InternalServiceException`
- `OperationTimeoutException`

GetCatalogImportStatus 動作 (Python: get_catalog_import_status)

擷取遷移操作的狀態。

請求

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

欲遷移的目錄的 ID。目前，此項目應為 AWS 帳戶 ID。

回應

- `ImportStatus` – CatalogImportStatus 物件。

  指定的目錄遷移的狀態。

錯誤

- `InternalServiceException`
- `OperationTimeoutException`

資料表最佳化工具 API

資料表最佳化程式 API 描述了啟用壓縮以提升讀取效能的 AWS Glue API。

資料類型

- TableOptimizer 結構
- TableOptimizerConfiguration 結構
- TableOptimizerRun 結構
- RunMetrics 結構
- BatchGetTableOptimizerEntry 結構
- BatchTableOptimizer 結構
- BatchGetTableOptimizerError 結構

TableOptimizer 結構

包含與資料表相關聯之最佳化工具的相關詳細資料。

欄位

- `type` – UTF-8 字串 (有效值：`compaction="COMPACTION"`)。

資料表最佳化工具類型。目前唯一有效的值為：compaction。

- configuration – TableOptimizerConfiguration 物件。

  建立或更新資料表最佳化工具時，會指定的 TableOptimizerConfiguration 物件。

- lastRun – TableOptimizerRun 物件。

  表示上次執行之資料表最佳化工具的 TableOptimizerRun 物件。

TableOptimizerConfiguration 結構

包含資料表最佳化工具組態的相關詳細資料。您會在建立或更新資料表最佳化工具時傳遞此組態。

欄位

- roleArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組，且需符合 Single-line string pattern。

  呼叫者傳遞的角色，可提供服務權限，以代表呼叫者更新與最佳化工具相關聯的資源。

- enabled – 布林值。

  無論資料表最佳化是否已啟用。

TableOptimizerRun 結構

包含資料表最佳化工具執行的詳細資料。

欄位

- eventType – UTF-8 字串 (有效值：starting="STARTING" | completed="COMPLETED" | failed="FAILED" | in_progress="IN_PROGRESS")。

  表示資料表最佳化工具執行狀態的事件類型。

- startTimestamp – 時間戳記。

  表示 Lake Formation 內的壓縮任務開始時的 Epoch 時間戳記。

- endTimestamp – 時間戳記。

  表示壓縮任務結束時的 Epoch 時間戳記。

- `metrics` – RunMetrics 物件。

  包含最佳化工具執行之指標的 `RunMetrics` 物件。
- `error` – UTF-8 字串。

  最佳化工具執行期間發生的錯誤。

RunMetrics 結構

最佳化工具執行的指標。

欄位

- `NumberOfBytesCompacted` – UTF-8 字串。

  壓縮任務執行移除的位元組數。
- `NumberOfFilesCompacted` – UTF-8 字串。

  壓縮任務執行移除檔案數。
- `NumberOfDpus` – UTF-8 字串。

  任務花費的 DPU 小時數。
- `JobDurationInHour` – UTF-8 字串。

  任務的持續時間 (以小時為單位)。

BatchGetTableOptimizerEntry 結構

表示要在 `BatchGetTableOptimizer` 操作中擷取的資料表最佳化工具。

欄位

- `catalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表的目錄 ID。
- `databaseName`：UTF-8 字串，長度至少為 1 個位元組。

  資料表所在目錄的資料庫的名稱。

- tableName：UTF-8 字串，長度至少為 1 個位元組。

  資料表的名稱。

- type – UTF-8 字串 (有效值：compaction="COMPACTION")。

  資料表最佳化工具類型。

BatchTableOptimizer 結構

包含 BatchGetTableOptimizer 操作傳回之其中一個資料表最佳化工具的詳細資料。

欄位

- catalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表的目錄 ID。

- databaseName：UTF-8 字串，長度至少為 1 個位元組。

  資料表所在目錄的資料庫的名稱。

- tableName：UTF-8 字串，長度至少為 1 個位元組。

  資料表的名稱。

- tableOptimizer – TableOptimizer 物件。

  包含組態和上次執行之資料表最佳化工具相關詳細資料的 TableOptimizer 物件。

BatchGetTableOptimizerError 結構

包含 BatchGetTableOptimizer 操作傳回的錯誤清單中之其中一個錯誤的相關詳細資料。

欄位

- error – ErrorDetail 物件。

  包含與錯誤相關之程式碼和訊息詳細資料的 ErrorDetail 物件。

- catalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

資料表的目錄 ID。

- databaseName：UTF-8 字串，長度至少為 1 個位元組。

  資料表所在目錄的資料庫的名稱。

- tableName：UTF-8 字串，長度至少為 1 個位元組。

  資料表的名稱。

- type – UTF-8 字串 (有效值：compaction="COMPACTION")。

  資料表最佳化工具類型。

作業

- GetTableOptimizer 行動（Python：獲取表優化器）
- BatchGetTableOptimizer 行動（Python：批處理表優化器）
- ListTableOptimizerRuns 行動（Python：列表優化器運行）
- CreateTableOptimizer 行動（Python：創建表格優化器）
- DeleteTableOptimizer 行動（Python：刪除表格優化器）
- UpdateTableOptimizer 行動（Python：更新表優化器）

GetTableOptimizer 行動（Python：獲取表優化器）

傳回與指定資料表相關聯之所有最佳化工具的組態。

請求

- CatalogId – 必要：目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合 Single-line string pattern。

  資料表的目錄 ID。

- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表所在目錄的資料庫的名稱。

- TableName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

資料表的名稱。

- `Type` – 必要：UTF-8 字串 (有效值：`compaction="COMPACTION"`)。

  資料表最佳化工具類型。

回應

- `CatalogId` – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表的目錄 ID。
- `DatabaseName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在目錄的資料庫的名稱。
- `TableName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表的名稱。
- `TableOptimizer` – TableOptimizer 物件。

  與指定資料表相關聯的最佳化工具。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `AccessDeniedException`
- `InternalServiceException`

BatchGetTableOptimizer 行動（Python：批處理表優化器）

傳回指定資料表最佳化工具的組態。

請求

- `Entries` – 必要：一個 BatchGetTableOptimizerEntry 物件。

指定要擷取之資料表最佳化工具的 BatchGetTableOptimizerEntry 物件清單。

回應

- TableOptimizers – 一個 BatchTableOptimizer 物件陣列。

  BatchTableOptimizer 物件的清單。
- Failures – 一個 BatchGetTableOptimizerError 物件陣列。

  操作中的錯誤清單。

錯誤

- InternalServiceException

ListTableOptimizerRuns 行動（Python：列表優化器運行）

列出先前針對特定資料表之最佳化工具執行的歷史記錄。

請求

- CatalogId – 必要：目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合 Single-line string pattern。

  資料表的目錄 ID。
- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表所在目錄的資料庫的名稱。
- TableName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表的名稱。
- Type – 必要：UTF-8 字串 (有效值：compaction="COMPACTION")。

  資料表最佳化工具類型。目前唯一有效的值為：compaction。
- MaxResults – 數字 (整數)。

每次呼叫時最佳化工具執行的傳回數上限。

- NextToken – UTF-8 字串。

  接續符記，如果這是接續呼叫。

回應

- CatalogId – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表的目錄 ID。
- DatabaseName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在目錄的資料庫的名稱。
- TableName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表的名稱。
- NextToken – UTF-8 字串。

  用於將傳回的最佳化工具執行清單分頁的持續權杖，如果清單的目前區段不是最後區段則會傳回。
- TableOptimizerRuns – 一個 TableOptimizerRun 物件陣列。

  與資料表相關聯的最佳化工具執行清單。

錯誤

- EntityNotFoundException
- AccessDeniedException
- InvalidInputException
- InternalServiceException

CreateTableOptimizer 行動（Python：創建表格優化器）

針對特定函數建立新的資料表。compaction 為目前唯一支援的最佳化工具類型。

請求

- CatalogId – 必要：目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合 Single-line string pattern。

  資料表的目錄 ID。

- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表所在目錄的資料庫的名稱。

- TableName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表的名稱。

- Type – 必要：UTF-8 字串 (有效值：compaction="COMPACTION")。

  資料表最佳化工具類型。目前唯一有效的值為：compaction。

- TableOptimizerConfiguration – 必要：TableOptimizerConfiguration 物件。

  表示資料表最佳化工具組態的 TableOptimizerConfiguration 物件。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- InvalidInputException
- AccessDeniedException
- AlreadyExistsException
- InternalServiceException

DeleteTableOptimizer 行動（Python：刪除表格優化器）

針對資料表刪除最佳化工具和所有相關的中繼資料。系統將不會再針對該資料表執行最佳化。

請求

- `CatalogId` – 必要：目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合 Single-line string pattern。

  資料表的目錄 ID。

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表所在目錄的資料庫的名稱。

- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表的名稱。

- `Type` – 必要：UTF-8 字串 (有效值：`compaction="COMPACTION"`)。

  資料表最佳化工具類型。

回應

- 無回應參數。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `AccessDeniedException`
- `InternalServiceException`

UpdateTableOptimizer 行動（Python：更新表優化器）

針對現有的資料表最佳化工具更新組態。

請求

- `CatalogId` – 必要：目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合 Single-line string pattern。

資料表的目錄 ID。

- DatabaseName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表所在目錄的資料庫的名稱。

- TableName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表的名稱。

- Type – 必要：UTF-8 字串 (有效值：compaction="COMPACTION")。

  資料表最佳化工具類型。目前唯一有效的值為：compaction。

- TableOptimizerConfiguration – 必要：TableOptimizerConfiguration 物件。

  表示資料表最佳化工具組態的 TableOptimizerConfiguration 物件。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- InvalidInputException
- AccessDeniedException
- InternalServiceException

爬蟲程式和分類器 API

爬蟲程式和分類器 API 說明 AWS Glue 爬蟲程式與分類器資料類型，以及用於建立、刪除、更新和列出爬蟲程式或分類器的 API。

主題

- 分類器 API
- 爬蟲程式 API

- 資料欄統計資料 API
- 爬蟲程式排程器 API

分類器 API

Classifier API 說明 AWS Glue 分類器資料類型，以及用於建立、刪除、更新和列出分類器的 API。

資料類型

- Classifier 結構
- GrokClassifier 結構
- XMLClassifier 結構
- JsonClassifier 結構
- CsvClassifier 結構
- CreateGrokClassifierRequest 結構
- UpdateGrokClassifierRequest 結構
- CreateXMLClassifierRequest 結構
- UpdateXMLClassifierRequest 結構
- CreateJsonClassifierRequest 結構
- UpdateJsonClassifierRequest 結構
- CreateCsvClassifierRequest 結構
- UpdateCsvClassifierRequest 結構

Classifier 結構

探索任務會觸發分類器。分類器會檢查指定的檔案是否採用其能處理的格式。如果是，則分類器會以符合該資料格式的 `StructType` 物件形式，建立結構描述。

您可以使用 AWS Glue 提供的標準分類器，或者您可以編寫自己的分類器，以最佳的方式分類資料來源，並指定適當的結構描述來使用。分類器可以是 `grok` 分類器、`XML` 分類器或 `JSON` 分類器，或者透過 `Classifier` 物件中其中一個欄位指定的自訂 `CSV` 分類器。

欄位

- `GrokClassifier` – GrokClassifier 物件。

使用 grok 的分類器。

- XMLClassifier – XMLClassifier 物件。

  XML 內容的分類器。

- JsonClassifier – JsonClassifier 物件。

  JSON 內容的分類器。

- CsvClassifier – CsvClassifier 物件。

  逗點分隔值 (CSV) 的分類器。

GrokClassifier 結構

使用 grok 模式的分類器。

欄位

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分類器名稱。

- Classification – 必要：UTF-8 字串。

  分類器符合的資料格式識別符，例如 Twitter、JSON、Omniture 日誌等。

- CreationTime – 時間戳記。

  此分類器登錄時的時間。

- LastUpdated – 時間戳記。

  此分類器的上次更新時間。

- Version – 數字 (long)。

  此分類器的版本。

- GrokPattern – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 2048 個位元組，且需符合A Logstash Grok string pattern。

  此分類器套用到資料存放區的 grok 模式。如需詳細資訊，請參閱撰寫自訂分類器中的內建模式。

- `CustomPatterns` – UTF-8 字串，長度不可超過 16000 個位元組，需符合URI address multi-line string pattern。

  此分類器定義的選用自訂 grok 模式。如需詳細資訊，請參閱撰寫自訂分類器中的自訂模式。

XMLClassifier 結構

XML 內容的分類器。

欄位

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分類器名稱。

- `Classification` – 必要：UTF-8 字串。

  分類器符合的資料格式識別碼。

- `CreationTime` – 時間戳記。

  此分類器登錄時的時間。

- `LastUpdated` – 時間戳記。

  此分類器的上次更新時間。

- `Version` – 數字 (long)。

  此分類器的版本。

- `RowTag` – UTF-8 字串。

  XML 標籤，指定包含所剖析之 XML 文件中各記錄的元素。這個設定無法識別自我關閉的元素 (由 /> 關閉)。僅包含屬性的空白資料列元素若結尾為關閉標籤便能剖析 (例如，可以是 `<row item_a="A" item_b="B"></row>`，但不能是 `<row item_a="A" item_b="B" />`)。

JsonClassifier 結構

JSON 內容的分類器。

欄位

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分類器名稱。

- CreationTime – 時間戳記。

  此分類器登錄時的時間。

- LastUpdated – 時間戳記。

  此分類器的上次更新時間。

- Version – 數字 (long)。

  此分類器的版本。

- JsonPath – 必要：UTF-8 字串。

  為要分類的分類器定義 JSON 資料的 JsonPath 字串。AWS Glue 支援 JsonPath 的運算子子集，如撰寫 JsonPath 自訂分類器中所述。

CsvClassifier 結構

自訂 CSV 內容的分類器。

欄位

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分類器名稱。

- CreationTime – 時間戳記。

  此分類器登錄時的時間。

- LastUpdated – 時間戳記。

  此分類器的上次更新時間。

- Version – 數字 (long)。

  此分類器的版本。

- `Delimiter` - UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 1 個位元組，且需符合 Custom string pattern #10。

  表示用於分隔資料列中每個欄位項目的自訂符號。

- `QuoteSymbol` - UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 1 個位元組，且需符合 Custom string pattern #10。

  用來表示將內容結合成單一欄位值的自訂符號。其必須不同於欄位分隔符號。

- `ContainsHeader` – UTF-8 字串 (有效值：`UNKNOWN` | `PRESENT` | `ABSENT`)。

  表示 CSV 檔案是否包含標頭。

- `Header` – UTF-8 字串陣列。

  表示欄位名稱之字串的清單。

- `DisableValueTrimming` – 布林值。

  指定在確認欄位值類型之前不要裁剪值。預設值為 `true`。

- `AllowSingleColumn` – 布林值。

  啟用處理僅包含一個欄位的檔案。

- `CustomDatatypeConfigured` – 布林值。

  啟用要設定的自訂資料類型。

- `CustomDatatypes` – UTF-8 字串陣列。

  指定資料類型清單包括
  "BINARY"、"BOOLEAN"、"DATE"、"DECIMAL"、"DOUBLE"、"FLOAT"、"INT"、"LONG"、"SHORT"、"S

- `Serde` – UTF-8 字串 (有效值：`OpenCSVSerDe` | `LazySimpleSerDe` | `None`)。

  設定用於在分類器中處理 CSV 的 SerDe，並且將在資料型錄中套用該 Serde。有效值為 `OpenCSVSerDe`、`LazySimpleSerDe` 和 `None`。您可以指定希望爬蟲程式執行偵測時的 `None` 值。

CreateGrokClassifierRequest 結構

指定 `CreateClassifier` 要建立的 `grok` 分類器。

欄位

- `Classification` – 必要：UTF-8 字串。

  分類器符合的資料格式識別符，例如 Twitter、JSON、Omniture 日誌、Amazon CloudWatch Logs 等。
- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  新分類器名稱。
- `GrokPattern` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 2048 個位元組，且需符合A Logstash Grok string pattern。

  此分類器使用的 grok 模式。
- `CustomPatterns` – UTF-8 字串，長度不可超過 16000 個位元組，需符合URI address multi-line string pattern。

  此分類器使用的選用自訂 grok 模式。

UpdateGrokClassifierRequest 結構

指定當傳送到 `UpdateClassifier` 時要更新的 grok 分類器。

欄位

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  `GrokClassifier` 的名稱。
- `Classification` – UTF-8 字串。

  分類器符合的資料格式識別符，例如 Twitter、JSON、Omniture 日誌、Amazon CloudWatch Logs 等。
- `GrokPattern` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 2048 個位元組，需符合A Logstash Grok string pattern。

  此分類器使用的 grok 模式。
- `CustomPatterns` – UTF-8 字串，長度不可超過 16000 個位元組，需符合URI address multi-line string pattern。

此分類器使用的選用自訂 grok 模式。

CreateXMLClassifierRequest 結構

指定 `CreateClassifier` 要建立的 XML 分類器。

欄位

- `Classification` – 必要：UTF-8 字串。

  分類器符合的資料格式識別碼。

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分類器名稱。

- `RowTag` – UTF-8 字串。

  XML 標籤，指定包含所剖析之 XML 文件中各記錄的元素。這個設定無法識別自我關閉的元素 (由 `/>` 關閉)。僅包含屬性的空白資料列元素若結尾為關閉標籤便能剖析 (例如，可以是 `<row item_a="A" item_b="B"></row>`，但不能是 `<row item_a="A" item_b="B" />`)。

UpdateXMLClassifierRequest 結構

指定要更新的 XML 分類器。

欄位

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分類器名稱。

- `Classification` – UTF-8 字串。

  分類器符合的資料格式識別碼。

- `RowTag` – UTF-8 字串。

XML 標籤，指定包含所剖析之 XML 文件中各記錄的元素。這個設定無法識別自我關閉的元素 (由 /> 關閉)。僅包含屬性的空白資料列元素若結尾為關閉標籤便能剖析 (例如，可以是 `<row item_a="A" item_b="B"></row>`，但不能是 `<row item_a="A" item_b="B" />`)。

CreateJsonClassifierRequest 結構

指定 `CreateClassifier` 要建立的 JSON 分類器。

欄位

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分類器名稱。

- `JsonPath` – 必要：UTF-8 字串。

  為要分類的分類器定義 JSON 資料的 `JsonPath` 字串。AWS Glue 支援 JsonPath 的運算子子集，如撰寫 JsonPath 自訂分類器中所述。

UpdateJsonClassifierRequest 結構

指定要更新的 JSON 分類器。

欄位

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分類器名稱。

- `JsonPath` – UTF-8 字串。

  為要分類的分類器定義 JSON 資料的 `JsonPath` 字串。AWS Glue 支援 JsonPath 的運算子子集，如撰寫 JsonPath 自訂分類器中所述。

CreateCsvClassifierRequest 結構

指定 `CreateClassifier` 要建立的自訂 CSV 分類器。

欄位

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分類器名稱。

- Delimiter - UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 1 個位元組，且需符合 Custom string pattern #10。

  表示用於分隔資料列中每個欄位項目的自訂符號。

- QuoteSymbol - UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 1 個位元組，且需符合 Custom string pattern #10。

  用來表示將內容結合成單一欄位值的自訂符號。必須不同於欄位分隔符號。

- ContainsHeader – UTF-8 字串 (有效值：UNKNOWN | PRESENT | ABSENT)。

  表示 CSV 檔案是否包含標頭。

- Header – UTF-8 字串陣列。

  表示欄位名稱之字串的清單。

- DisableValueTrimming – 布林值。

  指定在確認欄位值類型之前不要裁剪值。預設值為 true。

- AllowSingleColumn – 布林值。

  啟用處理僅包含一個欄位的檔案。

- CustomDatatypeConfigured – 布林值。

  啟用自訂資料類型的組態。

- CustomDatatypes – UTF-8 字串陣列。

  建立支援的自訂資料類型清單。

- Serde – UTF-8 字串 (有效值：OpenCSVSerDe | LazySimpleSerDe | None)。

  設定用於在分類器中處理 CSV 的 SerDe，並且將在資料型錄中套用該 Serde。有效值為 OpenCSVSerDe、LazySimpleSerDe 和 None。您可以指定希望爬蟲程式執行偵測時的 None 值。

UpdateCsvClassifierRequest 結構

指定要更新的自訂 CSV 分類器。

欄位

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  分類器名稱。

- Delimiter - UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 1 個位元組，且需符合 Custom string pattern #10。

  表示用於分隔資料列中每個欄位項目的自訂符號。

- QuoteSymbol - UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 1 個位元組，且需符合 Custom string pattern #10。

  用來表示將內容結合成單一欄位值的自訂符號。其必須不同於欄位分隔符號。

- ContainsHeader – UTF-8 字串 (有效值：UNKNOWN | PRESENT | ABSENT)。

  表示 CSV 檔案是否包含標頭。

- Header – UTF-8 字串陣列。

  表示欄位名稱之字串的清單。

- DisableValueTrimming – 布林值。

  指定在確認欄位值類型之前不要裁剪值。預設值為 true。

- AllowSingleColumn – 布林值。

  啟用處理僅包含一個欄位的檔案。

- CustomDatatypeConfigured – 布林值。

  指定自訂資料類型的組態。

- CustomDatatypes – UTF-8 字串陣列。

  指定支援的自訂資料類型清單。

- Serde – UTF-8 字串 (有效值：OpenCSVSerDe | LazySimpleSerDe | None)。

設定用於在分類器中處理 CSV 的 SerDe，並且將在資料型錄中套用該 Serde。有效值為 `OpenCSVSerDe`、`LazySimpleSerDe` 和 `None`。您可以指定希望爬蟲程式執行偵測時的 `None` 值。

操作

- CreateClassifier 動作 (Python: create_classifier)
- DeleteClassifier 動作 (Python: delete_classifier)
- GetClassifier 動作 (Python: get_classifier)
- GetClassifiers 動作 (Python: get_classifiers)
- UpdateClassifier 動作 (Python: update_classifier)

CreateClassifier 動作 (Python: create_classifier)

在使用者的帳戶內建立分類器。可能是 `GrokClassifier`、`XMLClassifier`、`JsonClassifier`，或 `CsvClassifier`，視出現的要求欄位而定。

請求

- `GrokClassifier` – CreateGrokClassifierRequest 物件。

  指定要建立之分類器的 `GrokClassifier` 物件。
- `XMLClassifier` – CreateXMLClassifierRequest 物件。

  指定要建立之分類器的 `XMLClassifier` 物件。
- `JsonClassifier` – CreateJsonClassifierRequest 物件。

  指定要建立之分類器的 `JsonClassifier` 物件。
- `CsvClassifier` – CreateCsvClassifierRequest 物件。

  指定要建立之分類器的 `CsvClassifier` 物件。

回應

- 無回應參數。

錯誤

- `AlreadyExistsException`
- `InvalidInputException`
- `OperationTimeoutException`

DeleteClassifier 動作 (Python: delete_classifier)

從 Data Catalog 移除分類器。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要移除的分類器名稱。

回應

- 無回應參數。

錯誤

- `EntityNotFoundException`
- `OperationTimeoutException`

GetClassifier 動作 (Python: get_classifier)

擷取指定名稱的分類器。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要擷取的分類器名稱。

回應

- `Classifier` – 分類器 物件。

  要求的分類器。

錯誤

- `EntityNotFoundException`
- `OperationTimeoutException`

GetClassifiers 動作 (Python: get_classifiers)

列出 Data Catalog 中所有的分類器物件。

請求

- `MaxResults` – 數字 (整數)，不可小於 1，也不可以大於 1000。

  所要回傳的清單大小 (選用)。
- `NextToken` – UTF-8 字串。

  選擇性的接續符記。

回應

- `Classifiers` – 一個 分類器 物件陣列。

  要求的分類器物件清單。
- `NextToken` – UTF-8 字串。

  接續符記。

錯誤

- `OperationTimeoutException`

UpdateClassifier 動作 (Python: update_classifier)

修改現有的分類器 (GrokClassifier、XMLClassifier、JsonClassifier 或 CsvClassifier，視出現的欄位而定)。

請求

- GrokClassifier – UpdateGrokClassifierRequest 物件。

  含更新欄位的 GrokClassifier 物件。
- XMLClassifier – UpdateXMLClassifierRequest 物件。

  含更新欄位的 XMLClassifier 物件。
- JsonClassifier – UpdateJsonClassifierRequest 物件。

  含更新欄位的 JsonClassifier 物件。
- CsvClassifier – UpdateCsvClassifierRequest 物件。

  含更新欄位的 CsvClassifier 物件。

回應

- 無回應參數。

錯誤

- InvalidInputException
- VersionMismatchException
- EntityNotFoundException
- OperationTimeoutException

爬蟲程式 API

爬行者程式 API 會說明 AWS Glue 爬行者程式資料類型，以及用來建立、刪除、更新和列出檢索器的 API。

資料類型

- Crawler 結構
- Schedule 結構
- CrawlerTargets 結構
- S3Target 結構
- S3 DeltaCatalogTarget 結構
- S3 DeltaDirectTarget 結構
- JdbcTarget 結構
- MongoDBTarget 結構
- DynamoDBTarget 結構
- DeltaTarget 結構
- IcebergTarget 結構
- HudiTarget 結構
- CatalogTarget 結構
- CrawlerMetrics 結構
- CrawlerHistory 結構
- CrawlsFilter 結構
- SchemaChangePolicy 結構
- LastCrawlInfo 結構
- RecrawlPolicy 結構
- LineageConfiguration 結構
- LakeFormationConfiguration 結構

Crawler 結構

指定爬蟲程式，以檢驗資料來源並使用分類器嘗試判斷其結構描述。如果成功，爬蟲程式會將與資料來源有關的中繼資料記錄到 AWS Glue Data Catalog。

欄位

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

爬蟲程式的名稱。

- `Role` – UTF-8 字串。

  用來存取 Amazon Simple Storage Service (Amazon S3) 資料等客戶資源 IAM 角色的 Amazon Resource Name (ARN)。

- `Targets` – CrawlerTargets 物件。

  待編目的目標集合。

- `DatabaseName` – UTF-8 字串。

  爬蟲程式輸出存放所在的資料庫名稱。

- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  爬蟲程式的描述。

- `Classifiers` – UTF-8 字串陣列。

  UTF-8 字串的清單，可藉由這些字串指定要與爬蟲程式建立關聯的自訂分類器。

- `RecrawlPolicy` – RecrawlPolicy 物件。

  一種政策，指定是否要再次網路爬取整個資料集，或只網路爬取自上次執行爬蟲程式後新增的資料夾。

- `SchemaChangePolicy` – SchemaChangePolicy 物件。

  指定爬蟲程式更新及刪除行為的政策。

- `LineageConfiguration` – LineageConfiguration 物件。

  指定是否為爬蟲程式啟用資料歷程的組態。

- `State` – UTF-8 字串 (有效值：`READY` | `RUNNING` | `STOPPING`)。

  指出爬蟲程式是否正在執行，或是否正在等待執行。

- `TablePrefix` – UTF-8 字串，長度不可超過 128 個位元組。

  新增到所建立資料表名稱之前的字首。

- `Schedule` – 排程 物件。

  如為排程的爬蟲程式，也就是爬蟲程式執行的排程。

- `CrawlElapsedTime` – 數字 (long)。

  如果爬蟲程式正在執行，包含爬蟲程式上次啟動後經過的總時間。

- `CreationTime` – 時間戳記。

  爬蟲程式建立的時間。

- `LastUpdated` – 時間戳記。

  爬蟲程式上次更新的時間。

- `LastCrawl` – LastCrawlInfo 物件。

  最後一次編目的狀態，以及發生錯誤時的可能錯誤資訊。

- `Version` – 數字 (long)。

  爬蟲程式的版本。

- `Configuration` – UTF-8 字串。

  爬蟲程式組態資訊。此版本的 JSON 字串可讓使用者指定爬蟲程式的各種行為。如需詳細資訊，請參閱設定爬蟲程式組態選項。

- `CrawlerSecurityConfiguration` – UTF-8 字串，長度不可超過 128 個位元組。

  此爬蟲程式要使用的 `SecurityConfiguration` 結構。

- `LakeFormationConfiguration` – LakeFormationConfiguration 物件。

  指定爬行者程式是否應使用爬行者程式的 AWS Lake Formation 認證，而不是 IAM 角色登入資料。

Schedule 結構

排程物件，使用 `cron` 陳述式來將事件排程。

欄位

- `ScheduleExpression` – UTF-8 字串。

  用來指定排程的 `cron` 表達式 (請參閱適用於任務與爬蟲程式的依時排程。例如，如果要每天在 12:15 UTC 執行某項動作，您可以指定：`cron(15 12 * * ? *)`。

- `State` – UTF-8 字串 (有效值：`SCHEDULED` | `NOT_SCHEDULED` | `TRANSITIONING`)。

  排程的狀態。

CrawlerTargets 結構

指定要編目的資料存放區。

欄位

- `S3Targets` – 一個 S3Target 物件陣列。

  指定 Amazon Simple Storage Service (Amazon S3) 的目標。
- `JdbcTargets` – 一個 JdbcTarget 物件陣列。

  指定 JDBC 目標。
- `MongoDBTargets` – 一個 MongoDBTarget 物件陣列。

  指定 Amazon DocumentDB 或 MongoDB 目標。
- `DynamoDBTargets` – 一個 DynamoDBTarget 物件陣列。

  指定 Amazon DynamoDB 的目標。
- `CatalogTargets` – 一個 CatalogTarget 物件陣列。

  指定 AWS Glue Data Catalog 目標。
- `DeltaTargets` – 一個 DeltaTarget 物件陣列。

  指定 Delta 資料存放區目標。
- `IcebergTargets` – 一個 IcebergTarget 物件陣列。

  指定 Apache Iceberg 資料存放區目標。
- `HudiTargets` – 一個 HudiTarget 物件陣列。

  指定 Apache Hudi 資料存放區目標。

S3Target 結構

指定 Amazon Simple Storage Service (Amazon S3) 中資料存放區。

欄位

- `Path` – UTF-8 字串。

  至 Amazon S3 目標的路徑。

- `Exclusions` – UTF-8 字串陣列。

  用於排除於編目的 glob 模式清單。如需詳細資訊，請參閱使用爬蟲程式建立資料表目錄。

- `ConnectionName` – UTF-8 字串。

  連線的名稱，允許任務或爬蟲程式在 Amazon Virtual Private Cloud 環境 (Amazon VPC) 內存取 Amazon S3 中的資料。

- `SampleSize` – 數字 (整數)。

  設定在資料集中網路爬取範例檔案時，每個分葉資料夾中要編目的檔案數目。如果未設定，則會網路爬取所有檔案。有效值是介於 1 到 249 之間的整數。

- `EventQueueArn` – UTF-8 字串。

  有效的 Amazon SQS ARN。例如 `arn:aws:sqs:region:account:sqs`。

- `DlqEventQueueArn` – UTF-8 字串。

  有效的 Amazon 無法投遞 SQS ARN。例如 `arn:aws:sqs:region:account:deadLetterQueue`。

S3 DeltaCatalogTarget 結構

指定寫入「 AWS Glue 資料目錄」中 Delta Lake 資料來源的目標。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。

- `Inputs` – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。

- `PartitionKeys` – UTF-8 字串陣列。

  指定使用一系列索引鍵的原生分割。

- `Table` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫中資料表的名稱。

- `Database` – 必要：UTF-8 字串，需符合Custom string pattern #40。

要寫入之資料庫的名稱。

- `AdditionalOptions` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  指定連接器的其他連接選項。

- `SchemaChangePolicy` – CatalogSchemaChangePolicy 物件。

  可以針對爬蟲程式指定更新行為的政策。

S3 DeltaDirectTarget 結構

在中指定寫入 Delta 湖資料來源的目標 Amazon S3。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。

- `Inputs` – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。

- `PartitionKeys` – UTF-8 字串陣列。

  指定使用一系列索引鍵的原生分割。

- `Path` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入 Delta Lake 資料來源的 Amazon S3 路徑。

- `Compression` – 必要：UTF-8 字串 (有效值：`uncompressed="UNCOMPRESSED"` | `snappy="SNAPPY"`)。

  指定資料的壓縮方式。一般來說，如果資料具有標準副檔名，則不需要此項目。可能值為 `"gzip"` 和 `"bzip"`。

- `Format` – 必要：UTF-8 字串 (有效值：`json="JSON"` | `csv="CSV"` | `avro="AVRO"` | `orc="ORC"` | `parquet="PARQUET"` | `hudi="HUDI"` | `delta="DELTA"`)。

指定目標的資料輸出格式。

- `AdditionalOptions` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  指定連接器的其他連接選項。

- `SchemaChangePolicy` – DirectSchemaChangePolicy 物件。

  可以針對爬蟲程式指定更新行為的政策。

JdbcTarget 結構

指定要編目的 JDBC 資料存放區。

欄位

- `ConnectionName` – UTF-8 字串。

  用來連接到 JDBC 目標的連線名稱。

- `Path` – UTF-8 字串。

  JDBC 目標的路徑。

- `Exclusions` – UTF-8 字串陣列。

  用於排除於編目的 glob 模式清單。如需詳細資訊，請參閱使用爬蟲程式建立資料表目錄。

- `EnableAdditionalMetadata` – UTF-8 字串陣列。

  指定值 `RAWTYPES` 或 `COMMENTS`，以在表格回應中啟用其他中繼資料。`RAWTYPES` 提供本機層級的資料類型。`COMMENTS` 提供與資料庫中的資料欄或資料表關聯的註解。

  若您不需要其他中繼資料，請讓欄位保持空白。

MongoDBTarget 結構

指定要網路爬取的 Amazon DocumentDB 資料庫或 MongoDB 資料存放區。

欄位

- ConnectionName – UTF-8 字串。

  用來連接到 Amazon DocumentDB 或 MongoDB 目標的連線名稱。

- Path – UTF-8 字串。

  Amazon DocumentDB 或 MongoDB 目標 (資料庫/集合) 的路徑。

- ScanAll – 布林值。

  指出是否掃描所有記錄，還是從資料表中取樣資料列。當資料表不是高傳輸量資料表時，掃描所有記錄可能需要很長的時間。

  一個 true 值代表會掃描所有記錄，而一個 false 值代表會取樣記錄。如果未指定任何值，則預設值為 true。

DynamoDBTarget 結構

指定要抓取的 Amazon DynamoDB 資料表。

欄位

- Path – UTF-8 字串。

  所要抓取 DynamoDB 資料表的名稱。

- scanAll – 布林值。

  指出是否掃描所有記錄，還是從資料表中取樣資料列。當資料表不是高傳輸量資料表時，掃描所有記錄可能需要很長的時間。

  一個 true 值代表會掃描所有記錄，而一個 false 值代表會取樣記錄。如果未指定任何值，則預設值為 true。

- scanRate – 數字 (雙位數)。

  AWS Glue 爬行者程式所要使用的已設定讀取容量單位百分比。讀取容量單位是 DynamoDB 定義的術語，此數值可作為每秒可在該資料表上執行的讀取次數速率限制符號。

  有效值為 null 值或介於 0.1 到 1.5 之間的值。當使用者未提供值，且預設為已設定讀取容量單位的 0.5 (針對已佈建的資料表)，或最大設定讀取容量單位的 0.25 (針對使用隨需模式的資料表) 時，系統會使用 Null 值。

DeltaTarget 結構

指定用於網路爬取一個或多個 Delta 資料表的 Delta 資料存放區。

欄位

- `DeltaTables` – UTF-8 字串陣列。

  Delta 資料表的 Amazon S3 路徑清單。

- `ConnectionName` – UTF-8 字串。

  用來連接到 Delta 資料表目標的連線名稱。

- `WriteManifest` – 布林值。

  指定是否將資訊清單檔案寫入 Delta 資料表路徑。

- `CreateNativeDeltaTable` – 布林值。

  指定爬蟲程式是否要建立原生資料表，以便與支援直接查詢 Delta 交易記錄日誌的查詢引擎整合。

IcebergTarget 結構

指定 Apache Iceberg 資料來源，其中 Iceberg 資料表存放在 Amazon S3中。

欄位

- `Paths` – UTF-8 字串陣列。

  包含 Iceberg 中繼資料資料夾的一個或多個 Amazon S3 路徑。`s3://bucket/prefix`

- `ConnectionName` – UTF-8 字串。

  用來連線到 Iceberg 目標的連線名稱。

- `Exclusions` – UTF-8 字串陣列。

  用於排除於編目的 glob 模式清單。如需詳細資訊，請參閱使用爬蟲程式建立資料表目錄。

- `MaximumTraversalDepth` – 數字 (整數)。

  爬蟲可以遍歷以發現 Amazon S3 路徑中的 Iceberg 元數據文件夾的最大路徑深度。 Amazon S3 用來限制爬蟲程式執行時間。

HudiTarget 結構

指定 Apache Hudi 資料來源。

欄位

- `Paths` – UTF-8 字串陣列。

  Hudi 的 Amazon S3 位置字串陣列，每個位置字串都指出 Hudi 資料表中繼資料檔案所在的根資料夾。Hudi 資料夾可能位於根資料夾的子資料夾中。

  爬蟲程式將掃描路徑下所有資料夾中的 Hudi 資料夾。

- `ConnectionName` – UTF-8 字串。

  用來連線到 Hudi 目標的連線名稱。如果您的 Hudi 檔案存放在需要 VPC 授權的儲存貯體中，則可以在此處設定其連線屬性。

- `Exclusions` – UTF-8 字串陣列。

  用於排除於編目的 glob 模式清單。如需詳細資訊，請參閱使用爬蟲程式建立資料表目錄。

- `MaximumTraversalDepth` – 數字 (整數)。

  爬蟲可以遍歷以探索 Amazon S3 路徑中 Hudi 元數據文件夾的最大路徑深度。 Amazon S3 用來限制爬蟲程式執行時間。

CatalogTarget 結構

指定一個 AWS Glue Data Catalog 目標。

欄位

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要同步的資料庫名稱。

- `Tables` – 必要：UTF-8 字串的陣列，至少要有 1 個字串。

  要同步的資料表清單。

- `ConnectionName` – UTF-8 字串。

配對使用 Catalog 連接類型與 NETWORK 連接類型時，Amazon S3 支援的資料目錄資料表的連接名稱將作為編目的目標。

- EventQueueArn – UTF-8 字串。

  有效的 Amazon SQS ARN。例如 arn:aws:sqs:region:account:sqs。

- DlqEventQueueArn – UTF-8 字串。

  有效的 Amazon 無法投遞 SQS ARN。例如 arn:aws:sqs:region:account:deadLetterQueue。

CrawlerMetrics 結構

指定爬蟲程式的指標。

欄位

- CrawlerName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  爬蟲程式的名稱。

- TimeLeftSeconds – 數字 (雙精度浮點數)，不可大於 None (無)。

  完成執行中爬蟲程式的預估剩餘時間。

- StillEstimating – 布林值。

  假如爬蟲程式仍在預估完成執行所需要的時間，將顯示 True。

- LastRuntimeSeconds – 數字 (雙精度浮點數)，不可大於 None (無)。

  爬蟲程式最近一次執行的持續時間 (以秒為單位)。

- MedianRuntimeSeconds – 數字 (雙精度浮點數)，不可大於 None (無)。

  此爬蟲程式執行的中位數持續時間 (以秒為單位)。

- TablesCreated – 數字 (整數)，不可大於 None (無)。

  此爬蟲程式建立的資料表數量。

- TablesUpdated – 數字 (整數)，不可大於 None (無)。

  此爬蟲程式更新的資料表數量。

- TablesDeleted – 數字 (整數)，不可大於 None (無)。

  此爬蟲程式刪除的資料表數量。

CrawlerHistory 結構

包含爬蟲程式的執行資訊。

欄位

- CrawlId – UTF-8 字串。

  每次網路爬取的 UUID 識別碼。

- State – UTF-8 字串 (有效值：RUNNING | COMPLETED | FAILED | STOPPED)。

  網路爬取的狀態。

- StartTime – 時間戳記。

  開始編目的日期和時間。

- EndTime – 時間戳記。

  網路爬取結束的日期和時間。

- Summary – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  JSON 中特定網路爬取的執行摘要。包含已新增、更新或刪除的目錄資料表和分割區。

- ErrorMessage – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  如果發生錯誤，則為與此網路爬取相關聯的錯誤訊息。

- LogGroup – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組，且需符合 Log group string pattern。

  與編目相關聯的日誌群組。

- LogStream – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組，且需符合 Log-stream string pattern。

  與編目相關聯的日誌串流。

- MessagePrefix – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  有關此編目之 CloudWatch 訊息的前置詞。

- DPUHour – 數字 (雙精度浮點數)，不可大於 None (無)。

  網路爬取所使用的資料處理單位 (DPU) 的數目 (以小時為單位)。

CrawlsFilter 結構

欄位、比較運算子和值的清單，您可以用來篩選指定爬蟲程式的爬蟲程式執行。

欄位

- FieldName – UTF-8 字串 (有效值：CRAWL_ID | STATE | START_TIME | END_TIME | DPU_HOUR)。

  用來篩選特定爬蟲程式的爬蟲程式執行的索引鍵。每個欄位名稱的有效值為：

  - CRAWL_ID：代表網路爬取之 UUID 識別碼的字串。
  - STATE：代表網路爬取狀態的字串。
  - START_TIME 和 END_TIME：時間戳記，以毫秒為單位。
  - DPU_HOUR：用於網路爬取的資料處理單位 (DPU) 小時數。

- FilterOperator – UTF-8 字串 (有效值：GT | GE | LT | LE | EQ | NE)。

  對值進行操作的已定義比較程式。可用的運算子包括：

  - GT：大於。
  - GE：大於或等於。
  - LT：小於。
  - LE：小於或等於。
  - EQ：等於。
  - NE：不等於。

- FieldValue – UTF-8 字串。

  在網路爬取欄位上提供用於比較的值。

SchemaChangePolicy 結構

可以針對爬蟲程式指定更新和刪除行為的政策。

欄位

- `UpdateBehavior` – UTF-8 字串 (有效值：`LOG` | `UPDATE_IN_DATABASE`)。

  爬蟲程式找到變更結構描述時的更新行為。
- `DeleteBehavior` – UTF-8 字串 (有效值：`LOG` | `DELETE_FROM_DATABASE` | `DEPRECATE_IN_DATABASE`)。

  爬蟲程式找到刪除物件時的刪除行為。

LastCrawlInfo 結構

關於最近一次編目的狀態和錯誤探索。

欄位

- `Status` – UTF-8 字串 (有效值：`SUCCEEDED` | `CANCELLED` | `FAILED`)。

  最近一次編目的狀態。
- `ErrorMessage` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  如果發生錯誤，則為最後一次編目的錯誤資訊。
- `LogGroup` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組，且需符合 Log group string pattern。

  最後一次編目的日誌群組。
- `LogStream` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組，且需符合 Log-stream string pattern。

  最後一次編目的日誌串流。
- `MessagePrefix` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  此爬蟲程式相關訊息的字首。

- `StartTime` – 時間戳記。

  爬蟲程式開始的時間。

RecrawlPolicy 結構

在第一次網路爬取完成後網路爬取 Amazon S3 資料來源時，指定是要再次網路爬取整個資料集，還是只網路爬取自上次爬蟲程式執行以來新增的資料夾。如需詳細資訊，請參閱開發人員指南中的 AWS Glue中的增量網路爬取。

欄位

- `RecrawlBehavior` – UTF-8 字串 (有效值：`CRAWL_EVERYTHING` | `CRAWL_NEW_FOLDERS_ONLY` | `CRAWL_EVENT_MODE`)。

  指定是否要再次網路爬取整個資料集，或只網路爬取自上次執行爬蟲程式後新增的資料夾。

  值為 `CRAWL_EVERYTHING` 指定再次網路爬取整個資料集。

  值為 `CRAWL_NEW_FOLDERS_ONLY` 指定只網路爬取自上次執行爬蟲程式之後，已新增的資料夾。

  `CRAWL_EVENT_MODE` 值會指定只網路爬取 Amazon S3 事件所識別的變更。

LineageConfiguration 結構

指定爬蟲程式的資料歷程組態設定。

欄位

- `CrawlerLineageSettings` – UTF-8 字串 (有效值：`ENABLE` | `DISABLE`)。

  指定是否啟用爬蟲程式的資料歷程。有效的 值如下：

  - ENABLE：啟用爬蟲程式的資料歷程
  - DISABLE：停用爬蟲程式的資料歷程

LakeFormationConfiguration 結構

指 AWS Lake Formation 定爬行者程式的組態設定值。

欄位

- `UseLakeFormationCredentials` – 布林值。

  指定是否要使用爬行者程式的 AWS Lake Formation 認證，而不是 IAM 角色登入資料。

- `AccountId` – UTF-8 字串，長度不可超過 12 個位元組。

  跨帳戶網路爬取的必要項目。對於與目標資料相同的帳戶網路爬取，則可以將其保留為 null。

作業

- CreateCrawler 動作（Python：創建履帶）
- DeleteCrawler 行動（Python：刪除履帶）
- GetCrawler 行動（Python：獲取履帶）
- GetCrawlers 行動（Python：獲取爬蟲）
- GetCrawlerMetrics 動作（Python：獲取履帶程序度量）
- UpdateCrawler 行動（Python：更新 _ 爬蟲）
- StartCrawler 行動（Python：開始履帶）
- StopCrawler 行動（Python：停止履帶）
- BatchGetCrawlers 動作（Python：批處理抓取器）
- ListCrawlers 動作（Python：列表爬蟲）
- ListCrawls 動作（Python：列表爬行）

CreateCrawler 動作（Python：創建履帶）

建立新的爬蟲程式，為其指定目標、角色、組態和選用的排程。至少必須在 `s3Targets` 欄位、`jdbcTargets` 欄位或 `DynamoDBTargets` 欄位中指定一個抓取目標。

請求

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  新爬蟲程式的名稱。

- `Role` – 必要：UTF-8 字串。

新爬蟲程式用來存取客戶資源的 IAM 角色或某 IAM 角色的 Amazon Resource Name (ARN)。

- `DatabaseName` – UTF-8 字串。

  寫入結果的 AWS Glue 資料庫，例如：`arn:aws:daylight:us-east-1::database/sometable/*`。

- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  新爬蟲程式的描述。

- `Targets` – 必要：CrawlerTargets 物件。

  待編目的目標集合清單。

- `Schedule` – UTF-8 字串。

  用來指定排程的 `cron` 表達式 (請參閱適用於任務與爬蟲程式的依時排程。例如，如果要每天在 12:15 UTC 執行某項動作，您可以指定：`cron(15 12 * * ? *)`。

- `Classifiers` – UTF-8 字串陣列。

  使用者已註冊的自訂分類器清單。依預設，所有內建分類器均包含在編目內，但這些自訂分類器一律覆寫特定分類的預設分類器。

- `TablePrefix` – UTF-8 字串，長度不可超過 128 個位元組。

  用於為所建立之資料表建立目錄的資料表字首。

- `SchemaChangePolicy` – SchemaChangePolicy 物件。

  爬蟲程式的更新和刪除行為政策。

- `RecrawlPolicy` – RecrawlPolicy 物件。

  一種政策，指定是否要再次網路爬取整個資料集，或只網路爬取自上次執行爬蟲程式後新增的資料夾。

- `LineageConfiguration` – LineageConfiguration 物件。

  指定爬蟲程式的資料歷程組態設定。

- `LakeFormationConfiguration` – LakeFormationConfiguration 物件。

  指 AWS Lake Formation 定爬行者程式的組態設定值。

- `Configuration` – UTF-8 字串。

爬蟲程式組態資訊。此版本的 JSON 字串可讓使用者指定爬蟲程式的各種行為。如需詳細資訊，請參閱設定爬蟲程式組態選項。

- `CrawlerSecurityConfiguration` – UTF-8 字串，長度不可超過 128 個位元組。

  此爬蟲程式要使用的 `SecurityConfiguration` 結構。

- `Tags` – 金鑰值對的對應陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  要搭配此爬蟲程式要求使用的標籤。您可以使用標籤來限制對於爬蟲程式的存取情況。如需中標籤的詳細資訊 AWS Glue，請參閱開發人員指南AWS Glue中的「AWS 標籤」。

回應

- 無回應參數。

錯誤

- `InvalidInputException`
- `AlreadyExistsException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`

DeleteCrawler 行動（Python：刪除履帶）

從中移除指定的 AWS Glue Data Catalog爬行者程式 (除非爬行者程式狀態為)。`RUNNING`

請求

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要移除之爬蟲程式的名稱。

回應

- 無回應參數。

錯誤

- `EntityNotFoundException`
- `CrawlerRunningException`
- `SchedulerTransitioningException`
- `OperationTimeoutException`

GetCrawler 行動（Python：獲取履帶）

擷取特定爬蟲程式的中繼資料。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要擷取中繼資料之爬蟲程式的名稱。

回應

- `Crawler` – 爬蟲程式 物件。

  特定爬蟲程式的中繼資料。

錯誤

- `EntityNotFoundException`
- `OperationTimeoutException`

GetCrawlers 行動（Python：獲取爬蟲）

從客戶帳戶中定義之所有爬蟲程式擷取中繼資料。

請求

- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  每次呼叫要傳回的爬蟲程式數量。
- NextToken – UTF-8 字串。

  接續符記，如果這是接續要求。

回應

- Crawlers – 一個 爬蟲程式 物件陣列。

  爬蟲程式中繼資料清單。
- NextToken – UTF-8 字串。

  持續字元，如果傳回的清單沒有達到此客戶帳戶中定義的結尾。

錯誤

- OperationTimeoutException

GetCrawlerMetrics 動作（Python：獲取履帶程序度量）

擷取指定爬蟲程式的指標。

請求

- CrawlerNameList – UTF-8 字串的陣列，不可超過 100 個字串。

  要擷取指標之爬蟲程式的名稱清單。
- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  所要回傳清單的大小上限。
- NextToken – UTF-8 字串。

  接續符記，如果這是接續呼叫。

回應

- `CrawlerMetricsList` – 一個 CrawlerMetrics 物件陣列。

  指定爬蟲程式的指標清單。

- `NextToken` – UTF-8 字串。

  接續字元，如果傳回的清單未包含最後一個可用指標。

錯誤

- `OperationTimeoutException`

UpdateCrawler 行動（Python：更新 _ 爬蟲）

更新爬蟲程式。如果爬蟲程式執行中，您必須先使用 `StopCrawler` 停止爬蟲程式，然後再更新。

請求

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  新爬蟲程式的名稱。

- `Role` – UTF-8 字串。

  新爬蟲程式用來存取客戶資源的 IAM 角色或某 IAM 角色的 Amazon Resource Name (ARN)。

- `DatabaseName` – UTF-8 字串。

  儲存結果的 AWS Glue 資料庫，例如：`arn:aws:daylight:us-east-1::database/sometable/*`。

- `Description` – UTF-8 字串，長度不可超過 2048 個位元組，且需符合 URI address multi-line string pattern。

  新爬蟲程式的描述。

- `Targets` – CrawlerTargets 物件。

  待編目的目標清單。

- `Schedule` – UTF-8 字串。

用來指定排程的 cron 表達式 (請參閱適用於任務與爬蟲程式的依時排程。例如，如果要每天在 12:15 UTC 執行某項動作，您可以指定：cron(15 12 * * ? *)。

- Classifiers – UTF-8 字串陣列。

  使用者已註冊的自訂分類器清單。依預設，所有內建分類器均包含在編目內，但這些自訂分類器一律覆寫特定分類的預設分類器。

- TablePrefix – UTF-8 字串，長度不可超過 128 個位元組。

  用於為所建立之資料表建立目錄的資料表字首。

- SchemaChangePolicy – SchemaChangePolicy 物件。

  爬蟲程式的更新和刪除行為政策。

- RecrawlPolicy – RecrawlPolicy 物件。

  一種政策，指定是否要再次網路爬取整個資料集，或只網路爬取自上次執行爬蟲程式後新增的資料夾。

- LineageConfiguration – LineageConfiguration 物件。

  指定爬蟲程式的資料歷程組態設定。

- LakeFormationConfiguration – LakeFormationConfiguration 物件。

  指 AWS Lake Formation 定爬行者程式的組態設定值。

- Configuration – UTF-8 字串。

  爬蟲程式組態資訊。此版本的 JSON 字串可讓使用者指定爬蟲程式的各種行為。如需詳細資訊，請參閱設定爬蟲程式組態選項。

- CrawlerSecurityConfiguration – UTF-8 字串，長度不可超過 128 個位元組。

  此爬蟲程式要使用的 SecurityConfiguration 結構。

回應

- 無回應參數。

錯誤

- InvalidInputException

- VersionMismatchException
- EntityNotFoundException
- CrawlerRunningException
- OperationTimeoutException

StartCrawler 行動（Python：開始履帶）

使用指定的爬蟲程式開始編目，無論排程。如果爬行者程式已經在執行中，會傳回. CrawlerRunningException

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要啟動的爬蟲程式名稱。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- CrawlerRunningException
- OperationTimeoutException

StopCrawler 行動（Python：停止履帶）

如果指定的爬蟲程式正在執行中，停止編目。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要停止的爬蟲程式名稱。

回應

- 無回應參數。

錯誤

- `EntityNotFoundException`
- `CrawlerNotRunningException`
- `CrawlerStoppingException`
- `OperationTimeoutException`

BatchGetCrawlers 動作（Python：批處理抓取器）

為指定的爬蟲程式名稱清單，傳回資源中繼資料的清單。呼叫 `ListCrawlers` 操作之後，您便可以呼叫此操作來存取您已授與許可的資料。此操作支援所有 IAM 許可，包括使用標籤的許可條件。

請求

- `CrawlerNames` – 必要：UTF-8 字串的陣列，不可超過 100 個字串。

  爬蟲程式名稱清單，可能是從 `ListCrawlers` 操作傳回的名稱。

回應

- `Crawlers` – 一個 爬蟲程式 物件陣列。

  爬蟲程式定義的清單。
- `CrawlersNotFound` – UTF-8 字串的陣列，不可超過 100 個字串。

  未尋獲爬蟲程式的名稱清單。

錯誤

- `InvalidInputException`
- `OperationTimeoutException`

ListCrawlers 動作（Python：列表爬蟲）

擷取此 AWS 帳戶中所有爬行者程式資源的名稱，或具有指定標籤的資源。您可運用此操作，查看帳戶下有哪些可用資源及其名稱。

此操作會接收您可在回應時做為篩選條件的選用 Tags 欄位，因此已標記的資源可分組進行擷取。如果您選擇使用標籤進行篩選，則此時只會擷取包含該標籤的資源。

請求

- `MaxResults` – 數字 (整數)，不可小於 1，也不可以大於 1000。

  所要回傳清單的大小上限。

- `NextToken` – UTF-8 字串。

  接續符記，如果這是接續要求。

- `Tags` – 金鑰值對的對應陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  指定只傳回包含這些標籤的資源。

回應

- `CrawlerNames` – UTF-8 字串的陣列，不可超過 100 個字串。

  這個帳戶下所有爬蟲程式的名稱，或是使用指定標籤的爬蟲程式。

- `NextToken` – UTF-8 字串。

  接續字元，如果傳回的清單未包含最後一個可用指標。

錯誤

- `OperationTimeoutException`

ListCrawls 動作（Python：列表爬行）

傳回指定爬蟲程式的所有網路爬取。僅傳回自爬蟲程式歷史記錄功能啟動日期以來發生的網路爬取，而且最多只會保留 12 個月的網路爬取。不會傳回較舊的網路爬取。

您可以使用此 API 來：

- 擷取指定爬蟲程式的所有網路爬取。
- 在有限的計數內擷取指定爬蟲程式的所有網路爬取。
- 擷取特定時間範圍內指定爬蟲程式的所有網路爬取。
- 擷取具有特定狀態、網路爬取 ID 或 DPU 小時值的指定爬蟲程式的所有網路爬取。

請求

- `CrawlerName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  您希望擷取其執行的爬蟲程式名稱。

- `MaxResults` – 數字 (整數)，不可小於 1，也不可以大於 1000。

  回傳結果的數量上限。預設值為 20，最大值為 100。

- `Filters` – 一個 CrawlsFilter 物件陣列。

  依照您在下列 `CrawlsFilter` 物件的清單中指定的條件篩選網路爬取。

- `NextToken` – UTF-8 字串。

  接續符記，如果這是接續呼叫。

回應

- `Crawls` – 一個 CrawlerHistory 物件陣列。

  `CrawlerHistory` 物件的清單，代表符合您條件的網路爬取執行。

- `NextToken` – UTF-8 字串。

  為一種接續符記，用於將傳回的符記清單分頁，而如果清單目前的區段不是最後區段就會傳回。

錯誤

- EntityNotFoundException
- OperationTimeoutException
- InvalidInputException

資料欄統計資料 API

資料欄統計資料 API 將說明傳回資料表資料欄統計資料的 AWS Glue API。

資料類型

- ColumnStatisticsTaskRun 結構
- ColumnStatisticsTaskRunningException 結構
- ColumnStatisticsTaskNotRunningException 結構
- ColumnStatisticsTaskStoppingException 結構

ColumnStatisticsTaskRun 結構

顯示資料欄統計資料執行之詳細資料的物件。

欄位

- CustomerId – UTF-8 字串，長度不可超過 12 個位元組。

  AWS 帳戶 ID。
- ColumnStatisticsTaskRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  適用於特定資料欄統計資料任務執行的識別碼。
- DatabaseName – UTF-8 字串。

  資料表所在的資料庫。
- TableName – UTF-8 字串。

  產生資料欄統計資料之資料表的名稱。
- ColumnNameList – UTF-8 字串陣列。

欄名稱之清單。若未提供，則依預設系統將會使用資料表的所有資料欄名稱。

- CatalogID – 目錄 ID 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在的 Data Catalog 的 ID。若沒有提供，則依預設會使用 AWS 帳戶 ID。

- Role – UTF-8 字串。

  服務用於產生統計資料的 IAM 角色。

- SampleSize：數字 (雙位數)，不可大於 100。

  用來產生統計資料的資料列百分比。若未提供，則系統將會使用整個資料表產生統計資料。

- SecurityConfiguration – UTF-8 字串，長度不可超過 128 個位元組。

  用於加密資料欄統計資料任務執行之 CloudWatch 日誌的安全組態名稱。

- NumberOfWorkers – 數字 (整數)，至少為 1。

  用於產生資料欄統計資料的工作者數量。此工作已預先設定為自動擴展至 25 個執行個體。

- WorkerType – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於產生統計資料的工作者類型。預設值為 g.1x。

- Status – UTF-8 字串 (有效值：STARTING | RUNNING | SUCCEEDED | FAILED | STOPPED)。

  任務執行的狀態。

- CreationTime – 時間戳記。

  此任務建立的時間。

- LastUpdated – 時間戳記。

  此任務上次修改的時間點。

- StartTime – 時間戳記。

  任務的開始時間。

- EndTime – 時間戳記。

  任務的結束時間。

- `ErrorMessage` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  工作的錯誤訊息。

- `DPUSeconds` – 數字 (雙精度浮點數)，不可大於 None (無)。

  所有自動擴展之工作者的計算 DPU 用量 (以秒為單位)。

ColumnStatisticsTaskRunningException 結構

當您在執行資料欄統計資料產生工作期間，嘗試啟動其他工作時發生的例外狀況。

欄位

- `Message` – UTF-8 字串。

  說明問題的訊息。

ColumnStatisticsTaskNotRunningException 結構

當您在沒有任務執行期間，嘗試停止任務執行時發生的例外狀況。

欄位

- `Message` – UTF-8 字串。

  說明問題的訊息。

ColumnStatisticsTaskStoppingException 結構

當您嘗試停止任務執行時發生的例外狀況。

欄位

- `Message` – UTF-8 字串。

  說明問題的訊息。

操作

- StartColumnStatisticsTaskRun 動作 (Python：start_column_statistics_task_run)
- GetColumnStatisticsTaskRun 動作 (Python：get_column_statistics_task_run)
- GetColumnStatisticsTaskRuns 動作 (Python：get_column_statistics_task_runs)
- ListColumnStatisticsTaskRuns 動作 (Python：list_column_statistics_task_runs)
- StopColumnStatisticsTaskRun 動作 (Python：stop_column_statistics_task_run)

StartColumnStatisticsTaskRun 動作 (Python：start_column_statistics_task_run)

針對指定的資料表和資料欄，啟動資料欄統計資料任務執行。

請求

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表所在的資料庫名稱。
- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  用於產生統計資料的資料表名稱。
- `ColumnNameList` – UTF-8 字串陣列。

  用於產生統計資料之資料欄名稱的清單。若未提供，則依預設系統將會使用資料表的所有資料欄名稱。
- `Role` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  服務用於產生統計資料的 IAM 角色。
- `SampleSize`：數字 (雙位數)，不可大於 100。

  用來產生統計資料的資料列百分比。若未提供，則系統將會使用整個資料表產生統計資料。
- `CatalogID` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料表所在之 Data Catalog 的 ID。若沒有提供，則依預設會使用 AWS 帳戶 ID。

- SecurityConfiguration – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於加密資料欄統計資料任務執行之 CloudWatch 日誌的安全組態名稱。

回應

- ColumnStatisticsTaskRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料欄統計資料任務執行的識別碼。

錯誤

- AccessDeniedException
- EntityNotFoundException
- ColumnStatisticsTaskRunningException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- InvalidInputException

GetColumnStatisticsTaskRun 動作 (Python：get_column_statistics_task_run)

取得已知任務執行 ID 之任務執行的相關中繼資料/資訊。

請求

- ColumnStatisticsTaskRunId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  適用於特定資料欄統計資料任務執行的識別碼。

回應

- ColumnStatisticsTaskRun – ColumnStatisticsTaskRun 物件。

  表示資料欄統計資料執行之詳細資料的 ColumnStatisticsTaskRun 物件。

錯誤

- EntityNotFoundException
- OperationTimeoutException
- InvalidInputException

GetColumnStatisticsTaskRuns 動作 (Python：get_column_statistics_task_runs)

擷取與指定資料表相關聯之所有執行的相關資訊。

請求

- DatabaseName – 必要：UTF-8 字串。

  資料表所在的資料庫名稱。
- TableName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表的名稱。
- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  回應的大小上限。
- NextToken – UTF-8 字串。

  接續符記，如果這是接續呼叫。

回應

- ColumnStatisticsTaskRuns – 一個 ColumnStatisticsTaskRun 物件陣列。

  資料欄統計資料任務執行的清單。
- NextToken – UTF-8 字串。

  持續權杖 (如果尚未傳回所有任務執行)。

錯誤

- OperationTimeoutException

ListColumnStatisticsTaskRuns 動作 (Python：list_column_statistics_task_runs)

列出特定帳戶的所有任務執行。

請求

- `MaxResults` – 數字 (整數)，不可小於 1，也不可以大於 1000。

  回應的大小上限。
- `NextToken` – UTF-8 字串。

  接續符記，如果這是接續呼叫。

回應

- `ColumnStatisticsTaskRunIds` – UTF-8 字串的陣列，不可超過 100 個字串。

  資料欄統計資料任務執行 ID 的清單。
- `NextToken` – UTF-8 字串。

  持續權杖 (如果尚未傳回所有任務執行 ID)。

錯誤

- `OperationTimeoutException`

StopColumnStatisticsTaskRun 動作 (Python：stop_column_statistics_task_run)

停止指定資料表的任務執行。

請求

- `DatabaseName` – 必要：UTF-8 字串。

  資料表所在的資料庫名稱。
- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表的名稱。

回應

- 無回應參數。

錯誤

- `EntityNotFoundException`
- `ColumnStatisticsTaskNotRunningException`
- `ColumnStatisticsTaskStoppingException`
- `OperationTimeoutException`

爬蟲程式排程器 API

爬蟲程式排程器 API 說明 AWS Glue 爬蟲程式資料類型，以及用於建立、刪除、更新和列出爬蟲程式的 API。

資料類型

- Schedule 結構

Schedule 結構

排程物件，使用 `cron` 陳述式來將事件排程。

欄位

- `ScheduleExpression` – UTF-8 字串。

  用來指定排程的 `cron` 表達式 (請參閱適用於任務與爬蟲程式的依時排程。例如，如果要每天在 12:15 UTC 執行某項動作，您可以指定：`cron(15 12 * * ? *)`。
- `State` – UTF-8 字串 (有效值：`SCHEDULED` | `NOT_SCHEDULED` | `TRANSITIONING`)。

  排程的狀態。

操作

- UpdateCrawlerSchedule 動作 (Python: update_crawler_schedule)

- StartCrawlerSchedule 動作 (Python: start_crawler_schedule)
- StopCrawlerSchedule 動作 (Python: stop_crawler_schedule)

UpdateCrawlerSchedule 動作 (Python: update_crawler_schedule)

使用 cron 表達式來更新爬蟲程式排程。

請求

- CrawlerName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要更新排程之爬蟲程式的名稱。
- Schedule – UTF-8 字串。

  更新的 cron 表達式，用來指定排程 (請參閱適用於任務與爬蟲程式的依時排程。例如，如果要每天在 12:15 UTC 執行某項動作，您可以指定：cron(15 12 * * ? *)。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- InvalidInputException
- VersionMismatchException
- SchedulerTransitioningException
- OperationTimeoutException

StartCrawlerSchedule 動作 (Python: start_crawler_schedule)

將指定爬蟲程式的排程狀態變更為 SCHEDULED (除非爬蟲程式已在執行中，或排程狀態已經是 SCHEDULED。)

請求

- `CrawlerName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要排程的爬蟲程式的名稱。

回應

- 無回應參數。

錯誤

- `EntityNotFoundException`
- `SchedulerRunningException`
- `SchedulerTransitioningException`
- `NoScheduleException`
- `OperationTimeoutException`

StopCrawlerSchedule 動作 (Python: stop_crawler_schedule)

將指定爬蟲程式的排程狀態設定為 `NOT_SCHEDULED`，但如果爬蟲程式已在執行中，此動作不會停止爬蟲程式。

請求

- `CrawlerName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要更新其排程狀態的爬蟲程式的名稱。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- SchedulerNotRunningException
- SchedulerTransitioningException
- OperationTimeoutException

自動產生 ETL 指令碼 API

ETL 產生指令碼 API 說明在 AWS Glue 產生 ETL 指令碼的資料類型與 API。

資料類型

- CodeGenNode 結構
- CodeGenNodeArg 結構
- CodeGenEdge 結構
- Location 結構
- CatalogEntry 結構
- MappingEntry 結構

CodeGenNode 結構

代表有向無環圖 (DAG) 中的節點

欄位

- Id – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Identifier string pattern。

  節點圖中獨特唯一的節點識別符。
- NodeType – 必要：UTF-8 字串。

  節點的所屬類型。
- Args – 必要：一個 CodeGenNodeArg 物件陣列，不可超過 50 個結構。

  節點的屬性，形式為名稱值對組。

- `LineNumber` – 數字 (整數)。

  節點的行號。

CodeGenNodeArg 結構

節點的引數或屬性。

欄位

- `Name` – 必要：UTF-8 字串。

  引數或屬性的名稱。

- `Value` – 必要：UTF-8 字串。

  引數或屬性的值。

- `Param` – 布林值。

  如果值做為參數使用，則為 true。

CodeGenEdge 結構

代表有向無環圖 (DAG) 中的方向性邊緣。

欄位

- `Source` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Identifier string pattern。

  邊緣開始節點的 ID。

- `Target` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Identifier string pattern。

  邊緣結束節點的 ID。

- `TargetParameter` – UTF-8 字串。

  邊緣的目標。

Location 結構

資源的位置。

欄位

- `Jdbc` – CodeGenNodeArg 物件陣列，不可超過 50 個結構。

  JDBC 位置。
- `S3` – CodeGenNodeArg 物件陣列，不可超過 50 個結構。

  Amazon Simple Storage Service (Amazon S3) 的位置。
- `DynamoDB` – CodeGenNodeArg 物件陣列，不可超過 50 個結構。

  Amazon DynamoDB 資料表位置。

CatalogEntry 結構

在 AWS Glue Data Catalog 中指定資料表的定義。

欄位

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料表中繼資料所在的資料庫。
- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  有問題的資料表的名稱。

MappingEntry 結構

定義映射。

欄位

- `SourceTable` – UTF-8 字串。

  來源資料表的名稱。

- `SourcePath` – UTF-8 字串。

  來源路徑。
- `SourceType` – UTF-8 字串。

  來源類型。
- `TargetTable` – UTF-8 字串。

  目標資料表。
- `TargetPath` – UTF-8 字串。

  目標路徑。
- `TargetType` – UTF-8 字串。

  目標類型。

操作

- CreateScript 動作 (Python: create_script)
- GetDataflowGraph 動作 (Python: get_dataflow_graph)
- GetMapping 動作 (Python: get_mapping)
- GetPlan 動作 (Python: get_plan)

CreateScript 動作 (Python: create_script)

將有向無環圖 (DAG) 轉換為程式碼。

請求

- `DagNodes` – 一個 CodeGenNode 物件陣列。

  DAG 中節點的清單。
- `DagEdges` – 一個 CodeGenEdge 物件陣列。

  DAG 中邊緣的清單。
- `Language` – UTF-8 字串 (有效值：`PYTHON` | `SCALA`)。

  從 DAG 所產生程式碼的程式設計語言。

回應

- `PythonScript` – UTF-8 字串。

  從 DAG 產生的 Python 指令碼。
- `ScalaCode` – UTF-8 字串。

  從 DAG 產生的 Scala 程式碼。

錯誤

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetDataflowGraph 動作 (Python: get_dataflow_graph)

將 Python 指令碼轉換為有向無環圖 (DAG)。

請求

- `PythonScript` – UTF-8 字串。

  要轉換的 Python 指令碼。

回應

- `DagNodes` – 一個 CodeGenNode 物件陣列。

  在所產生 DAG 中的節點的清單。
- `DagEdges` – 一個 CodeGenEdge 物件陣列。

  在所產生 DAG 中的邊緣的清單。

錯誤

- `InvalidInputException`
- `InternalServiceException`

- OperationTimeoutException

GetMapping 動作 (Python: get_mapping)

建立映射。

請求

- Source – 必要：CatalogEntry 物件。

  指定來源資料表。
- Sinks – 一個 CatalogEntry 物件陣列。

  目標資料表的清單。
- Location – 位置 物件。

  映射用的參數。

回應

- Mapping – 必要：一個 MappingEntry 物件。

  指定目標的映射的清單。

錯誤

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- EntityNotFoundException

GetPlan 動作 (Python: get_plan)

取得程式碼以進行指定的映射。

請求

- Mapping – 必要：一個 MappingEntry 物件。

來源資料表與目標資料表的映射的清單。

- `Source` – 必要：CatalogEntry 物件。

  來源資料表。

- `Sinks` – 一個 CatalogEntry 物件陣列。

  目標資料表。

- `Location` – 位置 物件。

  映射用的參數。

- `Language` – UTF-8 字串 (有效值：PYTHON | SCALA)。

  用來進行映射的程式碼的程式設計語言。

- `AdditionalPlanOptionsMap` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  用於保存額外選用鍵-值參數的映射。

  目前支援下列鍵值對：

  - `inferSchema` — 指定針對 AWS Glue 任務產生的預設指令碼，將 `inferSchema` 設定為 true 或 false。例如，若要將 `inferSchema` 設定為 true，請傳遞鍵值對：

    ```
    --additional-plan-options-map '{"inferSchema":"true"}'
    ```

回應

- `PythonScript` – UTF-8 字串。

  用來進行映射的 Python 指令碼。

- `ScalaCode` – UTF-8 字串。

  用來進行映射的 Scala 程式碼。

錯誤

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

視覺化任務 API

視覺化工作 API 可讓您使用 JSON 物件 (代表工作的視覺化設定) 中的 AWS Glue API 來建立資料整合 AWS Glue 工作。

提供給建立或更新工作 API 的清單，以便在 AWS Glue Studio 中註冊已建立的工作的 DAG，並產生相關聯的程式碼。`CodeGenConfigurationNodes`

資料類型

- CodeGenConfigurationNode 結構
- JDBC ConnectorOptions 結構
- StreamingDataPreviewOptions 結構
- AthenaConnectorSource 結構
- JDBC ConnectorSource 結構
- SparkConnectorSource 結構
- CatalogSource 結構
- MySQL CatalogSource 結構
- PostgreSQL CatalogSource 結構
- OracleSQL CatalogSource 結構
- 微软结构 ServerCatalogSource
- CatalogKinesisSource 結構
- DirectKinesisSource 結構
- KinesisStreamingSourceOptions 結構
- CatalogKafkaSource 結構
- DirectKafkaSource 結構
- KafkaStreamingSourceOptions 結構
- RedshiftSource 結構

- AmazonRedshiftSource 結構
- AmazonRedshiftNodeData 結構
- AmazonRedshiftAdvancedOption 結構
- 選項結構
- S3 CatalogSource 結構
- S3 SourceAdditionalOptions 結構
- S3 CsvSource 結構
- DirectJDBCSource 結構
- S3 DirectSourceAdditionalOptions 結構
- S3 JsonSource 結構
- S3 ParquetSource 結構
- S3 DeltaSource 結構
- S3 CatalogDeltaSource 結構
- CatalogDeltaSource 結構
- S3 HudiSource 結構
- S3 CatalogHudiSource 結構
- CatalogHudiSource 結構
- DynamoDB CatalogSource 結構
- RelationalCatalogSource 結構
- JDBC ConnectorTarget 結構
- SparkConnectorTarget 結構
- BasicCatalogTarget 結構
- MySQL CatalogTarget 結構
- PostgreSQL CatalogTarget 結構
- OracleSQL CatalogTarget 結構
- 微软结构 ServerCatalogTarget
- RedshiftTarget 結構
- AmazonRedshiftTarget 結構
- UpsertRedshiftTargetOptions 結構
- S3 CatalogTarget 結構

- S3 GlueParquetTarget 結構
- CatalogSchemaChangePolicy 結構
- S3 DirectTarget 結構
- S3 HudiCatalogTarget 結構
- S3 HudiDirectTarget 結構
- S3 DeltaCatalogTarget 結構
- S3 DeltaDirectTarget 結構
- DirectSchemaChangePolicy 結構
- ApplyMapping 結構
- Mapping 結構
- SelectFields 結構
- DropFields 結構
- RenameField 結構
- Spigot 結構
- Join 結構
- JoinColumn 結構
- SplitFields 結構
- SelectFromCollection 結構
- FillMissingValues 結構
- Filter 結構
- FilterExpression 結構
- FilterValue 結構
- CustomCode 結構
- SparkSQL 結構
- SqlAlias 結構
- DropNullFields 結構
- NullCheckBoxList 結構
- NullValueField 結構
- Datatype 結構
- Merge 結構

- Union 結構
- PIIDetection 結構
- Aggregate 結構
- DropDuplicates 結構
- GovernedCatalogTarget 結構
- GovernedCatalogSource 結構
- AggregateOperation 結構
- GlueSchema 結構
- GlueStudioSchemaColumn 結構
- GlueStudioColumn 結構
- DynamicTransform 結構
- TransformConfigParameter 結構
- EvaluateDataQuality 結構
- DQ ResultsPublishingOptions 結構
- DQ StopJobOnFailureOptions 結構
- EvaluateDataQualityMultiFrame 結構
- 配方結構
- RecipeReference 結構
- SnowflakeNodeData 結構
- SnowflakeSource 結構
- SnowflakeTarget 結構
- ConnectorDataSource 結構
- ConnectorDataTarget 結構

CodeGenConfigurationNode 結構

`CodeGenConfigurationNode` 列舉所有有效的節點類型。僅可以填入一個成員變數。

欄位

- `AthenaConnectorSource` – AthenaConnectorSource 物件。

  指定 Amazon Athena 資料來源的連接器。

- `JDBCConnectorSource` – JDBC ConnectorSource 物件。

  指定 JDBC 資料來源的連接器。

- `SparkConnectorSource` – SparkConnectorSource 物件。

  指定 Apache Spark 資料來源的連接器。

- `CatalogSource` – CatalogSource 物件。

  指定「資料目錄」中的 AWS Glue 資料倉庫。

- `RedshiftSource` – RedshiftSource 物件。

  指定 Amazon Redshift 資料存放區。

- `S3CatalogSource` – S3 CatalogSource 物件。

  在資料目錄中指定 Amazon S3 資 AWS Glue 料存放區。

- `S3CsvSource` – S3 CsvSource 物件。

  指定存放在 Amazon S3 中的命令分隔值 (CSV) 資料存放區。

- `S3JsonSource` – S3 JsonSource 物件。

  指定儲存在 Amazon S3 中的 JSON 資料存放區。

- `S3ParquetSource` – S3 ParquetSource 物件。

  指定存放在 Amazon S3 中的 Apache Parquet 資料存放區。

- `RelationalCatalogSource` – RelationalCatalogSource 物件。

  指定資料目錄中的關聯式目錄資 AWS Glue 料倉庫。

- `DynamoDBCatalogSource` – DynamoDB CatalogSource 物件。

  在資料目錄中指定 DynamoDBC 目錄資料存放區。 AWS Glue

- `JDBCConnectorTarget` – JDBC ConnectorTarget 物件。

  指定以 Apache Parquet 直欄式儲存寫入 Amazon S3 的資料目標。

- `SparkConnectorTarget` – SparkConnectorTarget 物件。

  指定使用 Apache Spark 連接器的目標。

- `CatalogTarget` – BasicCatalogTarget 物件。

指定使用「 AWS Glue 資料目錄」表格的目標。

- `RedshiftTarget` – RedshiftTarget 物件。

  指定使用 Amazon Redshift 的目標。

- `S3CatalogTarget` – S3 CatalogTarget 物件。

  指定使用資料目錄寫入 Amazon S3 的 AWS Glue 資料目標。

- `S3GlueParquetTarget` – S3 GlueParquetTarget 物件。

  指定以 Apache Parquet 直欄式儲存寫入 Amazon S3 的資料目標。

- `S3DirectTarget` – S3 DirectTarget 物件。

  指定寫入 Amazon S3 的資料目標。

- `ApplyMapping` – ApplyMapping 物件。

  指定將資料來源中的資料屬性索引鍵映射至資料目標中資料屬性索引鍵的轉換。您可以重新命名索引鍵、修改索引鍵的資料類型，以及選擇要從資料集中捨棄哪些索引鍵。

- `SelectFields` – SelectFields 物件。

  指定選擇要保留之資料屬性索引鍵的轉換。

- `DropFields` – DropFields 物件。

  指定選擇要捨棄之資料屬性索引鍵的轉換。

- `RenameField` – RenameField 物件。

  指定重新命名單一資料屬性索引鍵的轉換。

- `Spigot` – Spigot 物件。

  指定將資料範例寫入 Amazon S3 儲存貯體的轉換。

- `Join` – Join 物件。

  使用指定資料屬性索引鍵上的比較片語，將兩個資料集聯結為一個資料集。可以使用內、外、左、右、左半、左反聯結。

- `SplitFields` – SplitFields 物件。

  指定將資料屬性索引鍵分割成兩個 `DynamicFrames` 的轉換。輸出是 `DynamicFrames` 的集合：一個具有所選資料屬性索引鍵，另一個具有其餘資料屬性索引鍵。

- `SelectFromCollection` – SelectFromCollection 物件。

  指定從 `DynamicFrames` 的集合選擇一個 `DynamicFrame` 的轉換。輸出為所選的 `DynamicFrame`。

- `FillMissingValues` – FillMissingValues 物件。

  指定如下轉換：尋找遺失值之資料集中的記錄，並新增具有由插補決定值的新欄位。輸入資料集會用於訓練機器學習模型，以決定遺失值應該是什麼。

- `Filter` – Filter 物件。

  指定根據篩選條件將資料集分割成兩個的轉換。

- `CustomCode` – CustomCode 物件。

  指定使用您提供的自訂程式碼來執行資料轉換的轉換。輸出是的集合 DynamicFrames。

- `SparkSQL` – SparkSQL 物件。

  指定轉換，其中輸入使用 Spark SQL 語法的 SQL 查詢來轉換資料。輸出是單個 `DynamicFrame`。

- `DirectKinesisSource` – DirectKinesisSource 物件。

  指定直接的 Amazon Kinesis 資料來源。

- `DirectKafkaSource` – DirectKafkaSource 物件。

  指定 Apache Kafka 資料存放區。

- `CatalogKinesisSource` – CatalogKinesisSource 物件。

  在資料目錄中指定 Kinesis AWS Glue 資料來源。

- `CatalogKafkaSource` – CatalogKafkaSource 物件。

  指定 Data Catalog 中的 Apache Kafka 資料存放區。

- `DropNullFields` – DropNullFields 物件。

  指定轉換，如果資料行中的所有值都為「null」(空)，則從資料集中刪除此行。默認情況下， AWS Glue Studio 將識別空對象，但一些值，如空字符串，字符串是「null」，-1 個整數或其他佔位符，如零，不會自動識別為空值。

- `Merge` – Merge 物件。

  指定根據指定的主索引鍵來合併此 `DynamicFrame` 與暫存 `DynamicFrame` 以識別記錄的轉換。重複的記錄 (具有相同主索引鍵的記錄) 不會被刪除重複資料。

- `Union` – UNION 物件。

  指定將兩個或多個資料集中的列合併為單一結果的轉換。

- `PIIDetection` – PIIDetection 物件。

  指定用於標識、刪除或遮罩 PII 資料的轉換。

- `Aggregate` – Aggregate 物件。

  指定轉換，依照所選欄位來分組行，並依照指定函數計算彙總值。

- `DropDuplicates` – DropDuplicates 物件。

  指定用於從資料集刪除重複資料行的轉換。

- `GovernedCatalogTarget` – GovernedCatalogTarget 物件。

  指定資料目標寫入受管目錄。

- `GovernedCatalogSource` – GovernedCatalogSource 物件。

  指定受管 Data Catalog 中的 資料來源。

- `MicrosoftSQLServerCatalogSource` – 微软 ServerCatalogSource 物件。

  指定 AWS Glue Data Catalog 中的 Microsoft SQL 伺服器資料來源。

- `MySQLCatalogSource` – MySQL CatalogSource 物件。

  指定資料目錄中的 MySQL 資 AWS Glue 料來源。

- `OracleSQLCatalogSource` – OracleSQL CatalogSource 物件。

  在「資料目錄」中指定 Oracle AWS Glue 資料來源。

- `PostgreSQLCatalogSource` – PostgreSQL CatalogSource 物件。

  在「資料目錄」中指定 PostgresSQL 資 AWS Glue 料來源。

- `MicrosoftSQLServerCatalogTarget` – 微软 ServerCatalogTarget 物件。

  指定使用 Microsoft SQL 的目標。

- `MySQLCatalogTarget` – MySQL CatalogTarget 物件。

  指定使用 MySQL 的目標。

- `OracleSQLCatalogTarget` – OracleSQL CatalogTarget 物件。

指定使用 Oracle SQL 的目標。

- `PostgreSQLCatalogTarget` – PostgreSQL CatalogTarget 物件。

  指定使用 Postgres SQL 的目標。

- `DynamicTransform` – DynamicTransform 物件。

  指定使用者建立的自訂視覺化轉換。

- `EvaluateDataQuality` – EvaluateDataQuality 物件。

  指定資料品質評估標準。

- `S3CatalogHudiSource` – S3 CatalogHudiSource 物件。

  指定已在資料目錄中註冊的 Hudi AWS Glue 資料來源。資料來源必須儲存在中 Amazon S3。

- `CatalogHudiSource` – CatalogHudiSource 物件。

  指定已在資料目錄中註冊的 Hudi AWS Glue 資料來源。

- `S3HudiSource` – S3 HudiSource 物件。

  指定儲存於 Amazon S3中的 Hudi 資料來源。

- `S3HudiCatalogTarget` – S3 HudiCatalogTarget 物件。

  指定寫入資料目錄中 Hudi 資料來源的 AWS Glue 目標。

- `S3HudiDirectTarget` – S3 HudiDirectTarget 物件。

  指定寫入中 Amazon S3 Hudi 資料來源的目標。

- `S3CatalogDeltaSource` – S3 CatalogDeltaSource 物件。

  指定已在「資料目錄」中註冊的 Delta 湖資 AWS Glue 料來源。資料來源必須儲存在中 Amazon S3。

- `CatalogDeltaSource` – CatalogDeltaSource 物件。

  指定已在「資料目錄」中註冊的 Delta 湖資 AWS Glue 料來源。

- `S3DeltaSource` – S3 DeltaSource 物件。

  指定儲存於中的三角洲湖資料來源 Amazon S3。

- `S3DeltaCatalogTarget` – S3 DeltaCatalogTarget 物件。

指定寫入資料目錄中 Delta Lake 資料來源的 AWS Glue 目標。

- `S3DeltaDirectTarget` – S3 DeltaDirectTarget 物件。

  在中指定寫入 Delta 湖資料來源的目標 Amazon S3。

- `AmazonRedshiftSource` – AmazonRedshiftSource 物件。

  指定在 Amazon Redshift 中寫入資料來源的目標。

- `AmazonRedshiftTarget` – AmazonRedshiftTarget 物件。

  指定在 Amazon Redshift 中寫入資料目標的目標。

- `EvaluateDataQualityMultiFrame` – EvaluateDataQualityMultiFrame 物件。

  指定資料品質評估標準。允許多個輸入資料，並會傳回動態影格集合。

- `Recipe` – Recipe 物件。

  指定 AWS Glue DataBrew 配方節點。

- `SnowflakeSource` – SnowflakeSource 物件。

  指定 Snowflake 資料來源。

- `SnowflakeTarget` – SnowflakeTarget 物件。

  指定寫入 Snowflake 資料來源的目標。

- `ConnectorDataSource` – ConnectorDataSource 物件。

  指定使用標準連線選項產生的來源。

- `ConnectorDataTarget` – ConnectorDataTarget 物件。

  指定使用標準連線選項產生的目標。

JDBC ConnectorOptions 結構

連接器的其他連接選項。

欄位

- `FilterPredicate` – UTF-8 字串，需符合Custom string pattern #40。

  額外條件子句，用於篩選來源的資料。例如：

```
BillingCity='Mountain View'
```

當您使用查詢，而不是資料表名稱，您應該驗證查詢是否適用於指定的 filterPredicate。

- PartitionColumn – UTF-8 字串，需符合Custom string pattern #40。

  用於分割的整數資料行名稱。此選項僅適用於包含在 lowerBound、upperBound 以及 numPartitions 中。此選項的運作方式與 Spark SQL JDBC 讀取器相同。

- LowerBound – 數字 (long)，不可大於 None (無)。

  用來決定分割區步幅的 partitionColumn 最小值。

- UpperBound – 數字 (long)，不可大於 None (無)。

  用來決定分割區步幅的 partitionColumn 最大值。

- NumPartitions – 數字 (long)，不可大於 None (無)。

  分割區數。這個值，搭配 lowerBound (包含) 及 upperBound (不含)，形成用於分割 partitionColumn 而產生之 WHERE 子句表達式的分割區步幅。

- JobBookmarkKeys – UTF-8 字串陣列。

  用於排序之任務書籤索引鍵的名稱。

- JobBookmarkKeysSortOrder – UTF-8 字串，需符合Custom string pattern #40。

  指定升冪或降冪排序順序。

- DataTypeMapping – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串 (有效值：ARRAY | BIGINT | BINARY | BIT | BLOB | BOOLEAN | CHAR | CLOB | DATALINK | DATE | DECIMAL | DISTINCT | DOUBLE | FLOAT | INTEGER | JAVA_OBJECT | LONGNVARCHAR | LONGVARBINARY | LONGVARCHAR | NCHAR | NCLOB | NULL | NUMERIC | NVARCHAR | OTHER | REAL | REF | REF_CURSOR | ROWID | SMALLINT | SQLXML | STRUCT | TIME | TIME_WITH_TIMEZONE | TIMESTAMP | TIMESTAMP_WITH_TIMEZONE | TINYINT | VARBINARY | VARCHAR)。

  每個值都是 UTF-8 字串 (有效值：DATE | STRING | TIMESTAMP | INT | FLOAT | LONG | BIGDECIMAL | BYTE | SHORT | DOUBLE)。

  自訂資料類型映射，用於建置從 JDBC 資料類型到 AWS Glue 資料類型的映射。例如，該選項通過調用驅動程序的ResultSet.getString()方法FLOAT將 JDBC String 類型的數據字段"dataTypeMapping":{"FLOAT":"STRING"}映射到 Java 類型，並使用它來構建 AWS Glue

記錄。ResultSet 物件是由每個驅動程式實作，因此行為是特定於您使用的驅動程式。請參閱 JDBC 驅動程式的文件，瞭解驅動程式如何執行轉換。

StreamingDataPreviewOptions 結構

指定與資料預覽相關的選項，以檢視資料範例。

欄位

- PollingTime – 數字 (長)，至少為 10。

  輪詢時間 (以毫秒為單位)。
- RecordPollingLimit – 數字 (長)，至少為 1。

  輪詢的記錄數上限。

AthenaConnectorSource 結構

指定 Amazon Athena 資料來源的連接器。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料來源的名稱。
- ConnectionName – 必要：UTF-8 字串，需符合Custom string pattern #40。

  與連接器相關聯之連線的名稱。
- ConnectorName – 必要：UTF-8 字串，需符合Custom string pattern #40。

  協助存取 Studio 中資料存 AWS Glue 放區的連接器名稱。
- ConnectionType – 必要：UTF-8 字串，需符合Custom string pattern #40。

  連線類型，例如 marketplace.athena 或 custom.athena，指定連線到 Amazon Athena 資料存放區。
- ConnectionTable – UTF-8 字串，需符合Custom string pattern #41。

  資料來源中的資料表名稱。
- SchemaName – 必要：UTF-8 字串，需符合Custom string pattern #40。

要讀取的 CloudWatch 日誌群組名稱。例如：/aws-glue/jobs/output。

- OutputSchemas – 一個 GlueSchema 物件陣列。

  指定自訂 Athena 來源的資料架構。

JDBC ConnectorSource 結構

指定 JDBC 資料來源的連接器。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料來源的名稱。

- ConnectionName – 必要：UTF-8 字串，需符合Custom string pattern #40。

  與連接器相關聯之連線的名稱。

- ConnectorName – 必要：UTF-8 字串，需符合Custom string pattern #40。

  協助存取 Studio 中資料存 AWS Glue 放區的連接器名稱。

- ConnectionType – 必要：UTF-8 字串，需符合Custom string pattern #40。

  連線類型，例如 marketplace.jdbc 或 custom.jdbc，同時指定與 JDBC 資料存放區的連線。

- AdditionalOptions – JDBC ConnectorOptions 物件。

  連接器的其他連接選項。

- ConnectionTable – UTF-8 字串，需符合Custom string pattern #41。

  資料來源中的資料表名稱。

- Query – UTF-8 字串，需符合Custom string pattern #42。

  要從中取得資料的資料表或 SQL 查詢。您可以指定 ConnectionTable 或 query，但不能同時指定兩者。

- OutputSchemas – 一個 GlueSchema 物件陣列。

  指定自訂 JDBC 來源的資料架構。

SparkConnectorSource 結構

指定 Apache Spark 資料來源的連接器。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料來源的名稱。
- ConnectionName – 必要：UTF-8 字串，需符合Custom string pattern #40。

  與連接器相關聯之連線的名稱。
- ConnectorName – 必要：UTF-8 字串，需符合Custom string pattern #40。

  協助存取 Studio 中資料存 AWS Glue 放區的連接器名稱。
- ConnectionType – 必要：UTF-8 字串，需符合Custom string pattern #40。

  連接的類型，如 marketplace.spark 或 custom.spark，指定 Apache Spark 資料存放區的連線。
- AdditionalOptions – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  連接器的其他連接選項。
- OutputSchemas – 一個 GlueSchema 物件陣列。

  指定自訂 spark 來源的資料架構。

CatalogSource 結構

指定「資料目錄」中的 AWS Glue 資料倉庫。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料存放區的名稱。
- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

要讀取之資料庫的名稱。

- `Table` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫中資料表的名稱。

MySQL CatalogSource 結構

指定資料目錄中的 MySQL 資 AWS Glue 料來源。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料來源的名稱。

- `Database` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫的名稱。

- `Table` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫中資料表的名稱。

PostgreSQL CatalogSource 結構

在「資料目錄」中指定 PostgresSQL 資 AWS Glue 料來源。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料來源的名稱。

- `Database` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫的名稱。

- `Table` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫中資料表的名稱。

OracleSQL CatalogSource 結構

在「資料目錄」中指定 Oracle AWS Glue 資料來源。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料來源的名稱。
- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫的名稱。
- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫中資料表的名稱。

微软结构 ServerCatalogSource

指定 AWS Glue Data Catalog 中的 Microsoft SQL 伺服器資料來源。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料來源的名稱。
- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫的名稱。
- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫中資料表的名稱。

CatalogKinesisSource 結構

在資料目錄中指定 Kinesis AWS Glue 資料來源。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

資料來源的名稱。

- `WindowSize` – 數字 (整數)，不可大於 None (無)。

  處理每個微批次的時間量。

- `DetectSchema` – 布林值。

  是否自動從傳入資料確定結構描述。

- `Table` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫中資料表的名稱。

- `Database` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫的名稱。

- `StreamingOptions` – KinesisStreamingSourceOptions 物件。

  Kinesis 串流資料來源的其他選項。

- `DataPreviewOptions` – StreamingDataPreviewOptions 物件。

  資料預覽的其他選項。

DirectKinesisSource 結構

指定直接的 Amazon Kinesis 資料來源。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料來源的名稱。

- `WindowSize` – 數字 (整數)，不可大於 None (無)。

  處理每個微批次的時間量。

- `DetectSchema` – 布林值。

  是否自動從傳入資料確定結構描述。

- `StreamingOptions` – KinesisStreamingSourceOptions 物件。

  Kinesis 串流資料來源的其他選項。

- `DataPreviewOptions` – StreamingDataPreviewOptions 物件。

  資料預覽的其他選項。

KinesisStreamingSourceOptions 結構

Amazon Kinesis 串流資料來源的其他選項。

欄位

- `EndpointUrl` – UTF-8 字串，需符合Custom string pattern #40。

  Kinesis 端點的 URL。

- `StreamName` – UTF-8 字串，需符合Custom string pattern #40。

  Kinesis 資料串流的名稱。

- `Classification` – UTF-8 字串，需符合Custom string pattern #40。

  選擇性分類。

- `Delimiter` – UTF-8 字串，需符合Custom string pattern #40。

  指定分隔符號字元。

- `StartingPosition` – UTF-8 字串 (有效值：`latest="LATEST"` | `trim_horizon="TRIM_HORIZON"` | `earliest="EARLIEST"` | `timestamp="TIMESTAMP"`)。

  Kinesis 資料串流中要從中讀取資料的起始位置。可能的值包括 `"latest"`、`"trim_horizon"`、`"earliest"` 或 `yyyy-mm-ddTHH:MM:SSZ` 模式中 UTC 格式的時間戳記字串 (其中 Z 代表以 +/- 表示的 UTC 時區偏移。例如："2023-04-04T08:00:00-04:00")。預設值為 `"latest"`。

  注意：只有 4.0 AWS Glue 版或更新版本才支援使用 UTC 格式的時間戳記字串作為「起始位置」的值。

- `MaxFetchTimeInMs` – 數字 (long)，不可大於 None (無)。

  工作執行程式從 Kinesis 資料串流讀取目前批次記錄所花費的時間上限 (毫秒)。在這段時間內可以進行多個 `GetRecords` API 呼叫。預設值為 `1000`。

- `MaxFetchRecordsPerShard` – 數字 (long)，不可大於 None (無)。

每個微批次在 Kinesis 資料串流中，每個碎片可擷取的最大記錄數。注意：如果串流工作已讀取 Kinesis 的額外記錄 (在相同的 Get-record 呼叫中)，用戶端可能會超過此限制。如果MaxFetchRecordsPerShard需要嚴格，那麼它需要是MaxRecordPerRead. 預設值為100000。

- MaxRecordPerRead – 數字 (long)，不可大於 None (無)。

  要從每個 getRecords 操作的 Kinesis 資料串流中擷取的記錄數量上限。預設值為 10000。

- AddIdleTimeBetweenReads – 布林值。

  增加兩個連續 getRecords 操作之間的時間延遲。預設值為 "False"。此選項僅在 Glue 2.0 及以上版本上才可設定。

- IdleTimeBetweenReadsInMs – 數字 (long)，不可大於 None (無)。

  連續兩個 getRecords 操作之間的最小延遲時間，以毫秒為單位指定。預設值為 1000。此選項僅在 Glue 2.0 及以上版本上才可設定。

- DescribeShardInterval – 數字 (long)，不可大於 None (無)。

  兩個 ListShards API 調用之間的最短時間間隔，以供腳本考慮重新分片。預設值為 1s。

- NumRetries – 數字 (整數)，不可大於 None (無)。

  Kinesis Data Streams API 請求的重試數上限。預設值為 3。

- RetryIntervalMs – 數字 (long)，不可大於 None (無)。

  重試 Kinesis Data Streams API 呼叫之前的冷卻時間期間 (以毫秒為單位)。預設值為 1000。

- MaxRetryIntervalMs – 數字 (long)，不可大於 None (無)。

  Kinesis Data Streams API 呼叫之兩次重試之間的最大冷卻時間期間 (以毫秒為單位)。預設值為 10000。

- AvoidEmptyBatches – 布林值。

  避免建立空白微批次任務，方法是在批次開始之前檢查 Kinesis 資料串流中是否有未讀取的資料。預設值為 "False"。

- StreamArn – UTF-8 字串，需符合Custom string pattern #40。

  Kinesis 資料串流的 Amazon Resource Name (ARN)。

- RoleArn – UTF-8 字串，需符合Custom string pattern #40。

使用 AWS Security Token Service (AWS STS) 擔任之角色的 Amazon Resource Name (ARN)。此角色必須具有描述或讀取 Kinesis 資料串流記錄操作的許可。存取不同帳戶中的資料串流時，您必須使用此參數。搭配 "awsSTSSessionName" 使用。

- RoleSessionName – UTF-8 字串，需符合Custom string pattern #40。

  使用 AWS STS 擔任角色之工作階段的識別符。存取不同帳戶中的資料串流時，您必須使用此參數。搭配 "awsSTSRoleARN" 使用。

- AddRecordTimestamp – UTF-8 字串，需符合Custom string pattern #40。

  當此選項設定為 'true' 時，資料輸出將包含一個名為 "__src_timestamp" 的額外資料欄，其指示串流收到相應記錄的時間。預設值為 'false'。4.0 AWS Glue 版或更新版本支援此選項。

- EmitConsumerLagMetrics – UTF-8 字串，需符合Custom string pattern #40。

  當此選項設定為 'true' 時，對於每個批次，它會發出串流接收到的最舊記錄到達時間之間的持續時間的 AWS Glue 指標。 CloudWatch該指標的名稱是「膠合. 驅動程序. maxConsumerLagInMs」. 預設值為 'false'。在 AWS Glue 4.0 版或更新版中支援此選項。

- StartingTimestamp – UTF-8 字串。

  Kinesis 資料串流中開始讀取資料之記錄的時間戳記。可能的值是 yyyy-mm-ddTHH:MM:SSZ 模式的 UTC 格式的時間戳記字串 (其中 Z 代表以 +/- 表示的 UTC 時區偏移。例如："2023-04-04T08:00:00+08:00")。

CatalogKafkaSource 結構

指定 Data Catalog 中的 Apache Kafka 資料存放區。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料存放區的名稱。

- WindowSize – 數字 (整數)，不可大於 None (無)。

  處理每個微批次的時間量。

- DetectSchema – 布林值。

  是否自動從傳入資料確定結構描述。

- `Table` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫中資料表的名稱。

- `Database` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫的名稱。

- `StreamingOptions` – KafkaStreamingSourceOptions 物件。

  指定串流選項。

- `DataPreviewOptions` – StreamingDataPreviewOptions 物件。

  指定與資料預覽相關的選項，以檢視資料範例。

DirectKafkaSource 結構

指定 Apache Kafka 資料存放區。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料存放區的名稱。

- `StreamingOptions` – KafkaStreamingSourceOptions 物件。

  指定串流選項。

- `WindowSize` – 數字 (整數)，不可大於 None (無)。

  處理每個微批次的時間量。

- `DetectSchema` – 布林值。

  是否自動從傳入資料確定結構描述。

- `DataPreviewOptions` – StreamingDataPreviewOptions 物件。

  指定與資料預覽相關的選項，以檢視資料範例。

KafkaStreamingSourceOptions 結構

其他串流選項。

欄位

- `BootstrapServers` – UTF-8 字串，需符合Custom string pattern #40。

  自舉伺服器 URL 的清單，例如 `b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094`。此選項必須在 API 呼叫中指定，或在 Data Catalog 的資料表中繼資料中定義。

- `SecurityProtocol` – UTF-8 字串，需符合Custom string pattern #40。

  用來與代理程式通訊的協定。可能的值為 `"SSL"` 或 `"PLAINTEXT"`。

- `ConnectionName` – UTF-8 字串，需符合Custom string pattern #40。

  連線的名稱。

- `TopicName` – UTF-8 字串，需符合Custom string pattern #40。

  在 Apache Kafka 中指定的主題名稱。您必須指定至少 1 個 `"topicName"`、`"assign"` 或 `"subscribePattern"`。

- `Assign` – UTF-8 字串，需符合Custom string pattern #40。

  要取用的特定 `TopicPartitions`。您必須指定至少 1 個 `"topicName"`、`"assign"` 或 `"subscribePattern"`。

- `SubscribePattern` – UTF-8 字串，需符合Custom string pattern #40。

  識別要訂閱的主題清單的 Java regex 字串。您必須指定至少 1 個 `"topicName"`、`"assign"` 或 `"subscribePattern"`。

- `Classification` – UTF-8 字串，需符合Custom string pattern #40。

  選擇性分類。

- `Delimiter` – UTF-8 字串，需符合Custom string pattern #40。

  指定分隔符號字元。

- `StartingOffsets` – UTF-8 字串，需符合Custom string pattern #40。

  要從中讀取資料的 Kafka 主題的起始位置。可能的值為 `"earliest"` 或 `"latest"`。預設值為 `"latest"`。

- `EndingOffsets` – UTF-8 字串，需符合Custom string pattern #40。

批次查詢結束時的終點。可能值為 "latest" 或指定每個 TopicPartition 結束偏移的 JSON 字串。

- PollTimeoutMs – 數字 (long)，不可大於 None (無)。

  在 Spark 任務執行器中從 Kafka 輪詢資料的逾時 (以毫秒為單位)。預設值為 512。

- NumRetries – 數字 (整數)，不可大於 None (無)。

  擷取 Kafka 位移失敗之前，要重試的次數。預設值為 3。

- RetryIntervalMs – 數字 (long)，不可大於 None (無)。

  重試擷取 Kafka 偏移量之前等待的時間 (毫秒)。預設值為 10。

- MaxOffsetsPerTrigger – 數字 (long)，不可大於 None (無)。

  每個觸發程序間隔所處理之偏移數目上限的速率限制。指定的偏移總數會按比例跨 topicPartitions 或不同磁碟區而分割。預設值為 null，這表示消費者讀取所有偏移，直到已知的最新偏移。

- MinPartitions – 數字 (整數)，不可大於 None (無)。

  從 Kafka 讀取所需的分割區最小數量。預設值為 null，這表示 Spark 分割區的數量等於 Kafka 分割區的數量。

- IncludeHeaders – 布林值。

  是否包括 Kafka 標頭。當選項設定為「true」時，資料輸出將包含一個名為「glue_streaming_kafka_headers」的額外欄，其類型為 Array[Struct(key: String, value: String)]。預設值為 "false"。此選項僅適用於 3.0 AWS Glue 版或更高版本。

- AddRecordTimestamp – UTF-8 字串，需符合Custom string pattern #40。

  當此選項設定為 'true' 時，資料輸出將包含一個名為 "__src_timestamp" 的額外資料欄，其指示主題收到相應記錄的時間。預設值為 'false'。4.0 AWS Glue 版或更新版本支援此選項。

- EmitConsumerLagMetrics – UTF-8 字串，需符合Custom string pattern #40。

  當此選項設置為 'true' 時，對於每個批次，它將發出主題接收到的最舊記錄和到達時間之間的持續時間的 AWS Glue 指標。 CloudWatch該指標的名稱是「膠合. 驅動程序. maxConsumerLagInMs」. 預設值為 'false'。在 AWS Glue 4.0 版或更新版中支援此選項。

- StartingTimestamp – UTF-8 字串。

Kafka 主題中開始讀取資料之記錄的時間戳記。可能的值是 `yyyy-mm-ddTHH:MM:SSZ` 模式的 UTC 格式的時間戳記字串 (其中 Z 代表以 +/- 表示的 UTC 時區偏移。例如："2023-04-04T08:00:00+08:00")。

只能設定 `StartingTimestamp` 或 `StartingOffsets`。

RedshiftSource 結構

指定 Amazon Redshift 資料存放區。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  Amazon Redshift 資料存放區的名稱。
- `Database` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取的資料庫。
- `Table` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取的資料庫資料表。
- `RedshiftTmpDir` – UTF-8 字串，需符合Custom string pattern #40。

  從資料庫複製時，可用來暫存臨時資料的 Amazon S3 路徑。
- `TmpDirIAMRole` – UTF-8 字串，需符合Custom string pattern #40。

  具有許可的 IAM 角色。

AmazonRedshiftSource 結構

指定 Amazon Redshift 來源。

欄位

- `Name` – UTF-8 字串，需符合Custom string pattern #43。

  Amazon Redshift 來源的名稱。
- `Data` – AmazonRedshiftNodeData 物件。

指定 Amazon Reshift 來源節點的資料。

AmazonRedshiftNodeData 結構

指定 Amazon Redshift 節點。

欄位

- `AccessType` – UTF-8 字串，需符合Custom string pattern #39。

  Redshift 連線的存取類型。可以是直接連線或型錄連線。
- `SourceType` – UTF-8 字串，需符合Custom string pattern #39。

  用來指定特定資料表是來源還是自訂查詢的來源類型。
- `Connection` – 選項 物件。

  Redshift 叢集的 AWS Glue 連線。
- `Schema` – 選項 物件。

  使用直接連線時的 Redshift 結構描述名稱。
- `Table` – 選項 物件。

  使用直接連線時的 Redshift 資料表名稱。
- `CatalogDatabase` – 選項 物件。

  使用 AWS Glue 資料目錄時資料目錄資料庫的名稱。
- `CatalogTable` – 選項 物件。

  使用 AWS Glue 資料目錄時的「資料目錄」表格名稱。
- `CatalogRedshiftSchema` – UTF-8 字串。

  使用資料型錄時的 Redshift 結構描述名稱。
- `CatalogRedshiftTable` – UTF-8 字串。

  要讀取的資料庫資料表。
- `TempDir` – UTF-8 字串，需符合Custom string pattern #40。

  從資料庫複製時，可用來暫存臨時資料的 Amazon S3 路徑。

- `IamRole` – 選項 物件。

  選用。連線到 S3 時使用的角色名稱。當保留空白時，IAM 角色預設為任務上的角色。

- `AdvancedOptions` – 一個 AmazonRedshiftAdvancedOption 物件陣列。

  連線至 Redshift 叢集時的選用值。

- `SampleQuery` – UTF-8 字串。

  當 SourceType是「查詢」時，用於從 Redshift 源獲取數據的 SQL。

- `PreAction` – UTF-8 字串。

  執行帶有 upsert 的 MERGE 或 APPEND 之前使用的 SQL。

- `PostAction` – UTF-8 字串。

  執行帶有 upsert 的 MERGE 或 APPEND 之前使用的 SQL。

- `Action` – UTF-8 字串。

  指定寫入 Redshift 叢集的方式。

- `TablePrefix` – UTF-8 字串，需符合Custom string pattern #39。

  指定資料表的字首。

- `Upsert` – 布林值。

  執行 APPEND 時，對 Redshift 接收器使用的動作。

- `MergeAction` – UTF-8 字串，需符合Custom string pattern #39。

  確定如何處理 Redshift 接收器中的 MERGE 時使用的動作。

- `MergeWhenMatched` – UTF-8 字串，需符合Custom string pattern #39。

  當現有記錄與新記錄相符時，確定如何處理 Redshift 接收器中的 MERGE 時使用的動作。

- `MergeWhenNotMatched` – UTF-8 字串，需符合Custom string pattern #39。

  當現有記錄與新記錄不符時，確定如何處理 Redshift 接收器中的 MERGE 時使用的動作。

- `MergeClause` – UTF-8 字串。

  在自訂合併中用於處理相符記錄的 SQL。

- `CrawlerConnection` – UTF-8 字串。

指定與所用型錄資料表相關聯的連線名稱。

- `TableSchema` – 一個 選項 物件陣列。

  指定節點的結構描述輸出陣列。

- `StagingTable` – UTF-8 字串。

  執行帶有 upsert 的 MERGE 或 APPEND 時使用的臨時暫存資料表名稱。

- `SelectedColumns` – 一個 選項 物件陣列。

  當執行帶有 upsert 的 MERGE 或 APPEND 時，用於確定相符記錄的資料欄名稱清單。

AmazonRedshiftAdvancedOption 結構

連線至 Redshift 叢集時指定選用值。

欄位

- `Key` – UTF-8 字串。

  其他連線選項的金鑰。

- `Value` – UTF-8 字串。

  其他連線選項的值。

選項結構

指定選項值。

欄位

- `Value` – UTF-8 字串，需符合Custom string pattern #40。

  指定選項的值。

- `Label` – UTF-8 字串，需符合Custom string pattern #40。

  指定選項的標籤。

- `Description` – UTF-8 字串，需符合Custom string pattern #40。

指定選項的描述。

S3 CatalogSource 結構

在資料目錄中指定 Amazon S3 資 AWS Glue 料存放區。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料存放區的名稱。

- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取的資料庫。

- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取的資料庫資料表。

- PartitionPredicate – UTF-8 字串，需符合Custom string pattern #40。

  滿足此述詞的分割區會被刪除。這些分割區中仍在保留期間內的檔案不會被刪除。設定為 "" – 預設為空值。

- AdditionalOptions – S3 SourceAdditionalOptions 物件。

  指定其他連接選項。

S3 SourceAdditionalOptions 結構

指定 Amazon S3 資料存放區的其他連線選項。

欄位

- BoundedSize – 數字 (long)。

  設定要處理之資料集的目標大小上限 (以位元組為單位)。

- BoundedFiles – 數字 (long)。

  設定要處理的檔案目標數目的上限。

S3 CsvSource 結構

指定存放在 Amazon S3 中的命令分隔值 (CSV) 資料存放區。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料存放區的名稱。

- `Paths` – 必要：UTF-8 字串陣列。

  要讀取的 Amazon S3 路徑清單。

- `CompressionType` – UTF-8 字串 (有效值：`gzip="GZIP"` | `bzip2="BZIP2"`)。

  指定資料的壓縮方式。一般來說，如果資料具有標準副檔名，則不需要此項目。可能值為 `"gzip"` 和 `"bzip"`。

- `Exclusions` – UTF-8 字串陣列。

  包含要排除之 Unix 樣式 glob 模式 JSON 清單的字串。例如，"[\"**.pdf\"]" 會排除所有 PDF 檔案。

- `GroupSize` – UTF-8 字串，需符合Custom string pattern #40。

  目標群組大小 (以位元組為單位)。系統會根據輸入資料大小和叢集大小來計算預設值。當輸入檔案數少於 50,000 個時，`"groupFiles"` 必須設定為 `"inPartition"` 才能讓此設定生效。

- `GroupFiles` – UTF-8 字串，需符合Custom string pattern #40。

  當輸入含有超過 50,000 個檔案時，預設會開啟分組檔案。若要在少於 50,000 個檔案時開啟分組，請將此參數設定為 "inPartition"。若要在超過 50,000 個檔案時停用分組，請將此參數設定為 `"none"`。

- `Recurse` – 布林值。

  如果設定為 True，則會遞迴讀取指定路徑下所有子目錄中的檔案。

- `MaxBand` – 數字 (整數)，不可大於 None (無)。

  此選項可控制 s3 清單可能會在多長時間 (以毫秒為單位) 後變得一致。修改時間戳記落在最後 maxBand 毫秒內的檔案會在使用時特別追蹤， JobBookmarks以解決 Amazon S3 最終一致性。使用者大多不需要設定此選項。預設值為 900000 毫秒或 15 分鐘。

- `MaxFilesInBand` – 數字 (整數)，不可大於 None (無)。

此選項會指定從最後 maxBand 秒內所要儲存的檔案數上限。如果超過此數量，系統就會略過額外的檔案，等下一個任務執行到來再處理。

- `AdditionalOptions` – S3 DirectSourceAdditionalOptions 物件。

  指定其他連接選項。

- `Separator` – 必要：UTF-8 字串 (有效值：`comma="COMMA"` | `ctrla="CTRLA"` | `pipe="PIPE"` | `semicolon="SEMICOLON"` | `tab="TAB"`)。

  指定分隔符號字元。預設值為逗號：","，但您仍可指定任何其他字元。

- `Escaper` – UTF-8 字串，需符合Custom string pattern #41。

  指定用於逸出的字元。只有在讀取 CSV 檔案時，才會使用此選項。預設值為 none。若啟用，後面緊接的字元會維持現狀，除了一小組眾所皆知的逸出字元 (`\n`、`\r`、`\t` 與 `\0`) 以外。

- `QuoteChar` – 必要：UTF-8 字串 (有效值：`quote="QUOTE"` | `quillemet="QUILLEMET"` | `single_quote="SINGLE_QUOTE"` | `disabled="DISABLED"`)。

  指定用於引用的字元。預設為雙引號：`'"'`。將之設為 `-1` 可完全關閉引用功能。

- `Multiline` – 布林值。

  布林值，用以指定單項記錄是否可以跨越多行。當欄位內含引用的新行字元時，可能就會發生這種情況。若有任何記錄跨越多行，請務必將此選項設為 True。預設值為 `False`，如此在剖析時會更加積極地分割檔案。

- `WithHeader` – 布林值。

  布林值，指定是否要將第一行做為標頭。預設值為 `False`。

- `WriteHeader` – 布林值。

  布林值，指定是否要將標頭寫入輸入之中。預設值為 `True`。

- `SkipFirst` – 布林值。

  布林值，指定是否要略過第一個資料行。預設值為 `False`。

- `OptimizePerformance` – 布林值。

  指定是否要使用進階 SIMD CSV 讀取器，以及 Apache Arrow 為基礎的直欄式記憶體格式。僅在 3.0 AWS Glue 版本中可用。

- `OutputSchemas` – 一個 GlueSchema 物件陣列。

指定自訂 S3 CSV 來源的資料架構。

DirectJDBCSource 結構

指定直接 JDBC 來源連線。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  JDBC 來源連線的名稱。
- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  JDBC 來源連線的資料庫。
- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

  JDBC 來源連線的資料表。
- ConnectionName – 必要：UTF-8 字串，需符合Custom string pattern #40。

  JDBC 來源的連線名稱。
- ConnectionType – 必要：UTF-8 字串 (有效值：sqlserver | mysql | oracle | postgresql | redshift)。

  JDBC 來源的連線類型。
- RedshiftTmpDir – UTF-8 字串，需符合Custom string pattern #40。

  JDBC Redshift 來源的暫存目錄。

S3 DirectSourceAdditionalOptions 結構

指定 Amazon S3 資料存放區的其他連線選項。

欄位

- BoundedSize – 數字 (long)。

  設定要處理之資料集的目標大小上限 (以位元組為單位)。
- BoundedFiles – 數字 (long)。

設定要處理的檔案目標數目的上限。

- EnableSamplePath – 布林值。

  設定選項啟用範例路徑。

- SamplePath – UTF-8 字串，需符合Custom string pattern #40。

  如果啟用，則會指定範例路徑。

S3 JsonSource 結構

指定儲存在 Amazon S3 中的 JSON 資料存放區。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料存放區的名稱。

- Paths – 必要：UTF-8 字串陣列。

  要讀取的 Amazon S3 路徑清單。

- CompressionType – UTF-8 字串 (有效值：gzip="GZIP" | bzip2="BZIP2")。

  指定資料的壓縮方式。一般來說，如果資料具有標準副檔名，則不需要此項目。可能值為 "gzip" 和 "bzip"。

- Exclusions – UTF-8 字串陣列。

  包含要排除之 Unix 樣式 glob 模式 JSON 清單的字串。例如，"[\"**.pdf\"]" 會排除所有 PDF 檔案。

- GroupSize – UTF-8 字串，需符合Custom string pattern #40。

  目標群組大小 (以位元組為單位)。系統會根據輸入資料大小和叢集大小來計算預設值。當輸入檔案數少於 50,000 個時，"groupFiles" 必須設定為 "inPartition" 才能讓此設定生效。

- GroupFiles – UTF-8 字串，需符合Custom string pattern #40。

  當輸入含有超過 50,000 個檔案時，預設會開啟分組檔案。若要在少於 50,000 個檔案時開啟分組，請將此參數設定為 "inPartition"。若要在超過 50,000 個檔案時停用分組，請將此參數設定為 "none"。

- Recurse – 布林值。

如果設定為 True，則會遞迴讀取指定路徑下所有子目錄中的檔案。

- `MaxBand` – 數字 (整數)，不可大於 None (無)。

  此選項可控制 s3 清單可能會在多長時間 (以毫秒為單位) 後變得一致。修改時間戳記落在最後 maxBand 毫秒內的檔案會在使用時特別追蹤， JobBookmarks以解決 Amazon S3 最終一致性。使用者大多不需要設定此選項。預設值為 900000 毫秒或 15 分鐘。

- `MaxFilesInBand` – 數字 (整數)，不可大於 None (無)。

  此選項會指定從最後 maxBand 秒內所要儲存的檔案數上限。如果超過此數量，系統就會略過額外的檔案，等下一個任務執行到來再處理。

- `AdditionalOptions` – S3 DirectSourceAdditionalOptions 物件。

  指定其他連接選項。

- `JsonPath` – UTF-8 字串，需符合Custom string pattern #40。

  定義 JsonPath JSON 資料的字串。

- `Multiline` – 布林值。

  布林值，用以指定單項記錄是否可以跨越多行。當欄位內含引用的新行字元時，可能就會發生這種情況。若有任何記錄跨越多行，請務必將此選項設為 True。預設值為 `False`，如此在剖析時會更加積極地分割檔案。

- `OutputSchemas` – 一個 GlueSchema 物件陣列。

  指定自訂 S3 JSON 來源的資料架構。

S3 ParquetSource 結構

指定存放在 Amazon S3 中的 Apache Parquet 資料存放區。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料存放區的名稱。

- `Paths` – 必要：UTF-8 字串陣列。

  要讀取的 Amazon S3 路徑清單。

- `CompressionType` – UTF-8 字串 (有效值：`snappy="SNAPPY"` | `lzo="LZO"` | `gzip="GZIP"` | `uncompressed="UNCOMPRESSED"` | `none="NONE"`)。

  指定資料的壓縮方式。一般來說，如果資料具有標準副檔名，則不需要此項目。可能值為 `"gzip"` 和 `"bzip"`。

- `Exclusions` – UTF-8 字串陣列。

  包含要排除之 Unix 樣式 glob 模式 JSON 清單的字串。例如，"[\"**.pdf\"]" 會排除所有 PDF 檔案。

- `GroupSize` – UTF-8 字串，需符合Custom string pattern #40。

  目標群組大小 (以位元組為單位)。系統會根據輸入資料大小和叢集大小來計算預設值。當輸入檔案數少於 50,000 個時，`"groupFiles"` 必須設定為 `"inPartition"` 才能讓此設定生效。

- `GroupFiles` – UTF-8 字串，需符合Custom string pattern #40。

  當輸入含有超過 50,000 個檔案時，預設會開啟分組檔案。若要在少於 50,000 個檔案時開啟分組，請將此參數設定為 "inPartition"。若要在超過 50,000 個檔案時停用分組，請將此參數設定為 `"none"`。

- `Recurse` – 布林值。

  如果設定為 True，則會遞迴讀取指定路徑下所有子目錄中的檔案。

- `MaxBand` – 數字 (整數)，不可大於 None (無)。

  此選項可控制 s3 清單可能會在多長時間 (以毫秒為單位) 後變得一致。修改時間戳記落在最後 maxBand 毫秒內的檔案會在使用時特別追蹤， JobBookmarks以解決 Amazon S3 最終一致性。使用者大多不需要設定此選項。預設值為 900000 毫秒或 15 分鐘。

- `MaxFilesInBand` – 數字 (整數)，不可大於 None (無)。

  此選項會指定從最後 maxBand 秒內所要儲存的檔案數上限。如果超過此數量，系統就會略過額外的檔案，等下一個任務執行到來再處理。

- `AdditionalOptions` – S3 DirectSourceAdditionalOptions 物件。

  指定其他連接選項。

- `OutputSchemas` – 一個 GlueSchema 物件陣列。

  指定自訂 S3 Parquet 來源的資料架構。

S3 DeltaSource 結構

指定儲存於中的三角洲湖資料來源 Amazon S3。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  Delta Lake 來源的名稱。
- Paths – 必要：UTF-8 字串陣列。

  要讀取的 Amazon S3 路徑清單。
- AdditionalDeltaOptions – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  指定其他連接選項。
- AdditionalOptions – S3 DirectSourceAdditionalOptions 物件。

  指定連接器的其他選項。
- OutputSchemas – 一個 GlueSchema 物件陣列。

  指定 Delta Lake 來源的資料結構描述。

S3 CatalogDeltaSource 結構

指定已在「資料目錄」中註冊的 Delta 湖資 AWS Glue 料來源。資料來源必須儲存在中 Amazon S3。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  Delta Lake 資料來源的名稱。
- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫的名稱。
- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

要讀取之資料庫中資料表的名稱。

- `AdditionalDeltaOptions` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  指定其他連接選項。

- `OutputSchemas` – 一個 GlueSchema 物件陣列。

  指定 Delta Lake 來源的資料結構描述。

CatalogDeltaSource 結構

指定已在「資料目錄」中註冊的 Delta 湖資 AWS Glue 料來源。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  Delta Lake 資料來源的名稱。

- `Database` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫的名稱。

- `Table` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫中資料表的名稱。

- `AdditionalDeltaOptions` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  指定其他連接選項。

- `OutputSchemas` – 一個 GlueSchema 物件陣列。

  指定 Delta Lake 來源的資料結構描述。

S3 HudiSource 結構

指定儲存於 Amazon S3中的 Hudi 資料來源。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  Hudi 來源的名稱。
- Paths – 必要：UTF-8 字串陣列。

  要讀取的 Amazon S3 路徑清單。
- AdditionalHudiOptions – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  指定其他連接選項。
- AdditionalOptions – S3 DirectSourceAdditionalOptions 物件。

  指定連接器的其他選項。
- OutputSchemas – 一個 GlueSchema 物件陣列。

  指定 Hudi 來源的資料結構描述。

S3 CatalogHudiSource 結構

指定已在資料目錄中註冊的 Hudi AWS Glue 資料來源。Hudi 資料來源必須儲存在中 Amazon S3。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  Hudi 資料來源的名稱。
- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫的名稱。
- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

要讀取之資料庫中資料表的名稱。

- `AdditionalHudiOptions` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  指定其他連接選項。

- `OutputSchemas` – 一個 GlueSchema 物件陣列。

  指定 Hudi 來源的資料結構描述。

CatalogHudiSource 結構

指定已在資料目錄中註冊的 Hudi AWS Glue 資料來源。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  Hudi 資料來源的名稱。

- `Database` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫的名稱。

- `Table` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫中資料表的名稱。

- `AdditionalHudiOptions` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  指定其他連接選項。

- `OutputSchemas` – 一個 GlueSchema 物件陣列。

  指定 Hudi 來源的資料結構描述。

DynamoDB CatalogSource 結構

在資料目錄中指定 DynamoDB 資 AWS Glue 料來源。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料來源的名稱。
- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫的名稱。
- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫中資料表的名稱。

RelationalCatalogSource 結構

指定 AWS Glue Data Catalog 中的關聯式資料庫資料來源。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料來源的名稱。
- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫的名稱。
- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取之資料庫中資料表的名稱。

JDBC ConnectorTarget 結構

指定以 Apache Parquet 直欄式儲存寫入 Amazon S3 的資料目標。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

資料目標的名稱。

- `Inputs` – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。

- `ConnectionName` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  與連接器相關聯之連線的名稱。

- `ConnectionTable` – 必要：UTF-8 字串，需符合Custom string pattern #41。

  資料目標中的資料表名稱。

- `ConnectorName` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  將要使用的連接器名稱。

- `ConnectionType` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  連線類型，例如 marketplace.jdbc 或 custom.jdbc，指定與 JDBC 資料目標的連線。

- `AdditionalOptions` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  連接器的其他連接選項。

- `OutputSchemas` – 一個 GlueSchema 物件陣列。

  指定 JDBC 目標的資料架構。

SparkConnectorTarget 結構

指定使用 Apache Spark 連接器的目標。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。

- `Inputs` – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

輸入到資料目標的節點。

- `ConnectionName` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  Apache Spark 連接器的連線名稱。

- `ConnectorName` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  Apache Spark 連接器的名稱。

- `ConnectionType` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  連接的類型，如 marketplace.spark 或 custom.spark，指定 Apache Spark 資料存放區的連線。

- `AdditionalOptions` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  連接器的其他連接選項。

- `OutputSchemas` – 一個 GlueSchema 物件陣列。

  指定自訂 spark 目標的資料架構。

BasicCatalogTarget 結構

指定使用「 AWS Glue 資料目錄」表格的目標。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。

- `Inputs` – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。

- `Database` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  從清單中選擇包含要用作目標之資料表的資料庫。此資料庫必須存在於 Data Catalog 中。

- `Table` – 必要：UTF-8 字串，需符合Custom string pattern #40。

定義輸出資料結構描述的資料表。此資料表必須已存在於 Data Catalog 中。

MySQL CatalogTarget 結構

指定使用 MySQL 的目標。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。

- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。

- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫的名稱。

- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫中資料表的名稱。

PostgreSQL CatalogTarget 結構

指定使用 Postgres SQL 的目標。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。

- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。

- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫的名稱。

- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

要寫入之資料庫中資料表的名稱。

OracleSQL CatalogTarget 結構

指定使用 Oracle SQL 的目標。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。

- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。

- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫的名稱。

- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫中資料表的名稱。

微软结构 ServerCatalogTarget

指定使用 Microsoft SQL 的目標。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。

- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。

- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫的名稱。

- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

要寫入之資料庫中資料表的名稱。

RedshiftTarget 結構

指定使用 Amazon Redshift 的目標。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。

- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。

- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫的名稱。

- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫中資料表的名稱。

- RedshiftTmpDir – UTF-8 字串，需符合Custom string pattern #40。

  從資料庫複製時，可用來暫存臨時資料的 Amazon S3 路徑。

- TmpDirIAMRole – UTF-8 字串，需符合Custom string pattern #40。

  具有許可的 IAM 角色。

- UpsertRedshiftOptions – UpsertRedshiftTargetOptions 物件。

  寫入 Redshift 目標時設定 upsert 操作的選項集。

AmazonRedshiftTarget 結構

指定 Amazon Redshift 目標。

欄位

- Name – UTF-8 字串，需符合Custom string pattern #43。

Amazon Redshift 目標的名稱。

- `Data` – AmazonRedshiftNodeData 物件。

  指定 Amazon Redshift 目標節點的資料。

- `Inputs`：UTF-8 字串陣列，不可小於 1，也不可超過 1 個字串。

  輸入到資料目標的節點。

UpsertRedshiftTargetOptions 結構

寫入 Redshift 目標時設定 upsert 操作的選項。

欄位

- `TableLocation` – UTF-8 字串，需符合Custom string pattern #40。

  Redshift 資料表的實體位置。

- `ConnectionName` – UTF-8 字串，需符合Custom string pattern #40。

  用來寫入 Redshift 的連線名稱。

- `UpsertKeys` – UTF-8 字串陣列。

  用於確定是執行更新還是插入的金鑰。

S3 CatalogTarget 結構

指定使用資料目錄寫入 Amazon S3 的 AWS Glue 資料目標。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。

- `Inputs` – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。

- `PartitionKeys` – UTF-8 字串陣列。

指定使用一系列索引鍵的原生分割。

- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫中資料表的名稱。

- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫的名稱。

- SchemaChangePolicy – CatalogSchemaChangePolicy 物件。

  可以針對爬蟲程式指定更新行為的政策。

S3 GlueParquetTarget 結構

指定以 Apache Parquet 直欄式儲存寫入 Amazon S3 的資料目標。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。

- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。

- PartitionKeys – UTF-8 字串陣列。

  指定使用一系列索引鍵的原生分割。

- Path – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入的單一 Amazon S3 路徑。

- Compression – UTF-8 字串 (有效值：snappy="SNAPPY" | lzo="LZO" | gzip="GZIP" | uncompressed="UNCOMPRESSED" | none="NONE")。

  指定資料的壓縮方式。一般來說，如果資料具有標準副檔名，則不需要此項目。可能值為 "gzip" 和 "bzip"。

- SchemaChangePolicy – DirectSchemaChangePolicy 物件。

  可以針對爬蟲程式指定更新行為的政策。

CatalogSchemaChangePolicy 結構

可以針對爬蟲程式指定更新行為的政策。

欄位

- EnableUpdateCatalog – 布林值。

  爬蟲程式找到變更的結構描述時是否使用指定的更新行為。
- UpdateBehavior – UTF-8 字串 (有效值：UPDATE_IN_DATABASE | LOG)。

  爬蟲程式找到變更結構描述時的更新行為。

S3 DirectTarget 結構

指定寫入 Amazon S3 的資料目標。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。
- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。
- PartitionKeys – UTF-8 字串陣列。

  指定使用一系列索引鍵的原生分割。
- Path – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入的單一 Amazon S3 路徑。
- Compression – UTF-8 字串，需符合Custom string pattern #40。

  指定資料的壓縮方式。一般來說，如果資料具有標準副檔名，則不需要此項目。可能值為 "gzip" 和 "bzip"。
- Format – 必要：UTF-8 字串 (有效值：json="JSON" | csv="CSV" | avro="AVRO" | orc="ORC" | parquet="PARQUET" | hudi="HUDI" | delta="DELTA")。

  指定目標的資料輸出格式。

- `SchemaChangePolicy` – DirectSchemaChangePolicy 物件。

  可以針對爬蟲程式指定更新行為的政策。

S3 HudiCatalogTarget 結構

指定寫入資料目錄中 Hudi 資料來源的 AWS Glue 目標。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。
- `Inputs` – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。
- `PartitionKeys` – UTF-8 字串陣列。

  指定使用一系列索引鍵的原生分割。
- `Table` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫中資料表的名稱。
- `Database` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫的名稱。
- `AdditionalOptions` – 必要：金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  指定連接器的其他連接選項。
- `SchemaChangePolicy` – CatalogSchemaChangePolicy 物件。

  可以針對爬蟲程式指定更新行為的政策。

S3 HudiDirectTarget 結構

指定寫入中 Amazon S3 Hudi 資料來源的目標。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。

- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。

- Path – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入 Hudi 資料來源的 Amazon S3 路徑。

- Compression – 必要：UTF-8 字串 (有效值：gzip="GZIP" | lzo="LZO" | uncompressed="UNCOMPRESSED" | snappy="SNAPPY")。

  指定資料的壓縮方式。一般來說，如果資料具有標準副檔名，則不需要此項目。可能值為 "gzip" 和 "bzip"。

- PartitionKeys – UTF-8 字串陣列。

  指定使用一系列索引鍵的原生分割。

- Format – 必要：UTF-8 字串 (有效值：json="JSON" | csv="CSV" | avro="AVRO" | orc="ORC" | parquet="PARQUET" | hudi="HUDI" | delta="DELTA")。

  指定目標的資料輸出格式。

- AdditionalOptions – 必要：金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  指定連接器的其他連接選項。

- SchemaChangePolicy – DirectSchemaChangePolicy 物件。

  可以針對爬蟲程式指定更新行為的政策。

S3 DeltaCatalogTarget 結構

指定寫入資料目錄中 Delta Lake 資料來源的 AWS Glue 目標。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。
- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。
- PartitionKeys – UTF-8 字串陣列。

  指定使用一系列索引鍵的原生分割。
- Table – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫中資料表的名稱。
- Database – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫的名稱。
- AdditionalOptions – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  指定連接器的其他連接選項。
- SchemaChangePolicy – CatalogSchemaChangePolicy 物件。

  可以針對爬蟲程式指定更新行為的政策。

S3 DeltaDirectTarget 結構

在中指定寫入 Delta 湖資料來源的目標 Amazon S3。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。
- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。

- `PartitionKeys` – UTF-8 字串陣列。

  指定使用一系列索引鍵的原生分割。

- `Path` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入 Delta Lake 資料來源的 Amazon S3 路徑。

- `Compression` – 必要：UTF-8 字串 (有效值：`uncompressed="UNCOMPRESSED"` | `snappy="SNAPPY"`)。

  指定資料的壓縮方式。一般來說，如果資料具有標準副檔名，則不需要此項目。可能值為 `"gzip"` 和 `"bzip"`。

- `Format` – 必要：UTF-8 字串 (有效值：`json="JSON"` | `csv="CSV"` | `avro="AVRO"` | `orc="ORC"` | `parquet="PARQUET"` | `hudi="HUDI"` | `delta="DELTA"`)。

  指定目標的資料輸出格式。

- `AdditionalOptions` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  指定連接器的其他連接選項。

- `SchemaChangePolicy` – DirectSchemaChangePolicy 物件。

  可以針對爬蟲程式指定更新行為的政策。

DirectSchemaChangePolicy 結構

可以針對爬蟲程式指定更新行為的政策。

欄位

- `EnableUpdateCatalog` – 布林值。

  爬蟲程式找到變更的結構描述時是否使用指定的更新行為。

- `UpdateBehavior` – UTF-8 字串 (有效值：`UPDATE_IN_DATABASE` | `LOG`)。

  爬蟲程式找到變更結構描述時的更新行為。

- `Table` – UTF-8 字串，需符合Custom string pattern #40。

指定資料庫中套用結構描述變更政策的資料表。

- Database – UTF-8 字串，需符合Custom string pattern #40。

  指定套用結構描述變更政策的資料庫。

ApplyMapping 結構

指定將資料來源中的資料屬性索引鍵映射至資料目標中資料屬性索引鍵的轉換。您可以重新命名索引鍵、修改索引鍵的資料類型，以及選擇要從資料集中捨棄哪些索引鍵。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。

- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  由其節點名稱識別的資料輸入。

- Mapping – 必要：一個 映射 物件。

  將資料來源中的資料屬性索引鍵映射至資料目標中的資料屬性索引鍵。

Mapping 結構

指定資料屬性索引鍵的映射。

欄位

- ToKey – UTF-8 字串，需符合Custom string pattern #40。

  套用映射之後，資料行應具備的名稱。可以與 FromPath 相同。

- FromPath – UTF-8 字串陣列。

  要修改的資料表或資料行。

- FromType – UTF-8 字串，需符合Custom string pattern #40。

  要修改之資料的類型。

- ToType – UTF-8 字串，需符合Custom string pattern #40。

要修改資料的資料類型。

- Dropped – 布林值。

  若此值為 true，則移除資料行。

- Children – 一個 映射 物件陣列。

  僅適用於巢套資料結構。如果要變更父結構，同時變更其某個子結構，則可以填寫此資料結構。它也是 Mapping，但其 FromPath 將是父結構的 FromPath，再加上來自此結構的 FromPath。

  對於子部件，假設您的結構如下：

  ```
  { "FromPath": "OuterStructure", "ToKey": "OuterStructure", "ToType":
  "Struct", "Dropped": false, "Chidlren": [{ "FromPath": "inner", "ToKey":
  "inner", "ToType": "Double", "Dropped": false, }] }
  ```

  您可以指定看起來類似如下的 Mapping：

  ```
  { "FromPath": "OuterStructure", "ToKey": "OuterStructure", "ToType":
  "Struct", "Dropped": false, "Chidlren": [{ "FromPath": "inner", "ToKey":
  "inner", "ToType": "Double", "Dropped": false, }] }
  ```

SelectFields 結構

指定選擇要保留之資料屬性索引鍵的轉換。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。

- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  由其節點名稱識別的資料輸入。

- Paths – 必要：UTF-8 字串陣列。

  資料結構中變數的 JSON 路徑。

DropFields 結構

指定選擇要捨棄之資料屬性索引鍵的轉換。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。
- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  由其節點名稱識別的資料輸入。
- Paths – 必要：UTF-8 字串陣列。

  資料結構中變數的 JSON 路徑。

RenameField 結構

指定重新命名單一資料屬性索引鍵的轉換。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。
- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  由其節點名稱識別的資料輸入。
- SourcePath – 必要：UTF-8 字串陣列。

  來源資料的資料結構中變數的 JSON 路徑。
- TargetPath – 必要：UTF-8 字串陣列。

  目標資料的資料結構中變數的 JSON 路徑。

Spigot 結構

指定將資料範例寫入 Amazon S3 儲存貯體的轉換。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。

- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  由其節點名稱識別的資料輸入。

- Path – 必要：UTF-8 字串，需符合Custom string pattern #40。

  Amazon S3 中的路徑，其中轉換會將記錄子集從資料集寫入 Amazon S3 儲存貯體中的 JSON 檔案。

- Topk – 數字 (整數)，不可大於 100。

  指定要從資料集開始寫入的記錄數目。

- Prob – 數字 (雙字)，不可大於 1。

  挑選任何給定記錄的概率 (最大值為 1 的小數值)。值 1 表示從資料集讀取的每一列應包含在範例輸出中。

Join 結構

使用指定資料屬性索引鍵上的比較片語，將兩個資料集聯結為一個資料集。可以使用內、外、左、右、左半、左反聯結。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。

- Inputs – 必要：UTF-8 字串的陣列，不可小於 2 或超過 2 個字串。

  由其節點名稱識別的資料輸入。

- JoinType – 必要：UTF-8 字串 (有效值：`equijoin="EQUIJOIN"` | `left="LEFT"` | `right="RIGHT"` | `outer="OUTER"` | `leftsemi="LEFT_SEMI"` | `leftanti="LEFT_ANTI"`)。

  指定要在資料集上執行的聯結類型。

- Columns – 必要：JoinColumn 物件陣列，不小於 2 個結構，也不大於 2 個結構。

要聯結的兩個資料行的清單。

JoinColumn 結構

指定要聯結的資料行。

欄位

- From – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要接合的資料行。
- Keys – 必要：UTF-8 字串陣列。

  要聯結之資料行的索引鍵。

SplitFields 結構

指定將資料屬性索引鍵分割成兩個 DynamicFrames 的轉換。輸出是 DynamicFrames 的集合：一個具有所選資料屬性索引鍵，另一個具有其餘資料屬性索引鍵。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。
- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  由其節點名稱識別的資料輸入。
- Paths – 必要：UTF-8 字串陣列。

  資料結構中變數的 JSON 路徑。

SelectFromCollection 結構

指定從 DynamicFrames 的集合選擇一個 DynamicFrame 的轉換。輸出為所選的 DynamicFrame。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

轉換節點的名稱。

- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  由其節點名稱識別的資料輸入。
- Index – 必要：數字 (整數)，不可大於 None (無)。

  要選取 DynamicFrame 的索引。

FillMissingValues 結構

指定如下轉換：尋找遺失值之資料集中的記錄，並新增具有由插補決定值的新欄位。輸入資料集會用於訓練機器學習模型，以決定遺失值應該是什麼。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。
- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  由其節點名稱識別的資料輸入。
- ImputedPath – 必要：UTF-8 字串，需符合Custom string pattern #40。

  輸入資料集的資料結構中變數的 JSON 路徑。
- FilledPath – UTF-8 字串，需符合Custom string pattern #40。

  填充資料集之資料結構中變數的 JSON 路徑。

Filter 結構

指定根據篩選條件將資料集分割成兩個的轉換。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。
- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

由其節點名稱識別的資料輸入。

- `LogicalOperator` – 必要：UTF-8 字串 (有效值：`AND` | `OR`)。

  透過比較索引鍵值與指定值來篩選列的運算子。

- `Filters` – 必要：一個 FilterExpression 物件。

  指定篩選條件表達式。

FilterExpression 結構

指定篩選條件表達式。

欄位

- `Operation` – 必要：UTF-8 字串 (有效值：`EQ` | `LT` | `GT` | `LTE` | `GTE` | `REGEX` | `ISNULL`)。

  要在表達式中執行的操作類型。

- `Negated` – 布林值。

  表達式是否被否定。

- `Values` – 必要：一個 FilterValue 物件。

  篩選條件值清單。

FilterValue 結構

代表在 `FilterExpression` 的值清單中的單一項目。

欄位

- `Type` – 必要：UTF-8 字串 (有效值：`COLUMNEXTRACTED` | `CONSTANT`)。

  篩選條件值的類型。

- `Value` – 必要：UTF-8 字串陣列。

  要關聯的值。

CustomCode 結構

指定使用您提供的自訂程式碼來執行資料轉換的轉換。輸出是的集合 DynamicFrames。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。
- Inputs – 必要：UTF-8 字串的陣列，至少要有 1 個字串。

  由其節點名稱識別的資料輸入。
- Code – 必要：UTF-8 字串，需符合Custom string pattern #35。

  用來執行資料轉換的自訂程式碼。
- ClassName – 必要：UTF-8 字串，需符合Custom string pattern #40。

  為自訂程式碼節點類別定義的名稱。
- OutputSchemas – 一個 GlueSchema 物件陣列。

  指定自訂代碼轉換的資料架構。

SparkSQL 結構

指定轉換，其中輸入使用 Spark SQL 語法的 SQL 查詢來轉換資料。輸出是單個 DynamicFrame。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。
- Inputs – 必要：UTF-8 字串的陣列，至少要有 1 個字串。

  由其節點名稱識別的資料輸入。您可以將表名與 SQL 查詢中使用的每個輸入節點相關聯。您選擇的名稱必須符合 Spark SQL 命名限制。
- SqlQuery – 必要：UTF-8 字串，需符合Custom string pattern #42。

  必須使用 Spark SQL 語法並返回單個資料集的 SQL 查詢。
- SqlAliases – 必要：一個 SqlAlias 物件。

別名清單。別名允許您指定要在 SQL 中為給定輸入使用的名稱。例如，您有一個名為 "" MyDataSource 的資料來源。如果你指定`From`為 MyDataSource，和 `Alias` as SqlName，那麼在你的 SQL 中你可以這樣做：

```
select * from SqlName
```

並從中獲取數據 MyDataSource。

- `OutputSchemas` – 一個 GlueSchema 物件陣列。

  指定 SparkSQL 轉換的資料架構。

SqlAlias 結構

代表在 `SqlAliases` 的值清單中的單一項目。

欄位

- `From` – 必要：UTF-8 字串，需符合Custom string pattern #39。

  資料表或其中的資料行。

- `Alias` – 必要：UTF-8 字串，需符合Custom string pattern #41。

  提供給資料表或其中之資料行的暫時名稱。

DropNullFields 結構

指定轉換，如果資料行中的所有值都為「null」(空)，則從資料集中刪除此行。默認情況下， AWS Glue Studio 將識別空對象，但一些值，如空字符串，字符串是「null」，-1 個整數或其他佔位符，如零，不會自動識別為空值。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。

- `Inputs` – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  由其節點名稱識別的資料輸入。

- `NullCheckBoxList` – NullCheckBoxList 物件。

  表示某些值是否被識別為空值以進行移除的結構。

- `NullTextList` – NullValueField 物件陣列，不可超過 50 個結構。

  一種結構；指定結 NullValueField 構清單，代表自訂 Null 值 (例如零或其他值) 做為資料集唯一的空預留位置。

  此 `DropNullFields` 轉換只會在空預留位置和資料類型的值都符合資料時才移除自訂的空值。

NullCheckBoxList 結構

表示某些值是否被識別為空值以進行移除。

欄位

- `IsEmpty` – 布林值。

  指定一個空字串被視為空值。

- `IsNullString` – 布林值。

  指定拼寫為單字 null 的值被視為空值。

- `IsNegOne` – 布林值。

  指定 -1 的整數值被視為空值。

NullValueField 結構

代表自訂的空值，例如零或用作資料集唯一的空預留位置的其他值。

欄位

- `Value` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  空預留位置的值。

- `Datatype` – 必要：Datatype 物件。

  值的資料類型。

Datatype 結構

代表該值的資料類型的結構。

欄位

- `Id` – 必要：UTF-8 字串，需符合Custom string pattern #39。

  值的資料類型。
- `Label` – 必要：UTF-8 字串，需符合Custom string pattern #39。

  指派給資料類型的標籤。

Merge 結構

指定根據指定的主索引鍵來合併此 `DynamicFrame` 與暫存 `DynamicFrame` 以識別記錄的轉換。重複的記錄 (具有相同主索引鍵的記錄) 不會被刪除重複資料。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。
- `Inputs` – 必要：UTF-8 字串的陣列，不可小於 2 或超過 2 個字串。

  由其節點名稱識別的資料輸入。
- `Source` – 必要：UTF-8 字串，需符合Custom string pattern #39。

  來源 `DynamicFrame`，它將與暫存 `DynamicFrame` 合併。
- `PrimaryKeys` – 必要：UTF-8 字串陣列。

  要從來源和暫存動態影格比對記錄的主索引鍵欄位清單。

Union 結構

指定將兩個或多個資料集中的列合併為單一結果的轉換。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

轉換節點的名稱。

- `Inputs` – 必要：UTF-8 字串的陣列，不可小於 2 或超過 2 個字串。

  輸入到轉換的節點 ID。

- `UnionType` – 必要：UTF-8 字串 (有效值：`ALL` | `DISTINCT`)。

  指示 Union 轉換的類型。

  指定`ALL`將資料來源中的所有列聯結至產生的列 DynamicFrame。產生的聯集不會移除重複的資料列。

  指`DISTINCT`定移除結果中的重複列 DynamicFrame。

PIIDetection 結構

指定用於標識、刪除或遮罩 PII 資料的轉換。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。

- `Inputs` – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到轉換的節點 ID。

- `PiiType` – 必要：UTF-8 字串 (有效值：`RowAudit` | `RowMasking` | `ColumnAudit` | `ColumnMasking`)。

  指示 PIIDetection 轉換的類型。

- `EntityTypesToDetect` – 必要：UTF-8 字串陣列。

  指示 PIIDetection 轉換將標識為 PII 資料的實體類型。

  PII 類型實體包括：
  PERSON_NAME、DATE、USA_SNN、EMAIL、USA_ITIN、USA_PASSPORT_NUMBER、PHONE_NUM

- `OutputColumnName` – UTF-8 字串，需符合Custom string pattern #40。

  針對將在行內包含偵測到的任何實體類型，指示輸出列名。

- `SampleFraction` – 數字 (雙字)，不可大於 1。

  指示掃描 PII 實體時要採樣的資料部分。
- `ThresholdFraction` – 數字 (雙字)，不可大於 1。

  針對要將列標識為 PII 資料時，指示必須滿足的資料部分。
- `MaskValue` – UTF-8 字串，長度不可超過 256 個位元組，且需符合 Custom string pattern #37。

  針對偵測到的實體指示替換值。

Aggregate 結構

指定轉換，依照所選欄位來分組行，並依照指定函數計算彙總值。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。
- `Inputs` – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  指定欄位和行作為彙總轉換輸入。
- `Groups` – 必要：UTF-8 字串陣列。

  指定分組所依據的欄位。
- `Aggs` – 必要：AggregateOperation 物件陣列，不小於 1 個結構，也不大於 30 個結構。

  指定要在指定欄位執行的彙總函數。

DropDuplicates 結構

指定用於從資料集刪除重複資料行的轉換。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  轉換節點的名稱。
- `Inputs` – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

由其節點名稱識別的資料輸入。

- `Columns` – UTF-8 字串陣列。

  重複時要合併或刪除的列名。

GovernedCatalogTarget 結構

指定使用資料目錄寫入 Amazon S3 的 AWS Glue 資料目標。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料目標的名稱。

- `Inputs` – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  輸入到資料目標的節點。

- `PartitionKeys` – UTF-8 字串陣列。

  指定使用一系列索引鍵的原生分割。

- `Table` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫中資料表的名稱。

- `Database` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要寫入之資料庫的名稱。

- `SchemaChangePolicy` – CatalogSchemaChangePolicy 物件。

  可以針對受管目錄指定更新行為的政策。

GovernedCatalogSource 結構

指定受控資料目錄中的 AWS Glue 資料倉庫。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

資料存放區的名稱。

- `Database` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取的資料庫。

- `Table` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  要讀取的資料庫資料表。

- `PartitionPredicate` – UTF-8 字串，需符合Custom string pattern #40。

  滿足此述詞的分割區會被刪除。這些分割區中仍在保留期間內的檔案不會被刪除。設定為 "" – 預設為空值。

- `AdditionalOptions` – S3 SourceAdditionalOptions 物件。

  指定其他連接選項。

AggregateOperation 結構

指定執行彙總轉換中彙總所需的參數集。

欄位

- `Column` – 必要：UTF-8 字串陣列。

  指定要套用彙總函數的資料集上的資料欄。

- `AggFunc`–必要：UTF-8 字串 (有效值：`avg` | `countDistinct` | `count` | `first` | `last` | `kurtosis` | `max` | `min` | `skewness` | `stddev_samp` | `stddev_pop` | `sum` | `sumDistinct` | `var_samp` | `var_pop`)。

  指定要套用的彙總函數。

  可能的彙總函數包括：avg
  countDistinct、count、first、last、kurtosis、max、min、skewness、stddev_samp、stddev_pop、sum、s

GlueSchema 結構

當無法由 AWS Glue決定架構時，指定使用者定義的架構。

欄位

- `Columns` – 一個 GlueStudioSchemaColumn 物件陣列。

  指定組成結構定義的資料行定 AWS Glue 義。

GlueStudioSchemaColumn 結構

指定結構定義中的單一 AWS Glue 資料行。

欄位

- `Name` – 必要：UTF-8 字串，長度不可超過 1024 個位元組，且需符合 Single-line string pattern。

  AWS Glue 工作室結構描述中的資料行名稱。
- `Type` – UTF-8 字串，長度不可超過 131072 個位元組，需符合Single-line string pattern。

  在 AWS Glue Studio 架構中此列的配置單元類型。

GlueStudioColumn 結構

在 AWS Glue工作室中指定一列。

欄位

- `Key` – 必要：UTF-8 字串，需符合Custom string pattern #41。

  在 AWS Glue 工作室列的關鍵。
- `FullPath` – 必要：UTF-8 字串陣列。

  T AWS Glue 工作室中資料行的完整網址。
- `Type` – 必要：UTF-8 字串 (有效值：`array="ARRAY"` | `bigint="BIGINT"` | `bigint array="BIGINT_ARRAY"` | `binary="BINARY"` | `binary array="BINARY_ARRAY"` | `boolean="BOOLEAN"` | `boolean array="BOOLEAN_ARRAY"` | `byte="BYTE"` | `byte array="BYTE_ARRAY"` | `char="CHAR"` | `char array="CHAR_ARRAY"` | `choice="CHOICE"` | `choice array="CHOICE_ARRAY"` | `date="DATE"` | `date array="DATE_ARRAY"` | `decimal="DECIMAL"` | `decimal array="DECIMAL_ARRAY"` | `double="DOUBLE"` | `double array="DOUBLE_ARRAY"` | `enum="ENUM"` | `enum array="ENUM_ARRAY"` | `float="FLOAT"` | `float array="FLOAT_ARRAY"` | `int="INT"` | `int array="INT_ARRAY"`

| interval="INTERVAL" | interval array="INTERVAL_ARRAY" | long="LONG" | long array="LONG_ARRAY" | object="OBJECT" | short="SHORT" | short array="SHORT_ARRAY" | smallint="SMALLINT" | smallint array="SMALLINT_ARRAY" | string="STRING" | string array="STRING_ARRAY" | timestamp="TIMESTAMP" | timestamp array="TIMESTAMP_ARRAY" | tinyint="TINYINT" | tinyint array="TINYINT_ARRAY" | varchar="VARCHAR" | varchar array="VARCHAR_ARRAY" | null="NULL" | unknown="UNKNOWN" | unknown array="UNKNOWN_ARRAY")。

T AWS Glue 工作室中資料行的類型。

- Children – 結構的陣列。

  T AWS Glue 工作室中父專欄的子項。

DynamicTransform 結構

指定執行動態轉換所需的參數集。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #40。

  指定動態轉換的名稱。

- TransformName – 必要：UTF-8 字串，需符合Custom string pattern #40。

  指定動態轉換在 AWS Glue Studio 視覺化編輯器中顯示的名稱。

- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  指定所需動態轉換的輸入。

- Parameters – 一個 TransformConfigParameter 物件陣列。

  指定動態轉換的參數。

- FunctionName – 必要：UTF-8 字串，需符合Custom string pattern #40。

  指定動態轉換的函數名稱。

- Path – 必要：UTF-8 字串，需符合Custom string pattern #40。

  指定動態轉換來源檔案和組態檔案的路徑。

- Version – UTF-8 字串，需符合Custom string pattern #40。

此欄位未使用，且會在未來版本中移除。

- `OutputSchemas` – 一個 GlueSchema 物件陣列。

  指定動態轉換的資料結構描述。

TransformConfigParameter 結構

指定動態轉換組態檔案的參數。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #40。

  指定動態轉換組態檔案的參數名稱。

- `Type` – 必要：UTF-8 字串 (有效值：`str="STR"` | `int="INT"` | `float="FLOAT"` | `complex="COMPLEX"` | `bool="BOOL"` | `list="LIST"` | `null="NULL"`)。

  指定動態轉換組態檔案的參數類型。

- `ValidationRule` – UTF-8 字串，需符合Custom string pattern #40。

  指定動態轉換組態檔案的驗證規則。

- `ValidationMessage` – UTF-8 字串，需符合Custom string pattern #40。

  指定動態轉換組態檔案的驗證訊息。

- `Value` – UTF-8 字串陣列。

  指定動態轉換組態檔案的參數值。

- `ListType` – UTF-8 字串 (有效值：`str="STR"` | `int="INT"` | `float="FLOAT"` | `complex="COMPLEX"` | `bool="BOOL"` | `list="LIST"` | `null="NULL"`)。

  指定動態轉換組態檔案的參數類型清單。

- `IsOptional` – 布林值。

  指定參數是否為選用或未在動態轉換組態檔案中。

EvaluateDataQuality 結構

指定資料品質評估標準。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料品質評估的名稱。
- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  資料品質評估的輸入。
- Ruleset – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 65536 個位元組，且需符合Custom string pattern #38。

  資料品質評估的規則集。
- Output – UTF-8 字串 (有效值：PrimaryInput | EvaluationResults)。

  資料品質評估的是輸出。
- PublishingOptions – DQ ResultsPublishingOptions 物件。

  設定結果發佈方式的選項。
- StopJobOnFailureOptions – DQ StopJobOnFailureOptions 物件。

  設定資料品質評估失敗時如何停止任務的選項。

DQ ResultsPublishingOptions 結構

設定資料品質評估結果發佈方式的選項。

欄位

- EvaluationContext – UTF-8 字串，需符合Custom string pattern #39。

  評估的內容。
- ResultsS3Prefix – UTF-8 字串，需符合Custom string pattern #40。

  附加到結果前面的 Amazon S3 字首。
- CloudWatchMetricsEnabled – 布林值。

啟用資料品質結果的指標。

- `ResultsPublishingEnabled` – 布林值。

  啟用發佈資料品質結果。

DQ StopJobOnFailureOptions 結構

設定資料品質評估失敗時如何停止任務的選項。

欄位

- `StopJobOnFailureTiming` – UTF-8 字串 (有效值：`Immediate` | `AfterDataLoad`)。

  資料品質評估失敗時停止任務的時機。選項為「立即」或 AfterDataLoad。

EvaluateDataQualityMultiFrame 結構

指定資料品質評估標準。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  資料品質評估的名稱。
- `Inputs` – 必要：UTF-8 字串的陣列，至少要有 1 個字串。

  資料品質評估的輸入。此清單中的第一個輸入是主資料來源。
- `AdditionalDataSources` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #43。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  除主資料來源以外的所有資料來源的別名。
- `Ruleset` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 65536 個位元組，且需符合Custom string pattern #38。

  資料品質評估的規則集。
- `PublishingOptions` – DQ ResultsPublishingOptions 物件。

設定結果發佈方式的選項。

- AdditionalOptions – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串 (有效值：performanceTuning.caching="CacheOption" | observations.scope="ObservationsOption")。

  每個值都是 UTF-8 字串。

  設定轉換執行期行為的選項。

- StopJobOnFailureOptions – DQ StopJobOnFailureOptions 物件。

  設定資料品質評估失敗時如何停止任務的選項。

配方結構

在 AWS Glue 工作中使用 AWS Glue DataBrew 配方的 AWS Glue Studio 節點。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  AWS Glue 工作室節點的名稱。

- Inputs – 必要：UTF-8 字串的陣列，不可小於 1 或超過 1 個字串。

  作為配方節點輸入的節點，由 ID 識別。

- RecipeReference – 必要：RecipeReference 物件。

  節點所使用之 DataBrew 配方的參考。

RecipeReference 結構

一個 AWS Glue DataBrew 配方的參考。

欄位

- RecipeArn – 必要：UTF-8 字串，需符合Custom string pattern #40。

  DataBrew 食譜的 ARN。

- RecipeVersion – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 16 個位元組。

DataBrew 配 RecipeVersion 方的。

SnowflakeNodeData 結構

指定 AWS Glue 工作室中雪花節點的配置。

欄位

- `SourceType` – UTF-8 字串，需符合Custom string pattern #39。

  指定如何指定擷取的資料。有效值："table"、 "query"。
- `Connection` – 選項 物件。

  指定與雪花端點的 AWS Glue 資料目錄連線。
- `Schema` – UTF-8 字串。

  指定節點使用的 Snowflake 資料庫結構描述。
- `Table` – UTF-8 字串。

  指定節點使用的 Snowflake 資料表。
- `Database` – UTF-8 字串。

  指定節點使用的 Snowflake 資料庫。
- `TempDir` – UTF-8 字串，需符合Custom string pattern #40。

  目前未使用。
- `IamRole` – 選項 物件。

  目前未使用。
- `AdditionalOptions` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #40。

  每個值都是 UTF-8 字串，需符合 Custom string pattern #40。

  指定傳遞至 Snowflake 連接器的其他選項。如果在此節點的其他位置指定了選項，這會具有優先順序。
- `SampleQuery` – UTF-8 字串。

SQL 字串，用於擷取具有 query sourcetype 的資料。

- PreAction – UTF-8 字串。

  SQL 字串會在 Snowflake 連接器執行其標準動作之前執行。

- PostAction – UTF-8 字串。

  SQL 字串會在 Snowflake 連接器執行其標準動作之後執行。

- Action – UTF-8 字串。

  指定在使用預先存在的資料寫入資料表時要採取的動作。有效值：append、merge、truncate、drop。

- Upsert – 布林值。

  當動作為 append 時使用。指定資料列已存在時的解析行為。如果為 true，將更新預先存在的資料列。如果為 false，將插入這些資料列。

- MergeAction – UTF-8 字串，需符合Custom string pattern #39。

  指定合併動作。有效值：simple、custom。如果為簡單，合併行為由 MergeWhenMatched 和 MergeWhenNotMatched 定義。如果為自訂，由 MergeClause 定義。

- MergeWhenMatched – UTF-8 字串，需符合Custom string pattern #39。

  指定合併時如何解析與預先存在的資料相符的記錄。有效值： update、delete。

- MergeWhenNotMatched – UTF-8 字串，需符合Custom string pattern #39。

  指定合併時如何處理與預先存在的資料不相符的記錄。有效值：insert、none。

- MergeClause – UTF-8 字串。

  指定自訂合併行為的 SQL 陳述式。

- StagingTable – UTF-8 字串。

  執行 merge 或 upsert append 動作時使用的暫存資料表名稱。資料會寫入此資料表，然後由產生的後置動作移至 table。

- SelectedColumns – 一個 選項 物件陣列。

  在偵測合併和更新插入相符項時，指定合併起來用於識別記錄的資料欄。具有 value、label 和 description 金鑰的結構清單。每個結構都描述了一個資料欄。

- AutoPushdown – 布林值。

指定是否啟用自動查詢下推。如果下推已啟用，則在 Spark 上執行查詢時，若查詢的一部分可以「向下推」到 Snowflake 伺服器，它便會被下推。這可改善某些查詢的效能。

- `TableSchema` – 一個 選項 物件陣列。

  手動定義節點的目標結構描述。具有 `value`、`label` 和 `description` 金鑰的結構清單。每個結構都定義了一個資料欄。

SnowflakeSource 結構

指定 Snowflake 資料來源。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  Snowflake 資料來源的名稱。

- `Data` – 必要：SnowflakeNodeData 物件。

  Snowflake 資料來源的組態。

- `OutputSchemas` – 一個 GlueSchema 物件陣列。

  指定輸出資料的使用者定義結構描述。

SnowflakeTarget 結構

指定 Snowflake 目標。

欄位

- `Name` – 必要：UTF-8 字串，需符合Custom string pattern #43。

  Snowflake 目標的名稱。

- `Data` – 必要：SnowflakeNodeData 物件。

  指定 Snowflake 目標節點的資料。

- `Inputs`：UTF-8 字串陣列，不可小於 1，也不可超過 1 個字串。

  輸入到資料目標的節點。

ConnectorDataSource 結構

指定使用標準連線選項產生的來源。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

  此來源節點的名稱。
- ConnectionType – 必要：UTF-8 字串，需符合Custom string pattern #40。

  的connectionType，如提供給基礎 AWS Glue 庫。此節點類型支援下列連線類型：
  - opensearch
  - azuresql
  - azurecosmos
  - bigquery
  - saphana
  - teradata
  - vertica
- Data – 必要：金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  針對節點指定連線選項的對應。您可以在 AWS Glue 文件的「連接參數」一節中找到對應連接類型的標準連接選項。
- OutputSchemas – 一個 GlueSchema 物件陣列。

  指定此來源的資料結構描述。

ConnectorDataTarget 結構

指定使用標準連線選項產生的目標。

欄位

- Name – 必要：UTF-8 字串，需符合Custom string pattern #43。

此目標節點的名稱。

- ConnectionType – 必要：UTF-8 字串，需符合Custom string pattern #40。

  的connectionType，如提供給基礎 AWS Glue 庫。此節點類型支援下列連線類型：

  - opensearch
  - azuresql
  - azurecosmos
  - bigquery
  - saphana
  - teradata
  - vertica
- Data – 必要：金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  針對節點指定連線選項的對應。您可以在 AWS Glue 文件的「連接參數」一節中找到對應連接類型的標準連接選項。
- Inputs：UTF-8 字串陣列，不可小於 1，也不可超過 1 個字串。

  輸入到資料目標的節點。

任務 API

Jobs API 說明在 AWS Glue 中的任務資料類型，包括用於任務、任務執行以及觸發條件的 API。

主題

- 任務
- 任務執行
- 觸發

任務

作業 API 描述與中建立、更新、刪除或檢視工作相關的資料類型和 API AWS Glue。

資料類型

- Job 結構
- ExecutionProperty 結構
- NotificationProperty 結構
- JobCommand 結構
- ConnectionsList 結構
- JobUpdate 結構
- SourceControlDetails 結構

Job 結構

指定任務定義。

欄位

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  您指派給此任務定義的名稱。
- JobMode – UTF-8 字串 (有效值：SCRIPT="" | VISUAL="" | NOTEBOOK="")。

  描述如何建立工作的模式。有效的 值如下：
  - SCRIPT-工作是使用 AWS Glue Studio 指令碼編輯器建立的。
  - VISUAL-工作是使用 AWS Glue Studio 視覺化編輯器建立的。
  - NOTEBOOK-工作是使用互動式工作階段筆記本建立的。

  當JobMode欄位遺失或為 null 時，SCRIPT會指派為預設值。
- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  任務的描述。
- LogUri – UTF-8 字串。

  此欄位保留供日後使用。
- Role – UTF-8 字串。

與此任務相關聯 IAM 角色的名稱或 Amazon Resource Name (ARN)。

- `CreatedOn` – 時間戳記。

  此任務定義的建立日期和時間。

- `LastModifiedOn` – 時間戳記。

  此任務定義上一次修改的時間點。

- `ExecutionProperty` – ExecutionProperty 物件。

  `ExecutionProperty`，指定此任務可同時執行的最大數量。

- `Command` – JobCommand 物件。

  執行這個任務的 `JobCommand`。

- `DefaultArguments` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  此任務每次執行的預設引數，以名稱值對的方式指定。

  您可以在這裡指定自己的工作執行腳本消耗的參數，以及 AWS Glue 本身消耗的參數。

  可以記錄任務引數。不要將純文字祕密當做引數傳遞。如果您想要將密碼保留在 Job 中，請從「 AWS Glue 連線」 AWS Secrets Manager 或其他密碼管理機制擷取密碼。

  如需如何指定和取用自有任務引數的資訊，請參閱本開發人員指南中的使用 Python 呼叫 AWS Glue API 主題。

  如需有關設定 Spark 任務時可提供給此欄位之引數的資訊，請參閱開發人員指南中的 Special Parameters Used by AWS Glue 主題。

  如需有關設定 Ray 任務時可提供給此欄位之引數的資訊，請參閱開發人員指南中的 Using job parameters in Ray jobs。

- `NonOverridableArguments` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

在任務執行中提供任務引數時，此任務未被覆寫的引數，以名稱值對的方式指定。

- `Connections` – ConnectionsList 物件。

  用於此任務的連線。

- `MaxRetries` – 數字 (整數)。

  JobRun 失敗後重試此工作的次數上限。

- `AllocatedCapacity` – 數字 (整數)。

  此欄位已作廢。請改用 `MaxCapacity`。

  配置給此工作執行的 AWS Glue 資料處理單元 (DPU) 數目。您可以分配至少 2 個 DPU，預設值為 10。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價頁面。

- `Timeout` – 數字 (整數)，至少為 1。

  任務逾時 (以分鐘為單位)。此為任務執行在停止並進入 `TIMEOUT` 狀態前可以消耗資源的最大時間。批次工作的預設值為 2,880 分鐘 (48 小時)。

  串流工作的逾時值必須小於 7 天或 10080 分鐘。當值保留空白時，如果您尚未設定維護時段，工作將在 7 天後重新啟動。如果您已設定維護時段，則維護時段將在 7 天後重新啟動。

- `MaxCapacity` – 數字 (雙位數)。

  對於 Glue 1.0 版或更早版本的工作，使用標準 Worker 類型，即此工作執行時可配置的 AWS Glue 資料處理單元 (DPU) 數目。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價頁面。

  對於 Glue 2.0 版或更新版本的任務，您不能指定 `Maximum capacity`。反之，您必須指定 `Worker type` 與 `Number of workers`。

  如果使用 `WorkerType` 和 `NumberOfWorkers`，請勿設定 `MaxCapacity`。

  可配置給 `MaxCapacity` 的值取決於您執行的是 Python shell 任務、Apache Spark ETL 任務或 Apache Spark Streaming ETL 任務：

  - 當您指定 Python shell 任務 (`JobCommand.Name="pythonshell"`) 時，您可以擇一分配 0.0625 或 1 個 DPU。預設為 0.0625 個 DPU。

  - 指定 Apache Spark ETL 任務 (`JobCommand.Name="glueetl"`) 或 Apache Spark Streaming ETL 任務 (`JobCommand.Name="gluestreaming"`) 時，您可以配置 2 到 100 個 DPU。預設值是 10 個 DPU。此任務類型沒有小數的 DPU 分配。

- `WorkerType` – UTF-8 字串 (有效值：`Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""`)。

  將在任務執行時分配的預先定義工作者類型。接受 Spark 任務的 G.1X、G.2X、G.4X、G.8X 或 G.025X 值。接受 Ray 任務的 Z.2X 值。

  - 對於 `G.1X` 工作者類型，每個工作者都會映射到 1 個 DPU (4 個 vCPU、16 GB 記憶體)，外加 84 GB 磁碟 (大約 34 GB 可用)，並為每個工作者提供 1 個執行器。我們建議將此工作者類型用於資料轉換、聯結和查詢等工作負載，以提供可擴展且符合成本效益的方式來執行大部分任務。
  - 對於 `G.2X` 工作者類型，每個工作者都會映射到 2 個 DPU (8 個 vCPU、32 GB 記憶體)，外加 128 GB 磁碟 (大約 77 GB 可用)，並為每個工作者提供 1 個執行器。我們建議將此工作者類型用於資料轉換、聯結和查詢等工作負載，以提供可擴展且符合成本效益的方式來執行大部分任務。
  - 對於 `G.4X` 工作者類型，每個工作者都會映射到 4 個 DPU (16 個 vCPU、64 GB 記憶體)，外加 256 GB 磁碟 (大約 235 GB 可用)，並為每個工作者提供 1 個執行器。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此 Worker 類型僅適用於以下 AWS 區域 3.0 或更新 AWS Glue 版本的 Spark ETL 工作：美國東部 (俄亥俄)、美國東部 (維吉尼亞北部)、美國西部 (奧勒岡)、亞太區域 (新加坡)、亞太區域 (雪梨)、亞太區域 (東京)、加拿大 (中部)、歐洲 (法蘭克福)、歐洲 (愛爾蘭) 和歐洲 (斯德哥爾摩)。
  - 對於 `G.8X` 工作者類型，每個工作者都會映射到 8 個 DPU (32 個 vCPU、128 GB 記憶體)，外加 512 GB 磁碟 (大約 487 GB 可用)，並為每個工作者提供 1 個執行器。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此背景工作類型僅適用於 3.0 AWS Glue 版或更新版本的 Spark ETL 工作，與`G.4X`背景工作者類型支援的相同 AWS 區域。
  - 對於 `G.025X` 工作者類型，每個工作者都會映射到 0.25 個 DPU (2 個 vCPU、4 GB 記憶體)，外加 84 GB 磁碟 (大約 34 GB 可用)，並為每個工作者提供 1 個執行器。我們建議低容量串流任務採用這種工作者類型。此背景工作類型僅適用於 3.0 AWS Glue 版串流工作。
  - 對於 `Z.2X` 工作者類型，每個工作者都會映射到 2 個 M-DPU (8 個 vCPU、64 GB 記憶體)，外加 128 GB 磁碟 (大約 120 GB 可用)，並根據自動縮放器提供最多 8 個 Ray 工作者。

- `NumberOfWorkers` – 數字 (整數)。

  當任務執行時所配置的已定義 `workerType` 的工作者數目。

- `SecurityConfiguration` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

與此任務搭配使用的 SecurityConfiguration 結構名稱。

- NotificationProperty – NotificationProperty 物件。

  指定任務通知的組態屬性。

- Running – 布林值。

  此欄位保留供日後使用。

- GlueVersion – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #20。

  在星火作業中，GlueVersion確定阿帕奇星火和 Python 的版本，在作業中 AWS Glue 可用。Python 版本指示針對 Spark 類型任務支援的版本。

  Ray 任務應將 GlueVersion 設定為 4.0 或更高版本。不過，Ray 任務中可用的 Ray、Python 和其他程式庫的版本由 Job 命令的 Runtime 參數決定。

  如需有關可用版 AWS Glue 本以及對應 Spark 和 Python 版本的詳細資訊，請參閱開發人員指南中的 Glue 版本。

  建立時未指定 Glue 版本的任務，預設為 Glue 0.9。

- CodeGenConfigurationNodes – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #39。

  每個值都是 CodeGenConfigurationNode 物件。

  Glue Studio 視覺化組件和 Glue Studio 代碼的產生都根據有向無循環圖的表示形式。

- ExecutionClass – UTF-8 字串，長度不可超過 16 個位元組 (有效值：FLEX="" | STANDARD="")。

  表示任務執行使用的是標準執行類別還是彈性執行類別。標準執行類別非常適合需要快速任務啟動和專用資源的時間敏感型工作負載。

  彈性執行類別適用於開始和完成時間可能會有所變化的時間敏感型任務。

  只有 AWS Glue 版本 3.0 及更新版本和指令類型glueetl的工作才能設定ExecutionClass為FLEX。彈性執行類別可用於 Spark 任務。

- SourceControlDetails – SourceControlDetails 物件。

任務原始檔控制組態的詳細資訊，可允許與遠端儲存庫雙向同步任務成品。

- `MaintenanceWindow` – UTF-8 字串，需符合Custom string pattern #30。

  此欄位指定串流工作的維護時段的星期幾和小時。 AWS Glue 定期執行維護活動。在這些維護時段期間， AWS Glue 將需要重新啟動您的串流工作。

  AWS Glue 將在指定維護時間的 3 小時內重新啟動工作。例如，如果您在格林威治標準時間星期一上午 10:00 設定維護時段，您的作業將在格林威治標準時間上午 10:00 至下午 1:00 之間重新啟動。

- `ProfileName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與工作相關聯的 AWS Glue 使用情況設定檔名稱。

ExecutionProperty 結構

任務的執行屬性。

欄位

- `MaxConcurrentRuns` – 數字 (整數)。

  一項任務可同時執行的最大數量。預設為 1。達到此閾值時，會傳回錯誤。可指定的最大值由服務限制來控制。

NotificationProperty 結構

指定通知的組態屬性。

欄位

- `NotifyDelayAfter` – 數字 (整數)，至少為 1。

  任務執行開始後，在傳送任務執行延遲通知之前所要等待的分鐘數。

JobCommand 結構

指定任務執行時執行的程式碼。

欄位

- Name – UTF-8 字串。

  任務命令的名稱。用於 Apache Spark ETL 任務時，必須使用 glueetl。用於 Python shell 任務時，必須使用 pythonshell。用於 Apache Spark Streaming ETL 任務時，必須是 gluestreaming。對於 Ray 任務，這一定是 glueray。

- ScriptLocation – UTF-8 字串，長度不可超過 400000 個位元組。

  指定指向執行任務指令碼的 Amazon Simple Storage Service (Amazon S3) 路徑。

- PythonVersion – UTF-8 字串，需符合Custom string pattern #21。

  用於執行 Python Shell 任務的 Python 版本。允許的值是 2 或 3。

- Runtime：UTF-8 字串，長度不可超過 64 個位元組，且需符合 Custom string pattern #29。

  在 Ray 任務中，執行期用於指定環境中可用的 Ray、Python 和其他程式庫的版本。此欄位不用於其他任務類型。如需支援的執行階段環境值，請參閱 AWS Glue 開發人員指南中的支援的 Ray 執行階段

ConnectionsList 結構

指定任務所使用的連線。

欄位

- Connections – UTF-8 字串陣列。

  任務所使用連線的清單。

JobUpdate 結構

指定用於更新現有任務定義的資訊。此資訊將完全覆寫之前的任務定義。

欄位

- JobMode – UTF-8 字串 (有效值：SCRIPT="" | VISUAL="" | NOTEBOOK="")。

  描述如何建立工作的模式。有效的 值如下：

  - SCRIPT-工作是使用 AWS Glue Studio 指令碼編輯器建立的。

- VISUAL-工作是使用 AWS Glue Studio 視覺化編輯器建立的。
- NOTEBOOK-工作是使用互動式工作階段筆記本建立的。

當JobMode欄位遺失或為 null 時，SCRIPT會指派為預設值。

- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  正在定義的任務說明。

- LogUri – UTF-8 字串。

  此欄位保留供日後使用。

- Role – UTF-8 字串。

  與此任務關聯之 IAM 角色的名稱或 Amazon Resource Name (ARN)(必要)。

- ExecutionProperty – ExecutionProperty 物件。

  ExecutionProperty，指定此任務可同時執行的最大數量。

- Command – JobCommand 物件。

  負責執行此任務的 JobCommand (必要)。

- DefaultArguments – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  此任務每次執行的預設引數，以名稱值對的方式指定。

  您可以在這裡指定自己的工作執行腳本消耗的參數，以及 AWS Glue 本身消耗的參數。

  可以記錄任務引數。不要將純文字祕密當做引數傳遞。如果您想要將密碼保留在 Job 中，請從「 AWS Glue 連線」 AWS Secrets Manager 或其他密碼管理機制擷取密碼。

  如需如何指定和取用自有任務引數的資訊，請參閱本開發人員指南中的使用 Python 呼叫 AWS Glue API 主題。

  如需有關設定 Spark 任務時可提供給此欄位之引數的資訊，請參閱開發人員指南中的 Special Parameters Used by AWS Glue 主題。

如需有關設定 Ray 任務時可提供給此欄位之引數的資訊，請參閱開發人員指南中的 Using job parameters in Ray jobs。

- NonOverridableArguments – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  在任務執行中提供任務引數時，此任務未被覆寫的引數，以名稱值對的方式指定。

- Connections – ConnectionsList 物件。

  用於此任務的連線。

- MaxRetries – 數字 (整數)。

  如果此任務失敗，可重試的次數上限。

- AllocatedCapacity – 數字 (整數)。

  此欄位已作廢。請改用 MaxCapacity。

  要配置給此工作的 AWS Glue 資料處理單元 (DPU) 數目。您可以分配至少 2 個 DPU，預設值為 10。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價頁面。

- Timeout – 數字 (整數)，至少為 1。

  任務逾時 (以分鐘為單位)。此為任務執行在停止並進入 TIMEOUT 狀態前可以消耗資源的最大時間。批次工作的預設值為 2,880 分鐘 (48 小時)。

  串流工作的逾時值必須小於 7 天或 10080 分鐘。當值保留空白時，如果您尚未設定維護時段，工作將在 7 天後重新啟動。如果您已設定維護時段，則維護時段將在 7 天後重新啟動。

- MaxCapacity – 數字 (雙位數)。

  對於 Glue 1.0 版或更早版本的工作，使用標準 Worker 類型，即此工作執行時可配置的 AWS Glue 資料處理單元 (DPU) 數目。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價頁面。

  對於 Glue 2.0+ 版任務，您不能指定 Maximum capacity。反之，您必須指定 Worker type 與 Number of workers。

  如果使用 WorkerType 和 NumberOfWorkers，請勿設定 MaxCapacity。

可配置給 MaxCapacity 的值取決於您執行的是 Python shell 任務、Apache Spark ETL 任務或 Apache Spark Streaming ETL 任務：

- 當您指定 Python shell 任務 (JobCommand.Name="pythonshell") 時，您可以擇一分配 0.0625 或 1 個 DPU。預設為 0.0625 個 DPU。
- 指定 Apache Spark ETL 任務 (JobCommand.Name="glueetl") 或 Apache Spark Streaming ETL 任務 (JobCommand.Name="gluestreaming") 時，您可以配置 2 到 100 個 DPU。預設值是 10 個 DPU。此任務類型沒有小數的 DPU 分配。

- WorkerType – UTF-8 字串 (有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="")。

  將在任務執行時分配的預先定義工作者類型。接受 Spark 任務的 G.1X、G.2X、G.4X、G.8X 或 G.025X 值。接受 Ray 任務的 Z.2X 值。

  - 對於 G.1X 工作者類型，每個工作者都會映射到 1 個 DPU (4 個 vCPU、16 GB 記憶體)，外加 84 GB 磁碟 (大約 34 GB 可用)，並為每個工作者提供 1 個執行器。我們建議將此工作者類型用於資料轉換、聯結和查詢等工作負載，以提供可擴展且符合成本效益的方式來執行大部分任務。
  - 對於 G.2X 工作者類型，每個工作者都會映射到 2 個 DPU (8 個 vCPU、32 GB 記憶體)，外加 128 GB 磁碟 (大約 77 GB 可用)，並為每個工作者提供 1 個執行器。我們建議將此工作者類型用於資料轉換、聯結和查詢等工作負載，以提供可擴展且符合成本效益的方式來執行大部分任務。
  - 對於 G.4X 工作者類型，每個工作者都會映射到 4 個 DPU (16 個 vCPU、64 GB 記憶體)，外加 256 GB 磁碟 (大約 235 GB 可用)，並為每個工作者提供 1 個執行器。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此 Worker 類型僅適用於以下 AWS 區域 3.0 或更新 AWS Glue 版本的 Spark ETL 工作：美國東部 (俄亥俄)、美國東部 (維吉尼亞北部)、美國西部 (奧勒岡)、亞太區域 (新加坡)、亞太區域 (雪梨)、亞太區域 (東京)、加拿大 (中部)、歐洲 (法蘭克福)、歐洲 (愛爾蘭) 和歐洲 (斯德哥爾摩)。
  - 對於 G.8X 工作者類型，每個工作者都會映射到 8 個 DPU (32 個 vCPU、128 GB 記憶體)，外加 512 GB 磁碟 (大約 487 GB 可用)，並為每個工作者提供 1 個執行器。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此背景工作類型僅適用於 3.0 AWS Glue 版或更新版本的 Spark ETL 工作，與G.4X背景工作者類型支援的相同 AWS 區域。
  - 對於 G.025X 工作者類型，每個工作者都會映射到 0.25 個 DPU (2 個 vCPU、4 GB 記憶體)，外加 84 GB 磁碟 (大約 34 GB 可用)，並為每個工作者提供 1 個執行器。我們建議低容量串流任務採用這種工作者類型。此背景工作類型僅適用於 3.0 AWS Glue 版串流工作。
  - 對於 Z.2X 工作者類型，每個工作者都會映射到 2 個 M-DPU (8 個 vCPU、64 GB 記憶體)，外加 128 GB 磁碟 (大約 120 GB 可用)，並根據自動縮放器提供最多 8 個 Ray 工作者。

- NumberOfWorkers – 數字 (整數)。

當任務執行時所配置的已定義 workerType 的工作者數目。

- SecurityConfiguration – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與此任務搭配使用的 SecurityConfiguration 結構名稱。

- NotificationProperty – NotificationProperty 物件。

  指定任務通知的組態屬性。

- GlueVersion – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #20。

  在星火作業中，GlueVersion確定阿帕奇星火和 Python 的版本，在作業中 AWS Glue 可用。Python 版本指示針對 Spark 類型任務支援的版本。

  Ray 任務應將 GlueVersion 設定為 4.0 或更高版本。不過，Ray 任務中可用的 Ray、Python 和其他程式庫的版本由 Job 命令的 Runtime 參數決定。

  如需有關可用版 AWS Glue 本以及對應 Spark 和 Python 版本的詳細資訊，請參閱開發人員指南中的 Glue 版本。

  建立時未指定 Glue 版本的任務，預設為 Glue 0.9。

- CodeGenConfigurationNodes – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #39。

  每個值都是 CodeGenConfigurationNode 物件。

  Glue Studio 視覺化組件和 Glue Studio 代碼的產生都根據有向無循環圖的表示形式。

- ExecutionClass – UTF-8 字串，長度不可超過 16 個位元組 (有效值：FLEX="" | STANDARD="")。

  表示任務執行使用的是標準執行類別還是彈性執行類別。標準執行類別非常適合需要快速的任務啟動和專用資源的時間敏感型工作負載。

  彈性執行類別適用於開始和完成時間可能會有所變化的時間敏感型任務。

  只有 AWS Glue 版本 3.0 及更新版本和指令類型glueetl的工作才能設定ExecutionClass為FLEX。彈性執行類別可用於 Spark 任務。

- `SourceControlDetails` – SourceControlDetails 物件。

  任務原始檔控制組態的詳細資訊，可允許與遠端儲存庫雙向同步任務成品。

- `MaintenanceWindow` – UTF-8 字串，需符合Custom string pattern #30。

  此欄位指定串流工作的維護時段的星期幾和小時。 AWS Glue 定期執行維護活動。在這些維護時段期間， AWS Glue 將需要重新啟動您的串流工作。

  AWS Glue 將在指定維護時間的 3 小時內重新啟動工作。例如，如果您在格林威治標準時間星期一上午 10:00 設定維護時段，您的作業將在格林威治標準時間上午 10:00 至下午 1:00 之間重新啟動。

- `ProfileName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與工作相關聯的 AWS Glue 使用情況設定檔名稱。

SourceControlDetails 結構

任務原始檔控制組態的詳細資訊，可允許與遠端儲存庫雙向同步任務成品。

欄位

- `Provider` – UTF-8 字串。

  遠端儲存庫的提供者。

- `Repository` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組。

  任務成品所在遠端儲存庫的名稱。

- `Owner` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組。

  任務成品所在遠端儲存庫的擁有者。

- `Branch` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組。

  遠端儲存庫中可供自由選用的分支。

- `Folder` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組。

  遠端儲存庫中可供自由選用的資料夾。

- `LastCommitId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組。

  遠端儲存庫中遞交的最後一個遞交 ID。

- `LastSyncTimestamp` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組。

  最近一次執行任務同步作業的日期和時間。

- `AuthStrategy` – UTF-8 字串。

  驗證類型，可以是儲存在 AWS Secrets Manager 中的驗證權杖，也可以是個人存取權杖。

- `AuthToken` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組。

  授權字符的值。

作業

- CreateJob 行動（Python：創建工作）
- UpdateJob 行動（Python：更新工作）
- GetJob 行動（Python：獲取工作）
- GetJobs 行動（Python：獲取工作）
- DeleteJob 行動（Python：刪除工作）
- ListJobs 行動（Python：列表工作）
- BatchGetJobs 行動（Python：批處理工作）

CreateJob 行動（Python：創建工作）

建立新任務定義。

請求

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  您指派給此任務定義的名稱。它在您的 帳戶中必須是唯一的。

- `JobMode` – UTF-8 字串 (有效值：`SCRIPT=""` | `VISUAL=""` | `NOTEBOOK=""`)。

  描述如何建立工作的模式。有效的 值如下：

  - `SCRIPT`-工作是使用 AWS Glue Studio 指令碼編輯器建立的。
  - `VISUAL`-工作是使用 AWS Glue Studio 視覺化編輯器建立的。
  - `NOTEBOOK`-工作是使用互動式工作階段筆記本建立的。

當JobMode欄位遺失或為 null 時，SCRIPT會指派為預設值。

- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  正在定義的任務說明。

- LogUri – UTF-8 字串。

  此欄位保留供日後使用。

- Role – 必要：UTF-8 字串。

  與此任務相關聯 IAM 角色的名稱或 Amazon Resource Name (ARN)。

- ExecutionProperty – ExecutionProperty 物件。

  ExecutionProperty，指定此任務可同時執行的最大數量。

- Command – 必要：JobCommand 物件。

  執行這個任務的 JobCommand。

- DefaultArguments – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  此任務每次執行的預設引數，以名稱值對的方式指定。

  您可以在這裡指定自己的工作執行腳本消耗的參數，以及 AWS Glue 本身消耗的參數。

  可以記錄任務引數。不要將純文字祕密當做引數傳遞。如果您想要將密碼保留在 Job 中，請從「 AWS Glue 連線」 AWS Secrets Manager 或其他密碼管理機制擷取密碼。

  如需如何指定和取用自有任務引數的資訊，請參閱本開發人員指南中的使用 Python 呼叫 AWS Glue API 主題。

  如需有關設定 Spark 任務時可提供給此欄位之引數的資訊，請參閱開發人員指南中的 Special Parameters Used by AWS Glue 主題。

  如需有關設定 Ray 任務時可提供給此欄位之引數的資訊，請參閱開發人員指南中的 Using job parameters in Ray jobs。

- NonOverridableArguments – 金鑰值對的映射陣列。

每個金鑰都是 UTF-8 字串。

每個值都是 UTF-8 字串。

在任務執行中提供任務引數時，此任務未被覆寫的引數，以名稱值對的方式指定。

- `Connections` – ConnectionsList 物件。

  用於此任務的連線。

- `MaxRetries` – 數字 (整數)。

  如果此任務失敗，可重試的次數上限。

- `AllocatedCapacity` – 數字 (整數)。

  此參數已棄用。請改用 `MaxCapacity`。

  要配置給此 Job 的 AWS Glue 資料處理單元 (DPU) 數目。您可以分配至少 2 個 DPU，預設值為 10。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價頁面。

- `Timeout` – 數字 (整數)，至少為 1。

  任務逾時 (以分鐘為單位)。此為任務執行在停止並進入 `TIMEOUT` 狀態前可以消耗資源的最大時間。批次工作的預設值為 2,880 分鐘 (48 小時)。

  串流工作的逾時值必須小於 7 天或 10080 分鐘。當值保留空白時，如果您尚未設定維護時段，工作將在 7 天後重新啟動。如果您已設定維護時段，則維護時段將在 7 天後重新啟動。

- `MaxCapacity` – 數字 (雙位數)。

  對於 Glue 1.0 版或更早版本的工作，使用標準 Worker 類型，即此工作執行時可配置的 AWS Glue 資料處理單元 (DPU) 數目。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價頁面。

  對於 Glue 2.0+ 版任務，您不能指定 `Maximum capacity`。反之，您必須指定 `Worker type` 與 `Number of workers`。

  如果使用 `WorkerType` 和 `NumberOfWorkers`，請勿設定 `MaxCapacity`。

  可配置給 `MaxCapacity` 的值取決於您執行的是 Python shell 任務、Apache Spark ETL 任務或 Apache Spark Streaming ETL 任務：

  - 當您指定 Python shell 任務 (JobCommand.Name="pythonshell") 時，您可以擇一分配 0.0625 或 1 個 DPU。預設為 0.0625 個 DPU。
  - 指定 Apache Spark ETL 任務 (JobCommand.Name="glueetl") 或 Apache Spark Streaming ETL 任務 (JobCommand.Name="gluestreaming") 時，您可以配置 2 到 100 個 DPU。預設值是 10 個 DPU。此任務類型沒有小數的 DPU 分配。
- SecurityConfiguration – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與此任務搭配使用的 SecurityConfiguration 結構名稱。
- Tags – 金鑰值對的對應陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  要搭配此任務使用的標籤。您可以使用標籤來限制對於任務的存取情況。如需中標籤的詳細資訊 AWS Glue，請參閱開發人員指南AWS Glue中的「AWS 標籤」。
- NotificationProperty – NotificationProperty 物件。

  指定任務通知的組態屬性。
- GlueVersion – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #20。

  在星火作業中，GlueVersion確定阿帕奇星火和 Python 的版本，在作業中 AWS Glue 可用。Python 版本指示針對 Spark 類型任務支援的版本。

  Ray 任務應將 GlueVersion 設定為 4.0 或更高版本。不過，Ray 任務中可用的 Ray、Python 和其他程式庫的版本由 Job 命令的 Runtime 參數決定。

  如需有關可用版 AWS Glue 本以及對應 Spark 和 Python 版本的詳細資訊，請參閱開發人員指南中的 Glue 版本。

  建立時未指定 Glue 版本的任務，預設為 Glue 0.9。
- NumberOfWorkers – 數字 (整數)。

  當任務執行時所配置的已定義 workerType 的工作者數目。
- WorkerType – UTF-8 字串 (有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="")。

將在任務執行時分配的預先定義工作者類型。接受 Spark 任務的 G.1X、G.2X、G.4X、G.8X 或 G.025X 值。接受 Ray 任務的 Z.2X 值。

- 對於 G.1X 工作者類型，每個工作者都會映射到 1 個 DPU (4 個 vCPU、16 GB 記憶體)，外加 84 GB 磁碟 (大約 34 GB 可用)，並為每個工作者提供 1 個執行器。我們建議將此工作者類型用於資料轉換、聯結和查詢等工作負載，以提供可擴展且符合成本效益的方式來執行大部分任務。
- 對於 G.2X 工作者類型，每個工作者都會映射到 2 個 DPU (8 個 vCPU、32 GB 記憶體)，外加 128 GB 磁碟 (大約 77 GB 可用)，並為每個工作者提供 1 個執行器。我們建議將此工作者類型用於資料轉換、聯結和查詢等工作負載，以提供可擴展且符合成本效益的方式來執行大部分任務。
- 對於 G.4X 工作者類型，每個工作者都會映射到 4 個 DPU (16 個 vCPU、64 GB 記憶體)，外加 256 GB 磁碟 (大約 235 GB 可用)，並為每個工作者提供 1 個執行器。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此 Worker 類型僅適用於以下 AWS 區域 3.0 或更新 AWS Glue 版本的 Spark ETL 工作：美國東部 (俄亥俄)、美國東部 (維吉尼亞北部)、美國西部 (奧勒岡)、亞太區域 (新加坡)、亞太區域 (雪梨)、亞太區域 (東京)、加拿大 (中部)、歐洲 (法蘭克福)、歐洲 (愛爾蘭) 和歐洲 (斯德哥爾摩)。
- 對於 G.8X 工作者類型，每個工作者都會映射到 8 個 DPU (32 個 vCPU、128 GB 記憶體)，外加 512 GB 磁碟 (大約 487 GB 可用)，並為每個工作者提供 1 個執行器。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此背景工作類型僅適用於 3.0 AWS Glue 版或更新版本的 Spark ETL 工作，與G.4X背景工作者類型支援的相同 AWS 區域。
- 對於 G.025X 工作者類型，每個工作者都會映射到 0.25 個 DPU (2 個 vCPU、4 GB 記憶體)，外加 84 GB 磁碟 (大約 34 GB 可用)，並為每個工作者提供 1 個執行器。我們建議低容量串流任務採用這種工作者類型。此背景工作類型僅適用於 3.0 AWS Glue 版串流工作。
- 對於 Z.2X 工作者類型，每個工作者都會映射到 2 個 M-DPU (8 個 vCPU、64 GB 記憶體)，外加 128 GB 磁碟 (大約 120 GB 可用)，並根據自動縮放器提供最多 8 個 Ray 工作者。

- `CodeGenConfigurationNodes` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，需符合 Custom string pattern #39。

  每個值都是 CodeGenConfigurationNode 物件。

  Glue Studio 視覺化組件和 Glue Studio 代碼的產生都根據有向無循環圖的表示形式。

- `ExecutionClass` – UTF-8 字串，長度不可超過 16 個位元組 (有效值：`FLEX=""` | `STANDARD=""`)。

  表示任務執行使用的是標準執行類別還是彈性執行類別。標準執行類別非常適合需要快速的任務啟動和專用資源的時間敏感型工作負載。

彈性執行類別適用於開始和完成時間可能會有所變化的時間敏感型任務。

只有 AWS Glue 版本 3.0 及更新版本和指令類型glueetl的工作才能設定ExecutionClass為FLEX。彈性執行類別可用於 Spark 任務。

- SourceControlDetails – SourceControlDetails 物件。

  任務原始檔控制組態的詳細資訊，可允許與遠端儲存庫雙向同步任務成品。

- MaintenanceWindow – UTF-8 字串，需符合Custom string pattern #30。

  此欄位指定串流工作的維護時段的星期幾和小時。 AWS Glue 定期執行維護活動。在這些維護時段期間， AWS Glue 將需要重新啟動您的串流工作。

  AWS Glue 將在指定維護時間的 3 小時內重新啟動工作。例如，如果您在格林威治標準時間星期一上午 10:00 設定維護時段，您的作業將在格林威治標準時間上午 10:00 至下午 1:00 之間重新啟動。

- ProfileName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與工作相關聯的 AWS Glue 使用情況設定檔名稱。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  為此任務定義而提供的唯一名稱。

錯誤

- InvalidInputException
- IdempotentParameterMismatchException
- AlreadyExistsException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentModificationException

UpdateJob 行動（Python：更新工作）

更新現有的任務定義。此資訊將完全覆寫之前的任務定義。

請求

- JobName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要更新之任務定義的名稱。

- JobUpdate – 必要：JobUpdate 物件。

  指定用於更新任務定義的值。未指定的組態將被移除或重置為預設值。

- ProfileName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與工作相關聯的 AWS Glue 使用情況設定檔名稱。

回應

- JobName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  傳回已更新之任務定義的名稱。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException

GetJob 行動（Python：獲取工作）

擷取現有的任務定義。

請求

- JobName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欲擷取的任務定義的名稱。

回應

- Job – 任務 物件。

  要求的任務定義。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetJobs 行動（Python：獲取工作）

擷取所有目前的任務定義。

請求

- NextToken – UTF-8 字串。

  接續符記，如果這是接續呼叫。
- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  回應的大小上限。

回應

- Jobs – 一個 任務 物件陣列。

  任務定義的清單。

- NextToken – UTF-8 字串。

  持續符記 (如果尚未傳回所有任務定義)。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

DeleteJob 行動（Python：刪除工作）

刪除指定的任務定義。如果找不到此任務定義，不會擲出例外狀況。

請求

- JobName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欲刪除的任務定義的名稱。

回應

- JobName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  已刪除之任務定義的名稱。

錯誤

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

ListJobs 行動（Python：列表工作）

擷取此 AWS 帳號中所有工作資源的名稱，或具有指定標籤的資源。您可運用此操作，查看帳戶下有哪些可用資源及其名稱。

此操作會接收您可在回應時做為篩選條件的選用 Tags 欄位，因此已標記的資源可分組進行擷取。如果您選擇使用標籤進行篩選，則此時只會擷取包含該標籤的資源。

請求

- NextToken – UTF-8 字串。

  接續符記，如果這是接續要求。
- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  所要回傳清單的大小上限。
- Tags – 金鑰值對的對應陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  指定只傳回包含這些標籤的資源。

回應

- JobNames – UTF-8 字串陣列。

  這個帳戶下所有任務的名稱，或是使用指定標籤的任務。
- NextToken – UTF-8 字串。

  接續字元，如果傳回的清單未包含最後一個可用指標。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

BatchGetJobs 行動（Python：批處理工作）

為指定的動作名稱清單，傳回資源中繼資料的清單。呼叫 ListJobs 操作之後，您便可以呼叫此操作來存取您已授與許可的資料。此操作支援所有 IAM 許可，包括使用標籤的許可條件。

請求

- JobNames – 必要：UTF-8 字串陣列。

  任務名稱清單，可能是從 ListJobs 操作傳回的名稱。

回應

- Jobs – 一個 任務 物件陣列。

  任務定義的清單。
- JobsNotFound – UTF-8 字串陣列。

  找不到任務名稱清單。

錯誤

- InternalServiceException
- OperationTimeoutException
- InvalidInputException

任務執行

作業執行 API 描述了中啟動、停止或檢視工作執行以及重設工作書籤相關的資料類型和 API AWS Glue。Job 流程和作業執行可存取 90 天的工作執行歷程記錄。

資料類型

- JobRun 結構
- Predecessor 結構
- JobBookmarkEntry 結構
- BatchStopJobRunSuccessfulSubmission 結構
- BatchStopJobRunError 結構

- NotificationProperty 結構

JobRun 結構

包含關於任務執行的資訊。

欄位

- Id – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  此次任務執行的 ID。
- Attempt – 數字 (整數)。

  嘗試執行此項任務的次數。
- PreviousRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  此任務先前執行作業的 ID。例如，使用 StartJobRun 動作所指定的 JobRunId。
- TriggerName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  起始此次任務執行的觸發條件的名稱。
- JobName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  此次執行所用的任務定義名稱。
- JobMode – UTF-8 字串 (有效值：SCRIPT="" | VISUAL="" | NOTEBOOK="")。

  描述如何建立工作的模式。有效的 值如下：
  - SCRIPT-工作是使用 AWS Glue Studio 指令碼編輯器建立的。
  - VISUAL-工作是使用 AWS Glue Studio 視覺化編輯器建立的。
  - NOTEBOOK-工作是使用互動式工作階段筆記本建立的。

  當JobMode欄位遺失或為 null 時，SCRIPT會指派為預設值。
- StartedOn – 時間戳記。

  此次任務執行開始的日期和時間。

- `LastModifiedOn` – 時間戳記。

  此次任務執行上次修改的時間。

- `CompletedOn` – 時間戳記。

  此次任務執行完成的日期和時間。

- `JobRunState`— UTF-8 字串 (有效值：`STARTINGRUNNING`| `STOPPING` | `STOPPED` | `SUCCEEDED` | `FAILED` | `TIMEOUT` | `ERROR` | `WAITING` |`EXPIRED`)。

  任務執行目前的狀態。如需異常終止之任務狀態的詳細資訊，請參閱 AWS Glue 任務執行狀態。

- `Arguments` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  與此次執行相關的任務引數。處理此任務執行時，這些引數會取代任務定義本身已設定的預設引數。

  您可以在這裡指定自己的工作執行腳本消耗的參數，以及 AWS Glue 本身消耗的參數。

  可以記錄任務引數。不要將純文字祕密當做引數傳遞。如果您想要將密碼保留在 Job 中，請從「 AWS Glue 連線」 AWS Secrets Manager 或其他密碼管理機制擷取密碼。

  如需如何指定和取用自有任務引數的資訊，請參閱本開發人員指南中的使用 Python 呼叫 AWS Glue API 主題。

  如需有關設定 Spark 任務時可提供給此欄位之引數的資訊，請參閱開發人員指南中的 Special Parameters Used by AWS Glue 主題。

  如需有關設定 Ray 任務時可提供給此欄位之引數的資訊，請參閱開發人員指南中的 Using job parameters in Ray jobs。

- `ErrorMessage` – UTF-8 字串。

  與此次任務執行相關的錯誤訊息。

- `PredecessorRuns` – 一個 前置任務 物件陣列。

  此次任務執行的前置任務的清單。

- `AllocatedCapacity` – 數字 (整數)。

  此欄位已作廢。請改用 `MaxCapacity`。

分配給此 JobRun單元的 AWS Glue 資料處理單元 (DPU) 數目。可分配 2 到 100 DPU，預設值為 10。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價頁面。

- `ExecutionTime` – 數字 (整數)。

  任務執行消耗資源所需的時間 (以秒為單位)。

- `Timeout` – 數字 (整數)，至少為 1。

  JobRun 逾時 (以分鐘為單位)。此為任務執行在停止並進入 `TIMEOUT` 狀態前可以消耗資源的最大時間。此值會覆寫父任務所設的逾時值。

  串流工作的逾時值必須小於 7 天或 10080 分鐘。當值保留空白時，如果您尚未設定維護時段，工作將在 7 天後重新啟動。如果您已設定維護時段，則維護時段將在 7 天後重新啟動。

- `MaxCapacity` – 數字 (雙位數)。

  對於 Glue 1.0 版或更早版本的工作，使用標準 Worker 類型，即此工作執行時可配置的 AWS Glue 資料處理單元 (DPU) 數目。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價頁面。

  對於 Glue 2.0+ 版任務，您不能指定 `Maximum capacity`。反之，您必須指定 `Worker type` 與 `Number of workers`。

  如果使用 `WorkerType` 和 `NumberOfWorkers`，請勿設定 `MaxCapacity`。

  可配置給 `MaxCapacity` 的值取決於您執行的是 Python shell 任務、Apache Spark ETL 任務或 Apache Spark Streaming ETL 任務：

  - 當您指定 Python shell 任務 (`JobCommand.Name="pythonshell"`) 時，您可以擇一分配 0.0625 或 1 個 DPU。預設為 0.0625 個 DPU。
  - 指定 Apache Spark ETL 任務 (`JobCommand.Name="glueetl"`) 或 Apache Spark Streaming ETL 任務 (`JobCommand.Name="gluestreaming"`) 時，您可以配置 2 到 100 個 DPU。預設值是 10 個 DPU。此任務類型沒有小數的 DPU 分配。

- `WorkerType` – UTF-8 字串 (有效值：`Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""`)。

  將在任務執行時分配的預先定義工作者類型。接受 Spark 任務的 G.1X、G.2X、G.4X、G.8X 或 G.025X 值。接受 Ray 任務的 Z.2X 值。

- 對於 G.1X 工作者類型，每個工作者都會映射到 1 個 DPU (4 個 vCPU、16 GB 記憶體)，外加 84 GB 磁碟 (大約 34 GB 可用)，並為每個工作者提供 1 個執行器。我們建議將此工作者類型用於資料轉換、聯結和查詢等工作負載，以提供可擴展且符合成本效益的方式來執行大部分任務。
- 對於 G.2X 工作者類型，每個工作者都會映射到 2 個 DPU (8 個 vCPU、32 GB 記憶體)，外加 128 GB 磁碟 (大約 77 GB 可用)，並為每個工作者提供 1 個執行器。我們建議將此工作者類型用於資料轉換、聯結和查詢等工作負載，以提供可擴展且符合成本效益的方式來執行大部分任務。
- 對於 G.4X 工作者類型，每個工作者都會映射到 4 個 DPU (16 個 vCPU、64 GB 記憶體)，外加 256 GB 磁碟 (大約 235 GB 可用)，並為每個工作者提供 1 個執行器。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此 Worker 類型僅適用於以下 AWS 區域 3.0 或更新 AWS Glue 版本的 Spark ETL 工作：美國東部 (俄亥俄)、美國東部 (維吉尼亞北部)、美國西部 (奧勒岡)、亞太區域 (新加坡)、亞太區域 (雪梨)、亞太區域 (東京)、加拿大 (中部)、歐洲 (法蘭克福)、歐洲 (愛爾蘭) 和歐洲 (斯德哥爾摩)。
- 對於 G.8X 工作者類型，每個工作者都會映射到 8 個 DPU (32 個 vCPU、128 GB 記憶體)，外加 512 GB 磁碟 (大約 487 GB 可用)，並為每個工作者提供 1 個執行器。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此背景工作類型僅適用於 3.0 AWS Glue 版或更新版本的 Spark ETL 工作，與G.4X背景工作者類型支援的相同 AWS 區域。
- 對於 G.025X 工作者類型，每個工作者都會映射到 0.25 個 DPU (2 個 vCPU、4 GB 記憶體)，外加 84 GB 磁碟 (大約 34 GB 可用)，並為每個工作者提供 1 個執行器。我們建議低容量串流任務採用這種工作者類型。此背景工作類型僅適用於 3.0 AWS Glue 版串流工作。
- 對於 Z.2X 工作者類型，每個工作者都會映射到 2 個 M-DPU (8 個 vCPU、64 GB 記憶體)，外加 128 GB 磁碟 (大約 120 GB 可用)，並根據自動縮放器提供最多 8 個 Ray 工作者。

- NumberOfWorkers – 數字 (整數)。

  當任務執行時所配置的已定義 workerType 的工作者數目。

- SecurityConfiguration – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  可與此任務執行搭配使用的 SecurityConfiguration 結構名稱。

- LogGroupName – UTF-8 字串。

  可在 Amazon 中 CloudWatch 使用的伺服器端加密之安全日誌記錄的日誌群組名稱 AWS KMS。此名稱可為 /aws-glue/jobs/，在此情況下預設加密為 NONE。若您新增角色名稱及 SecurityConfiguration 名稱 (即 /aws-glue/jobs-yourRoleName-yourSecurityConfigurationName/)，則該安全組態將用於加密此日誌群組。

- NotificationProperty – NotificationProperty 物件。

指定任務執行通知的組態屬性。

- `GlueVersion` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #20。

  在星火作業中，`GlueVersion`確定阿帕奇星火和 Python 的版本，在作業中 AWS Glue 可用。Python 版本指示針對 Spark 類型任務支援的版本。

  Ray 任務應將 `GlueVersion` 設定為 `4.0` 或更高版本。不過，Ray 任務中可用的 Ray、Python 和其他程式庫的版本由 Job 命令的 `Runtime` 參數決定。

  如需有關可用版 AWS Glue 本以及對應 Spark 和 Python 版本的詳細資訊，請參閱開發人員指南中的 Glue 版本。

  建立時未指定 Glue 版本的任務，預設為 Glue 0.9。

- `DPUSeconds` – 數字 (雙位數)。

  此欄位可針對使用執行類別的工作執行`FLEX`或啟用 Auto Scaling 時設定，並代表工作執行生命週期期間每個執行程式執行的總時間 (以秒為單位) 乘以 DPU 因子 (1 代表`G.1X`、2 代表`G.2X`，或者工作程式為 0.25)。`G.025X`此值可能不同於 Auto Scaling 任務案例中的 `executionEngineRuntime` * `MaxCapacity`，因為在給定時間執行的執行程序數量可能少於 `MaxCapacity`。因此，`DPUSeconds` 的值有可能小於 `executionEngineRuntime` * `MaxCapacity`。

- `ExecutionClass` – UTF-8 字串，長度不可超過 16 個位元組 (有效值：`FLEX=""` | `STANDARD=""`)。

  表示任務執行使用的是標準執行類別還是彈性執行類別。標準執行類別非常適合需要快速的任務啟動和專用資源的時間敏感型工作負載。

  彈性執行類別適用於開始和完成時間可能會有所變化的時間敏感型任務。

  只有 AWS Glue 版本 3.0 及更新版本和指令類型`glueetl`的工作才能設定`ExecutionClass`為`FLEX`。彈性執行類別可用於 Spark 任務。

- `MaintenanceWindow` – UTF-8 字串，需符合Custom string pattern #30。

  此欄位指定串流工作的維護時段的星期幾和小時。 AWS Glue 定期執行維護活動。在這些維護時段期間， AWS Glue 將需要重新啟動您的串流工作。

AWS Glue 將在指定維護時間的 3 小時內重新啟動工作。例如，如果您在格林威治標準時間星期一上午 10:00 設定維護時段，您的作業將在格林威治標準時間上午 10:00 至下午 1:00 之間重新啟動。

- `ProfileName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與工作執行相關聯的 AWS Glue 使用情況設定檔名稱。

Predecessor 結構

任務執行，在用來觸發此任務執行的觸發條件的述詞中使用。

欄位

- `JobName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  前置任務執行所用的任務定義名稱。
- `RunId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  前置任務執行作業的任務執行 ID。

JobBookmarkEntry 結構

定義可供任務繼續處理的位置點。

欄位

- `JobName` – UTF-8 字串。

  正在討論的任務的名稱。
- `Version` – 數字 (整數)。

  任務的版本。
- `Run` – 數字 (整數)。

  執行 ID 編號。
- `Attempt` – 數字 (整數)。

嘗試 ID 編號。

- PreviousRunId – UTF-8 字串。

  與之前任務執行相關聯的唯一執行識別符。

- RunId – UTF-8 字串。

  執行 ID 編號。

- JobBookmark – UTF-8 字串。

  書籤本身。

BatchStopJobRunSuccessfulSubmission 結構

記錄停止了指定 JobRun 的成功要求。

欄位

- JobName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  先前停止的任務執行中所用的任務定義名稱。

- JobRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  已停止之任務執行的 JobRunId。

BatchStopJobRunError 結構

記錄在嘗試停止指定的任務執行時所發生的錯誤。

欄位

- JobName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  正在討論的任務執行中所用的任務定義名稱。

- JobRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

任務執行有問題的 JobRunId。

- ErrorDetail – ErrorDetail 物件。

  指定關於所發生錯誤的詳細資訊。

NotificationProperty 結構

指定通知的組態屬性。

欄位

- NotifyDelayAfter – 數字 (整數)，至少為 1。

  任務執行開始後，在傳送任務執行延遲通知之前所要等待的分鐘數。

作業

- StartJobRun 行動（Python：開始工作運行）
- BatchStopJobRun 行動（Python：批處理停工運行）
- GetJobRun 行動（Python：獲取工作運行）
- GetJobRuns 行動（Python：獲取工作運行）
- GetJobBookmark 行動（Python：獲取工作書籤）
- GetJobBookmarks 行動（Python：獲取工作書籤）
- ResetJobBookmark 行動（Python：重置工作書籤）

StartJobRun 行動（Python：開始工作運行）

使用任務定義開始任務執行。

請求

- JobName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欲使用的任務定義的名稱。
- JobRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

要重試之先前 JobRun 的 ID。

- `Arguments` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  與此次執行相關的任務引數。處理此任務執行時，這些引數會取代任務定義本身已設定的預設引數。

  您可以在這裡指定自己的工作執行腳本消耗的參數，以及 AWS Glue 本身消耗的參數。

  可以記錄任務引數。不要將純文字祕密當做引數傳遞。如果您想要將密碼保留在 Job 中，請從「 AWS Glue 連線」 AWS Secrets Manager 或其他密碼管理機制擷取密碼。

  如需如何指定和取用自有任務引數的資訊，請參閱本開發人員指南中的使用 Python 呼叫 AWS Glue API 主題。

  如需有關設定 Spark 任務時可提供給此欄位之引數的資訊，請參閱開發人員指南中的 Special Parameters Used by AWS Glue 主題。

  如需有關設定 Ray 任務時可提供給此欄位之引數的資訊，請參閱開發人員指南中的 Using job parameters in Ray jobs。

- `AllocatedCapacity` – 數字 (整數)。

  此欄位已作廢。請改用 `MaxCapacity`。

  要配置給此 JobRun單元的 AWS Glue 資料處理單元 (DPU) 數目。您可以分配至少 2 個 DPU，預設值為 10。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價頁面。

- `Timeout` – 數字 (整數)，至少為 1。

  JobRun 逾時 (以分鐘為單位)。此為任務執行在停止並進入 `TIMEOUT` 狀態前可以消耗資源的最大時間。此值會覆寫父任務所設的逾時值。

  串流工作的逾時值必須小於 7 天或 10080 分鐘。當值保留空白時，如果您尚未設定維護時段，工作將在 7 天後重新啟動。如果您已設定維護時段，則維護時段將在 7 天後重新啟動。

- `MaxCapacity` – 數字 (雙位數)。

對於 Glue 1.0 版或更早版本的工作，使用標準 Worker 類型，即此工作執行時可配置的 AWS Glue 資料處理單元 (DPU) 數目。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價頁面。

對於 Glue 2.0+ 版任務，您不能指定 `Maximum capacity`。反之，您必須指定 `Worker type` 與 `Number of workers`。

如果使用 `WorkerType` 和 `NumberOfWorkers`，請勿設定 `MaxCapacity`。

可配置給 `MaxCapacity` 的值取決於您執行的是 Python shell 任務、Apache Spark ETL 任務或 Apache Spark Streaming ETL 任務：

- 當您指定 Python shell 任務 (`JobCommand.Name="pythonshell"`) 時，您可以擇一分配 0.0625 或 1 個 DPU。預設為 0.0625 個 DPU。
- 指定 Apache Spark ETL 任務 (`JobCommand.Name="glueetl"`) 或 Apache Spark Streaming ETL 任務 (`JobCommand.Name="gluestreaming"`) 時，您可以配置 2 到 100 個 DPU。預設值是 10 個 DPU。此任務類型沒有小數的 DPU 分配。

- `SecurityConfiguration` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  可與此任務執行搭配使用的 `SecurityConfiguration` 結構名稱。

- `NotificationProperty` – NotificationProperty 物件。

  指定任務執行通知的組態屬性。

- `WorkerType` – UTF-8 字串 (有效值：`Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""`)。

  將在任務執行時分配的預先定義工作者類型。接受 Spark 任務的 G.1X、G.2X、G.4X、G.8X 或 G.025X 值。接受 Ray 任務的 Z.2X 值。

  - 對於 `G.1X` 工作者類型，每個工作者都會映射到 1 個 DPU (4 個 vCPU、16 GB 記憶體)，外加 84 GB 磁碟 (大約 34 GB 可用)，並為每個工作者提供 1 個執行器。我們建議將此工作者類型用於資料轉換、聯結和查詢等工作負載，以提供可擴展且符合成本效益的方式來執行大部分任務。
  - 對於 `G.2X` 工作者類型，每個工作者都會映射到 2 個 DPU (8 個 vCPU、32 GB 記憶體)，外加 128 GB 磁碟 (大約 77 GB 可用)，並為每個工作者提供 1 個執行器。我們建議將此工作者類型用於資料轉換、聯結和查詢等工作負載，以提供可擴展且符合成本效益的方式來執行大部分任務。
  - 對於 `G.4X` 工作者類型，每個工作者都會映射到 4 個 DPU (16 個 vCPU、64 GB 記憶體)，外加 256 GB 磁碟 (大約 235 GB 可用)，並為每個工作者提供 1 個執行器。我們建議工作負載包含最嚴

苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此 Worker 類型僅適用於以下 AWS 區域 3.0 或更新 AWS Glue 版本的 Spark ETL 工作：美國東部 (俄亥俄)、美國東部 (維吉尼亞北部)、美國西部 (奧勒岡)、亞太區域 (新加坡)、亞太區域 (雪梨)、亞太區域 (東京)、加拿大 (中部)、歐洲 (法蘭克福)、歐洲 (愛爾蘭) 和歐洲 (斯德哥爾摩)。

- 對於 G.8X 工作者類型，每個工作者都會映射到 8 個 DPU (32 個 vCPU、128 GB 記憶體)，外加 512 GB 磁碟 (大約 487 GB 可用)，並為每個工作者提供 1 個執行器。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此背景工作類型僅適用於 3.0 AWS Glue 版或更新版本的 Spark ETL 工作，與G.4X背景工作者類型支援的相同 AWS 區域。
- 對於 G.025X 工作者類型，每個工作者都會映射到 0.25 個 DPU (2 個 vCPU、4 GB 記憶體)，外加 84 GB 磁碟 (大約 34 GB 可用)，並為每個工作者提供 1 個執行器。我們建議低容量串流任務採用這種工作者類型。此背景工作類型僅適用於 3.0 AWS Glue 版串流工作。
- 對於 Z.2X 工作者類型，每個工作者都會映射到 2 個 M-DPU (8 個 vCPU、64 GB 記憶體)，外加 128 GB 磁碟 (大約 120 GB 可用)，並根據自動縮放器提供最多 8 個 Ray 工作者。

- NumberOfWorkers – 數字 (整數)。

  當任務執行時所配置的已定義 workerType 的工作者數目。

- ExecutionClass – UTF-8 字串，長度不可超過 16 個位元組 (有效值：FLEX="" | STANDARD="")。

  表示任務執行使用的是標準執行類別還是彈性執行類別。標準執行類別非常適合需要快速的任務啟動和專用資源的時間敏感型工作負載。

  彈性執行類別適用於開始和完成時間可能會有所變化的時間敏感型任務。

  只有 AWS Glue 版本 3.0 及更新版本和指令類型glueetl的工作才能設定ExecutionClass為FLEX。彈性執行類別可用於 Spark 任務。

- ProfileName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與工作執行相關聯的 AWS Glue 使用情況設定檔名稱。

回應

- JobRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  指派給此次任務執行的 ID。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentRunsExceededException

BatchStopJobRun 行動（Python：批處理停工運行）

針對指定的任務定義停止一個或多個任務執行作業。

請求

- JobName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  用於停止任務執行的任務定義的名稱。
- JobRunIds – 必要：UTF-8 字串的陣列，不可小於 1 或超過 25 個字串。

  因該項任務定義而應停止的 JobRunIds 清單。

回應

- SuccessfulSubmissions – 一個 BatchStopJobRunSuccessfulSubmission 物件陣列。

  已成功提交以 JobRuns 供停止的清單。
- Errors – 一個 BatchStopJobRunError 物件陣列。

  在嘗試停止 JobRuns 時所發生錯誤的清單，包括針對每項錯誤列出的 JobRunId，以及關於錯誤的詳細資訊。

錯誤

- InvalidInputException
- InternalServiceException

- OperationTimeoutException

GetJobRun 行動（Python：獲取工作運行）

擷取特定任務執行作業的中繼資料。Job 流程和作業執行可存取 90 天的工作執行歷程記錄。

請求

- JobName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  正在執行的任務定義的名稱。
- RunId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  此次任務執行的 ID。
- PredecessorsIncluded – 布林值。

  如果應傳回前置任務執行作業的清單，則此值為 true。

回應

- JobRun – JobRun 物件。

  所要求任務執行作業的中繼資料。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetJobRuns 行動（Python：獲取工作運行）

擷取特定任務定義所有執行作業的中繼資料。

請求

- JobName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要擷取其所有任務執行作業資料的任務定義的名稱。

- NextToken – UTF-8 字串。

  接續符記，如果這是接續呼叫。

- MaxResults— 數字 (整數)，不小於 1 或大於 200。

  回應的大小上限。

回應

- JobRuns – 一個 JobRun 物件陣列。

  任務執行中繼資料物件的清單。

- NextToken – UTF-8 字串。

  持續符記 (如果尚未傳回所有重新要求的任務執行作業的結果)。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetJobBookmark 行動（Python：獲取工作書籤）

傳回任務書籤項目的資訊。

如需有關啟用和使用任務書籤的詳細資訊，請參閱：

- 使用任務書籤追蹤處理的資料
- 使用的 Job 參數 AWS Glue

- 任務結構

請求

- JobName – 必要：UTF-8 字串。

  正在討論的任務的名稱。
- Version – 數字 (整數)。

  任務的版本。
- RunId – UTF-8 字串。

  與此任務執行相關聯的唯一執行識別符。

回應

- JobBookmarkEntry – JobBookmarkEntry 物件。

  一種結構，定義任務可繼續處理的點。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ValidationException

GetJobBookmarks 行動（Python：獲取工作書籤）

傳回任務書籤項目的資訊。此清單是按照遞減的版本號碼排序的。

如需有關啟用和使用任務書籤的詳細資訊，請參閱：

- 使用任務書籤追蹤處理的資料
- 使用的 Job 參數 AWS Glue
- 任務結構

請求

- JobName – 必要：UTF-8 字串。

  正在討論的任務的名稱。
- MaxResults – 數字 (整數)。

  回應的大小上限。
- NextToken – 數字 (整數)。

  接續符記，如果這是接續呼叫。

回應

- JobBookmarkEntries – 一個 JobBookmarkEntry 物件陣列。

  任務書籤項目的清單，定義工作可以繼續處理的點。
- NextToken – 數字 (整數)。

  接續符記，如果傳回所有項目，其值為 1；如果沒有傳回所有請求的任務執行，則傳回 > 1 的值。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

ResetJobBookmark 行動（Python：重置工作書籤）

重設書籤項目。

如需有關啟用和使用任務書籤的詳細資訊，請參閱：

- 使用任務書籤追蹤處理的資料
- 使用的 Job 參數 AWS Glue
- 任務結構

請求

- JobName – 必要：UTF-8 字串。

  正在討論的任務的名稱。
- RunId – UTF-8 字串。

  與此任務執行相關聯的唯一執行識別符。

回應

- JobBookmarkEntry – JobBookmarkEntry 物件。

  重設的書籤項目。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

觸發

觸發器 API 描述了與中建立、更新或刪除以及啟動和停止工作觸發器相關的資料類型和 API AWS Glue。

資料類型

- Trigger 結構
- TriggerUpdate 結構
- Predicate 結構
- Condition 結構
- Action 結構
- EventBatchingCondition 結構

Trigger 結構

關於特定觸發條件的資訊。

欄位

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  觸發條件的名稱。
- WorkflowName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與觸發條件相關聯的工作流程名稱。
- Id – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  保留以供日後使用。
- Type – UTF-8 字串 (有效值：SCHEDULED | CONDITIONAL | ON_DEMAND | EVENT)。

  此處觸發條件的類型。
- State – UTF-8 字串 (有效值：CREATING | CREATED | ACTIVATING | ACTIVATED | DEACTIVATING | DEACTIVATED | DELETING | UPDATING)。

  觸發條件目前的狀態。
- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  此觸發條件的說明。
- Schedule – UTF-8 字串。

  用來指定排程的 cron 表達式 (請參閱適用於任務與爬蟲程式的依時排程。例如，如果要每天在 12:15 UTC 執行某項動作，您可以指定：cron(15 12 * * ? *)。
- Actions – 一個 動作 物件陣列。

  此觸發條件起始的動作。
- Predicate – 述詞 物件。

  此觸發條件的述詞，定義了觸發的時間點。

- `EventBatchingCondition` – EventBatching條件 物件。

  觸發事件觸發之前必須滿足的 Batch 條件 (接收到的指定 EventBridge 事件數或批次時間範圍到期)。

TriggerUpdate 結構

一種結構，用來提供更新觸發條件時所使用的資訊。此物件在更新先前的觸發條件定義時，會完全覆寫掉之前的內容。

欄位

- `Name` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  保留以供日後使用。

- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  此觸發條件的說明。

- `Schedule` – UTF-8 字串。

  用來指定排程的 `cron` 表達式 (請參閱適用於任務與爬蟲程式的依時排程。例如，如果要每天在 12:15 UTC 執行某項動作，您可以指定：`cron(15 12 * * ? *)`。

- `Actions` – 一個 動作 物件陣列。

  此觸發條件起始的動作。

- `Predicate` – 述詞 物件。

  此觸發條件的述詞，定義了觸發的時間點。

- `EventBatchingCondition` – EventBatching條件 物件。

  觸發事件觸發之前必須滿足的 Batch 條件 (接收到的指定 EventBridge 事件數或批次時間範圍到期)。

Predicate 結構

定義觸發條件的述詞，此述詞會決定觸發的時間點。

欄位

- Logical – UTF-8 字串 (有效值：AND | ANY)。

  如果只列出一個條件，則為選用欄位。如果列出了多個條件，則此為必要欄位。
- Conditions – 一個 條件 物件陣列。

  觸發條件的清單，這些條件決定了觸發的時間點。

Condition 結構

定義觸發的條件。

欄位

- LogicalOperator – UTF-8 字串 (有效值：EQUALS)。

  邏輯運算子。
- JobName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  任務的名稱，此條件會套用至該任務的 JobRuns，此觸發條件也會等待起始該任務。
- State— UTF-8 字串 (有效值：STARTINGRUNNING| STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT | ERROR | WAITING |EXPIRED)。

  條件的狀態。目前，觸發條件可聆聽的任務狀態只有 SUCCEEDED、STOPPED、FAILED 和 TIMEOUT。觸發條件可聆聽的爬蟲程式狀態只有 SUCCEEDED、FAILED 和 CANCELLED。
- CrawlerName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要套用此條件的爬取程式名稱。
- CrawlState – UTF-8 字串 (有效值：RUNNING | CANCELLING | CANCELLED | SUCCEEDED | FAILED | ERROR)。

  要套用此條件的爬取程式狀態。

Action 結構

定義觸發條件要起始的動作。

欄位

- JobName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要執行的任務的名稱。

- Arguments – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  此觸發條件觸發時所使用的任務引數。處理此任務執行時，這些引數會取代任務定義本身已設定的預設引數。

  您可以在這裡指定自己的工作執行腳本消耗的參數，以及 AWS Glue 本身消耗的參數。

  如需如何指定和取用自有任務引數的資訊，請參閱本開發人員指南中的使用 Python 呼叫 AWS Glue API 主題。

  如需設定工作所 AWS Glue 耗用之索引鍵值配對的詳細資訊，請參閱開發人員指南中的「使用的特殊參數」 AWS Glue主題。

- Timeout – 數字 (整數)，至少為 1。

  JobRun 逾時 (以分鐘為單位)。此為任務執行在停止並進入 TIMEOUT 狀態前可以消耗資源的最大時間。預設值為 2,880 分鐘 (48 小時)。此會覆寫父任務所設的逾時值。

- SecurityConfiguration – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  可與此動作搭配使用的 SecurityConfiguration 結構名稱。

- NotificationProperty – NotificationProperty 物件。

  指定任務執行通知的組態屬性。

- CrawlerName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要搭配此動作使用的爬取程式名稱。

EventBatchingCondition 結構

觸發事件觸發之前必須滿足的 Batch 條件 (接收到的指定 EventBridge 事件數或批次時間範圍到期)。

欄位

- `BatchSize` – 必要：數字 (整數)，不可小於 1，也不可以大於 100。

  EventBridge 事件觸發 EventBridge 前必須從 Amazon 接收的事件數目。
- `BatchWindow` – 數字 (整數)，不可小於 1，也不可以大於 900。

  EventBridge 事件觸發器觸發後的時間範圍 (以秒為單位)。時段在接收到第一個事件時啟動。

作業

- CreateTrigger 動作（Python：創建觸發器）
- StartTrigger 行動（Python：啟動觸發器）
- GetTrigger 行動（Python：獲取觸發器）
- GetTriggers 行動（Python：獲取觸發器）
- UpdateTrigger 行動（Python：更新觸發器）
- StopTrigger 行動（Python：停止觸發器）
- DeleteTrigger 動作（Python：刪除觸發器）
- ListTriggers 行動（Python：列表觸發器）
- BatchGetTriggers 行動（Python：批處理觸發器）

CreateTrigger 動作（Python：創建觸發器）

建立新的觸發條件。

請求

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  觸發條件的名稱。
- `WorkflowName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

與觸發條件相關聯的工作流程名稱。

- Type – 必要：UTF-8 字串 (有效值：SCHEDULED | CONDITIONAL | ON_DEMAND | EVENT)。

  新觸發條件的類型。

- Schedule – UTF-8 字串。

  用來指定排程的 cron 表達式 (請參閱適用於任務與爬蟲程式的依時排程。例如，如果要每天在 12:15 UTC 執行某項動作，您可以指定：cron(15 12 * * ? *)。

  當觸發類型為 SCHEDULED 時，此欄位為必要。

- Predicate – 述詞 物件。

  用來指定新觸發條件觸發時間點的述詞。

  當觸發類型為 CONDITIONAL 時，此欄位為必要。

- Actions – 必要：一個 動作 物件。

  此觸發條件觸發時所起始的動作。

- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  新觸發條件的說明。

- StartOnCreation – 布林值。

  設定 true，即可在建立時啟動 SCHEDULED 和 CONDITIONAL 觸發。True 不支援 ON_DEMAND 觸發。

- Tags – 金鑰值對的對應陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  要搭配此觸發條件使用的標籤。您可以使用標籤來限制對於觸發條件的存取情況。如需中標籤的詳細資訊 AWS Glue，請參閱開發人員指南AWS Glue中的「AWS 標籤」。

- EventBatchingCondition – EventBatching條件 物件。

  觸發事件觸發之前必須滿足的 Batch 條件 (接收到的指定 EventBridge 事件數或批次時間範圍到期)。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  觸發條件的名稱。

錯誤

- `AlreadyExistsException`
- `EntityNotFoundException`
- `InvalidInputException`
- `IdempotentParameterMismatchException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `ConcurrentModificationException`

StartTrigger 行動（Python：啟動觸發器）

啟動現有的觸發條件。請參閱觸發任務，以了解如何啟動不同類型觸發條件的資訊。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  所要啟動觸發條件的名稱。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  已啟動的觸發條件的名稱。

錯誤

- InvalidInputException
- InternalServiceException
- EntityNotFoundException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentRunsExceededException

GetTrigger 行動（Python：獲取觸發器）

擷取觸發條件的定義。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要擷取的觸發條件的名稱。

回應

- Trigger – 觸發條件 物件。

  要求的觸發條件定義。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetTriggers 行動（Python：獲取觸發器）

取得與任務相關的所有觸發條件。

請求

- NextToken – UTF-8 字串。

  接續符記，如果這是接續呼叫。
- DependentJobName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  擷取觸發之任務的名稱。將會傳回可起始此項任務的觸發條件，如果沒有此等觸發條件，將傳回所有的觸發條件。
- MaxResults— 數字 (整數)，不小於 1 或大於 200。

  回應的大小上限。

回應

- Triggers – 一個 觸發條件 物件陣列。

  指定任務適用的觸發條件的清單。
- NextToken – UTF-8 字串。

  持續符記 (如果尚未傳回所有要求的觸發條件)。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

UpdateTrigger 行動（Python：更新觸發器）

更新觸發定義。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

所要更新觸發條件的名稱。

- `TriggerUpdate` – 必要：TriggerUpdate 物件。

  新的值，用來更新觸發條件。

回應

- `Trigger` – 觸發條件 物件。

  產生的觸發條件定義。

錯誤

- `InvalidInputException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

StopTrigger 行動（Python：停止觸發器）

停止指定的觸發條件。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  所要停止的觸發條件的名稱。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  已停止的觸發條件的名稱。

錯誤

- InvalidInputException
- InternalServiceException
- EntityNotFoundException
- OperationTimeoutException
- ConcurrentModificationException

DeleteTrigger 動作（Python：刪除觸發器）

刪除指定的觸發條件。如果找不到此觸發條件，就不會拋出例外狀況。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  所要刪除的觸發條件的名稱。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  已刪除的觸發條件的名稱。

錯誤

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException

ListTriggers 行動（Python：列表觸發器）

檢索此 AWS 帳戶中所有觸發器資源的名稱，或具有指定標籤的資源。您可運用此操作，查看帳戶下有哪些可用資源及其名稱。

此操作會接收您可在回應時做為篩選條件的選用 Tags 欄位，因此已標記的資源可分組進行擷取。如果您選擇使用標籤進行篩選，則此時只會擷取包含該標籤的資源。

請求

- NextToken – UTF-8 字串。

  接續符記，如果這是接續要求。

- DependentJobName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要擷取其觸發條件的工作的名稱。這時會傳回可起始此任務的觸發條件。如果這時沒有這種觸發條件，則會傳回所有的觸發條件。

- MaxResults— 數字 (整數)，不小於 1 或大於 200。

  所要回傳清單的大小上限。

- Tags – 金鑰值對的對應陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  指定只傳回包含這些標籤的資源。

回應

- TriggerNames – UTF-8 字串陣列。

  這個帳戶下所有觸發條件的名稱，或是使用指定標籤的觸發條件。

- NextToken – UTF-8 字串。

  接續字元，如果傳回的清單未包含最後一個可用指標。

錯誤

- EntityNotFoundException
- InvalidInputException
- InternalServiceException

- OperationTimeoutException

BatchGetTriggers 行動（Python：批處理觸發器）

為指定的觸發條件名稱清單，傳回資源中繼資料的清單。呼叫 ListTriggers 操作之後，您便可以呼叫此操作來存取您已授與許可的資料。此操作支援所有 IAM 許可，包括使用標籤的許可條件。

請求

- TriggerNames – 必要：UTF-8 字串陣列。

  觸發條件名稱清單，可能是從 ListTriggers 操作傳回的名稱。

回應

- Triggers – 一個 觸發條件 物件陣列。

  觸發條件定義的清單。
- TriggersNotFound – UTF-8 字串陣列。

  找不到觸發條件名稱清單。

錯誤

- InternalServiceException
- OperationTimeoutException
- InvalidInputException

互動式工作階段 API

互動式工作階段 API 描述了與使用 AWS Glue 互動式工作階段建置和測試擷取、轉換和載入 (ETL) 指令碼以進行資料整合相關的 AWS Glue API。

資料類型

- Session 結構
- SessionCommand 結構

- Statement 結構
- StatementOutput 結構
- StatementOutputData 結構
- ConnectionsList 結構

Session 結構

遠端 Spark 執行階段環境執行的期間。

欄位

- `Id` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  工作階段的 ID。
- `CreatedOn` – 時間戳記。

  建立工作階段的時間和日期。
- `Status` – UTF-8 字串 (有效值：`PROVISIONING` | `READY` | `FAILED` | `TIMEOUT` | `STOPPING` | `STOPPED`)。

  工作階段狀態。
- `ErrorMessage` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  工作階段期間顯示的錯誤訊息。
- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  工作階段的描述。
- `Role` – UTF-8 字串，長度不可小於 20 個位元組，也不可以超過 2048 個位元組，且需符合 Custom string pattern #26。

  與此工作階段相關聯 IAM 角色的名稱或 Amazon Resource Name (ARN)。
- `Command` – SessionCommand 物件。

  指令物件。請 SessionCommand參閱。

- `DefaultArguments` – 鍵值對的映射陣列，不超過 75 對。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，需符合 Custom string pattern #27。

  每個值都是 UTF-8 字串，長度不可超過 4096 個位元組，需符合 URI address multi-line string pattern。

  金鑰值對的映射陣列。上限為 75 對。

- `Connections` – ConnectionsList 物件。

  用於工作階段的連線數量。

- `Progress` – 數字 (雙位數)。

  工作階段的程式碼執行進度。

- `MaxCapacity` – 數字 (雙位數)。

  工作執行時可配置的 AWS Glue 資料處理單元 (DPU) 數目。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。

- `SecurityConfiguration` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要與會話一起使用的 SecurityConfiguration 結構的名稱。

- `GlueVersion` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #20。

  該 AWS Glue 版本決定了阿帕奇星火和 Python AWS Glue 支持的版本。必 GlueVersion 須大於 2.0。

- `DataAccessId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 36 個位元組。

  工作階段的資料存取 ID。

- `PartitionId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 36 個位元組。

  工作階段的分割區 ID。

- `NumberOfWorkers` – 數字 (整數)。

  用於工作階段的已定義 `WorkerType` 的工作者數量。

- `WorkerType` – UTF-8 字串 (有效值：`Standard=""` | `G.1X=""` | `G.2X=""` | `G.025X=""` | `G.4X=""` | `G.8X=""` | `Z.2X=""`)。

  工作階段執行時分配的預先定義工作者類型。接受 Spark 工作階段的 `G.1X`、`G.2X`、`G.4X` 或 `G.8X` 值。接受 Ray 工作階段的 `Z.2X` 值。

- `CompletedOn` – 時間戳記。

  此工作階段完成的日期和時間。

- `ExecutionTime` – 數字 (雙位數)。

  工作階段執行的總時間。

- `DPUSeconds` – 數字 (雙位數)。

  工作階段所使用的 DPU (公式： ExecutionTime * MaxCapacity)。

- `IdleTimeout` – 數字 (整數)。

  工作階段逾時前閒置的分鐘數。

- `ProfileName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與工作階段相關聯的 AWS Glue 使用情況設定檔名稱。

SessionCommand 結構

執行任務的 `SessionCommand`。

欄位

- `Name` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  指定的名稱 SessionCommand。可以是 'glueetl' 或 'gluestreaming'。

- `PythonVersion` – UTF-8 字串，需符合Custom string pattern #21。

  指定 Python 版本。Python 版本指示針對 Spark 類型任務支援的版本。

Statement 結構

在工作階段中發生的特定動作陳述式或請求。

欄位

- `Id` – 數字 (整數)。

  陳述式的 ID。
- `Code` – UTF-8 字串。

  陳述式的執行程式碼。
- `State` – UTF-8 字串 (有效值：`WAITING` | `RUNNING` | `AVAILABLE` | `CANCELLING` | `CANCELLED` | `ERROR`)。

  處理請求時的狀態。
- `Output` – StatementOutput 物件。

  JSON 格式的輸出。
- `Progress` – 數字 (雙位數)。

  程式碼執行進度。
- `StartedOn` – 數字 (long)。

  開始任務定義的 unix 時間和日期。
- `CompletedOn` – 數字 (long)。

  完成任務定義的 unix 時間和日期。

StatementOutput 結構

JSON 格式的程式碼執行輸出。

欄位

- `Data` – StatementOutputData 物件。

  程式碼執行輸出。
- `ExecutionCount` – 數字 (整數)。

輸出的執行計數。

- `Status` – UTF-8 字串 (有效值：`WAITING` | `RUNNING` | `AVAILABLE` | `CANCELLING` | `CANCELLED` | `ERROR`)。

  程式碼執行輸出的狀態。

- `ErrorName` – UTF-8 字串。

  輸出中的錯誤名稱。

- `ErrorValue` – UTF-8 字串。

  輸出的錯誤值。

- `Traceback` – UTF-8 字串陣列。

  輸出的回溯。

StatementOutputData 結構

JSON 格式的程式碼執行輸出。

欄位

- `TextPlain` – UTF-8 字串。

  文字格式的程式碼執行輸出。

ConnectionsList 結構

指定任務所使用的連線。

欄位

- `Connections` – UTF-8 字串陣列。

  任務所使用連線的清單。

作業

- CreateSession 行動（Python：創建會話）

- StopSession 行動（Python：停止會話）
- DeleteSession 行動（Python：刪除會話）
- GetSession 行動（Python：獲取會話）
- ListSessions 行動（Python：列表會話）
- RunStatement 行動（Python：運行語句）
- CancelStatement 行動（Python：取消語句）
- GetStatement 行動（Python：獲取語句）
- ListStatements 行動（Python：列表語句）

CreateSession 行動（Python：創建會話）

建立新的工作階段。

請求

請求建立新的工作階段。

- `Id` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  工作階段請求的 ID。
- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  工作階段的描述。
- `Role` – 必要：UTF-8 字串，長度不可小於 20 個位元組，也不可以超過 2048 個位元組，且需符合Custom string pattern #26。

  IAM 角色 ARN
- `Command` – 必要：SessionCommand 物件。

  執行任務的 `SessionCommand`。
- `Timeout` – 數字 (整數)，至少為 1。

  工作階段逾時前的分鐘數。Spark ETL 任務的預設值為 48 小時 (2880 分鐘)，這是此任務類型的工作階段存留期上限。如需其他任務類型，請參閱文件。

- `IdleTimeout` – 數字 (整數)，至少為 1。

  工作階段逾時前閒置的分鐘數。Spark ETL 任務的預設值為逾時值。如需其他任務類型，請參閱文件。

- `DefaultArguments` – 鍵值對的映射陣列，不超過 75 對。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，需符合 Custom string pattern #27。

  每個值都是 UTF-8 字串，長度不可超過 4096 個位元組，需符合 URI address multi-line string pattern。

  金鑰值對的映射陣列。上限為 75 對。

- `Connections` – ConnectionsList 物件。

  用於工作階段的連線數量。

- `MaxCapacity` – 數字 (雙位數)。

  工作執行時可配置的 AWS Glue 資料處理單元 (DPU) 數目。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。

- `NumberOfWorkers` – 數字 (整數)。

  用於工作階段的已定義 `WorkerType` 的工作者數量。

- `WorkerType` – UTF-8 字串 (有效值：`Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""`)。

  將在任務執行時分配的預先定義工作者類型。接受 Spark 任務的 G.1X、G.2X、G.4X 或 G.8X 值。接受 Ray 筆記本的 Z.2X 值。

  - 對於 `G.1X` 工作者類型，每個工作者都會映射到 1 個 DPU (4 個 vCPU、16 GB 記憶體)，外加 84 GB 磁碟 (大約 34 GB 可用)，並為每個工作者提供 1 個執行器。我們建議將此工作者類型用於資料轉換、聯結和查詢等工作負載，以提供可擴展且符合成本效益的方式來執行大部分任務。
  - 對於 `G.2X` 工作者類型，每個工作者都會映射到 2 個 DPU (8 個 vCPU、32 GB 記憶體)，外加 128 GB 磁碟 (大約 77 GB 可用)，並為每個工作者提供 1 個執行器。我們建議將此工作者類型用於資料轉換、聯結和查詢等工作負載，以提供可擴展且符合成本效益的方式來執行大部分任務。
  - 對於 `G.4X` 工作者類型，每個工作者都會映射到 4 個 DPU (16 個 vCPU、64 GB 記憶體)，外加 256 GB 磁碟 (大約 235 GB 可用)，並為每個工作者提供 1 個執行器。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此 Worker 類型僅適用於以下 AWS 區域

3.0 或更新 AWS Glue 版本的 Spark ETL 工作：美國東部 (俄亥俄)、美國東部 (維吉尼亞北部)、美國西部 (奧勒岡)、亞太區域 (新加坡)、亞太區域 (雪梨)、亞太區域 (東京)、加拿大 (中部)、歐洲 (法蘭克福)、歐洲 (愛爾蘭) 和歐洲 (斯德哥爾摩)。

- 對於 G.8X 工作者類型，每個工作者都會映射到 8 個 DPU (32 個 vCPU、128 GB 記憶體)，外加 512 GB 磁碟 (大約 487 GB 可用)，並為每個工作者提供 1 個執行器。我們建議工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務採用這種工作者類型。此背景工作類型僅適用於 3.0 AWS Glue 版或更新版本的 Spark ETL 工作，與G.4X背景工作者類型支援的相同 AWS 區域。
- 對於 Z.2X 工作者類型，每個工作者都會映射到 2 個 M-DPU (8 個 vCPU、64 GB 記憶體)，外加 128 GB 磁碟 (大約 120 GB 可用)，並根據自動縮放器提供最多 8 個 Ray 工作者。

- SecurityConfiguration – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  要與會話使用的 SecurityConfiguration 結構的名稱

- GlueVersion – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #20。

  該 AWS Glue 版本決定了阿帕奇星火和 Python AWS Glue 支持的版本。必 GlueVersion 須大於 2.0。

- DataAccessId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 36 個位元組。

  工作階段的資料存取 ID。

- PartitionId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 36 個位元組。

  工作階段的分割區 ID。

- Tags – 金鑰值對的映射陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  屬於工作階段的鍵值對 (標籤) 的映射。

- RequestOrigin – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  請求的來源。

- ProfileName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

與工作階段相關聯的 AWS Glue 使用情況設定檔名稱。

回應

- `Session` – Session (工作階段) 物件。

  在回應中傳回工作階段物件。

錯誤

- `AccessDeniedException`
- `IdempotentParameterMismatchException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `ValidationException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`

StopSession 行動（Python：停止會話）

停止工作階段。

請求

- `Id` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欲停止的工作階段 ID。

- `RequestOrigin` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  請求的來源。

回應

- Id – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  傳回已停止的工作階段 ID。

錯誤

- AccessDeniedException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- IllegalSessionStateException
- ConcurrentModificationException

DeleteSession 行動（Python：刪除會話）

刪除工作階段。

請求

- Id – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欲刪除的工作階段 ID。
- RequestOrigin – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  刪除工作階段請求的來源名稱。

回應

- Id – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  傳回已刪除的工作階段 ID。

錯誤

- AccessDeniedException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- IllegalSessionStateException
- ConcurrentModificationException

GetSession 行動（Python：獲取會話）

擷取工作階段。

請求

- Id – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  工作階段的 ID。
- RequestOrigin – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  請求的來源。

回應

- Session – Session (工作階段) 物件。

  回應中傳回的工作階段物件。

錯誤

- AccessDeniedException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

- `InvalidInputException`

ListSessions 行動（Python：列表會話）

擷取工作階段的清單。

請求

- `NextToken` – UTF-8 字串，長度不可超過 400000 個位元組。

  用於下一組結果的字符，如果沒有更多結果則為 null。
- `MaxResults` – 數字 (整數)，不可小於 1，也不可以大於 1000。

  結果的數量上限。
- `Tags` – 金鑰值對的映射陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  屬於工作階段的標籤。
- `RequestOrigin` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  請求的來源。

回應

- `Ids` – UTF-8 字串陣列。

  傳回工作階段的 ID。
- `Sessions` – 一個 Session (工作階段) 物件陣列。

  傳回工作階段物件。
- `NextToken` – UTF-8 字串，長度不可超過 400000 個位元組。

  用於下一組結果的字符，如果沒有更多結果則為 null。

錯誤

- AccessDeniedException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

RunStatement 行動（Python：運行語句）

執行陳述式。

請求

- SessionId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要執行之陳述式的工作階段 ID。
- Code – 必要：UTF-8 字串，長度不可超過 68000 個位元組。

  要執行的陳述式程式碼。
- RequestOrigin – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  請求的來源。

回應

- Id – 數字 (整數)。

  傳回執行之陳述式的 ID。

錯誤

- EntityNotFoundException
- AccessDeniedException
- InternalServiceException
- OperationTimeoutException

- InvalidInputException
- ValidationException
- ResourceNumberLimitExceededException
- IllegalSessionStateException

CancelStatement 行動（Python：取消語句）

取消陳述式。

請求

- SessionId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要取消之陳述式的工作階段 ID。
- Id – 必要：數字 (整數)。

  要取消之陳述式的 ID。
- RequestOrigin – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  取消陳述式請求的來源。

回應

- 無回應參數。

錯誤

- AccessDeniedException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- IllegalSessionStateException

GetStatement 行動（Python：獲取語句）

擷取陳述式。

請求

- SessionId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  陳述式的工作階段 ID。
- Id – 必要：數字 (整數)。

  陳述式的 ID。
- RequestOrigin – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  請求的來源。

回應

- Statement – 陳述式 物件。

  傳回陳述式。

錯誤

- AccessDeniedException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- IllegalSessionStateException

ListStatements 行動（Python：列表語句）

列出工作階段的陳述式。

請求

- SessionId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  陳述式的工作階段 ID。

- RequestOrigin – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  列出陳述式請求的來源。

- NextToken – UTF-8 字串，長度不可超過 400000 個位元組。

  接續符記，如果這是接續呼叫。

回應

- Statements – 一個 陳述式 物件陣列。

  傳回陳述式清單。

- NextToken – UTF-8 字串，長度不可超過 400000 個位元組。

  接續字符 (如果尚未傳回所有陳述)。

錯誤

- AccessDeniedException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- IllegalSessionStateException

開發端點 API

開發端點 API 描述了與使用自定義進行測試相關的 AWS Glue API DevEndpoint。

資料類型

- DevEndpoint 結構
- DevEndpointCustomLibraries 結構

DevEndpoint 結構

開發端點，可供開發人員從遠端進行除錯、轉換及載入 (ETL) 指令碼。

欄位

- `EndpointName` – UTF-8 字串。

  `DevEndpoint` 的名稱。

- `RoleArn` – UTF-8 字串，需符合AWS IAM ARN string pattern。

  此 `DevEndpoint` 中所使用 IAM 角色的 Amazon Resource Name (ARN)。

- `SecurityGroupIds` – UTF-8 字串陣列。

  此 `DevEndpoint` 中所使用的安全群組識識別碼清單。

- `SubnetId` – UTF-8 字串。

  此 `DevEndpoint` 的子網路 ID。

- `YarnEndpointAddress` – UTF-8 字串。

  此 `DevEndpoint` 所用的 YARN 端點地址。

- `PrivateAddress` – UTF-8 字串。

  此為當 `DevEndpoint` 是在 VPC 中建立，透過 VPC 存取 `DevEndpoint` 時私有 IP 地址。`PrivateAddress` 欄位只會在您於 VPC 中建立 `DevEndpoint` 時出現。

- `ZeppelinRemoteSparkInterpreterPort` – 數字 (整數)。

  供遠端 Apache Spark 解譯器使用的 Apache Zeppelin 通訊埠。

- `PublicAddress` – UTF-8 字串。

  此 `DevEndpoint` 所用的公有 IP 地址。只有當您建立非虛擬私有雲端 (VPC) `PublicAddress` 時，`DevEndpoint` 欄位才會出現。

- `Status` – UTF-8 字串。

  此 DevEndpoint 的目前狀態。

- `WorkerType` – UTF-8 字串 (有效值：`Standard=""` | `G.1X=""` | `G.2X=""` | `G.025X=""` | `G.4X=""` | `G.8X=""` | `Z.2X=""`)。

  配置給開發端點的預先定義工作者類型。可接受值為標準、G.1X 或 G.2X

  - 用於 `Standard` 工作者類型時，每個工作者提供 4 個 vCPU、16 GB 的記憶體和 50 GB 磁碟，以及每個工作者 2 個執行器。
  - 用於 `G.1X` 工作者類型時，每個工作者會映射到 1 個 DPU (4 vCPU、16 GB 的記憶體和 64 GB 磁碟)，並為每個工作者提供 1 個執行器。我們建議記憶體密集型任務採用這種工作者類型。
  - 用於 `G.2X` 工作者類型時，每個工作者會映射到 2 個 DPU (8 vCPU、32 GB 的記憶體和 128 GB 磁碟)，並為每個工作者提供 1 個執行器。我們建議記憶體密集型任務採用這種工作者類型。

  已知問題：使用 `G.2X` `WorkerType` 組態建立開發端點時，開發端點的 Spark 驅動程式將在 4 個 vCPU、16 GB 記憶體和 64 GB 磁碟上執行。

- `GlueVersion` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #20。

  Glue 版本確定了阿帕奇星火和 Python 的 AWS Glue 支持的版本。Python 版本會指示在開發端點上執行您 ETL 指令碼時支援的版本。

  如需有關可用版 AWS Glue 本以及對應 Spark 和 Python 版本的詳細資訊，請參閱開發人員指南中的 Glue 版本。

  在沒有指定 Glue 版本情況下建立的開發端點，預設為 Glue 0.9。

  您可以使用 `CreateDevEndpoint` 或 `UpdateDevEndpoint` API 中的 `Arguments` 參數，指定開發端點的 Python 版本支援。如果未提供引數，版本預設為 Python 2。

- `NumberOfWorkers` – 數字 (整數)。

  配置給開發端點之已定義 `workerType` 的工作者數目。

  您可以為 `G.1X` 定義的工作者數目上限是 299，為 `G.2X` 定義的數目上限則是 149。

- `NumberOfNodes` – 數字 (整數)。

  配置給此DevEndpoint單元的 AWS Glue 資料處理單元 (DPU) 數目。

- ~~`AvailabilityZone` – UTF-8 字串。~~

DevEndpoint它所在的 AWS 可用區域。

- VpcId – UTF-8 字串。

  此 DevEndpoint 所用虛擬私有雲端 (VPC) 的 ID。

- ExtraPythonLibsS3Path – UTF-8 字串。

  指向 Amazon S3 儲存貯體中一或多個 Python 程式庫的路徑，此儲存貯體應載入您的 DevEndpoint。多個值必須是以英文逗號分隔的完整路徑。

  > **Note**
  >
  > 您只能搭配純 Python 程式庫使用 DevEndpoint。目前尚未支援使用 C 擴充功能的程式庫 (例如 pandas Python 資料分析程式庫)。

- ExtraJarsS3Path – UTF-8 字串。

  指向 S3 儲存貯體中一或多個 Java .jar 檔案的路徑，該 S3 儲存貯體應載入您的 DevEndpoint。

  > **Note**
  >
  > 您只能搭配純 Java/Scala 程式庫使用 DevEndpoint。

- FailureReason – UTF-8 字串。

  此 DevEndpoint 目前故障的原因。

- LastUpdateStatus – UTF-8 字串。

  上次更新的狀態。

- CreatedTimestamp – 時間戳記。

  建立此 DevEndpoint 項目的時間點。

- LastModifiedTimestamp – 時間戳記。

  此 DevEndpoint 上次修改的時間點。

- PublicKey – UTF-8 字串。

  此 DevEndpoint 用來進行身分驗證所使用的公有金鑰。由於推薦使用屬性是公用金鑰，此屬性的提供是為了回溯相容性。

- PublicKeys – UTF-8 字串的陣列，不可超過 5 個字串。

  該 DevEndpoints 用來進行身分驗證所使用的公有金鑰清單。此屬性的使用優先於單一公有公鑰，因為公有金鑰允許您為每個用戶端使用不同的私有金鑰。

  > **Note**
  >
  > 如果您之前是搭配公有金鑰建立端點，則您必須移除該金鑰，才能設定的公有金鑰。搭配 deletePublicKeys 屬性中的公用金鑰，以及 addPublicKeys 屬性清單中的新金鑰清單，呼叫 UpdateDevEndpoint API 操作。

- SecurityConfiguration – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  可與此 DevEndpoint 搭配使用的 SecurityConfiguration 結構名稱。

- Arguments – 金鑰值對的映射陣列，不超過 100 對。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  用於設定 DevEndpoint 的引數映射。

  有效引數為：

  - "--enable-glue-datacatalog": ""

  您可以使用 CreateDevEndpoint 或 UpdateDevEndpoint API 中的 Arguments 參數，指定開發端點的 Python 版本支援。如果未提供引數，版本預設為 Python 2。

DevEndpointCustomLibraries 結構

要載入開發端點的自訂程式庫。

欄位

- ExtraPythonLibsS3Path – UTF-8 字串。

  至 Amazon Simple Storage Service (Amazon S3) 儲存貯體中一個或多個 Python 程式庫的路徑，該儲存貯體應載入您的 DevEndpoint。多個值必須是以英文逗號分隔的完整路徑。

> **Note**
>
> 您只能搭配純 Python 程式庫使用 `DevEndpoint`。目前尚未支援使用 C 擴充功能的程式庫 (例如 pandas Python 資料分析程式庫)。

- `ExtraJarsS3Path` – UTF-8 字串。

  指向 S3 儲存貯體中一或多個 Java `.jar` 檔案的路徑，該 S3 儲存貯體應載入您的 `DevEndpoint`。

> **Note**
>
> 您只能搭配純 Java/Scala 程式庫使用 `DevEndpoint`。

作業

- CreateDevEndpoint 行動（Python：創建開發端點）
- UpdateDevEndpoint 行動（Python：更新 _ 開發端點）
- DeleteDevEndpoint 行動（Python：刪除開發端點）
- GetDevEndpoint 行動（Python：獲取 _ 開發端點）
- GetDevEndpoints 行動（Python：獲取開發端點）
- BatchGetDevEndpoints 行動（Python：批處理開發端點）
- ListDevEndpoints 行動（Python：列表開發端點）

CreateDevEndpoint 行動（Python：創建開發端點）

建立新的開發端點。

請求

- `EndpointName` – 必要：UTF-8 字串。

  要指派給新 `DevEndpoint` 的名稱。

- `RoleArn` – 必要：UTF-8 字串，需符合AWS IAM ARN string pattern。

  `DevEndpoint` 的 IAM 角色。

- SecurityGroupIds – UTF-8 字串陣列。

  安全群組的安全群組 ID，這些安全群組是要給新的 DevEndpoint 使用。

- SubnetId – UTF-8 字串。

  新 DevEndpoint 要使用的子網路 ID。

- PublicKey – UTF-8 字串。

  此 DevEndpoint 用來進行身分驗證所使用的公有金鑰。由於推薦使用屬性是公用金鑰，此屬性的提供是為了回溯相容性。

- PublicKeys – UTF-8 字串的陣列，不可超過 5 個字串。

  該開發端點用來進行身分驗證所使用的公有金鑰清單。此屬性的使用優先於單一公有公鑰，因為公有金鑰允許您為每個用戶端使用不同的私有金鑰。

  > **Note**
  >
  > 如果您之前是搭配公有金鑰建立端點，則您必須移除該金鑰，才能設定的公有金鑰。搭配 deletePublicKeys 屬性中的公用金鑰，以及 addPublicKeys 屬性清單中的新金鑰清單，呼叫 UpdateDevEndpoint API。

- NumberOfNodes – 數字 (整數)。

  要配置給此DevEndpoint單元的 AWS Glue 資料處理單元 (DPU) 數目。

- WorkerType – UTF-8 字串 (有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="")。

  配置給開發端點的預先定義工作者類型。可接受值為標準、G.1X 或 G.2X

  - 用於 Standard 工作者類型時，每個工作者提供 4 個 vCPU、16 GB 的記憶體和 50 GB 磁碟，以及每個工作者 2 個執行器。
  - 用於 G.1X 工作者類型時，每個工作者會映射到 1 個 DPU (4 vCPU、16 GB 的記憶體和 64 GB 磁碟)，並為每個工作者提供 1 個執行器。我們建議記憶體密集型任務採用這種工作者類型。
  - 用於 G.2X 工作者類型時，每個工作者會映射到 2 個 DPU (8 vCPU、32 GB 的記憶體和 128 GB 磁碟)，並為每個工作者提供 1 個執行器。我們建議記憶體密集型任務採用這種工作者類型。

  已知問題：使用 G.2X WorkerType 組態建立開發端點時，開發端點的 Spark 驅動程式將在 4 個 vCPU、16 GB 記憶體和 64 GB 磁碟上執行。

- `GlueVersion` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #20。

  Glue 版本確定了阿帕奇星火和 Python 的 AWS Glue 支持的版本。Python 版本會指示在開發端點上執行您 ETL 指令碼時支援的版本。

  如需有關可用版 AWS Glue 本以及對應 Spark 和 Python 版本的詳細資訊，請參閱開發人員指南中的 Glue 版本。

  在沒有指定 Glue 版本情況下建立的開發端點，預設為 Glue 0.9。

  您可以使用 `CreateDevEndpoint` 或 `UpdateDevEndpoint` API 中的 `Arguments` 參數，指定開發端點的 Python 版本支援。如果未提供引數，版本預設為 Python 2。

- `NumberOfWorkers` – 數字 (整數)。

  配置給開發端點之已定義 `workerType` 的工作者數目。

  您可以為 `G.1X` 定義的工作者數目上限是 299，為 `G.2X` 定義的數目上限則是 149。

- `ExtraPythonLibsS3Path` – UTF-8 字串。

  指向 Amazon S3 儲存貯體中一或多個 Python 程式庫的路徑，此儲存貯體應載入您的 `DevEndpoint`。多個值必須是以英文逗號分隔的完整路徑。

  > **Note**
  >
  > 您只能搭配純 Python 程式庫使用 `DevEndpoint`。目前尚未支援使用 C 延伸模組的程式庫 (例如 pandas Python 資料分析程式庫)。

- `ExtraJarsS3Path` – UTF-8 字串。

  指向 S3 儲存貯體中一或多個 Java `.jar` 檔案的路徑，該 S3 儲存貯體應載入您的 `DevEndpoint`。

- `SecurityConfiguration` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  可與此 `DevEndpoint` 搭配使用的 `SecurityConfiguration` 結構名稱。

- `Tags` – 金鑰值對的對應陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

要搭配此 DevEndpoint 使用的標籤。您可以使用標籤來限制對 DevEndpoint. 如需中標籤的詳細資訊 AWS Glue，請參閱開發人員指南AWS Glue中的「AWS 標籤」。

- `Arguments` – 金鑰值對的映射陣列，不超過 100 對。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  用於設定 `DevEndpoint` 的引數映射。

回應

- `EndpointName` – UTF-8 字串。

  指派給新的 `DevEndpoint` 的名稱。
- `Status` – UTF-8 字串。

  新的 `DevEndpoint` 的目前狀態。
- `SecurityGroupIds` – UTF-8 字串陣列。

  指派給新的 `DevEndpoint` 的安全群組。
- `SubnetId` – UTF-8 字串。

  指派給新 `DevEndpoint` 的子網路 ID。
- `RoleArn` – UTF-8 字串，需符合AWS IAM ARN string pattern。

  指派給新 `DevEndpoint` 之角色的 Amazon Resource Name (ARN)。
- `YarnEndpointAddress` – UTF-8 字串。

  此 `DevEndpoint` 所用 YARN 端點的地址。
- `ZeppelinRemoteSparkInterpreterPort` – 數字 (整數)。

  供遠端 Apache Spark 解譯器使用的 Apache Zeppelin 通訊埠。
- `NumberOfNodes` – 數字 (整數)。

  配置給此 DevEndpoint單元的 AWS Glue 資料處理單元 (DPU) 數目。
- `WorkerType` – UTF-8 字串 (有效值：`Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""`)。

配置給開發端點的預先定義工作者類型。可能是標準、G.1X 或 G.2X 的值。

- `GlueVersion` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #20。

  Glue 版本確定了阿帕奇星火和 Python 的 AWS Glue 支持的版本。Python 版本會指示在開發端點上執行您 ETL 指令碼時支援的版本。

  如需有關可用版 AWS Glue 本以及對應 Spark 和 Python 版本的詳細資訊，請參閱開發人員指南中的 Glue 版本。

- `NumberOfWorkers` – 數字 (整數)。

  配置給開發端點之已定義 `workerType` 的工作者數目。

- `AvailabilityZone` – UTF-8 字串。

  `DevEndpoint`它所在的 AWS 可用區域。

- `VpcId` – UTF-8 字串。

  此 `DevEndpoint` 所用虛擬私有雲端 (VPC) 的 ID。

- `ExtraPythonLibsS3Path` – UTF-8 字串。

  至 S3 儲存貯體中一個或多個 Python 程式庫的路徑，該 S3 儲存貯體將載入您的 `DevEndpoint`。

- `ExtraJarsS3Path` – UTF-8 字串。

  至 S3 儲存貯體中一個或多個 Java `.jar` 檔案的路徑，該 S3 儲存貯體將載入您的 `DevEndpoint`。

- `FailureReason` – UTF-8 字串。

  此 `DevEndpoint` 目前故障的原因。

- `SecurityConfiguration` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  可與此 `DevEndpoint` 搭配使用的 `SecurityConfiguration` 結構名稱。

- `CreatedTimestamp` – 時間戳記。

  此 `DevEndpoint` 建立的時間點。

- `Arguments` – 金鑰值對的對應陣列，不超過 100 對。

  每個金鑰都是 UTF-8 字串。

每個值都是 UTF-8 字串。

用於設定 DevEndpoint 的引數映射。

有效引數為：

- "--enable-glue-datacatalog": ""

您可以使用 CreateDevEndpoint 或 UpdateDevEndpoint API 中的 Arguments 參數，指定開發端點的 Python 版本支援。如果未提供引數，版本預設為 Python 2。

錯誤

- AccessDeniedException
- AlreadyExistsException
- IdempotentParameterMismatchException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ValidationException
- ResourceNumberLimitExceededException

UpdateDevEndpoint 行動（Python：更新 _ 開發端點）

更新指定的開發端點。

請求

- EndpointName – 必要：UTF-8 字串。

  要更新之 DevEndpoint 的名稱。
- PublicKey – UTF-8 字串。

  DevEndpoint 將使用的公有金鑰。
- AddPublicKeys – UTF-8 字串的陣列，不可超過 5 個字串。

  DevEndpoint 將使用的公有金鑰清單。

- DeletePublicKeys – UTF-8 字串的陣列，不可超過 5 個字串。

  要從 DevEndpoint 刪除的公有金鑰清單。

- CustomLibraries – DevEndpointCustomLibraries 物件。

  要載入 DevEndpoint 中的自訂 Python 或 Java 程式庫。

- UpdateEtlLibraries – 布林值。

  如果要載入開發端點中的自訂程式庫的清單需要更新，此值為 True，如果不需更新則為 False。

- DeleteArguments – UTF-8 字串陣列。

  要從用於設定 DevEndpoint 之引數映射中刪除的引數金鑰清單。

- AddArguments – 金鑰值對的對應陣列，不超過 100 對。

  每個金鑰都是 UTF-8 字串。

  每個值都是 UTF-8 字串。

  要新增用於設定 DevEndpoint 之引數映射的引數映射。

  有效引數為：

  - "--enable-glue-datacatalog": ""

  您可以使用 CreateDevEndpoint 或 UpdateDevEndpoint API 中的 Arguments 參數，指定開發端點的 Python 版本支援。如果未提供引數，版本預設為 Python 2。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ValidationException

DeleteDevEndpoint 行動（Python：刪除開發端點）

刪除指定的開發端點。

請求

- `EndpointName` – 必要：UTF-8 字串。

  `DevEndpoint` 的名稱。

回應

- 無回應參數。

錯誤

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

GetDevEndpoint 行動（Python：獲取 _ 開發端點）

擷取關於所指定開發端點的資訊。

> **Note**
>
> 當您在虛擬私有雲端 (VPC) 建立開發端點時， AWS Glue 只會傳回一個私有 IP 地址，不會填入公有 IP 地址。當您建立非 VPC 開發端點時，只會 AWS Glue 傳回公用 IP 位址。

請求

- `EndpointName` – 必要：UTF-8 字串。

  用於擷取資訊之 `DevEndpoint` 的名稱。

回應

- `DevEndpoint` – DevEndpoint 物件。

  `DevEndpoint` 定義。

錯誤

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

GetDevEndpoints 行動（Python：獲取開發端點）

擷取此 AWS 帳戶中的所有開發端點。

> **Note**
>
> 當您在虛擬私有雲端 (VPC)建立開發端點時， AWS Glue 只會傳回一個私有 IP 地址，不會填入公有 IP 地址。當您建立非 VPC 開發端點時，只會 AWS Glue 傳回公用 IP 位址。

請求

- `MaxResults` – 數字 (整數)，不可小於 1，也不可以大於 1000。

  所要回傳資訊的檔案大小上限。
- `NextToken` – UTF-8 字串。

  接續符記，如果這是接續呼叫。

回應

- `DevEndpoints` – 一個 DevEndpoint 物件陣列。

  `DevEndpoint` 定義的清單。
- `NextToken` – UTF-8 字串。

持續符記 (如果尚未傳回所有 DevEndpoint 定義)。

錯誤

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException

BatchGetDevEndpoints 行動（Python：批處理開發端點）

為指定的開發端點名稱清單，傳回資源中繼資料的清單。呼叫 ListDevEndpoints 操作之後，您便可以呼叫此操作來存取您已授與許可的資料。此操作支援所有 IAM 許可，包括使用標籤的許可條件。

請求

- customerAccountId – UTF-8 字串。

  AWS 帳戶識別碼。
- DevEndpointNames – 必要：UTF-8 字串的陣列，不可小於 1 或超過 25 個字串。

  DevEndpoint 名稱清單，可能是從 ListDevEndpoint 操作傳回的名稱。

回應

- DevEndpoints – 一個 DevEndpoint 物件陣列。

  DevEndpoint 定義的清單。
- DevEndpointsNotFound – UTF-8 字串的陣列，不可小於 1 或超過 25 個字串。

  找不到DevEndpoints 清單。

錯誤

- AccessDeniedException
- InternalServiceException

- OperationTimeoutException
- InvalidInputException

ListDevEndpoints 行動（Python：列表開發端點）

擷取這個 AWS 帳戶下所有 DevEndpoint 資源的名稱，或是包含指定標籤的資源。您可運用此操作，查看帳戶下有哪些可用資源及其名稱。

此操作會接收您可在回應時做為篩選條件的選用 Tags 欄位，因此已標記的資源可分組進行擷取。如果您選擇使用標籤進行篩選，則此時只會擷取包含該標籤的資源。

請求

- NextToken – UTF-8 字串。

  接續符記，如果這是接續要求。
- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  所要回傳清單的大小上限。
- Tags – 金鑰值對的對應陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  指定只傳回包含這些標籤的資源。

回應

- DevEndpointNames – UTF-8 字串陣列。

  這個帳戶下所有 DevEndpoint 的名稱，或是使用指定標籤的 DevEndpoint。
- NextToken – UTF-8 字串。

  接續字元，如果傳回的清單未包含最後一個可用指標。

錯誤

- InvalidInputException

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

結構描述登錄檔

結構描述登錄 API 描述與在中使用結構描述相關的資料類型和 API AWS Glue。

資料類型

- RegistryId 結構
- RegistryListItem 結構
- MetadataInfo 結構
- OtherMetadataValueListItem 結構
- SchemaListItem 結構
- SchemaVersionListItem 結構
- MetadataKeyValuePair 結構
- SchemaVersionErrorItem 結構
- ErrorDetails 結構
- SchemaVersionNumber 結構
- SchemaId 結構

RegistryId 結構

包裝函式結構，可包含登錄檔名稱和 Amazon Resource Name (ARN)。

欄位

- RegistryName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  登錄檔的名稱。僅用於查閱。必須提供其中一個 RegistryArn 或 RegistryName。
- RegistryArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

要更新的登錄檔的 arn。必須提供其中一個 RegistryArn 或 RegistryName。

RegistryListItem 結構

包含登錄檔詳細資訊的結構。

欄位

- RegistryName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  登錄的名稱。
- RegistryArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  登錄檔的 Amazon Resource Name (ARN)。
- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  登錄檔描述。
- Status – UTF-8 字串 (有效值：AVAILABLE | DELETING)。

  登錄檔的狀態。
- CreatedTime – UTF-8 字串。

  建立登錄檔的資料。
- UpdatedTime – UTF-8 字串。

  更新登錄檔的日期。

MetadataInfo 結構

包含結構描述版本之中繼資料資訊的結構。

欄位

- MetadataValue – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 256 個位元組，且需符合 Custom string pattern #33。

中繼資料索引鍵的對應值。

- `CreatedTime` – UTF-8 字串。

  項目建立的時間。

- `OtherMetadataValueList` – 一個 OtherMetadataValueListItem 物件陣列。

  屬於相同中繼資料索引鍵的其他中繼資料。

OtherMetadataValueListItem 結構

包含屬於相同中繼資料索引鍵之結構描述版本的其他中繼資料的結構。

欄位

- `MetadataValue` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 256 個位元組，且需符合 Custom string pattern #33。

  屬於相同中繼資料索引鍵的其他中繼資料的中繼資料索引鍵的對應值。

- `CreatedTime` – UTF-8 字串。

  項目建立的時間。

SchemaListItem 結構

包含結構描述最少詳細資訊的物件。

欄位

- `RegistryName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  結構描述所在的登錄檔名稱。

- `SchemaName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  結構描述的名稱。

- `SchemaArn` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

結構描述的 Amazon Resource Name (ARN)。

- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  結構描述的描述。

- `SchemaStatus` – UTF-8 字串 (有效值：`AVAILABLE` | `PENDING` | `DELETING`)。

  結構描述的狀態。

- `CreatedTime` – UTF-8 字串。

  建立結構描述的日期和時間。

- `UpdatedTime` – UTF-8 字串。

  結構描述的更新日期和時間。

SchemaVersionListItem 結構

包含結構描述版本詳細資訊的物件。

欄位

- `SchemaArn` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  結構描述的 Amazon Resource Name (ARN)。

- `SchemaVersionId` – UTF-8 字串，長度不可小於 36 個位元組，也不可以超過 36 個位元組，且需符合 Custom string pattern #17。

  結構描述版本的唯一識別碼。

- `VersionNumber` – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  結構描述的版本編號。

- `Status` – UTF-8 字串 (有效值：`AVAILABLE` | `PENDING` | `FAILURE` | `DELETING`)。

  結構描述版本的狀態。

- `CreatedTime` – UTF-8 字串。

  建立結構描述版本的日期和時間。

MetadataKeyValuePair 結構

包含中繼資料的索引鍵值組的結構。

欄位

- `MetadataKey` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #33。

  中繼資料金鑰。
- `MetadataValue` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 256 個位元組，且需符合 Custom string pattern #33。

  中繼資料索引鍵的對應值。

SchemaVersionErrorItem 結構

包含結構描述版本上操作之錯誤詳細資訊的物件。

欄位

- `VersionNumber` – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  結構描述的版本編號。
- `ErrorDetails` – ErrorDetails 物件。

  結構描述版本的錯誤詳細資訊。

ErrorDetails 結構

包含錯誤詳細資訊的物件。

欄位

- `ErrorCode` – UTF-8 字串。

  錯誤的錯誤碼。
- `ErrorMessage` – UTF-8 字串。

  錯誤的錯誤訊息。

SchemaVersionNumber 結構

包含結構描述版本資訊的結構。

欄位

- `LatestVersion` – 布林值。

  可用於結構描述的最新版本。
- `VersionNumber` – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  結構描述的版本編號。

SchemaId 結構

結構描述登錄中結構描述的 AWS Glue 唯一識別碼。

欄位

- `SchemaArn` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  結構描述的 Amazon Resource Name (ARN)。必須提供其中一個 `SchemaArn` 或 `SchemaName`。
- `SchemaName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  結構描述的名稱。必須提供其中一個 `SchemaArn` 或 `SchemaName`。
- `RegistryName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  包含結構描述的結構描述登錄檔的名稱。

作業

- CreateRegistry 動作（Python：創建註冊表）
- CreateSchema 動作 (Python: 建立結構描述)
- GetSchema 行動（Python：獲取模式）
- ListSchemaVersions 行動（Python：列表模式版本）

- GetSchemaVersion 行動（Python：獲取模式版本）
- GetSchemaVersionsDiff 行動（Python：獲取模式版本差異）
- ListRegistries 動作（Python：列表註冊表）
- ListSchemas 動作（Python：列表模式）
- RegisterSchemaVersion 行動（Python：註冊模式版本）
- UpdateSchema 行動（Python：更新模式）
- CheckSchemaVersionValidity 操作（Python：檢查模式版本有效性）
- UpdateRegistry 行動（Python：更新註冊表）
- GetSchemaByDefinition 行動（Python：獲取模式定義）
- GetRegistry 行動（Python：獲取註冊表）
- PutSchemaVersionMetadata 行動（Python：放置版本的元數據）
- QuerySchemaVersionMetadata 行動（Python：查詢模式版本的元數據）
- RemoveSchemaVersionMetadata 行動（Python：刪除模式版本的元數據）
- DeleteRegistry 行動（Python：刪除註冊表）
- DeleteSchema 操作（Python：刪除模式）
- DeleteSchemaVersions 行動（Python：刪除模式版本）

CreateRegistry 動作（Python：創建註冊表）

建立一個新的登錄檔，可用於儲存結構描述的集合。

請求

- `RegistryName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Custom string pattern #18。

  要建立的登錄檔名稱最長為 255 個字元，並且只能包含字母、數字、連字號、下劃線、美元符號或雜湊符號。沒有空格。

- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  登錄檔描述。如果未提供描述，則此項不會有任何預設值。

- `Tags` – 金鑰值對的對應陣列，不超過 50 對。

每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

AWS 包含金鑰值組的標籤，可以透過主控台、命令列或 API 來搜尋。

回應

- `RegistryArn` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  新建立登錄檔的 Amazon Resource Name (ARN)。

- `RegistryName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  登錄的名稱。

- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  登錄檔描述。

- `Tags` – 金鑰值對的對應陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  登錄檔的標籤。

錯誤

- `InvalidInputException`
- `AccessDeniedException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`
- `ConcurrentModificationException`
- `InternalServiceException`

CreateSchema 動作 (Python: 建立結構描述)

建立新的結構描述集並註冊結構描述定義。如果結構描述集已經存在，但沒有實際註冊版本，則傳回錯誤。

建立結構描述集時，版本檢查點會設定為第一個版本。相容性模式「DISABLED」會限制在第一個結構描述版本之後新增任何其他結構描述版本。對於所有其他相容性模式，使用 `RegisterSchemaVersion` API 時，相容性設定的驗證只會從第二個版本開始套用。

如果在沒有 `RegistryId` 的情況下呼叫這個 API，這將在登錄檔資料庫資料表中建立一個 "default-registry" 的項目 (如果它不存在)。

請求

- `RegistryId` – RegistryId 物件。

  這是包含登錄檔識別欄位的包裝函式圖形。如果未提供此項，則會使用預設登錄檔。相同的 ARN 格式是：`arn:aws:glue:us-east-2:<customer id>:registry/default-registry:random-5-letter-id`。
- `SchemaName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Custom string pattern #18。

  要建立的結構描述名稱最長為 255 個字元，並且只能包含字母、數字、連字號、下劃線、美元符號或雜湊符號。沒有空格。
- `DataFormat` – 必要：UTF-8 字串 (有效值：`AVRO` | `JSON` | `PROTOBUF`)。

  結構描述定義的資料格式。目前支援 AVRO、JSON 和 PROTOBUF。
- `Compatibility` – UTF-8 字串 (有效值：`NONE` | `DISABLED` | `BACKWARD` | `BACKWARD_ALL` | `FORWARD` | `FORWARD_ALL` | `FULL` | `FULL_ALL`)。

  結構描述的相容性模式。可能值如下：
  - NONE：未套用相容性模式。您可以在開發案例中使用此選項，或者如果您不知道要套用至結構描述的相容性模式。任何新增的版本都會被接受，而不經過相容性檢查。
  - DISABLED：此相容性選項可防止特定結構描述的版本控制。您可以使用此選項來防止未來的結構描述版本控制。
  - BACKWARD：建議使用此相容性選項，因為它允許資料接收器同時讀取目前和前一個結構描述版本。這代表，例如，新的結構描述版本無法捨棄資料欄位或變更這些欄位的類型，因此使用以前版本的讀取器無法讀取它們。

- BACKWARD_ALL：此相容性選項允許資料接收器同時讀取目前和所有先前的結構描述版本。當您需要刪除欄位或新增選用欄位，並檢查與所有先前結構描述版本的相容性時，您可以使用此選項。
- FORWARD：此相容性選項允許資料接收器同時讀取目前和後一個結構描述版本，但不一定是較新的版本。當您需要新增欄位或刪除選用欄位，但只檢查最新結構描述版本的相容性時，您可以使用此選項。
- FORWARD_ALL：此相容性選項允許資料接收器讀取任何新註冊結構描述的生產者所寫入的資料。當您需要新增欄位或刪除選用欄位，並檢查與所有先前結構描述版本的相容性時，您可以使用此選項。
- FULL：此相容性選項允許資料接收器讀取使用前一個或下一個版本的結構描述，但不能讀取更舊或更新版本的生產者所寫入的資料。當您需要新增或移除選用欄位，但只檢查最新結構描述版本的相容性時，您可以使用此選項。
- FULL_ALL：此相容性選項允許資料接收器讀取生產者使用所有先前的結構描述版本所寫入的資料。當您需要新增或移除選用欄位，並檢查與所有先前結構描述版本的相容性時，您可以使用此選項。

- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  結構描述的選擇性說明。如果未提供描述，則此項不會有任何自動預設值。

- `Tags` – 金鑰值對的對應陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  AWS 包含金鑰值組的標籤，可以透過主控台、命令列或 API 來搜尋。如果指定，請遵循 AWS tags-on-create 模式。

- `SchemaDefinition` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 170000 個位元組，且需符合 Custom string pattern #32。

  為 `SchemaName` 使用 `DataFormat` 設定的結構描述定義。

回應

- `RegistryName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

登錄的名稱。

- RegistryArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  登錄檔的 Amazon Resource Name (ARN)。

- SchemaName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  結構描述的名稱。

- SchemaArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  結構描述的 Amazon Resource Name (ARN)。

- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  如果在建立時指定的結構描述。

- DataFormat – UTF-8 字串 (有效值：AVRO | JSON | PROTOBUF)。

  結構描述定義的資料格式。目前支援 AVRO、JSON 和 PROTOBUF。

- Compatibility – UTF-8 字串 (有效值：NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL)。

  結構描述的相容性模式。

- SchemaCheckpoint – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  檢查點的版本號碼 (上次變更相容性模式)。

- LatestSchemaVersion – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  與傳回結構描述定義相關聯結構描述的最新版本。

- NextSchemaVersion – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  與傳回結構描述定義相關聯結構描述的下一個版本。

- SchemaStatus – UTF-8 字串 (有效值：AVAILABLE | PENDING | DELETING)。

  結構描述的狀態。

- Tags – 金鑰值對的對應陣列，不超過 50 對。

每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

結構描述的標籤。

- SchemaVersionId – UTF-8 字串，長度不可小於 36 個位元組，也不可以超過 36 個位元組，且需符合 Custom string pattern #17。

  第一個結構描述版本的唯一識別碼。

- SchemaVersionStatus – UTF-8 字串 (有效值：AVAILABLE | PENDING | FAILURE | DELETING)。

  建立的第一個結構描述版本的狀態。

錯誤

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- AlreadyExistsException
- ResourceNumberLimitExceededException
- ConcurrentModificationException
- InternalServiceException

GetSchema 行動（Python：獲取模式）

詳細描述指定的結構描述。

請求

- SchemaId – 必要：SchemaId 物件。

  這是包含結構描述身分欄位的包裝函式結構。結構包含：

  - SchemaId$SchemaArn：模式的 Amazon 資源名稱（ARN）。必須提供 SchemaArn 或 SchemaName 和 RegistryName。

- SchemaId$SchemaName：模式的名稱。必須提供 SchemaArn 或 SchemaName 和 RegistryName。

回應

- RegistryName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  登錄的名稱。
- RegistryArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  登錄檔的 Amazon Resource Name (ARN)。
- SchemaName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  結構描述的名稱。
- SchemaArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  結構描述的 Amazon Resource Name (ARN)。
- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  如果在建立時指定，則為結構描述的描述。
- DataFormat – UTF-8 字串 (有效值：AVRO | JSON | PROTOBUF)。

  結構描述定義的資料格式。目前支援 AVRO、JSON 和 PROTOBUF。
- Compatibility – UTF-8 字串 (有效值：NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL)。

  結構描述的相容性模式。
- SchemaCheckpoint – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  檢查點的版本號碼 (上次變更相容性模式)。
- LatestSchemaVersion – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  與傳回結構描述定義相關聯結構描述的最新版本。

- NextSchemaVersion – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  與傳回結構描述定義相關聯結構描述的下一個版本。

- SchemaStatus – UTF-8 字串 (有效值：AVAILABLE | PENDING | DELETING)。

  結構描述的狀態。

- CreatedTime – UTF-8 字串。

  建立結構描述的日期和時間。

- UpdatedTime – UTF-8 字串。

  結構描述的更新日期和時間。

錯誤

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

ListSchemaVersions 行動（Python：列表模式版本）

以最少的資訊傳回您所建立的結構描述版本清單。結果中不會包含處於 [已刪除] 狀態的結構描述版本。如果沒有可用的結構描述版本，則會傳回空的結果。

請求

- SchemaId – 必要：SchemaId 物件。

  這是包含結構描述身分欄位的包裝函式結構。結構包含：

  - SchemaId$SchemaArn：模式的 Amazon 資源名稱（ARN）。必須提供 SchemaArn 或 SchemaName 和 RegistryName。
  - SchemaId$SchemaName：模式的名稱。必須提供 SchemaArn 或 SchemaName 和 RegistryName。

- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 100。

  每頁要求結果的數量上限。如果未提供該值，這將預設為每頁 25 個。

- NextToken – UTF-8 字串。

  接續符記，如果這是接續呼叫。

回應

- Schemas – 一個 SchemaVersionListItem 物件陣列。

  包含每個結構描述版本詳細資訊的 SchemaVersionList 物件陣列。
- NextToken – UTF-8 字串。

  為一種接續符記，用於將傳回的符記清單分頁，而如果清單目前的區段不是最後區段就會傳回。

錯誤

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

GetSchemaVersion 行動（Python：獲取模式版本）

根據建立或註冊結構描述的版本時指派的唯一 ID，取得指定的結構描述。結果中不會包含處於 [已刪除] 狀態的結構描述版本。

請求

- SchemaId – SchemaId 物件。

  這是包含結構描述身分欄位的包裝函式結構。結構包含：

  - SchemaId$SchemaArn：模式的 Amazon 資源名稱（ARN）。必須提供 SchemaArn 或 SchemaName 和 RegistryName。
  - SchemaId$SchemaName：模式的名稱。必須提供 SchemaArn 或 SchemaName 和 RegistryName。
- SchemaVersionId – UTF-8 字串，長度不可小於 36 個位元組，也不可以超過 36 個位元組，且需符合 Custom string pattern #17。

結構描述版本的 SchemaVersionId。透過結構描述 ID 擷取時，此欄位是必要欄位。必須提供此項目或 SchemaId 包裝函式。

- SchemaVersionNumber – SchemaVersionNumber 物件。

  結構描述的版本編號。

回應

- SchemaVersionId – UTF-8 字串，長度不可小於 36 個位元組，也不可以超過 36 個位元組，且需符合 Custom string pattern #17。

  結構描述版本的 SchemaVersionId。
- SchemaDefinition – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 170000 個位元組，且需符合 Custom string pattern #32。

  結構描述 ID 的結構描述定義。
- DataFormat – UTF-8 字串 (有效值：AVRO | JSON | PROTOBUF)。

  結構描述定義的資料格式。目前支援 AVRO、JSON 和 PROTOBUF。
- SchemaArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  結構描述的 Amazon Resource Name (ARN)。
- VersionNumber – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  結構描述的版本編號。
- Status – UTF-8 字串 (有效值：AVAILABLE | PENDING | FAILURE | DELETING)。

  結構描述版本的狀態。
- CreatedTime – UTF-8 字串。

  建立結構描述版本的日期和時間。

錯誤

- InvalidInputException
- AccessDeniedException

- EntityNotFoundException
- InternalServiceException

GetSchemaVersionsDiff 行動（Python：獲取模式版本差異）

擷取結構描述登錄檔中兩個預存結構描述版本之間指定差異類型的結構描述版本差異。

此 API 允許您比較相同結構描述下的兩個結構描述定義之間的兩個結構描述版本。

請求

- SchemaId – 必要：SchemaId 物件。

  這是包含結構描述身分欄位的包裝函式結構。結構包含：

  - SchemaId$SchemaArn：模式的 Amazon 資源名稱（ARN）。必須提供其中一個 SchemaArn 或 SchemaName。
  - SchemaId$SchemaName：模式的名稱。必須提供其中一個 SchemaArn 或 SchemaName。
- FirstSchemaVersionNumber – 必要：SchemaVersionNumber 物件。

  要比較的兩個結構描述版本中的第一個。
- SecondSchemaVersionNumber – 必要：SchemaVersionNumber 物件。

  要比較的兩個結構描述版本中的第二個。
- SchemaDiffType – 必要：UTF-8 字串 (有效值：SYNTAX_DIFF)。

  參考 SYNTAX_DIFF，這是目前支援的差異類型。

回應

- Diff – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 340000 個位元組，且需符合 Custom string pattern #32。

  模式作為 JsonPatch 格式字符串之間的差異。

錯誤

- InvalidInputException
- EntityNotFoundException

- AccessDeniedException
- InternalServiceException

ListRegistries 動作（Python：列表註冊表）

以最少的登錄檔資訊傳回您所建立的登錄檔清單。狀態為 Deleting 的登錄檔將不會包含在結果中。如果沒有可用的登錄檔，將傳回空白的結果。

請求

- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 100。

  每頁要求結果的數量上限。如果未提供該值，這將預設為每頁 25 個。
- NextToken – UTF-8 字串。

  接續符記，如果這是接續呼叫。

回應

- Registries – 一個 RegistryListItem 物件陣列。

  包含每個登錄檔最少詳細資訊的 RegistryDetailedListItem 物件陣列。
- NextToken – UTF-8 字串。

  為一種接續符記，用於將傳回的符記清單分頁，而如果清單目前的區段不是最後區段就會傳回。

錯誤

- InvalidInputException
- AccessDeniedException
- InternalServiceException

ListSchemas 動作（Python：列表模式）

傳回具有最少詳細資訊的結構描述清單。結果中不會包含處於 [正在刪除] 狀態的結構描述。如果沒有可用的結構描述，將傳回空的結果。

未提供 RegistryId 時，跨登錄檔的所有結構描述都將成為 API 回應的一部分。

請求

- `RegistryId` – RegistryId 物件。

  包裝函式結構，可包含登錄檔名稱和 Amazon Resource Name (ARN)。
- `MaxResults` – 數字 (整數)，不可小於 1，也不可以大於 100。

  每頁要求結果的數量上限。如果未提供該值，這將預設為每頁 25 個。
- `NextToken` – UTF-8 字串。

  接續符記，如果這是接續呼叫。

回應

- `Schemas` – 一個 SchemaListItem 物件陣列。

  包含每個結構描述詳細資訊的 `SchemaListItem` 物件陣列。
- `NextToken` – UTF-8 字串。

  為一種接續符記，用於將傳回的符記清單分頁，而如果清單目前的區段不是最後區段就會傳回。

錯誤

- `InvalidInputException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`

RegisterSchemaVersion 行動（Python：註冊模式版本）

將新版本加入至現有的結構描述。如果新版結構描述不符合結構描述集的相容性需求，則傳回錯誤。此 API 將不會建立新的結構描述集，而且如果結構描述集不存在於結構描述登錄檔中，將會傳回 404 錯誤。

如果這是要在結構描述登錄檔中註冊的第一個結構描述定義，則此 API 將儲存結構描述版本並立即傳回。否則，由於相容性模式，此呼叫可能會比其他操作執行更長的時間。您可以呼叫 `GetSchemaVersion` API 搭配 `SchemaVersionId` 來檢查相容性模式。

如果相同的結構描述定義已儲存在結構描述登錄檔作為版本，現有結構描述的結構描述 ID 會傳回給呼叫者。

請求

- SchemaId – 必要：SchemaId 物件。

  這是包含結構描述身分欄位的包裝函式結構。結構包含：

  - SchemaId$SchemaArn：模式的 Amazon 資源名稱 (ARN) 。必須提供 SchemaArn 或 SchemaName 和 RegistryName。
  - SchemaId$SchemaName：模式的名稱。必須提供 SchemaArn 或 SchemaName 和 RegistryName。
- SchemaDefinition – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 170000 個位元組，且需符合 Custom string pattern #32。

  為 SchemaName 使用 DataFormat 設定的結構描述定義。

回應

- SchemaVersionId – UTF-8 字串，長度不可小於 36 個位元組，也不可以超過 36 個位元組，且需符合 Custom string pattern #17。

  代表此結構描述版本的唯一 ID。
- VersionNumber – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  此結構描述的版本 (僅適用於同步流程，以防這是第一個版本)。
- Status – UTF-8 字串 (有效值：AVAILABLE | PENDING | FAILURE | DELETING)。

  結構描述版本的狀態。

錯誤

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- ResourceNumberLimitExceededException
- ConcurrentModificationException

- InternalServiceException

UpdateSchema 行動（Python：更新模式）

更新結構描述集的描述、相容性設定或版本檢查點。

若要更新相容性設定，呼叫將不會驗證整組結構描述版本與新相容性設定的相容性。如果有提供 Compatibility 的值，VersionNumber (檢查點) 也是必要的。API 將驗證檢查點版本號碼的一致性。

如果有提供 VersionNumber (檢查點) 的值，則 Compatibility 是選用的，這可以用來設定/重設結構描述的檢查點。

只有在結構描述處於 AVAILABLE 狀態時，才會發生此更新。

請求

- SchemaId – 必要：SchemaId 物件。

  這是包含結構描述身分欄位的包裝函式結構。結構包含：

  - SchemaId$SchemaArn：模式的 Amazon 資源名稱（ARN）。必須提供其中一個 SchemaArn 或 SchemaName。
  - SchemaId$SchemaName：模式的名稱。必須提供其中一個 SchemaArn 或 SchemaName。
- SchemaVersionNumber – SchemaVersionNumber 物件。

  檢查指向所需的版本號碼。必須提供其中一個 VersionNumber 或 Compatibility。
- Compatibility – UTF-8 字串 (有效值：NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL)。

  結構描述的新相容性設定。
- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  結構描述的新的描述。

回應

- SchemaArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

結構描述的 Amazon Resource Name (ARN)。

- SchemaName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  結構描述的名稱。

- RegistryName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  包含結構描述的登錄檔的名稱。

錯誤

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- ConcurrentModificationException
- InternalServiceException

CheckSchemaVersionValidity 操作（Python：檢查模式版本有效性）

驗證提供的結構描述。這個呼叫沒有副作用，它只是使用提供的結構描述，使用 DataFormat 做為格式來驗證。由於它不採用結構描述集名稱，因此不會執行相容性檢查。

請求

- DataFormat – 必要：UTF-8 字串 (有效值：AVRO | JSON | PROTOBUF)。

  結構描述定義的資料格式。目前支援 AVRO、JSON 和 PROTOBUF。

- SchemaDefinition – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 170000 個位元組，且需符合 Custom string pattern #32。

  必須驗證結構描述的定義。

回應

- Valid – 布林值。

如果結構描述是有效的則傳回 true，否則為 false。

- `Error` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 5000 個位元組。

  驗證失敗錯誤訊息。

錯誤

- `InvalidInputException`
- `AccessDeniedException`
- `InternalServiceException`

UpdateRegistry 行動（Python：更新註冊表）

更新用於儲存結構描述集合的現有登錄檔。更新的屬性與登錄檔相關，並且不會修改任何登錄檔內的結構描述。

請求

- `RegistryId` – 必要：RegistryId 物件。

  這是包裝函式結構，可包含登錄檔名稱和 Amazon Resource Name (ARN)。

- `Description` – 必要：描述字串，長度不可超過 2048 個位元組，需符合 URI address multi-line string pattern。

  登錄檔描述。如果未提供描述，則不會更新此欄位。

回應

- `RegistryName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  更新登錄檔的名稱。

- `RegistryArn` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  更新登錄檔的 Amazon Resource Name (ARN)。

錯誤

- `InvalidInputException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `ConcurrentModificationException`
- `InternalServiceException`

GetSchemaByDefinition 行動（Python：獲取模式定義）

由檢索結構描述`SchemaDefinition`。結構描述定義會傳送至結構描述登錄檔、進行標準化和雜湊。如果雜湊在 `SchemaName` 或 ARN (或預設登錄檔，如果沒有提供) 的範圍內相符，則傳回該結構描述的中繼資料。否則，返回 404 或 NotFound 錯誤。處於 `Deleted` 狀態的結構描述版本將不會包含在結果中。

請求

- `SchemaId` – 必要：SchemaId 物件。

  這是包含結構描述身分欄位的包裝函式結構。結構包含：

  - SchemaId$SchemaArn：模式的 Amazon 資源名稱（ARN）。必須提供其中一個 `SchemaArn` 或 `SchemaName`。
  - SchemaId$SchemaName：模式的名稱。必須提供其中一個 `SchemaArn` 或 `SchemaName`。
- `SchemaDefinition` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 170000 個位元組，且需符合 Custom string pattern #32。

  需要結構描述詳細資訊之結構描述的定義。

回應

- `SchemaVersionId` – UTF-8 字串，長度不可小於 36 個位元組，也不可以超過 36 個位元組，且需符合 Custom string pattern #17。

  結構描述版本的結構描述 ID。
- `SchemaArn` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

結構描述的 Amazon Resource Name (ARN)。

- DataFormat – UTF-8 字串 (有效值：AVRO | JSON | PROTOBUF)。

  結構描述定義的資料格式。目前支援 AVRO、JSON 和 PROTOBUF。

- Status – UTF-8 字串 (有效值：AVAILABLE | PENDING | FAILURE | DELETING)。

  結構描述版本的狀態。

- CreatedTime – UTF-8 字串。

  建立結構描述的日期和時間。

錯誤

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

GetRegistry 行動（Python：獲取註冊表）

詳細描述指定的登錄檔。

請求

- RegistryId – 必要：RegistryId 物件。

  這是包裝函式結構，可包含登錄檔名稱和 Amazon Resource Name (ARN)。

回應

- RegistryName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  登錄的名稱。

- RegistryArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

登錄檔的 Amazon Resource Name (ARN)。

- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  登錄檔描述。

- `Status` – UTF-8 字串 (有效值：`AVAILABLE` | `DELETING`)。

  登錄檔的狀態。

- `CreatedTime` – UTF-8 字串。

  建立登錄檔的日期和時間。

- `UpdatedTime` – UTF-8 字串。

  登錄檔的更新日期和時間。

錯誤

- `InvalidInputException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`

PutSchemaVersionMetadata 行動（Python：放置版本的元數據）

放置指定結構描述版本 ID 的中繼資料索引鍵值配對。每個結構描述版本最多允許 10 個索引鍵值配對。它們可以透過一個或多個呼叫來新增。

請求

- `SchemaId` – SchemaId 物件。

  結構描述的唯一 ID。

- `SchemaVersionNumber` – SchemaVersionNumber 物件。

  結構描述的版本編號。

- `SchemaVersionId` – UTF-8 字串，長度不可小於 36 個位元組，也不可以超過 36 個位元組，且需符合 Custom string pattern #17。

結構描述版本的唯一版本 ID。

- `MetadataKeyValue` – 必要：MetadataKeyValuePair 物件。

  中繼資料索引鍵的對應值。

回應

- `SchemaArn` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  結構描述的 Amazon Resource Name (ARN)。

- `SchemaName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  結構描述的名稱。

- `RegistryName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  登錄檔的名稱。

- `LatestVersion` – 布林值。

  結構描述的最新版本。

- `VersionNumber` – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  結構描述的版本編號。

- `SchemaVersionId` – UTF-8 字串，長度不可小於 36 個位元組，也不可以超過 36 個位元組，且需符合 Custom string pattern #17。

  結構描述版本的唯一版本 ID。

- `MetadataKey` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #33。

  中繼資料索引鍵。

- `MetadataValue` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 256 個位元組，且需符合 Custom string pattern #33。

  中繼資料索引鍵的值。

錯誤

- InvalidInputException
- AccessDeniedException
- AlreadyExistsException
- EntityNotFoundException
- ResourceNumberLimitExceededException

QuerySchemaVersionMetadata 行動（Python：查詢模式版本的元數據）

查詢結構描述版本的中繼資料資訊。

請求

- SchemaId – SchemaId 物件。

  包裝函式結構，可包含結構描述名稱和 Amazon Resource Name (ARN)。
- SchemaVersionNumber – SchemaVersionNumber 物件。

  結構描述的版本編號。
- SchemaVersionId – UTF-8 字串，長度不可小於 36 個位元組，也不可以超過 36 個位元組，且需符合 Custom string pattern #17。

  結構描述版本的唯一版本 ID。
- MetadataList – 一個 MetadataKeyValuePair 物件陣列。

  搜尋鍵-值對中繼資料，如果他們沒有提供所有中繼資料訊息將被提取。
- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 50。

  每頁要求結果的數量上限。如果未提供該值，這將預設為每頁 25 個。
- NextToken – UTF-8 字串。

  接續符記，如果這是接續呼叫。

回應

- MetadataInfoMap – 金鑰值對的映射陣列。

每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，需符合 Custom string pattern #33。

每個值都是 MetadataInfo 物件。

中繼資料索引鍵和關聯值的映射。

- SchemaVersionId – UTF-8 字串，長度不可小於 36 個位元組，也不可以超過 36 個位元組，且需符合 Custom string pattern #17。

  結構描述版本的唯一版本 ID。
- NextToken – UTF-8 字串。

  為一種接續符記，用於將傳回的符記清單分頁，而如果清單目前的區段不是最後區段就會傳回。

錯誤

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException

RemoveSchemaVersionMetadata 行動（Python：刪除模式版本的元數據）

從指定的結構描述版本 ID 的結構描述版本中繼資料中移除索引鍵值配對。

請求

- SchemaId – SchemaId 物件。

  包裝函式結構，可包含結構描述名稱和 Amazon Resource Name (ARN)。
- SchemaVersionNumber – SchemaVersionNumber 物件。

  結構描述的版本編號。
- SchemaVersionId – UTF-8 字串，長度不可小於 36 個位元組，也不可以超過 36 個位元組，且需符合 Custom string pattern #17。

  結構描述版本的唯一版本 ID。
- MetadataKeyValue – 必要：MetadataKeyValuePair 物件。

中繼資料索引鍵的值。

回應

- SchemaArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  結構描述的 Amazon Resource Name (ARN)。

- SchemaName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  結構描述的名稱。

- RegistryName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  登錄的名稱。

- LatestVersion – 布林值。

  結構描述的最新版本。

- VersionNumber – 數字 (長整數)，不可小於 1，也不可以大於 100000。

  結構描述的版本編號。

- SchemaVersionId – UTF-8 字串，長度不可小於 36 個位元組，也不可以超過 36 個位元組，且需符合 Custom string pattern #17。

  結構描述版本的版本 ID。

- MetadataKey – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #33。

  中繼資料索引鍵。

- MetadataValue – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 256 個位元組，且需符合 Custom string pattern #33。

  中繼資料索引鍵的值。

錯誤

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException

DeleteRegistry 行動（Python：刪除註冊表）

刪除包含結構描述及其所有版本的整個登錄檔。若要取得刪除操作的狀態，您可以在非同步呼叫之後呼叫 GetRegistry API。刪除登錄檔會停用登錄檔的所有線上作業，例如 UpdateRegistry、CreateSchema、UpdateSchema 以及 RegisterSchemaVersion API。

請求

- RegistryId – 必要：RegistryId 物件。

  這是包裝函式結構，可包含登錄檔名稱和 Amazon Resource Name (ARN)。

回應

- RegistryName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  要刪除之登錄檔的名稱。
- RegistryArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  要刪除之登錄檔的 Amazon Resource Name (ARN)。
- Status – UTF-8 字串 (有效值：AVAILABLE | DELETING)。

  登錄檔的狀態。成功的操作將傳回 Deleting 狀態。

錯誤

- InvalidInputException
- EntityNotFoundException
- AccessDeniedException

- `ConcurrentModificationException`

DeleteSchema 操作（Python：刪除模式）

刪除整個結構描述集，包括結構描述集及其所有版本。若要取得刪除操作的狀態，您可以在非同步呼叫之後呼叫 `GetSchema` API。刪除登錄檔會停用登錄檔的所有線上作業，例如 `GetSchemaByDefinition` 及 `RegisterSchemaVersion` API。

請求

- `SchemaId` – 必要：SchemaId 物件。

  這是包裝函式結構，可包含結構描述名稱和 Amazon Resource Name (ARN)。

回應

- `SchemaArn` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合 Custom string pattern #22。

  要刪除之結構描述的 Amazon Resource Name (ARN)。

- `SchemaName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #18。

  要刪除之結構描述的名稱。

- `Status` – UTF-8 字串 (有效值：`AVAILABLE` | `PENDING` | `DELETING`)。

  結構描述的狀態。

錯誤

- `InvalidInputException`
- `EntityNotFoundException`
- `AccessDeniedException`
- `ConcurrentModificationException`

DeleteSchemaVersions 行動（Python：刪除模式版本）

從指定的結構描述移除版本。可能會提供版本號碼或範圍。如果相容性模式禁止刪除必要的版本，例如 BACKWARDS_FULL，則會傳回錯誤。在此呼叫後呼叫 `GetSchemaVersions` API 將列出已刪除版本的狀態。

當版本號碼的範圍包含檢查指向的版本時，API 將傳回 409 衝突，並且不會繼續刪除。您必須首先使用 `DeleteSchemaCheckpoint` API 移除檢查點，然後再使用此 API。

您無法使用 `DeleteSchemaVersions` API 刪除結構描述集中的第一個結構描述版本。第一個結構描述版本只能由 `DeleteSchema` API 刪除。此操作也將刪除結構描述版本下連接的 `SchemaVersionMetadata`。將在資料庫上強制執行硬式刪除。

如果相容性模式禁止刪除必要的版本，例如 BACKWARDS_FULL，則會傳回錯誤。

請求

- `SchemaId` – 必要：SchemaId 物件。

  這是包裝函式結構，可包含結構描述名稱和 Amazon Resource Name (ARN)。
- `Versions` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 100000 個位元組，且需符合 Custom string pattern #34。

  可以提供一個版本範圍，其格式可能是：
  - 單一版本號碼，5
  - 範圍：5-8：刪除版本 5、6、7、8

回應

- `SchemaVersionErrors` – 一個 SchemaVersionErrorItem 物件陣列。

  `SchemaVersionErrorItem` 物件的清單，每個物件都包含一個錯誤和結構描述版本。

錯誤

- `InvalidInputException`
- `EntityNotFoundException`
- `AccessDeniedException`

- ConcurrentModificationException

工作流程

工作流程 API 描述了與中建立、更新或檢視工作流程相關的資料類型和 API AWS Glue。Job 流程和作業執行可存取 90 天的工作執行歷程記錄。

資料類型

- JobNodeDetails 結構
- CrawlerNodeDetails 結構
- TriggerNodeDetails 結構
- Crawl 結構
- Node 結構
- Edge 結構
- Workflow 結構
- WorkflowGraph 結構
- WorkflowRun 結構
- WorkflowRunStatistics 結構
- StartingEventBatchCondition 結構
- Blueprint 結構
- BlueprintDetails 結構
- LastActiveDefinition 結構
- BlueprintRun 結構

JobNodeDetails 結構

工作流程中顯示的 Job 節點詳細資訊。

欄位

- JobRuns – 一個 JobRun 物件陣列。

job 節點代表的任務資訊。

CrawlerNodeDetails 結構

工作流程中顯示的 Crawler 節點詳細資訊。

欄位

- `Crawls` – 一個 編目 物件陣列。

  crawl 節點代表的編目清單。

TriggerNodeDetails 結構

工作流程中顯示的 Trigger 節點詳細資訊。

欄位

- `Trigger` – 觸發條件 物件。

  trigger 節點代表的觸發資訊。

Crawl 結構

工作流程中的編目詳細資訊。

欄位

- `State` – UTF-8 字串 (有效值：`RUNNING` | `CANCELLING` | `CANCELLED` | `SUCCEEDED` | `FAILED` | `ERROR`)。

  爬蟲程式的狀態。
- `StartedOn` – 時間戳記。

  開始編目的日期和時間。
- `CompletedOn` – 時間戳記。

  完成編目的日期和時間。

- `ErrorMessage` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  與此編目相關聯的錯誤訊息。
- `LogGroup` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組，且需符合 Log group string pattern。

  與編目相關聯的日誌群組。
- `LogStream` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組，且需符合 Log-stream string pattern。

  與編目相關聯的日誌串流。

Node 結構

節點代表工作流程圖形上的 AWS Glue 元件 (觸發程式、爬行者程式或工作)。

欄位

- `Type` – UTF-8 字串 (有效值：`CRAWLER` | `JOB` | `TRIGGER`)。

  節點所表示的 AWS Glue 元件類型。
- `Name` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  節點所表示的 AWS Glue 元件名稱。
- `UniqueId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  工作流程內指派給節點的唯一 ID。
- `TriggerDetails` – TriggerNode詳情 物件。

  當節點代表 Trigger 時的觸發詳細資訊。
- `JobDetails` – JobNode詳情 物件。

  當節點代表 Job 時的任務詳細資訊。
- `CrawlerDetails` – CrawlerNode詳情 物件。

  當節點代表 Crawler 時的爬蟲程式詳細資訊。

Edge 結構

邊代表兩個元件之間的直接連接，這兩個 AWS Glue 元件是邊所屬工作流程的一部分。

欄位

- `SourceId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  節點開始所在工作流程的唯一節點。
- `DestinationId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  節點結束所在工作流程的唯一節點。

Workflow 結構

工作流程是為了完成複雜 ETL AWS Glue 工作而執行的多個相依工作和編目器的集合。每項工作流程都管理其所有任務和爬蟲程式的執行和監控。

欄位

- `Name` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  工作流程的名稱。
- `Description` – UTF-8 字串。

  工作流程的描述。
- `DefaultRunProperties` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串。

  要做為每項工作流程執行一部分使用的屬性集合。執行屬性可供工作流程中的每個任務使用。任務可以修改流程中下一個任務的屬性。
- `CreatedOn` – 時間戳記。

工作流程建立的日期和時間。

- `LastModifiedOn` – 時間戳記。

  工作流程上次修改的日期和時間。

- `LastRun` – WorkflowRun 物件。

  上次執行工作流程的資訊。

- `Graph` – WorkflowGraph 物件。

  以節點形式表示屬於工作流程的所有 AWS Glue 元件，以及它們之間作為邊的定向連接的圖形。

- `CreationStatus` – UTF-8 字串 (有效值：`CREATING` | `CREATED` | `CREATION_FAILED`)。

  工作流程的建立狀態。

- `MaxConcurrentRuns` – 數字 (整數)。

  您可以使用此參數來防止不想要的資料更新、控制成本，或在某些情況下，防止超過任何元件任務的並行執行次數上限。此參數若保留空白，即不限制並行工作流程執行的數目。

- `BlueprintDetails` – BlueprintDetails 物件。

  此結構指出建立此特定工作流程的藍圖詳細資訊。

WorkflowGraph 結構

工作流程圖代表完整的工作流程，包含工作流程中的所有 AWS Glue 元件，以及它們之間所有的導向連線。

欄位

- `Nodes` – 一個 節點 物件陣列。

  AWS Glue 元件清單屬於以節點表示的工作流程。

- `Edges` – 一個 Edge 物件陣列。

  屬於工作流程之節點間的所有導向連線清單。

WorkflowRun 結構

工作流程回合是執行提供所有執行時間資訊的工作流程。

欄位

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  已執行的工作流程名稱。

- WorkflowRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  此工作流程回合的 ID。

- PreviousRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  先前工作流程執行的 ID。

- WorkflowRunProperties – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串。

  在執行期間設定的工作流程回合屬性。

- StartedOn – 時間戳記。

  工作流程回合開始的日期和時間。

- CompletedOn – 時間戳記。

  工作流程回合完成的日期和時間。

- Status – UTF-8 字串 (有效值：RUNNING | COMPLETED | STOPPING | STOPPED | ERROR)。

  工作流程回合的狀態。

- ErrorMessage – UTF-8 字串。

  此錯誤訊息說明啟動工作流程執行時可能發生的任何錯誤。目前唯一的錯誤訊息是「並行執行超過工作流程：foo。」

- `Statistics` – WorkflowRun統計 物件。

  該回合的統計資料。

- `Graph` – WorkflowGraph 物件。

  以節點形式表示屬於工作流程的所有 AWS Glue 元件，以及它們之間作為邊的定向連接的圖形。

- `StartingEventBatchCondition` – StartingEventBatchCondition 物件。

  啟動工作流程執行的批次條件。

WorkflowRunStatistics 結構

工作流程回合統計資料，提供關於工作流程回合的統計資料。

欄位

- `TotalActions` – 數字 (整數)。

  工作流程回合的動作總數。

- `TimeoutActions` – 數字 (整數)。

  逾時動作的總數。

- `FailedActions` – 數字 (整數)。

  失敗動作的總數。

- `StoppedActions` – 數字 (整數)。

  已停止動作的總數。

- `SucceededActions` – 數字 (整數)。

  成功動作的總數。

- `RunningActions` – 數字 (整數)。

  正在執行狀態的動作總數。

- `ErroredActions` – 數字 (整數)。

  表示工作流程執行中處於 ERROR 狀態的任務執行計數。

- `WaitingActions` – 數字 (整數)。

表示工作流程執行中處於 WAITING 狀態的任務執行計數。

StartingEventBatchCondition 結構

啟動工作流程執行的批次條件。批次大小到達的事件數目 (在此情況下， BatchSize 成員為非零)，或批次視窗已過期 (在此情況下， BatchWindow 成員為非零)。

欄位

- `BatchSize` – 數字 (整數)。

  批次中的事件數目。
- `BatchWindow` – 數字 (整數)。

  批次間隔的持續時間 (以秒為單位)。

Blueprint 結構

藍圖的詳細資訊。

欄位

- `Name` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  藍圖的名稱。
- `Description` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組。

  藍圖的描述。
- `CreatedOn` – 時間戳記。

  藍圖註冊的日期和時間。
- `LastModifiedOn` – 時間戳記。

  上次修改藍圖的日期和時間。
- `ParameterSpec` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 131072 個位元組。

  JSON 字串，指出藍圖的參數規格清單。

- `BlueprintLocation` – UTF-8 字串。

  指定 Amazon S3 中發佈藍圖的路徑。

- `BlueprintServiceLocation` – UTF-8 字串。

  在 Amazon S3 中指定一個路徑，當您呼叫 `CreateBlueprint/UpdateBlueprint` 以將藍圖註冊到 AWS Glue時要複製藍圖的路徑。

- `Status` – UTF-8 字串 (有效值：`CREATING` | `ACTIVE` | `UPDATING` | `FAILED`)。

  藍圖註冊的狀態。

  - 正在建立 — 藍圖註冊正在進行中。
  - 使用中 — 藍圖已成功註冊。
  - 正在更新 — 正在進行藍圖註冊的更新。
  - 失敗 — 藍圖註冊失敗。

- `ErrorMessage` – UTF-8 字串。

  錯誤訊息。

- `LastActiveDefinition` – LastActive定義 物件。

  當藍圖有多個版本且最新版本發生某些錯誤時，此屬性會指出服務可用的上次成功藍圖定義。

BlueprintDetails 結構

藍圖的詳細資訊。

欄位

- `BlueprintName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  藍圖的名稱。

- `RunId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  此藍圖的執行 ID。

LastActiveDefinition 結構

當藍圖有多個版本且最新版本發生某些錯誤時，此屬性會指出服務可用的上次成功藍圖定義。

欄位

- `Description` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組。

  藍圖的描述。
- `LastModifiedOn` – 時間戳記。

  上次修改藍圖的日期和時間。
- `ParameterSpec` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 131072 個位元組。

  指定藍圖參數的 JSON 字串。
- `BlueprintLocation` – UTF-8 字串。

  在 Amazon S3 中指定 AWS Glue 開發人員在其中發佈藍圖的路徑。
- `BlueprintServiceLocation` – UTF-8 字串。

  在 Amazon S3 中指定一個路徑，當您建立或更新藍圖時從中複製藍圖。

BlueprintRun 結構

藍圖執行的詳細資訊

欄位

- `BlueprintName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  藍圖的名稱。
- `RunId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  此藍圖執行的執行 ID。
- `WorkflowName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

由於成功執行藍圖而建立的工作流程名稱。如果藍圖執行發生錯誤，則不會建立工作流程。

- State – UTF-8 字串 (有效值：RUNNING | SUCCEEDED | FAILED | ROLLING_BACK)。

  藍圖的執行狀態。可能值為：

  - 執行中 — 藍圖執行正在進行中。
  - 成功 — 藍圖執行成功完成。
  - 失敗 — 藍圖執行失敗且已完成回復。
  - 復原 — 藍圖執行失敗且正在進行回復。
- StartedOn – 時間戳記。

  藍圖執行開始的日期和時間。
- CompletedOn – 時間戳記。

  藍圖執行完成的日期和時間。
- ErrorMessage – UTF-8 字串。

  指出執行藍圖時看到的任何錯誤。
- RollbackErrorMessage – UTF-8 字串。

  如果在建立一個工作流程的實體有任何錯誤，我們會嘗試回復建立的實體直到該點並刪除它們。此屬性指出嘗試刪除所建立的實體時看到的錯誤。
- Parameters – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 131072 個位元組。

  藍圖參數為字串。您必須為每個索引鍵提供一個值，這是 Blueprint$ParameterSpec 中定義的參數規格所必需的。
- RoleArn – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 1024 個位元組，需符合 Custom string pattern #26。

  角色 ARN。此角色將由 AWS Glue 服務承擔，並將用來建立工作流程和工作流程的其他實體。

作業

- CreateWorkflow 動作 (Python: 建立工作流程)
- UpdateWorkflow 動作 (Python: 更新工作流程)
- DeleteWorkflow 動作 (Python: 刪除工作流程)

- GetWorkflow 行動（Python：獲取工作流程）
- ListWorkflows 動作 (Python: 清單工作流程)
- BatchGetWorkflows 操作（Python：批處理工作流程）
- GetWorkflowRun 行動（Python：工作流程運行）
- GetWorkflowRuns 行動（Python：獲取工作流程運行）
- GetWorkflowRunProperties 行動（Python：獲取工作流程 _ 運行屬性）
- PutWorkflowRunProperties 行動（Python：把工作流程 _ 運行屬性）
- CreateBlueprint 行動（Python：創建藍圖）
- UpdateBlueprint 行動（Python：更新藍圖）
- DeleteBlueprint 行動（Python：刪除藍圖）
- ListBlueprints 行動（Python：列表藍圖）
- BatchGetBlueprints 行動（Python：批處理藍圖）
- StartBlueprintRun 行動（Python：開始 _ 藍印運行）
- GetBlueprintRun 行動（Python：獲取藍印運行）
- GetBlueprintRuns 行動（Python：獲取藍印運行）
- StartWorkflowRun 行動（Python：開始工作流程運行）
- StopWorkflowRun 行動（Python：停止工作流程 _ 運行）
- ResumeWorkflowRun 行動（Python：恢復工作流程運行）

CreateWorkflow 動作 (Python: 建立工作流程)

建立新的工作流程。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要指派給工作流程的名稱。它在您帳戶中應該是唯一的。

- `Description` – UTF-8 字串。

  工作流程的描述。

- `DefaultRunProperties` – 金鑰值對的映射陣列。

每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

每個值都是 UTF-8 字串。

要做為每項工作流程執行一部分使用的屬性集合。

- Tags – 金鑰值對的對應陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  要用於這個工作流程的標籤。

- MaxConcurrentRuns – 數字 (整數)。

  您可以使用此參數來防止不想要的資料更新、控制成本，或在某些情況下，防止超過任何元件任務的並行執行次數上限。此參數若保留空白，即不限制並行工作流程執行的數目。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  提供為請求之一部分的工作流程名稱。

錯誤

- AlreadyExistsException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentModificationException

UpdateWorkflow 動作 (Python: 更新工作流程)

更新現有的工作流程。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要更新的工作流程名稱。

- Description – UTF-8 字串。

  工作流程的描述。

- DefaultRunProperties – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串。

  要做為每項工作流程執行一部分使用的屬性集合。

- MaxConcurrentRuns – 數字 (整數)。

  您可以使用此參數來防止不想要的資料更新、控制成本，或在某些情況下，防止超過任何元件任務的並行執行次數上限。此參數若保留空白，即不限制並行工作流程執行的數目。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  在輸入中指定的工作流程名稱。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException

- OperationTimeoutException
- ConcurrentModificationException

DeleteWorkflow 動作 (Python: 刪除工作流程)

刪除工作流程。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要刪除的工作流程名稱。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  在輸入中指定的工作流程名稱。

錯誤

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException

GetWorkflow 行動（Python：獲取工作流程）

擷取工作流程的資源中繼資料。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要擷取的工作流程名稱。

- IncludeGraph – 布林值。

  傳回工作流程資源中繼資料時，指定是否包含圖形。

回應

- Workflow – 工作流程 物件。

  工作流程的資源中繼資料。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

ListWorkflows 動作 (Python: 清單工作流程)

在帳戶中建立的工作流程清單名稱。

請求

- NextToken – UTF-8 字串。

  接續符記，如果這是接續要求。
- MaxResults— 數字 (整數)，不小於 1 或大於 25。

  所要回傳清單的大小上限。

回應

- Workflows – UTF-8 字串的陣列，不可小於 1 或超過 25 個字串。

  帳戶中的工作流程名稱清單。
- NextToken – UTF-8 字串。

  若未傳回所有的工作流程名稱，則為接續字符。

錯誤

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchGetWorkflows 操作（Python：批處理工作流程）

針對指定的工作流程清單，傳回資源中繼資料的清單。呼叫 ListWorkflows 操作之後，您便可以呼叫此操作來存取您已授與許可的資料。此操作支援所有 IAM 許可，包括使用標籤的許可條件。

請求

- Names – 必要：UTF-8 字串的陣列，不可小於 1 或超過 25 個字串。

  工作流程名稱清單，可能是從 ListWorkflows 操作傳回的名稱。
- IncludeGraph – 布林值。

  傳回工作流程資源中繼資料時，指定是否包含圖形。

回應

- Workflows – 工作流程 物件陣列，不小於 1 個結構，也不大於 25 個結構。

  工作流程資源中繼資料清單。
- MissingWorkflows – UTF-8 字串的陣列，不可小於 1 或超過 25 個字串。

  找不到工作流程名稱清單。

錯誤

- InternalServiceException
- OperationTimeoutException
- InvalidInputException

GetWorkflowRun 行動（Python：工作流程運行）

擷取指定工作流程回合的中繼資料。Job 流程和作業執行可存取 90 天的工作執行歷程記錄。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  正在執行的工作流程名稱。

- RunId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  工作流程回合的 ID。

- IncludeGraph – 布林值。

  指定回應是否包含工作流程圖。

回應

- Run – WorkflowRun 物件。

  請求的工作流程回合中繼資料。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetWorkflowRuns 行動（Python：獲取工作流程運行）

擷取指定工作流程所有回合的中繼資料。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  應傳回回合中繼資料的工作流程名稱。

- IncludeGraph – 布林值。

  指定回應是否包含工作流程圖。

- NextToken – UTF-8 字串。

  回應的大小上限。

- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  要包含在回應中的工作流程回合最大數量。

回應

- Runs – WorkflowRun 物件陣列，不小於 1 個結構，也不大於 1000 個結構。

  工作流程回合中繼資料物件的清單。

- NextToken – UTF-8 字串。

  接續字符 (如果尚未傳回所有請求的工作流程回合)。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetWorkflowRunProperties 行動（Python：獲取工作流程 _ 運行屬性）

擷取在執行期間設定的工作流程回合屬性。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  已執行的工作流程名稱。

- RunId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  應該傳回其回合屬性的工作流程回合 ID。

回應

- RunProperties – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串。

  在指定回合期間設定的工作流程回合屬性。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

PutWorkflowRunProperties 行動（Python：把工作流程 _ 運行屬性）

針對指定的工作流程回合放置指定的工作流程回合屬性。如果指定的回合已有屬性，則會覆寫該值，否則會將屬性新增到現有的屬性。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

已執行的工作流程名稱。

- RunId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  應該更新回合屬性的工作流程回合 ID。

- RunProperties – 必要：金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串。

  針對指定回合放置的屬性。

回應

- 無回應參數。

錯誤

- AlreadyExistsException
- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentModificationException

CreateBlueprint 行動（Python：創建藍圖）

使用註冊藍圖 AWS Glue。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

藍圖的名稱。

- Description – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組。

  藍圖的描述。

- BlueprintLocation – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 8192 個位元組，且需符合 Custom string pattern #28。

  指定 Amazon S3 中發佈藍圖的路徑。

- Tags – 金鑰值對的對應陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  要套用至此藍圖的標籤。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  傳回已註冊藍圖的名稱。

錯誤

- AlreadyExistsException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ResourceNumberLimitExceededException

UpdateBlueprint 行動（Python：更新藍圖）

更新已註冊的藍圖。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  藍圖的名稱。

- Description – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 512 個位元組。

  藍圖的描述。

- BlueprintLocation – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 8192 個位元組，且需符合 Custom string pattern #28。

  指定 Amazon S3 中發佈藍圖的路徑。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  傳回已更新的藍圖名稱。

錯誤

- EntityNotFoundException
- ConcurrentModificationException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- IllegalBlueprintStateException

DeleteBlueprint 行動（Python：刪除藍圖）

刪除現有的藍圖。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  欲刪除的藍圖名稱。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  傳回已刪除藍圖的名稱。

錯誤

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

ListBlueprints 行動（Python：列表藍圖）

列出帳戶中的所有藍圖名稱。

請求

- `NextToken` – UTF-8 字串。

  接續符記，如果這是接續要求。
- `MaxResults`— 數字 (整數)，不小於 1 或大於 25。

  所要回傳清單的大小上限。
- `Tags` – 金鑰值對的對應陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  依 AWS 資源標籤篩選清單。

回應

- Blueprints – UTF-8 字串陣列。

  帳戶中的藍圖名稱清單。

- NextToken – UTF-8 字串。

  持續符記 (如果並非所有藍圖名稱都已傳回)。

錯誤

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchGetBlueprints 行動（Python：批處理藍圖）

擷取有關藍圖清單的資訊。

請求

- Names – 必要：UTF-8 字串的陣列，不可小於 1 或超過 25 個字串。

  藍圖名稱清單。

- IncludeBlueprint – 布林值。

  指定回應是否包含藍圖。

- IncludeParameterSpec – 布林值。

  指定是否在回應中包含藍圖的參數做為 JSON 字串。

回應

- Blueprints – 一個 藍圖 物件陣列。

  以 Blueprints 物件傳回藍圖清單。

- MissingBlueprints – UTF-8 字串陣列。

傳回找不到的 BlueprintNames 清單。

錯誤

- InternalServiceException
- OperationTimeoutException
- InvalidInputException

StartBlueprintRun 行動（Python：開始 _ 藍印運行）

啟動指定工作流程的新執行。

請求

- BlueprintName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  藍圖的名稱。
- Parameters – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 131072 個位元組。

  指定參數為 BlueprintParameters 物件。
- RoleArn – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 1024 個位元組，需符合Custom string pattern #26。

  指定用於建立工作流程的 IAM 角色。

回應

- RunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  此藍圖執行的執行 ID。

錯誤

- InvalidInputException
- OperationTimeoutException

- InternalServiceException
- ResourceNumberLimitExceededException
- EntityNotFoundException
- IllegalBlueprintStateException

GetBlueprintRun 行動（Python：獲取藍印運行）

擷取藍圖執行的詳細資訊。

請求

- BlueprintName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #27。

  藍圖的名稱。

- RunId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  您要擷取的藍圖執行的執行 ID。

回應

- BlueprintRun – BlueprintRun 物件。

  其會傳回 BlueprintRun 物件。

錯誤

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetBlueprintRuns 行動（Python：獲取藍印運行）

擷取指定藍圖的藍圖執行詳細資訊

請求

- BlueprintName – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  藍圖的名稱。

- NextToken – UTF-8 字串。

  接續符記，如果這是接續要求。

- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  所要回傳清單的大小上限。

回應

- BlueprintRuns – 一個 BlueprintRun 物件陣列。

  傳回 BlueprintRun 物件的清單。

- NextToken – UTF-8 字串。

  持續符記 (如果並非所有藍圖都已傳回)。

錯誤

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException

StartWorkflowRun 行動（Python：開始工作流程運行）

啟動指定工作流程的新執行。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

要啟動的工作流程名稱。

- RunProperties – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 UTF-8 字串。

  針對新的工作流程執行的工作流程執行屬性。

回應

- RunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  新執行的 ID。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentRunsExceededException

StopWorkflowRun 行動（Python：停止工作流程 _ 運行）

停止執行指定的工作流程執行。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要停止的工作流程名稱。

- RunId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要停止的工作流程執行的 ID。

回應

- 無回應參數。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- IllegalWorkflowStateException

ResumeWorkflowRun 行動（Python：恢復工作流程運行）

重新啟動先前部分完成工作流程執行的所選節點，並繼續執行工作流程。會執行所選節點以及從所選節點下游的所有節點。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要繼續的工作流程名稱。

- RunId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要繼續的工作流程執行的 ID。

- NodeIds – 必要：UTF-8 字串陣列。

  您要重新啟動的節點 ID 清單。要重新啟動的節點必須在原始執行中嘗試執行。

回應

- RunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  指派給已繼續工作流程執行的新 ID。工作流程執行的每次繼續都會有一個新的執行 ID。
- NodeIds – UTF-8 字串陣列。

  實際重新啟動之節點的節點 ID 清單。

錯誤

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- ConcurrentRunsExceededException
- IllegalWorkflowStateException

使用設定檔

使用情況設定檔 API 說明與中建立、更新或檢視使用情況設定檔相關的資料類型和 API AWS Glue。

資料類型

- ProfileConfiguration 結構
- ConfigurationObject 結構
- UsageProfileDefinition 結構

ProfileConfiguration 結構

指定管理員在 AWS Glue 使用情況設定檔中設定的工作和階段作業值。

欄位

- SessionConfiguration – 金鑰值對的映射陣列。

每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

每個值都是 ConfigurationObject 物件。

AWS Glue 階段作業組態參數的索引鍵值對應。

- `JobConfiguration` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 ConfigurationObject 物件。

  AWS Glue 工作組態參數的鍵值對應。

ConfigurationObject 結構

指定管理員為 AWS Glue 使用情況設定檔中設定的每個工作或工作階段參數設定的值。

欄位

- `DefaultValue` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #31。

  參數的預設值。

- `AllowedValues` – UTF-8 字串陣列。

  允許的參數值清單。

- `MinValue` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #31。

  允許的參數的最小值。

- `MaxValue` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組，且需符合 Custom string pattern #31。

  允許的參數值上限。

UsageProfileDefinition 結構

描述 AWS Glue 使用情況設定檔。

欄位

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  使用情況設定檔的名稱。
- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  使用情況設定檔的說明。
- CreatedOn – 時間戳記。

  建立使用情況設定檔的日期和時間。
- LastModifiedOn – 時間戳記。

  上次修改使用情況設定檔的日期和時間。

作業

- CreateUsageProfile 行動（Python：創建 _ 使用 _ 配置文件）
- GetUsageProfile 行動（Python：獲取使用 _ 配置文件）
- UpdateUsageProfile 行動（Python：更新 _ 使用 _ 配置文件）
- DeleteUsageProfile 行動（Python：刪除使用 _ 配置文件）
- ListUsageProfiles 行動（Python：列表使用 _ 配置文件）

CreateUsageProfile 行動（Python：創建 _ 使用 _ 配置文件）

建立 AWS Glue 使用情況設定檔。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

使用情況設定檔的名稱。

- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  使用情況設定檔的說明。

- `Configuration` – 必要：ProfileConfiguration 物件。

  指定設定檔的工作和階段作業值的`ProfileConfiguration`物件。

- `Tags` – 金鑰值對的映射陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  套用至使用情況設定檔的標籤清單。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  已建立的使用情況設定檔名稱。

錯誤

- `InvalidInputException`
- `InternalServiceException`
- `AlreadyExistsException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `OperationNotSupportedException`

GetUsageProfile 行動（Python：獲取使用 _ 配置文件）

擷取有關指定 AWS Glue 使用情況設定檔的資訊。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要擷取的使用情況設定檔名稱。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  使用情況設定檔的名稱。
- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  使用情況設定檔的說明。
- Configuration – ProfileConfiguration 物件。

  指定設定檔的工作和階段作業值的ProfileConfiguration物件。
- CreatedOn – 時間戳記。

  建立使用情況設定檔的日期和時間。
- LastModifiedOn – 時間戳記。

  上次修改使用情況設定檔的日期和時間。

錯誤

- InvalidInputException
- InternalServiceException
- EntityNotFoundException
- OperationTimeoutException
- OperationNotSupportedException

UpdateUsageProfile 行動（Python：更新 _ 使用 _ 配置文件）

更新 AWS Glue 使用情況設定檔。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  使用情況設定檔的名稱。
- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  使用情況設定檔的說明。
- Configuration – 必要：ProfileConfiguration 物件。

  指定設定檔的工作和階段作業值的ProfileConfiguration物件。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  已更新的使用情況設定檔名稱。

錯誤

- InvalidInputException
- InternalServiceException
- EntityNotFoundException
- OperationTimeoutException
- OperationNotSupportedException
- ConcurrentModificationException

DeleteUsageProfile 行動（Python：刪除使用 _ 配置文件）

刪除 AWS Glue 指定的使用情況設定檔。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要刪除的使用情況設定檔名稱。

回應

- 無回應參數。

錯誤

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `OperationNotSupportedException`

ListUsageProfiles 行動（Python：列表使用 _ 配置文件）

列出所有 AWS Glue 使用情況設定檔。

請求

- `NextToken` – UTF-8 字串，長度不可超過 400000 個位元組。

  接續符記，如果這是接續呼叫，將會包含在內。
- `MaxResults`— 數字 (整數)，不小於 1 或大於 200。

  單一回應中要傳回的使用情況設定檔數目上限。

回應

- `Profiles` – 一個 UsageProfileDefinition 物件陣列。

  使用情況設定檔 (`UsageProfileDefinition`) 物件的清單。
- `NextToken` – UTF-8 字串，長度不可超過 400000 個位元組。

接續字元，如果目前清單區段不是最後一個，將會出現此接續字元。

錯誤

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `OperationNotSupportedException`

機器學習 API

機器學習 API 說明機器學習資料類型，包含建立、刪除或更新轉換或啟動機器學習任務執行的 API。

資料類型

- TransformParameters 結構
- EvaluationMetrics 結構
- MLTransform 結構
- FindMatchesParameters 結構
- FindMatchesMetrics 結構
- ConfusionMatrix 結構
- GlueTable 結構
- TaskRun 結構
- TransformFilterCriteria 結構
- TransformSortCriteria 結構
- TaskRunFilterCriteria 結構
- TaskRunSortCriteria 結構
- TaskRunProperties 結構
- FindMatchesTaskRunProperties 結構
- ImportLabelsTaskRunProperties 結構
- ExportLabelsTaskRunProperties 結構

- LabelingSetGenerationTaskRunProperties 結構
- SchemaColumn 結構
- TransformEncryption 結構
- 毫升UserDataEncryption 結構
- ColumnImportance 結構

TransformParameters 結構

與機器學習轉換相關聯的演算法特定參數。

欄位

- `TransformType` – 必要：UTF-8 字串 (有效值：`FIND_MATCHES`)。

  機器學習轉換的類型。

  如需機器學習轉換類型的資訊，請參閱建立機器學習轉換。
- `FindMatchesParameters` – FindMatches參數 物件。

  find matches 演算法的參數。

EvaluationMetrics 結構

評估指標會提供機器學習轉換品質的預估值。

欄位

- `TransformType` – 必要：UTF-8 字串 (有效值：`FIND_MATCHES`)。

  機器學習轉換的類型。
- `FindMatchesMetrics` – FindMatches度量 物件。

  find matches 演算法的評估指標。

MLTransform 結構

機器學習轉換的結構。

欄位

- `TransformId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  針對機器學習轉換產生的唯一轉換 ID。此 ID 保證是唯一的，且不會變更。

- `Name` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  機器學習轉換的使用者定義名稱。名稱不保證唯一，且可隨時變更。

- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  機器學習轉換的使用者定義描述長文。描述不保證唯一，且可隨時變更。

- `Status` – UTF-8 字串 (有效值：`NOT_READY` | `READY` | `DELETING`)。

  機器學習轉換目前的狀態。

- `CreatedOn` – 時間戳記。

  時間戳記。此機器學習轉換建立的時間和日期。

- `LastModifiedOn` – 時間戳記。

  時間戳記。此機器學習轉換最後一次修改的時間點。

- `InputRecordTables` – 一個 GlueTable 物件陣列，不可超過 10 個結構。

  轉換所使用的 AWS Glue 資料表定義清單。

- `Parameters` – TransformParameters 物件。

  `TransformParameters` 物件。您可以透過指定學習資料和各種權衡的偏好設定 (如精確率與回收率，或準確性與成本)，使用參數來調整 (自訂) 機器學習轉換的行為。

- `EvaluationMetrics` – EvaluationMetrics 物件。

  `EvaluationMetrics` 物件。評估指標會提供機器學習轉換品質的預估值。

- `LabelCount` – 數字 (整數)。

  針對此轉換產生的標籤檔案的 AWS Glue 計數識別碼。當您建立更佳的轉換時，您就可以反覆下載、標記和上傳標記檔案。

- `Schema` – 一個 SchemaColumn 物件陣列，不可超過 100 個結構。

表示欄和資料類型的鍵/值對映射，可以執行此轉換。已有 100 個欄的上限。

- `Role` – UTF-8 字串。

  IAM 角色的名稱或 Amazon Resource Name (ARN) 與所需的許可。所需許可包括 AWS Glue 資源的 AWS Glue 服務角色許可，以及轉換所需的 Amazon S3 許可。

  - 此角色需要 AWS Glue 服務角色權限才能允許存取中的資源 AWS Glue。請參閱連接政策到存取 AWS Glue的 IAM 使用者。
  - 此角色需要此轉換任務回合所用的 Amazon Simple Storage Service (Amazon S3) 來源、目標、暫時目錄、指令碼和任何程式庫的許可。

- `GlueVersion` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #20。

  此值決定 AWS Glue 此機器學習轉換與哪個版本相容。Glue 1.0 是建議大多數客戶使用的版本。如果未設定此值，Glue 相容性預設為 Glue 0.9。如需詳細資訊，請參閱開發人員指南中的 AWS Glue 版本。

- `MaxCapacity` – 數字 (雙位數)。

  為此轉換配置給工作執行的 AWS Glue 資料處理單元 (DPU) 數目。您可以分配從 2 到 100 個 DPU，預設值為 10。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價頁面。

  `MaxCapacity` 是 `NumberOfWorkers` 和 `WorkerType` 的互斥選項。

  - 如果設定 `NumberOfWorkers` 或 `WorkerType`，則無法設定 `MaxCapacity`。
  - 如果設定 `MaxCapacity`，則無法設定 `NumberOfWorkers` 或 `WorkerType`。
  - 如果設定 `WorkerType`，則 `NumberOfWorkers` 為必要 (反之亦然)。
  - `MaxCapacity` 和 `NumberOfWorkers` 都必須至少為 1。

  如果 `WorkerType` 欄位設成 `Standard` 以外的值，就會自動設定 `MaxCapacity` 欄位並且變成唯讀。

- `WorkerType` – UTF-8 字串 (有效值：`Standard=""` | `G.1X=""` | `G.2X=""` | `G.025X=""` | `G.4X=""` | `G.8X=""` | `Z.2X=""`)。

  此轉換執行任務時所配置的預先定義工作者類型。可接受值為標準、G.1X 或 G.2X

  - 用於 `Standard` 工作者類型時，每個工作者提供 4 個 vCPU、16 GB 的記憶體和 50 GB 磁碟，以及每個工作者 2 個執行器。

  - 用於 G.1X 工作者類型時，每個工作者提供 4 個 vCPU、16 GB 的記憶體和 64 GB 磁碟，以及每個工作者 1 個執行器。
  - 用於 G.2X 工作者類型時，每個工作者提供 8 個 vCPU、32 GB 的記憶體和 128 GB 磁碟，以及每個工作者 1 個執行器。

  MaxCapacity 是 NumberOfWorkers 和 WorkerType 的互斥選項。

  - 如果設定 NumberOfWorkers 或 WorkerType，則無法設定 MaxCapacity。
  - 如果設定 MaxCapacity，則無法設定 NumberOfWorkers 或 WorkerType。
  - 如果設定 WorkerType，則 NumberOfWorkers 為必要 (反之亦然)。
  - MaxCapacity 和 NumberOfWorkers 都必須至少為 1。
- NumberOfWorkers – 數字 (整數)。

  轉換執行任務時所配置的已定義 workerType 工作者數量。

  如果設定 WorkerType，則 NumberOfWorkers 為必要 (反之亦然)。
- Timeout – 數字 (整數)，至少為 1。

  機器學習轉換的逾時，以分鐘計。
- MaxRetries – 數字 (整數)。

  機器學習轉換 MLTaskRun 失敗後的重試次數上限。
- TransformEncryption – TransformEncryption 物件。

  適用於存取使用者資料的轉換 encryption-at-rest 設定。機器學習轉換可以使用 KMS 存取 Amazon S3 中加密的使用者資料。

FindMatchesParameters 結構

設定 find matches 轉換的參數。

欄位

- PrimaryKeyColumnName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 1024 個位元組，需符合 Single-line string pattern。

  唯一識別來源資料表中資料列的欄名。用於協助識別相符的記錄。
- PrecisionRecallTradeoff – 數字 (double)，不可大於 1.0。

調校轉換以取得精確率與回收率平衡時所選取的值。值為 0.5 表示無偏好；值為 1.0 表示專門針對精確率的偏差；值為 0.0 表示針對回收率的偏差。因為這是一種權衡，選擇接近 1.0 的值表示非常低的回收率，選擇接近 0.0 的值則會得到非常低的精確率。

精確率指標指出模型預測相符項目時的正確頻率。

回收率指標表示針對實際比對，模型預測相符項目的頻率。

- `AccuracyCostTradeoff` – 數字 (double)，不可大於 1.0。

  調校轉換以取得準確性與成本平衡時所選取的值。值為 0.5 表示系統持平考量準確性與成本。值為 1.0 表示專門針對準確性的偏差，這通常會導致較高的成本，有時會非常高。值為 0.0 表示專門針對成本的偏差，這會導致低準確性的 `FindMatches` 轉換，有時是無法接受的準確性。

  Accuracy (準確性) 會測量轉換找到真肯定和真否定的效果有多好。提高正確性需要更多的機器資源和成本。但也會增加回收率。

  成本會測量執行轉換要消耗多少運算資源上，以此計算出金額。

- `EnforceProvidedLabels` – 布林值。

  值，開啟或關閉以強制輸出符合使用者提供的標籤。如果此值為 `True`，則 `find matches` 轉換會強制輸出符合提供的標籤。結果會覆寫正常的合併結果。如果此值為 `False`，則 `find matches` 轉換不確保遵守所有提供的標籤，結果會倚賴訓練過的模型。

  請注意，將此值設定為 true 可能會增加合併執行的時間。

FindMatchesMetrics 結構

find matches 演算法的評估指標。測量機器學習轉換品質的方式，是讓您的轉換預測一些相符項目，並比較此結果與相同資料集的已知相符項目。品質指標是以您的部分資料為基礎，所以不精確。

欄位

- `AreaUnderPRCurve` – 數字 (double)，不可大於 1.0。

  在精確率/回收率曲線 (AUPRC) 下的區域是測量轉換整體品質的單一數字，與精確率與回收率的選擇無關。較高的值表示您有較具吸引力的精確率與回收率權衡。

  如需詳細資訊，請參閱 Wikipedia 中的 Precision and recall。

- `Precision` – 數字 (double)，不可大於 1.0。

精確率指標指出轉換預測相符項目的正確頻率。尤其，它會測量轉換從真陽性總可能性中找出真陽性的效果有多好。

如需詳細資訊，請參閱 Wikipedia 中的 Precision and recall。

- `Recall` – 數字 (double)，不可大於 1.0。

  回收率指標表示針對實際比對，轉換預測相符項目的頻率。尤其，它會測量轉換從來源資料總記錄中找出真陽性的效果有多好。

  如需詳細資訊，請參閱 Wikipedia 中的 Precision and recall。

- `F1` – 數字 (double)，不可大於 1.0。

  F1 指標上限指出轉換的準確性介於 0 和 1 之間，其中 1 為最佳準確性。

  如需詳細資訊，請參閱 Wikipedia 中的 F1 score。

- `ConfusionMatrix` – ConfusionMatrix 物件。

  混淆矩陣會顯示轉換準確預測的內容及其產生的錯誤類型。

  如需詳細資訊，請參閱 Wikipedia 的 Confusion matrix。

- `ColumnImportances` – 一個 ColumnImportance 物件陣列，不可超過 100 個結構。

  `ColumnImportance` 結構的清單，包含欄重要性指標，依重要性遞減順序排序。

ConfusionMatrix 結構

混淆矩陣會顯示轉換準確預測的內容及其產生的錯誤類型。

如需詳細資訊，請參閱 Wikipedia 的 Confusion matrix。

欄位

- `NumTruePositives` – 數字 (long)。

  在您轉換的混淆矩陣中，轉換在資料中正確找到的相符項目數量。

- `NumFalsePositives` – 數字 (long)。

  在您轉換的混淆矩陣中，轉換在資料中誤分類為相符項目的非相符項目數量。

- `NumTrueNegatives` – 數字 (long)。

在您轉換的混淆矩陣中，轉換在資料中正確拒絕的非相符項目數量。

- `NumFalseNegatives` – 數字 (long)。

  在您轉換的混淆矩陣中，轉換在資料中沒找到的相符項目數量。

GlueTable 結構

中用於輸入或輸出資料的資料庫和表格。 AWS Glue Data Catalog

欄位

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  AWS Glue Data Catalog中的資料庫名稱。

- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  AWS Glue Data Catalog中的資料表名稱。

- `CatalogId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  AWS Glue Data Catalog的唯一識別符。

- `ConnectionName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  連至 AWS Glue Data Catalog的連線名稱。

- `AdditionalOptions` – 金鑰值對的映射陣列，不少於 1 對，也不可大於 10 對。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是描述字串，長度不可超過 2048 個位元組，且需符合 URI address multi-line string pattern。

  資料表的其他選項。目前支援兩個金鑰：

  - `pushDownPredicate`：篩選分割區，而無需列出和讀取資料集中的所有檔案。

- `catalogPartitionPredicate`：使用 AWS Glue Data Catalog中的分割區索引以進行伺服器端分割區清理。

TaskRun 結構

與機器學習轉換相關聯的取樣參數。

欄位

- `TransformId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  轉換的唯一識別符。

- `TaskRunId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  此任務回合的唯一識別符。

- `Status` – UTF-8 字串 (有效值：`STARTING` | `RUNNING` | `STOPPING` | `STOPPED` | `SUCCEEDED` | `FAILED` | `TIMEOUT`)。

  請求任務回合目前的狀態。

- `LogGroupName` – UTF-8 字串。

  與此任務回合相關聯的安全記錄日誌群組名稱。

- `Properties` – TaskRun性質 物件。

  指定與此任務回合相關聯的組態屬性。

- `ErrorString` – UTF-8 字串。

  與此任務回合相關聯的錯誤字串清單。

- `StartedOn` – 時間戳記。

  此任務回合開始的日期和時間。

- `LastModifiedOn` – 時間戳記。

  請求任務回合上次更新的時間點。

- `CompletedOn` – 時間戳記。

請求任務回合上次完成的時間點。

- `ExecutionTime` – 數字 (整數)。

  任務執行消耗資源所需的時間 (以秒為單位)。

TransformFilterCriteria 結構

用來篩選機器學習轉換的條件。

欄位

- `Name` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用來篩選機器學習轉換的唯一轉換名稱。

- `TransformType` – UTF-8 字串 (有效值：`FIND_MATCHES`)。

  用來篩選機器學習轉換的機器學習轉換類型。

- `Status` – UTF-8 字串 (有效值：`NOT_READY` | `READY` | `DELETING`)。

  依轉換的上次已知狀態篩選機器學習轉換清單 (以指出轉換是否可用)。"NOT_READY (未就緒)"、"READY (就緒)" 或 "DELETING (正在刪除)" 其中之一。

- `GlueVersion` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #20。

  此值決定 AWS Glue 此機器學習轉換與哪個版本相容。Glue 1.0 是建議大多數客戶使用的版本。如果未設定此值，Glue 相容性預設為 Glue 0.9。如需詳細資訊，請參閱開發人員指南中的 AWS Glue 版本。

- `CreatedBefore` – 時間戳記。

  轉換建立前的時間和日期。

- `CreatedAfter` – 時間戳記。

  轉換建立後的時間和日期。

- `LastModifiedBefore` – 時間戳記。

  篩選出上次修改在此日期前的轉換。

- LastModifiedAfter – 時間戳記。

  篩選出上次修改在此日期後的轉換。

- Schema – 一個 SchemaColumn 物件陣列，不可超過 100 個結構。

  篩選出具有特定結構描述的資料集。Map<Column, Type> 物件是一個鍵/值對陣列，代表此轉換接受的結構描述，其中 Column 是欄的名稱，Type 是資料類型，例如整數或字串。已有 100 個欄的上限。

TransformSortCriteria 結構

與機器學習轉換相關聯的排序條件。

欄位

- Column – 必要：UTF-8 字串 (有效值：NAME | TRANSFORM_TYPE | STATUS | CREATED | LAST_MODIFIED)。

  要用在與機器學習轉換相關聯之排序條件中的欄。

- SortDirection – 必要：UTF-8 字串 (有效值：DESCENDING | ASCENDING)。

  要用在與機器學習轉換相關聯之排序條件中的排序方向。

TaskRunFilterCriteria 結構

用來篩選機器學習轉換任務回合的條件。

欄位

- TaskRunType – UTF-8 字串 (有效值：EVALUATION | LABELING_SET_GENERATION | IMPORT_LABELS | EXPORT_LABELS | FIND_MATCHES)。

  執行的任務類型。

- Status – UTF-8 字串 (有效值：STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT)。

  所執行任務目前的狀態。

- StartedBefore – 時間戳記。

篩選出在此日期前開始的任務回合。

- StartedAfter – 時間戳記。

  篩選出在此日期後開始的任務回合。

TaskRunSortCriteria 結構

用來排序機器學習轉換任務回合清單的排序條件。

欄位

- Column – 必要：UTF-8 字串 (有效值：TASK_RUN_TYPE | STATUS | STARTED)。

  用來排序機器學習轉換任務回合清單的欄。

- SortDirection – 必要：UTF-8 字串 (有效值：DESCENDING | ASCENDING)。

  用來排序機器學習轉換任務回合清單的排序方向。

TaskRunProperties 結構

任務回合的組態屬性。

欄位

- TaskType – UTF-8 字串 (有效值：EVALUATION | LABELING_SET_GENERATION | IMPORT_LABELS | EXPORT_LABELS | FIND_MATCHES)。

  執行的任務類型。

- ImportLabelsTaskRunProperties – ImportLabelsTaskRun性質 物件。

  匯入標籤任務回合的組態屬性。

- ExportLabelsTaskRunProperties – ExportLabelsTaskRun性質 物件。

  匯出標籤任務回合的組態屬性。

- LabelingSetGenerationTaskRunProperties – LabelingSetGenerationTaskRunProperties 物件。

  標籤集產生任務回合的組態屬性。

- `FindMatchesTaskRunProperties` – FindMatchesTaskRun性質 物件。

  Find Matches 任務回合的組態屬性。

FindMatchesTaskRunProperties 結構

指定 Find Matches 任務回合的組態屬性。

欄位

- `JobId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  Find Matches 任務回合的任務 ID。
- `JobName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  指派給 Find Matches 任務回合任務的名稱。
- `JobRunId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  Find Matches 任務回合的任務回合 ID。

ImportLabelsTaskRunProperties 結構

指定匯入標籤任務回合的組態屬性。

欄位

- `InputS3Path` – UTF-8 字串。

  Amazon Simple Storage Service (Amazon S3) 路徑，您會由此匯入標籤。
- `Replace` – 布林值。

  指出是否覆寫現有的標籤。

ExportLabelsTaskRunProperties 結構

指定匯出標籤任務回合的組態屬性。

欄位

- `OutputS3Path` – UTF-8 字串。

  Amazon Simple Storage Service (Amazon S3) 路徑，您會由此匯出標籤。

LabelingSetGenerationTaskRunProperties 結構

指定標籤集產生任務回合的組態屬性。

欄位

- `OutputS3Path` – UTF-8 字串。

  Amazon Simple Storage Service (Amazon S3) 路徑，您會在此產生標籤集。

SchemaColumn 結構

鍵/值對，代表要執行此轉換的欄和資料類型。`MLTransform` 的 `Schema` 參數最多可包含 100 個這些結構。

欄位

- `Name` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 1024 個位元組，需符合 Single-line string pattern。

  欄位的名稱。

- `DataType` – UTF-8 字串，長度不可超過 131072 個位元組，需符合Single-line string pattern。

  欄中的資料類型。

TransformEncryption 結構

適用於存取使用者資料的轉換 encryption-at-rest 設定。機器學習轉換可以使用 KMS 存取 Amazon S3 中加密的使用者資料。

此外，匯入的標籤和經過訓練的轉換現在可以使用客戶提供的 KMS 金鑰加密。

欄位

- MlUserDataEncryption – ML UserData 加密 物件。

  包含加密模式和客戶提供的 KMS 金鑰 ID 的 MLUserDataEncryption 物件。

- TaskRunSecurityConfigurationName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  安全組態的名稱。

毫升UserDataEncryption 結構

適用於存取使用者資料的轉換 encryption-at-rest 設定。

欄位

- MlUserDataEncryptionMode – 必要：UTF-8 字串 (有效值：DISABLED | SSE-KMS="SSEKMS")。

  套用至使用者資料的加密模式。有效的 值如下：

  - DISABLED：已停用加密
  - SSEKMS：針對存放在 Amazon S3 中的使用者資料，使用伺服器端加密搭配 AWS Key Management Service (SSE-KMS)。

- KmsKeyId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  客戶提供的 KMS 金鑰的 ID。

ColumnImportance 結構

包含欄名稱和欄重要性分數的結構。

欄重要性可協助您了解欄對模型的貢獻方式，藉由識別記錄中的哪些欄比其他欄更重要。

欄位

- ColumnName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

欄的名稱。

- `Importance` – 數字 (double)，不可大於 1.0。

  欄的欄重要性分數 (以十進位形式)。

作業

- CreateMLTransform 動作 (Python: create_ml_transform)
- UpdateMLTransform 動作 (Python: update_ml_transform)
- DeleteMLTransform 動作 (Python: delete_ml_transform)
- GetMLTransform 動作 (Python: get_ml_transform)
- GetMLTransforms 動作 (Python: get_ml_transforms)
- ListMLTransforms 動作 (Python: list_ml_transforms)
- 啟EvaluationTaskRun 動 ML 操作 (Python：開始計算任務運行)
- 啟LabelingSetGenerationTaskRun 動 ML 動作 (Python: 啟動 _ 標籤化 _ 設定產生 _ 任務執行)
- 獲取 ML TaskRun 操作 (Python：運行)
- 獲取 ML TaskRuns 操作 (Python：運行)
- 取消毫升TaskRun 操作 (Python：取消任務運行)
- StartExportLabelsTaskRun 操作 (Python：開始 _ 導出 _ 標籤 _ 任務 _ 運行)
- StartImportLabelsTaskRun 操作 (Python：開始導入標籤任務運行)

CreateMLTransform 動作 (Python: create_ml_transform)

建立 AWS Glue 機器學習轉換。此操作會建立轉換及培訓它的所有必要參數。

呼叫此操作，做為使用機器學習轉換程序的第一步 (例如 `FindMatches` 轉換)，以刪除重複的資料。除了想要用於演算法的參數，您還可以提供選用的 `Description`。

您還必須為代表您 AWS Glue 執行的工作指定特定參數，以便從資料中學習並建立高品質的機器學習轉換。這些參數包括 `Role`，以及選用的 `AllocatedCapacity`、`Timeout` 和 `MaxRetries`。如需詳細資訊，請參閱任務。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  建立轉換時給予它的唯一名稱。

- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  正在定義的機器學習轉換描述。預設為空字串。

- InputRecordTables – 必要：一個 GlueTable 物件陣列，不可超過 10 個結構。

  轉換所使用的 AWS Glue 資料表定義清單。

- Parameters – 必要：TransformParameters 物件。

  所用轉換類型的專屬演算法參數。條件性相依於轉換類型。

- Role – 必要：UTF-8 字串。

  IAM 角色的名稱或 Amazon Resource Name (ARN) 與所需的許可。所需許可包括 AWS Glue 資源的 AWS Glue 服務角色許可，以及轉換所需的 Amazon S3 許可。

  - 此角色需要 AWS Glue 服務角色權限才能允許存取中的資源 AWS Glue。請參閱連接政策到存取 AWS Glue的 IAM 使用者。
  - 此角色需要此轉換任務回合所用的 Amazon Simple Storage Service (Amazon S3) 來源、目標、暫時目錄、指令碼和任何程式庫的許可。

- GlueVersion – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #20。

  此值決定 AWS Glue 此機器學習轉換與哪個版本相容。Glue 1.0 是建議大多數客戶使用的版本。如果未設定此值，Glue 相容性預設為 Glue 0.9。如需詳細資訊，請參閱開發人員指南中的 AWS Glue 版本。

- MaxCapacity – 數字 (雙位數)。

  為此轉換配置給工作執行的 AWS Glue 資料處理單元 (DPU) 數目。您可以分配從 2 到 100 個 DPU，預設值為 10。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價頁面。

  MaxCapacity 是 NumberOfWorkers 和 WorkerType 的互斥選項。

  - 如果設定 NumberOfWorkers 或 WorkerType，則無法設定 MaxCapacity。

- 如果設定 MaxCapacity，則無法設定 NumberOfWorkers 或 WorkerType。
- 如果設定 WorkerType，則 NumberOfWorkers 為必要 (反之亦然)。
- MaxCapacity 和 NumberOfWorkers 都必須至少為 1。

如果 WorkerType 欄位設成 Standard 以外的值，就會自動設定 MaxCapacity 欄位並且變成唯讀。

如果 WorkerType 欄位設成 Standard 以外的值，就會自動設定 MaxCapacity 欄位並且變成唯讀。

- WorkerType – UTF-8 字串 (有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="")。

  執行此任務時所配置的預先定義工作者類型。可接受值為標準、G.1X 或 G.2X

  - 用於 Standard 工作者類型時，每個工作者提供 4 個 vCPU、16 GB 的記憶體和 50 GB 磁碟，以及每個工作者 2 個執行器。
  - 用於 G.1X 工作者類型時，每個工作者提供 4 個 vCPU、16 GB 的記憶體和 64 GB 磁碟，以及每個工作者 1 個執行器。
  - 用於 G.2X 工作者類型時，每個工作者提供 8 個 vCPU、32 GB 的記憶體和 128 GB 磁碟，以及每個工作者 1 個執行器。

  MaxCapacity 是 NumberOfWorkers 和 WorkerType 的互斥選項。

  - 如果設定 NumberOfWorkers 或 WorkerType，則無法設定 MaxCapacity。
  - 如果設定 MaxCapacity，則無法設定 NumberOfWorkers 或 WorkerType。
  - 如果設定 WorkerType，則 NumberOfWorkers 為必要 (反之亦然)。
  - MaxCapacity 和 NumberOfWorkers 都必須至少為 1。
- NumberOfWorkers – 數字 (整數)。

  執行此任務時所配置的已定義 workerType 工作者數目。

  如果設定 WorkerType，則 NumberOfWorkers 為必要 (反之亦然)。
- Timeout – 數字 (整數)，至少為 1。

  此轉換任務回合的逾時，以分鐘計。這是此轉換任務回合在終止並進入 TIMEOUT 狀態前，可取用資源的最長時間。預設值為 2,880 分鐘 (48 小時)。
- MaxRetries – 數字 (整數)。

任務回合失敗後，此轉換任務可重試的次數上限。

- `Tags` – 金鑰值對的對應陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  與此機器學習轉換搭配使用的標籤。您可以使用標籤來限制對機器學習轉換的存取情況。如需中標籤的詳細資訊 AWS Glue，請參閱開發人員指南AWS Glue中的「AWS 標籤」。

- `TransformEncryption` – TransformEncryption 物件。

  適用於存取使用者資料的轉換 encryption-at-rest 設定。機器學習轉換可以使用 KMS 存取 Amazon S3 中加密的使用者資料。

回應

- `TransformId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  針對轉換產生的唯一識別符。

錯誤

- `AlreadyExistsException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `AccessDeniedException`
- `ResourceNumberLimitExceededException`
- `IdempotentParameterMismatchException`

UpdateMLTransform 動作 (Python: update_ml_transform)

更新現有的機器學習轉換。呼叫此操作以調校演算法參數，取得更佳的結果。

呼叫此操作後，您就可以呼叫 StartMLEvaluationTaskRun 操作，評估新參數達到目標的效果有多好 (例如，改善機器學習轉換的品質，或讓它更為經濟實惠)。

請求

- TransformId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  建立轉換時所產生的唯一識別符。

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  建立轉換時授予它的唯一名稱。

- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  轉換的描述。預設為空字串。

- Parameters – TransformParameters 物件。

  所用轉換類型 (演算法) 的專屬組態參數。條件性相依於轉換類型。

- Role – UTF-8 字串。

  IAM 角色的名稱或 Amazon Resource Name (ARN) 與所需的許可。

- GlueVersion – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #20。

  此值決定 AWS Glue 此機器學習轉換與哪個版本相容。Glue 1.0 是建議大多數客戶使用的版本。如果未設定此值，Glue 相容性預設為 Glue 0.9。如需詳細資訊，請參閱開發人員指南中的 AWS Glue 版本。

- MaxCapacity – 數字 (雙位數)。

  為此轉換配置給工作執行的 AWS Glue 資料處理單元 (DPU) 數目。您可以分配從 2 到 100 個 DPU，預設值為 10。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價頁面。

  如果 WorkerType 欄位設成 Standard 以外的值，就會自動設定 MaxCapacity 欄位並且變成唯讀。

- WorkerType – UTF-8 字串 (有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="")。

  執行此任務時所配置的預先定義工作者類型。可接受值為標準、G.1X 或 G.2X

  - 用於 Standard 工作者類型時，每個工作者提供 4 個 vCPU、16 GB 的記憶體和 50 GB 磁碟，以及每個工作者 2 個執行器。
  - 用於 G.1X 工作者類型時，每個工作者提供 4 個 vCPU、16 GB 的記憶體和 64 GB 磁碟，以及每個工作者 1 個執行器。
  - 用於 G.2X 工作者類型時，每個工作者提供 8 個 vCPU、32 GB 的記憶體和 128 GB 磁碟，以及每個工作者 1 個執行器。
- NumberOfWorkers – 數字 (整數)。

  執行此任務時所配置的已定義 workerType 工作者數目。
- Timeout – 數字 (整數)，至少為 1。

  此轉換任務回合的逾時，以分鐘計。這是此轉換任務回合在終止並進入 TIMEOUT 狀態前，可取用資源的最長時間。預設值為 2,880 分鐘 (48 小時)。
- MaxRetries – 數字 (整數)。

  任務回合失敗後，此轉換任務可重試的次數上限。

回應

- TransformId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  已更新的轉換唯一識別符。

錯誤

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- AccessDeniedException

DeleteMLTransform 動作 (Python: delete_ml_transform)

刪除 AWS Glue 機器學習轉換。機器學習轉換是特殊的轉換類型，透過人類提供的範例學習，使用機器學習了解要執行的轉換詳細資訊。然後會儲存這些轉換。 AWS Glue如果不再需要某項轉換，您可以呼叫 `DeleteMLTransforms` 刪除它。不過，仍參照已刪除轉換的任何 AWS Glue 工作將不再成功。

請求

- `TransformId` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  要刪除轉換的唯一識別符。

回應

- `TransformId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  已刪除轉換的唯一識別符。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

GetMLTransform 動作 (Python: get_ml_transform)

取得 AWS Glue 機器學習轉換成品及其所有對應的中繼資料。機器學習轉換是特殊的轉換類型，透過人類提供的範例學習，使用機器學習了解要執行的轉換詳細資訊。然後會儲存這些轉換。 AWS Glue您可以呼叫 `GetMLTransform` 以擷取其中繼資料。

請求

- `TransformId` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

轉換的唯一識別符，於建立轉換時產生。

回應

- `TransformId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  轉換的唯一識別符，於建立轉換時產生。

- `Name` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  建立轉換時授予它的唯一名稱。

- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  轉換的描述。

- `Status` – UTF-8 字串 (有效值：`NOT_READY` | `READY` | `DELETING`)。

  轉換的上次已知狀態 (指出其是否可用)。"NOT_READY (未就緒)"、"READY (就緒)" 或 "DELETING (正在刪除)" 其中之一。

- `CreatedOn` – 時間戳記。

  建立轉換的日期和時間。

- `LastModifiedOn` – 時間戳記。

  轉換上次修改的日期和時間。

- `InputRecordTables` – 一個 GlueTable 物件陣列，不可超過 10 個結構。

  轉換所使用的 AWS Glue 資料表定義清單。

- `Parameters` – TransformParameters 物件。

  所用演算法的專屬組態參數。

- `EvaluationMetrics` – EvaluationMetrics 物件。

  最新的評估指標。

- `LabelCount` – 數字 (整數)。

此轉換可用的標籤數量。

- Schema – 一個 SchemaColumn 物件陣列，不可超過 100 個結構。

  Map<Column, Type> 物件，代表此轉換接受的結構描述。已有 100 個欄的上限。

- Role – UTF-8 字串。

  IAM 角色的名稱或 Amazon Resource Name (ARN) 與所需的許可。

- GlueVersion – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Custom string pattern #20。

  此值決定 AWS Glue 此機器學習轉換與哪個版本相容。Glue 1.0 是建議大多數客戶使用的版本。如果未設定此值，Glue 相容性預設為 Glue 0.9。如需詳細資訊，請參閱開發人員指南中的 AWS Glue 版本。

- MaxCapacity – 數字 (雙位數)。

  為此轉換配置給工作執行的 AWS Glue 資料處理單元 (DPU) 數目。您可以分配從 2 到 100 個 DPU，預設值為 10。DPU 是相對的處理能力，包含 4 個 vCPU 的運算容量和 16 GB 的記憶體。如需詳細資訊，請參閱 AWS Glue 定價頁面。

  如果 WorkerType 欄位設成 Standard 以外的值，就會自動設定 MaxCapacity 欄位並且變成唯讀。

- WorkerType – UTF-8 字串 (有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="")。

  執行此任務時所配置的預先定義工作者類型。可接受值為標準、G.1X 或 G.2X

  - 用於 Standard 工作者類型時，每個工作者提供 4 個 vCPU、16 GB 的記憶體和 50 GB 磁碟，以及每個工作者 2 個執行器。
  - 用於 G.1X 工作者類型時，每個工作者提供 4 個 vCPU、16 GB 的記憶體和 64 GB 磁碟，以及每個工作者 1 個執行器。
  - 用於 G.2X 工作者類型時，每個工作者提供 8 個 vCPU、32 GB 的記憶體和 128 GB 磁碟，以及每個工作者 1 個執行器。

- NumberOfWorkers – 數字 (整數)。

  執行此任務時所配置的已定義 workerType 工作者數目。

- Timeout – 數字 (整數)，至少為 1。

此轉換任務回合的逾時，以分鐘計。這是此轉換任務回合在終止並進入 TIMEOUT 狀態前，可取用資源的最長時間。預設值為 2,880 分鐘 (48 小時)。

- MaxRetries – 數字 (整數)。

  任務回合失敗後，此轉換任務可重試的次數上限。

- TransformEncryption – TransformEncryption 物件。

  適用於存取使用者資料的轉換 encryption-at-rest 設定。機器學習轉換可以使用 KMS 存取 Amazon S3 中加密的使用者資料。

錯誤

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

GetMLTransforms 動作 (Python: get_ml_transforms)

取得現有 AWS Glue 機器學習轉換的可排序、可篩選清單。機器學習轉換是特殊的轉換類型，透過人類提供的範例學習，使用機器學習了解要執行的轉換詳細資訊。然後會儲存這些轉換 AWS Glue，您可以透過呼叫GetMLTransforms擷取其中繼資料。

請求

- NextToken – UTF-8 字串。

  位移結果的分頁字符。

- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  回傳結果的數量上限。

- Filter – TransformFilter标准 物件。

  篩選轉換條件。

- Sort – TransformSort标准 物件。

  排序條件。

回應

- `Transforms` – 必要：一個 MLTransform 物件。

  機器學習轉換清單。

- `NextToken` – UTF-8 字串。

  如有多個結果可用，即為分頁字符。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

ListMLTransforms 動作 (Python: list_ml_transforms)

擷取此 AWS 帳戶中現有 AWS Glue 機器學習轉換的可排序、可篩選清單，或具有指定標籤的資源。此操作會接收您可用做為回應篩選條件的選用 `Tags` 欄位，因此已標記的資源可分組進行擷取。如果您選擇使用標籤進行篩選，則此時只會擷取包含該標籤的資源。

請求

- `NextToken` – UTF-8 字串。

  接續符記，如果這是接續要求。

- `MaxResults` – 數字 (整數)，不可小於 1，也不可以大於 1000。

  所要回傳清單的大小上限。

- `Filter` – TransformFilter标准 物件。

  `TransformFilterCriteria` 會用來篩選機器學習轉換。

- `Sort` – TransformSort标准 物件。

  `TransformSortCriteria` 會用來排序機器學習轉換。

- `Tags` – 金鑰值對的對應陣列，不超過 50 對。

每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

指定只傳回包含這些標籤的資源。

回應

- `TransformIds` – 必要：UTF-8 字串陣列。

  帳戶中所有機器學習轉換的識別符，或包含指定標籤的機器學習轉換。
- `NextToken` – UTF-8 字串。

  接續字元，如果傳回的清單未包含最後一個可用指標。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

啟EvaluationTaskRun 動 ML 操作（Python：開始計算任務運行）

啟動任務以評估轉換的品質。

當您提供標籤集做為事實的範例時， AWS Glue 機器學習會使用其中一些範例從中學習。其餘標籤做為測試預估品質使用。

傳回該回合的唯一識別符。您可以呼叫 `GetMLTaskRun`，以取得`EvaluationTaskRun` 統計資料的詳細資訊。

請求

- `TransformId` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  機器學習轉換的唯一識別符。

回應

- TaskRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與此回合相關聯的唯一識別符。

錯誤

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ConcurrentRunsExceededException
- MLTransformNotReadyException

啟LabelingSetGenerationTaskRun 動 ML 動作 (Python: 啟動 _ 標籤化 _ 設定產生 _ 任務執行)

針對您的機器學習轉換開始作用中學習工作流程，透過產生標籤集及新增標籤來改善轉換的品質。

當 StartMLLabelingSetGenerationTaskRun 完成後， AWS Glue 即會產生需要人類回答的「標籤集」或問題集。

如果是 FindMatches 轉換，這些問題會是這樣的形式：「分組這些資料列的正確方式是什麼，全部由相符的記錄組成？」

完成標記程序後，您可以呼叫 StartImportLabelsTaskRun 上傳標籤。完成 StartImportLabelsTaskRun 之後，機器學習轉換未來的所有回合都會使用新的改善標籤，並執行更高品質的轉換。

請求

- TransformId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  機器學習轉換的唯一識別符。

- `OutputS3Path` – 必要：UTF-8 字串。

  Amazon Simple Storage Service (Amazon S3) 路徑，您會在此產生標籤集。

回應

- `TaskRunId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與此任務回合相關聯的唯一回合識別符。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `ConcurrentRunsExceededException`

獲取 ML TaskRun 操作（Python：運行）

取得機器學習轉換上特定任務回合的詳細資訊。機器學習工作執行是代表您 AWS Glue 執行的非同步工作，做為各種機器學習工作流程的一部分。您可以呼叫具有 `GetMLTaskRun` 的 `TaskRunID` 及其父轉換的 `TransformID`，檢查任何任務回合的統計資料。

請求

- `TransformId` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  機器學習轉換的唯一識別符。

- `TaskRunId` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  任務回合的唯一識別符。

回應

- TransformId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  任務回合的唯一識別符。

- TaskRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與此回合相關聯的唯一回合識別符。

- Status – UTF-8 字串 (有效值：STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT)。

  此任務回合的狀態。

- LogGroupName – UTF-8 字串。

  與此任務回合相關聯的日誌群組名稱。

- Properties – TaskRun性質 物件。

  與任務回合相關聯的屬性清單。

- ErrorString – UTF-8 字串。

  與任務回合相關聯的錯誤字串。

- StartedOn – 時間戳記。

  此任務回合開始的日期和時間。

- LastModifiedOn – 時間戳記。

  此任務回合上次修改的日期和時間。

- CompletedOn – 時間戳記。

  此任務回合完成的日期和時間。

- ExecutionTime – 數字 (整數)。

  任務執行消耗資源所需的時間 (以秒為單位)。

錯誤

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

獲取 ML TaskRuns 操作（Python：運行）

取得機器學習轉換的回合清單。機器學習工作執行是代表您 AWS Glue 執行的非同步工作，做為各種機器學習工作流程的一部分。您可以呼叫具有其父轉換 TransformID 的 GetMLTaskRuns 以及本節所述的其他選用參數，取得可排序、可篩選的機器學習任務回合清單。

此操作會傳回必須分頁的歷史記錄回合清單。

請求

- TransformId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  機器學習轉換的唯一識別符。
- NextToken – UTF-8 字串。

  結果的分頁字符。預設值為空白。
- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  回傳結果的數量上限。
- Filter – TaskRunFilterCriteria 物件。

  任務回合的篩選條件，結構為 TaskRunFilterCriteria。
- Sort – TaskRunSortCriteria 物件。

  任務回合的排序條件，結構為 TaskRunSortCriteria。

回應

- TaskRuns – 一個 TaskRun 物件陣列。

與轉換相關聯的任務回合清單。

- NextToken – UTF-8 字串。

  如有多個結果可用，即為分頁字符。

錯誤

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

取消毫升TaskRun 操作（Python：取消任務運行）

取消 (停止) 任務回合。機器學習工作執行是代表您 AWS Glue 執行的非同步工作，做為各種機器學習工作流程的一部分。您可以呼叫具有任務回合父轉換 TransformID 的 CancelMLTaskRun 以及任務回合的 TaskRunId，隨時取消機器學習任務回合。

請求

- TransformId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  機器學習轉換的唯一識別符。

- TaskRunId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  所執行任務的唯一識別符。

回應

- TransformId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  機器學習轉換的唯一識別符。

- TaskRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  所執行任務的唯一識別符。

- Status – UTF-8 字串 (有效值：STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT)。

  此回合的狀態。

錯誤

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

StartExportLabelsTaskRun 操作（Python：開始 _ 導出 _ 標籤 _ 任務 _ 運行）

開始非同步任務以匯出特定轉換的所有標記資料。這個任務是唯一和標籤相關，但不屬於一般作用中學習工作流程的 API 呼叫。當您想要同時使用所有的現有標籤時，您通常會使用 StartExportLabelsTaskRun，例如當您想要移除或變更之前依現況提交的標籤時。此 API 操作接受您想要匯出標籤的 TransformId，以及標籤匯出目標的 Amazon Simple Storage Service (Amazon S3) 路徑。此操作會傳回 TaskRunId。您可以呼叫 GetMLTaskRun API 來檢查任務回合的狀態。

請求

- TransformId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  機器學習轉換的唯一識別符。

- OutputS3Path – 必要：UTF-8 字串。

  標籤匯出的目標 Amazon S3 路徑。

回應

- TaskRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  所執行任務的唯一識別符。

錯誤

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

StartImportLabelsTaskRun 操作（Python：開始導入標籤任務運行）

可讓您提供額外的標籤 (真實範例)，用於教授機器學習轉換和提升其品質。此 API 操作通常做為作用中學習工作流程的一部分，從 StartMLLabelingSetGenerationTaskRun 呼叫開始，最後提升機器學習轉換的品質。

完成 StartMLLabelingSetGenerationTaskRun 後， AWS Glue 機器學習即會產生一連串需要人類回答的問題。(在機器學習工作流程中，回答這些問題通常稱為「標記」)。如果是 FindMatches 轉換，這些問題會是這樣的形式：「分組這些資料列的正確方式是什麼，全部由相符的記錄組成？」完成標記程序後，使用者會呼叫 StartImportLabelsTaskRun 以上傳他們的解答/標籤。完成 StartImportLabelsTaskRun 後，機器學習轉換未來的所有回合都會使用新的改善標籤，並執行更高品質的轉換。

依預設，除非您將 Replace 設為 true，否則 StartMLLabelingSetGenerationTaskRun 會持續學習及合併您上傳的所有標籤。如果您將 Replace 設為 true，StartImportLabelsTaskRun 會刪除並忘記之前上傳的所有標籤，只學習您上傳的確切集合。如果您發現之前上傳了不正確的標籤，而且您認為它們對轉換品質有負面影響，更換標籤很有幫助。

您可以呼叫 GetMLTaskRun 操作來檢查任務回合的狀態。

請求

- TransformId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

機器學習轉換的唯一識別符。

- `InputS3Path` – 必要：UTF-8 字串。

  Amazon Simple Storage Service (Amazon S3) 路徑，您會由此匯入標籤。

- `ReplaceAllLabels` – 布林值。

  指出是否覆寫現有的標籤。

回應

- `TaskRunId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  所執行任務的唯一識別符。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`

Data Quality API

Data Quality API 描述資料品質類型，包含建立、刪除或更新資料品質規則集、執行和評估的 API。

資料類型

- DataSource 結構
- DataQualityRulesetListDetails 結構
- DataQualityTargetTable 結構
- DataQualityRulesetEvaluationRunDescription 結構
- DataQualityRulesetEvaluationRunFilter 結構
- DataQualityEvaluationRunAdditionalRunOptions 結構

- DataQualityRuleRecommendationRunDescription 結構
- DataQualityRuleRecommendationRunFilter 結構
- DataQualityResult 結構
- DataQualityAnalyzerResult 結構
- DataQualityObservation 結構
- MetricBasedObservation 結構
- DataQualityMetricValues 結構
- DataQualityRuleResult 結構
- DataQualityResultDescription 結構
- DataQualityResultFilterCriteria 結構
- DataQualityRulesetFilterCriteria 結構

DataSource 結構

您想要其資料品質結果的資料來源 (AWS Glue 表格)。

欄位

- `GlueTable` – 必要：GlueTable 物件。

  一 AWS Glue 張桌子。

DataQualityRulesetListDetails 結構

描述 `GetDataQualityRuleset` 傳回的資料品質規則集。

欄位

- `Name` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料品質規則集的名稱。
- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  資料品質規則集的描述。

- `CreatedOn` – 時間戳記。

  建立資料品質規則集的日期和時間。

- `LastModifiedOn` – 時間戳記。

  上次修改資料品質規則集的日期和時間。

- `TargetTable` – DataQualityTargetTable 物件。

  表示 AWS Glue 表格的物件。

- `RecommendationRunId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  從建議執行建立規則集時，會產生此執行 ID 以將兩者連結在一起。

- `RuleCount` – 數字 (整數)。

  規則集中的規則數目。

DataQualityTargetTable 結構

表示 AWS Glue 表格的物件。

欄位

- `TableName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  表格的名 AWS Glue 稱。

- `DatabaseName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料 AWS Glue 表所在的資料庫名稱。

- `CatalogId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  AWS Glue 資料表所在的目錄 ID。

DataQualityRulesetEvaluationRunDescription 結構

描述資料品質規則集評估執行的結果。

欄位

- `RunId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與此回合相關聯的唯一回合識別符。

- `Status` – UTF-8 字串 (有效值：`STARTING` | `RUNNING` | `STOPPING` | `STOPPED` | `SUCCEEDED` | `FAILED` | `TIMEOUT`)。

  此回合的狀態。

- `StartedOn` – 時間戳記。

  執行開始的日期和時間。

- `DataSource` – DataSource 物件。

  與執行相關聯的資料來源 (AWS Glue 表格)。

DataQualityRulesetEvaluationRunFilter 結構

篩選條件。

欄位

- `DataSource` – 必要：DataSource 物件。

  根據與執行關聯的資料來源 (AWS Glue 表格) 進行篩選。

- `StartedBefore` – 時間戳記。

  依在此時間之前開始的執行篩選結果。

- `StartedAfter` – 時間戳記。

  依在此時間之後開始的執行篩選結果。

DataQualityEvaluationRunAdditionalRunOptions 結構

您可以為評估執行指定的其他執行選項。

欄位

- `CloudWatchMetricsEnabled` – 布林值。

  是否啟用 CloudWatch 指標。

- `ResultsS3Prefix` – UTF-8 字串。

  用於存放結果的 Amazon S3 的字首。

- `CompositeRuleEvaluationMethod` – UTF-8 字串 (有效值：`COLUMN` | `ROW`)。

  將規則集中複合規則的計算方法設定為 ROW/欄

DataQualityRuleRecommendationRunDescription 結構

描述資料品質規則建議執行的結果。

欄位

- `RunId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與此回合相關聯的唯一回合識別符。

- `Status` – UTF-8 字串 (有效值：`STARTING` | `RUNNING` | `STOPPING` | `STOPPED` | `SUCCEEDED` | `FAILED` | `TIMEOUT`)。

  此回合的狀態。

- `StartedOn` – 時間戳記。

  此執行開始時的日期和時間。

- `DataSource` – DataSource 物件。

  與建議執行相關聯的資料來源 (AWS Glue 表格)。

DataQualityRuleRecommendationRunFilter 結構

列出資料品質建議執行的篩選條件。

欄位

- `DataSource` – 必要：DataSource 物件。

  根據指定的資料來源 (AWS Glue 表格) 進行篩選。

- `StartedBefore` – 時間戳記。

  根據在提供時間之前開始的結果進行篩選。

- `StartedAfter` – 時間戳記。

  根據在提供時間之後開始的結果進行篩選。

DataQualityResult 結構

描述資料品質結果。

欄位

- `ResultId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料品質結果的唯一結果 ID。

- `Score` – 數字 (double)，不可大於 1.0。

  彙總資料品質分數。表示通過的規則數目與規則總數的比率。

- `DataSource` – DataSource 物件。

  與資料品質結果相關聯的資料表 (若有)。

- `RulesetName` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與資料品質結果相關聯的規則集名稱。

- `EvaluationContext` – UTF-8 字串。

在 AWS Glue Studio 中的工作環境中，畫布中的每個節點通常被分配某種名稱，數據質量節點將具有名稱。如果存在多個節點，evaluationContext 可以區分節點。

- StartedOn – 時間戳記。

  此資料品質執行開始的日期和時間。

- CompletedOn – 時間戳記。

  此資料品質執行完成的日期和時間。

- JobName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與資料品質結果相關聯的任務名稱 (若有)。

- JobRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與資料品質結果相關聯的任務執行 ID (若有)。

- RulesetEvaluationRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  此資料品質結果的規則集評估的唯一執行 ID。

- RuleResults – 一個 DataQualityRuleResult 物件陣列，不可超過 2000 個結構。

  代表每個規則結果的 DataQualityRuleResult 物件清單。

- AnalyzerResults – 一個 DataQualityAnalyzerResult 物件陣列，不可超過 2000 個結構。

  代表每個分析器結果的 DataQualityAnalyzerResult 物件清單。

- Observations – DataQualityObservation 物件陣列，不可超過 50 個結構。

  代表評估規則和分析器後產生的觀測值的 DataQualityObservation 物件清單。

DataQualityAnalyzerResult 結構

描述資料品質分析器評估的結果。

欄位

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料品質分析器的名稱。

- Description – UTF-8 字串，長度不可超過 2048 個位元組，且需符合 URI address multi-line string pattern。

  資料品質分析器的說明。

- EvaluationMessage – UTF-8 字串，長度不可超過 2048 個位元組，且需符合 URI address multi-line string pattern。

  評估訊息。

- EvaluatedMetrics – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是一個數字 (double)。

  與分析器評估相關的測量結果對映。

DataQualityObservation 結構

描述評估規則和分析器之後產生的觀察。

欄位

- Description – UTF-8 字串，長度不可超過 2048 個位元組，且需符合 URI address multi-line string pattern。

  資料品質觀察的描述。

- MetricBasedObservation – MetricBasedObservation 物件。

  MetricBasedObservation代表根據評估資料品質指標之觀察的類型物件。

MetricBasedObservation 結構

說明根據評估的資料品質指標所產生的以量度為基礎的觀測。

欄位

- MetricName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於產生觀測的資料品質指標名稱。
- MetricValues – DataQualityMetricValues 物件。

  DataQualityMetricValues代表資料品質測量結果值分析的類型物件。
- NewRules – UTF-8 字串陣列。

  根據資料品質指標值，作為觀察的一部分產生的新資料品質規則清單。

DataQualityMetricValues 結構

根據歷史資料的分析說明資料品質測量結果值。

欄位

- ActualValue – 數字 (雙位數)。

  資料品質測量結果的實際值。
- ExpectedValue – 數字 (雙位數)。

  根據歷史資料的分析，資料品質指標的預期值。
- LowerLimit – 數字 (雙位數)。

  根據歷史資料的分析，資料品質測量結果值的下限。
- UpperLimit – 數字 (雙位數)。

  根據歷史資料的分析，資料品質測量結果值的上限。

DataQualityRuleResult 結構

描述資料品質規則評估的結果。

欄位

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料品質規則的名稱。

- Description – UTF-8 字串，長度不可超過 2048 個位元組，且需符合 URI address multi-line string pattern。

  資料品質規則的描述。

- EvaluationMessage – UTF-8 字串，長度不可超過 2048 個位元組，且需符合 URI address multi-line string pattern。

  評估訊息。

- Result – UTF-8 字串 (有效值：PASS | FAIL | ERROR)。

  規則的通過或失敗狀態。

- EvaluatedMetrics – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是一個數字 (double)。

  與規則評估相關聯的指標映射。

DataQualityResultDescription 結構

描述資料品質結果。

欄位

- ResultId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  此資料品質結果的唯一結果 ID。

- DataSource – DataSource 物件。

  與資料品質結果相關聯的資料表名稱。

- JobName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與資料品質結果相關聯的任務名稱。

- JobRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與資料品質結果相關聯的任務執行 ID。

- StartedOn – 時間戳記。

  此資料品質結果的執行開始時間。

DataQualityResultFilterCriteria 結構

用於傳回資料品質結果的條件。

欄位

- DataSource – DataSource 物件。

  依指定的資料來源篩選結果。例如，擷取 AWS Glue 資料表的所有結果。

- JobName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  依指定的任務名稱篩選結果。

- JobRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  依指定的任務執行 ID 篩選結果。

- StartedAfter – 時間戳記。

  依在此時間之後開始的執行篩選結果。

- StartedBefore – 時間戳記。

  依在此時間之前開始的執行篩選結果。

DataQualityRulesetFilterCriteria 結構

用於篩選資料品質規則集的條件。

欄位

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  規則集篩選條件的名稱。
- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  規則集篩選條件的描述。
- CreatedBefore – 時間戳記。

  篩選在此日期之前建立的規則集。
- CreatedAfter – 時間戳記。

  篩選在此日期之後建立的規則集。
- LastModifiedBefore – 時間戳記。

  篩選在此日期之前最後一次修改的規則集。
- LastModifiedAfter – 時間戳記。

  篩選在此日期之後最後一次修改的規則集。
- TargetTable – DataQualityTargetTable 物件。

  目標資料表的名稱和資料庫名稱。

作業

- StartDataQualityRulesetEvaluationRun 動作 (Python: 開始資料品質規則評估執行)
- CancelDataQualityRulesetEvaluationRun 動作 (Python: 取消資料品質規則評估執行)
- GetDataQualityRulesetEvaluationRun 操作（Python：獲取數據質量規則評估運行）
- ListDataQualityRulesetEvaluationRuns 操作（Python：列表數據質量規則評估運行）
- StartDataQualityRuleRecommendationRun 操作（Python：開始數據質量規則推薦運行）

- CancelDataQualityRuleRecommendationRun 操作（Python：取消數據質量規則推薦運行）
- GetDataQualityRuleRecommendationRun 操作（Python：獲取數據質量規則推薦運行）
- ListDataQualityRuleRecommendationRuns 操作（Python：列表數據質量規則推薦運行）
- GetDataQualityResult 操作（Python：獲取數據質量結果）
- BatchGetDataQualityResult 操作（Python：批處理數據質量結果）
- ListDataQualityResults 操作（Python：列表數據質量結果）
- CreateDataQualityRuleset 動作 (Python: 建立品質規則集)
- DeleteDataQualityRuleset 動作 (Python: 刪除資料品質規則集)
- GetDataQualityRuleset 行動（Python：獲取質量規則集）
- ListDataQualityRulesets 操作（Python：列表數據質量規則集）
- UpdateDataQualityRuleset 行動（Python：更新數據質量規則集）

StartDataQualityRulesetEvaluationRun 動作 (Python: 開始資料品質規則評估執行)

一旦您有規則集定義 (建議或您自己的規則集定義)，您可以呼叫此作業，針對資料來源 (AWS Glue 表格) 評估規則集。評估會計算您可以使用 `GetDataQualityResult` API 擷取的結果。

請求

- `DataSource` – 必要：DataSource 物件。

  與此執行相關聯的資料來源 (AWS Glue 表格)。
- `Role` – 必要：UTF-8 字串。

  提供 IAM 用來加密執行結果的角色。
- `NumberOfWorkers` – 數字 (整數)。

  在執行中使用的 `G.1X` 工作者數目。預設值為 5。
- `Timeout` – 數字 (整數)，至少為 1。

  執行逾時 (以分鐘為單位)。此為執行在停止並進入 `TIMEOUT` 狀態前可以消耗資源的最大時間。預設值為 2,880 分鐘 (48 小時)。
- `ClientToken` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

用於等冪性且建議將其設定為隨機 ID（例如 UUID），避免建立或啟動同一資源的多個執行個體。

- `AdditionalRunOptions` – DataQualityEvaluationRunAdditionalRunOptions 物件。

  您可以為評估執行指定的其他執行選項。

- `RulesetNames` – 必要：UTF-8 字串的陣列，不可小於 1，也不可超過 10 個字串。

  規則集名稱清單。

- `AdditionalDataSources` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 DataSource 物件。

  您可以為評估執行指定的其他資料來源的參考字串映射。

回應

- `RunId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與此回合相關聯的唯一回合識別符。

錯誤

- `InvalidInputException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `InternalServiceException`
- `ConflictException`

CancelDataQualityRulesetEvaluationRun 動作 (Python: 取消資料品質規則評估執行)

取消正針對資料來源評估規則集的執行。

請求

- RunId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  與此回合相關聯的唯一回合識別符。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

GetDataQualityRulesetEvaluationRun 操作（Python：獲取數據質量規則評估運行）

擷取針對資料來源評估規則集的特定執行。

請求

- RunId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  與此回合相關聯的唯一回合識別符。

回應

- RunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與此回合相關聯的唯一回合識別符。
- DataSource – DataSource 物件。

與此評估執行相關聯的資料來源 (資料 AWS Glue 表)。

- `Role` – UTF-8 字串。

  提供 IAM 用來加密執行結果的角色。

- `NumberOfWorkers` – 數字 (整數)。

  在執行中使用的 `G.1X` 工作者數目。預設值為 5。

- `Timeout` – 數字 (整數)，至少為 1。

  執行逾時 (以分鐘為單位)。此為執行在停止並進入 `TIMEOUT` 狀態前可以消耗資源的最大時間。預設值為 2,880 分鐘 (48 小時)。

- `AdditionalRunOptions` – DataQualityEvaluationRunAdditionalRunOptions 物件。

  您可以為評估執行指定的其他執行選項。

- `Status` – UTF-8 字串 (有效值：`STARTING` | `RUNNING` | `STOPPING` | `STOPPED` | `SUCCEEDED` | `FAILED` | `TIMEOUT`)。

  此回合的狀態。

- `ErrorString` – UTF-8 字串。

  與任務執行相關聯的錯誤字串。

- `StartedOn` – 時間戳記。

  此執行開始時的日期和時間。

- `LastModifiedOn` – 時間戳記。

  時間戳記。修改此資料品質規則建議執行的最後一個時間點。

- `CompletedOn` – 時間戳記。

  此任務執行完成的日期和時間。

- `ExecutionTime` – 數字 (整數)。

  執行消耗資源所需的時間 (以秒為單位)。

- `RulesetNames` – UTF-8 字串的陣列，不可小於 1，也不可超過 10 個字串。

  執行規則集名稱清單。此參數目前僅採用一個規則集名稱。

- `ResultIds` – UTF-8 字串的陣列，不可小於 1，也不可超過 10 個字串。

執行的資料品質結果的結果 ID 清單。

- `AdditionalDataSources` – 金鑰值對的映射陣列。

  每個金鑰都是 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  每個值都是 DataSource 物件。

  您可以為評估執行指定的其他資料來源的參考字串映射。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

ListDataQualityRulesetEvaluationRuns 操作（Python：列表數據質量規則評估運行）

列出符合篩選條件的所有執行，即針對資料來源評估規則集的執行。

請求

- `Filter` – DataQualityRulesetEvaluationRunFilter 物件。

  篩選條件。

- `NextToken` – UTF-8 字串。

  位移結果的分頁字符。

- `MaxResults` – 數字 (整數)，不可小於 1，也不可以大於 1000。

  回傳結果的數量上限。

回應

- `Runs` – 一個 DataQualityRulesetEvaluationRunDescription 物件陣列。

代表資料品質規則集執行的 DataQualityRulesetEvaluationRunDescription 物件清單。

- NextToken – UTF-8 字串。

  如有多個結果可用，即為分頁字符。

錯誤

- InvalidInputException
- OperationTimeoutException
- InternalServiceException

StartDataQualityRuleRecommendationRun 操作（Python：開始數據質量規則推薦運行）

啟動建議執行，當您不知道要撰寫哪些規則時，用來產生規則。 AWS Glue 資料品質會分析資料，並提出潛在規則集的建議。然後，您可以對規則集進行分類，並根據自己的喜好修改生成的規則集。

系統會在 90 天後自動刪除建議執行。

請求

- DataSource – 必要：DataSource 物件。

  與此執行相關聯的資料來源 (AWS Glue 表格)。

- Role – 必要：UTF-8 字串。

  提供 IAM 用來加密執行結果的角色。

- NumberOfWorkers – 數字 (整數)。

  在執行中使用的 G.1X 工作者數目。預設值為 5。

- Timeout – 數字 (整數)，至少為 1。

  執行逾時 (以分鐘為單位)。此為執行在停止並進入 TIMEOUT 狀態前可以消耗資源的最大時間。預設值為 2,880 分鐘 (48 小時)。

- CreatedRulesetName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

規則集的名稱。

- ClientToken – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於等冪性且建議將其設定為隨機 ID（例如 UUID），避免建立或啟動同一資源的多個執行個體。

回應

- RunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與此回合相關聯的唯一回合識別符。

錯誤

- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ConflictException

CancelDataQualityRuleRecommendationRun 操作（Python：取消數據質量規則推薦運行）

取消用於產生規則的指定建議執行。

請求

- RunId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  與此回合相關聯的唯一回合識別符。

回應

- 無回應參數。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

GetDataQualityRuleRecommendationRun 操作（Python：獲取數據質量規則推薦運行）

取得用來產生規則的指定建議執行。

請求

- `RunId` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  與此回合相關聯的唯一回合識別符。

回應

- `RunId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與此回合相關聯的唯一回合識別符。
- `DataSource` – DataSource 物件。

  與此執行相關聯的資料來源 (資料 AWS Glue 表)。
- `Role` – UTF-8 字串。

  提供 IAM 用來加密執行結果的角色。
- `NumberOfWorkers` – 數字 (整數)。

  在執行中使用的 `G.1X` 工作者數目。預設值為 5。
- `Timeout` – 數字 (整數)，至少為 1。

  執行逾時 (以分鐘為單位)。此為執行在停止並進入 `TIMEOUT` 狀態前可以消耗資源的最大時間。預設值為 2,880 分鐘 (48 小時)。

- Status – UTF-8 字串 (有效值：STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT)。

  此回合的狀態。

- ErrorString – UTF-8 字串。

  與任務執行相關聯的錯誤字串。

- StartedOn – 時間戳記。

  此執行開始時的日期和時間。

- LastModifiedOn – 時間戳記。

  時間戳記。修改此資料品質規則建議執行的最後一個時間點。

- CompletedOn – 時間戳記。

  此任務執行完成的日期和時間。

- ExecutionTime – 數字 (整數)。

  執行消耗資源所需的時間 (以秒為單位)。

- RecommendedRuleset – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 65536 個位元組。

  當啟動規則建議執行完成時，會建立建議的規則集 (一組規則)。此成員具有資料品質定義語言 (DQDL) 格式的規則。

- CreatedRulesetName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  執行建立的規則集的名稱。

錯誤

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

ListDataQualityRuleRecommendationRuns 操作（Python：列表數據質量規則推薦運行）

列出符合篩選條件的建議執行。

請求

- `Filter` – DataQualityRuleRecommendationRunFilter 物件。

  篩選條件。

- `NextToken` – UTF-8 字串。

  位移結果的分頁字符。

- `MaxResults` – 數字 (整數)，不可小於 1，也不可以大於 1000。

  回傳結果的數量上限。

回應

- `Runs` – 一個 DataQualityRuleRecommendationRunDescription 物件陣列。

  `DataQualityRuleRecommendationRunDescription` 物件的清單。

- `NextToken` – UTF-8 字串。

  如有多個結果可用，即為分頁字符。

錯誤

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

GetDataQualityResult 操作（Python：獲取數據質量結果）

擷取資料品質規則評估的結果。

請求

- ResultId – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料品質結果的唯一結果 ID。

回應

- ResultId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料品質結果的唯一結果 ID。
- Score – 數字 (double)，不可大於 1.0。

  彙總資料品質分數。表示通過的規則數目與規則總數的比率。
- DataSource – DataSource 物件。

  與資料品質結果相關聯的資料表 (若有)。
- RulesetName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與資料品質結果相關聯的規則集名稱。
- EvaluationContext – UTF-8 字串。

  在 AWS Glue Studio 中的工作環境中，畫布中的每個節點通常被分配某種名稱，數據質量節點將具有名稱。如果存在多個節點，evaluationContext 可以區分節點。
- StartedOn – 時間戳記。

  此資料品質結果執行開始的日期和時間。
- CompletedOn – 時間戳記。

  此資料品質結果執行完成的日期和時間。
- JobName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與資料品質結果相關聯的任務名稱 (若有)。

- JobRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與資料品質結果相關聯的任務執行 ID (若有)。
- RulesetEvaluationRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  與規則集評估相關聯的唯一執行 ID。
- RuleResults – 一個 DataQualityRuleResult 物件陣列，不可超過 2000 個結構。

  代表每個規則結果的 DataQualityRuleResult 物件清單。
- AnalyzerResults – 一個 DataQualityAnalyzerResult 物件陣列，不可超過 2000 個結構。

  代表每個分析器結果的 DataQualityAnalyzerResult 物件清單。
- Observations – DataQualityObservation 物件陣列，不可超過 50 個結構。

  代表評估規則和分析器後產生的觀測值的 DataQualityObservation 物件清單。

錯誤

- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- EntityNotFoundException

BatchGetDataQualityResult 操作（Python：批處理數據質量結果）

擷取指定結果 ID 的資料品質結果清單。

請求

- ResultIds – 必要：UTF-8 字串的陣列，不可小於 1，也不可超過 100 個字串。

  資料品質結果的唯一結果 ID 清單。

回應

- Results – 必要：一個 DataQualityResult 物件。

表示資料品質結果的 DataQualityResult 物件清單。

- ResultsNotFound – UTF-8 字串的陣列，不可小於 1，也不可超過 100 個字串。

  找不到結果的結果 ID 清單。

錯誤

- InvalidInputException
- OperationTimeoutException
- InternalServiceException

ListDataQualityResults 操作（Python：列表數據質量結果）

傳回您帳戶的所有資料品質執行結果。

請求

- Filter – DataQualityResultFilterCriteria 物件。

  篩選條件。

- NextToken – UTF-8 字串。

  位移結果的分頁字符。

- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  回傳結果的數量上限。

回應

- Results – 必要：一個 DataQualityResultDescription 物件。

  DataQualityResultDescription 物件的清單。

- NextToken – UTF-8 字串。

  如有多個結果可用，即為分頁字符。

錯誤

- InvalidInputException
- OperationTimeoutException
- InternalServiceException

CreateDataQualityRuleset 動作 (Python: 建立品質規則集)

建立套用至指定表格的 DQDL 規則的資料品質規則集。 AWS Glue

您可以使用資料品質定義語言 (DQDL) 建立規則集。如需詳細資訊，請參閱開 AWS Glue 發人員指南。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料品質規則集的唯一名稱。
- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  資料品質規則集的描述。
- Ruleset – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 65536 個位元組。

  資料品質定義語言 (DQDL) 規則集。如需詳細資訊，請參閱開 AWS Glue 發人員指南。
- Tags – 金鑰值對的映射陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  套用至資料品質規則集的標籤清單。
- TargetTable – DataQualityTargetTable 物件。

  與資料品質規則集關聯的目標資料表。
- RecommendationRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

建議執行的唯一執行 ID。

- ClientToken – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  用於等冪性且建議將其設定為隨機 ID（例如 UUID），避免建立或啟動同一資源的多個執行個體。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  資料品質規則集的唯一名稱。

錯誤

- InvalidInputException
- AlreadyExistsException
- OperationTimeoutException
- InternalServiceException
- ResourceNumberLimitExceededException

DeleteDataQualityRuleset 動作 (Python: 刪除資料品質規則集)

刪除資料品質規則集。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料品質規則集的名稱。

回應

- 無回應參數。

錯誤

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

GetDataQualityRuleset 行動（Python：獲取質量規則集）

透過識別符或名稱傳回現有的規則集。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  規則集的名稱。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  規則集的名稱。
- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  規則群組的描述。
- Ruleset – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 65536 個位元組。

  資料品質定義語言 (DQDL) 規則集。如需詳細資訊，請參閱開 AWS Glue 發人員指南。
- TargetTable – DataQualityTargetTable 物件。

  目標資料表的名稱和資料庫名稱。
- CreatedOn – 時間戳記。

  時間戳記。建立此資料品質規則集的時間和日期。

- `LastModifiedOn` – 時間戳記。

  時間戳記。修改此資料品質規則集的最後一個時間點。

- `RecommendationRunId` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  從建議執行建立規則集時，會產生此執行 ID 以將兩者連結在一起。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

ListDataQualityRulesets 操作（Python：列表數據質量規則集）

傳回指定表格清單的規則集分頁清單。 AWS Glue

請求

- `NextToken` – UTF-8 字串。

  位移結果的分頁字符。

- `MaxResults` – 數字 (整數)，不可小於 1，也不可以大於 1000。

  回傳結果的數量上限。

- `Filter` – DataQualityRulesetFilterCriteria 物件。

  篩選條件。

- `Tags` – 金鑰值對的映射陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  鍵/值對標籤清單。

回應

- `Rulesets` – 一個 DataQualityRulesetListDetails 物件陣列。

  指定表格清單中規則集的分頁清單。 AWS Glue
- `NextToken` – UTF-8 字串。

  如有多個結果可用，即為分頁字符。

錯誤

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

UpdateDataQualityRuleset 行動（Python：更新數據質量規則集）

更新指定的資料品質規則集。

請求

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  資料品質規則集的名稱。
- `Description` – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  規則群組的描述。
- `Ruleset` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 65536 個位元組。

  資料品質定義語言 (DQDL) 規則集。如需詳細資訊，請參閱開 AWS Glue 發人員指南。

回應

- `Name` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

資料品質規則集的名稱。

- Description – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  規則群組的描述。

- Ruleset – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 65536 個位元組。

  資料品質定義語言 (DQDL) 規則集。如需詳細資訊，請參閱開 AWS Glue 發人員指南。

錯誤

- EntityNotFoundException
- AlreadyExistsException
- IdempotentParameterMismatchException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ResourceNumberLimitExceededException

敏感資料偵測 API

敏感資料偵測 API 說明用於跨結構化資料的資料行和資料列偵測敏感資料的 API。

資料類型

- CustomEntityType 結構

CustomEntityType 結構

表示自訂模式的物件，可用於跨結構化資料的資料欄和資料列偵測敏感資料。

欄位

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

允許稍後擷取或刪除的自訂模式的名稱。此名稱在每個 AWS 帳戶中必須是唯一的。

- `RegexString` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  用於偵測自訂模式中敏感資料的規則運算式字串。

- `ContextWords` – UTF-8 字串的陣列，不可小於 1 或超過 20 個字串。

  內容文字清單。如果在規則運算式範疇內沒有找到這些內容文字，則不會將資料偵測為敏感資料。

  如果沒有內容文字，則只會檢查規則運算式。

操作

- CreateCustomEntityType 動作 (Python: create_custom_entity_type)
- DeleteCustomEntityType 動作 (Python: delete_custom_entity_type)
- GetCustomEntityType 動作 (Python: get_custom_entity_type)
- BatchGetCustomEntityTypes 動作 (Python: batch_get_custom_entity_types)
- ListCustomEntityTypes 動作 (Python: list_custom_entity_types)

CreateCustomEntityType 動作 (Python: create_custom_entity_type)

建立用於跨結構化資料的資料欄和資料列偵測敏感資料的自訂模式。

您建立的每個自訂模式都會指定一個規則運算式和一個可選的內容文字清單。如果沒有內容文字，則只會檢查規則運算式。

請求

- `Name` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  允許稍後擷取或刪除的自訂模式的名稱。此名稱在每個 AWS 帳戶中必須是唯一的。

- `RegexString` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  用於偵測自訂模式中敏感資料的規則運算式字串。

- ContextWords – UTF-8 字串的陣列，不可小於 1 或超過 20 個字串。

  內容文字清單。如果在規則運算式範疇內沒有找到這些內容文字，則不會將資料偵測為敏感資料。

  如果沒有內容文字，則只會檢查規則運算式。
- Tags – 金鑰值對的映射陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  套用至自訂實體類型的標籤清單。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  您建立的自訂模式的名稱。

錯誤

- AccessDeniedException
- AlreadyExistsException
- IdempotentParameterMismatchException
- InternalServiceException
- InvalidInputException
- OperationTimeoutException
- ResourceNumberLimitExceededException

DeleteCustomEntityType 動作 (Python: delete_custom_entity_type)

透過指定自訂模式的名稱來將其刪除。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

您要刪除的自訂模式的名稱。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  您刪除的自訂模式的名稱。

錯誤

- `EntityNotFoundException`
- `AccessDeniedException`
- `InternalServiceException`
- `InvalidInputException`
- `OperationTimeoutException`

GetCustomEntityType 動作 (Python: get_custom_entity_type)

透過指定自訂模式的名稱來擷取其詳細資訊。

請求

- Name – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  您要擷取的自訂模式的名稱。

回應

- Name – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  您擷取的自訂模式的名稱。
- `RegexString` – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

用於偵測自訂模式中敏感資料的規則運算式字串。

- `ContextWords` – UTF-8 字串的陣列，不可小於 1 或超過 20 個字串。

  建立自訂模式時指定的內容文字清單。如果在規則運算式範疇內沒有找到這些內容文字，則不會將資料偵測為敏感資料。

錯誤

- `EntityNotFoundException`
- `AccessDeniedException`
- `InternalServiceException`
- `InvalidInputException`
- `OperationTimeoutException`

BatchGetCustomEntityTypes 動作 (Python: batch_get_custom_entity_types)

擷取由名稱清單指定的自訂模式的詳細資訊。

請求

- `Names` – 必要：UTF-8 字串的陣列，不可小於 1 或超過 50 個字串。

  您要擷取的自訂模式的名稱清單。

回應

- `CustomEntityTypes` – 一個 CustomEntityType 物件陣列。

  表示已建立的自訂模式的 `CustomEntityType` 物件清單。
- `CustomEntityTypesNotFound` – UTF-8 字串的陣列，不可小於 1 或超過 50 個字串。

  未找到的自訂模式的名稱清單。

錯誤

- `InvalidInputException`

- InternalServiceException
- OperationTimeoutException

ListCustomEntityTypes 動作 (Python: list_custom_entity_types)

列出已建立的所有自訂模式。

請求

- NextToken – UTF-8 字串。

  位移結果的分頁字符。

- MaxResults – 數字 (整數)，不可小於 1，也不可以大於 1000。

  回傳結果的數量上限。

- Tags – 金鑰值對的映射陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  鍵/值對標籤清單。

回應

- CustomEntityTypes – 一個 CustomEntityType 物件陣列。

  表示自訂模式的 CustomEntityType 物件清單。

- NextToken – UTF-8 字串。

  如有多個結果可用，即為分頁字符。

錯誤

- InvalidInputException
- OperationTimeoutException
- InternalServiceException

在中標記 API AWS Glue

資料類型

- Tag 結構

Tag 結構

該Tag對象表示您可以分配給 AWS 資源的標籤。每個標籤皆包含由您定義的一個金鑰與一個選用值。

如需有關標籤和控制資源存取的詳細資訊 AWS Glue，請參閱開發人員指南中的AWS 標籤 AWS Glue和指定 AWS Glue 資源 ARN。

欄位

- key – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  標籤金鑰。如果要在物件上建立標籤，您必須擁有金鑰。這份金鑰會區分大小寫，而且前綴不可為 aws。

- value – UTF-8 字串，長度不可超過 256 個位元組。

  標籤值。如果要在物件上建立標籤，這個值為選用性。這個值區分大小寫，而且前綴不可為 aws。

作業

- TagResource 行動（Python：標籤資源）
- UntagResource 行動（Python：取消標籤資源）
- GetTags 行動（Python：獲取 _ 標籤）

TagResource 行動（Python：標籤資源）

為資源加上標籤。標籤是您可以指派給 AWS 資源的標籤。在中 AWS Glue，您只能標記某些資源。如需哪些資源可供標記的資訊，請參閱 AWS Glue中的AWS 標籤。

除了呼叫標籤相關 API 的標記許可外，您還需要呼叫連線上標記 API 的 glue:GetConnection 許可，以及呼叫資料庫上標記 API 的 glue:GetDatabase 許可。

請求

- ResourceArn – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合Custom string pattern #22。

  要向其中新增標籤的 AWS Glue 資源 ARN。如需有關資 AWS Glue 源 ARN 的詳細資訊，請參閱 AWS Glue ARN 字串模式。

- TagsToAdd – 必要：索引鍵/值對的對應陣列，不可超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  要新增至此資源的標籤。

回應

- 無回應參數。

錯誤

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- EntityNotFoundException

UntagResource 行動（Python：取消標籤資源）

從資源移除標籤。

請求

- ResourceArn – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合Custom string pattern #22。

  要從中移除這些標籤之資源的 Amazon Resource Name (ARN)。

- TagsToRemove – 必要：UTF-8 字串的陣列，不可超過 50 個字串。

要從此資源移除的標籤。

回應

- 無回應參數。

錯誤

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `EntityNotFoundException`

GetTags 行動（Python：獲取 _ 標籤）

擷取與資源關聯之標籤的清單。

請求

- `ResourceArn` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 10240 個位元組，且需符合Custom string pattern #22。

  要從中擷取這些標籤之資源的 Amazon Resource Name (ARN)。

回應

- `Tags` – 金鑰值對的映射陣列，不超過 50 對。

  每個金鑰均為 UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  每個值都是 UTF-8 字串，長度不可超過 256 個位元組。

  要求的標籤。

錯誤

- `InvalidInputException`

- InternalServiceException
- OperationTimeoutException
- EntityNotFoundException

常見資料類型

常見資料類型說明 AWS Glue中的其他常見資料類型。

Tag 結構

該Tag對象表示您可以分配給 AWS 資源的標籤。每個標籤皆包含由您定義的一個金鑰與一個選用值。

如需有關標籤和控制資源存取的詳細資訊 AWS Glue，請參閱開發人員指南中的AWS 標籤 AWS Glue和指定 AWS Glue 資源 ARN。

欄位

- key – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 128 個位元組。

  標籤金鑰。如果要在物件上建立標籤，您必須擁有金鑰。這份金鑰會區分大小寫，而且前綴不可為 aws。
- value – UTF-8 字串，長度不可超過 256 個位元組。

  標籤值。如果要在物件上建立標籤，這個值為選用性。這個值區分大小寫，而且前綴不可為 aws。

DecimalNumber 結構

包含十進制格式的數值。

欄位

- UnscaledValue – 必要：Blob。

  沒有單位的數值。
- Scale – 必要：數字 (整數)。

  決定無刻度值內小數點位置的刻度。

ErrorDetail 結構

包含錯誤的詳細資訊。

欄位

- ErrorCode – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  此錯誤相關的程式碼。
- ErrorMessage – 描述字串，長度不可超過 2048 個位元組，需符合URI address multi-line string pattern。

  描述錯誤的訊息。

PropertyPredicate 結構

定義屬性述詞。

欄位

- Key – 值字串，長度不可超過 1024 個位元組。

  屬性的金鑰。
- Value – 值字串，長度不可超過 1024 個位元組。

  屬性的值。
- Comparator – UTF-8 字串 (有效值：EQUALS | GREATER_THAN | LESS_THAN | GREATER_THAN_EQUALS | LESS_THAN_EQUALS)。

  用於將這個屬性與其他屬性比較的比較程式。

ResourceUri 結構

函數資源的 URI。

欄位

- ResourceType – UTF-8 字串 (有效值：JAR | FILE | ARCHIVE)。

資源的類型。

- `Uri` – 統一資源識別符 (uri)，長度不可小於 1 個位元組，也不可以超過 1024 個位元組，需符合URI address multi-line string pattern。

  存取資源的 URI。

ColumnStatistics 結構

代表資料表或分割區產生的欄層級統計資料。

欄位

- `ColumnName` – 必要：UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，且需符合Single-line string pattern。

  統計資料所屬的欄名稱。

- `ColumnType` – 必要：輸入名稱，長度不可超過 20000 個位元組，需符合 Single-line string pattern。

  欄的資料類型。

- `AnalyzedTime` – 必要：時間戳記。

  欄統計資料產生時的時間戳記。

- `StatisticsData` – 必要：ColumnStatisticsData 物件。

  `ColumnStatisticData` 物件，其中包含統計資料值。

ColumnStatisticsError 結構

封裝 `ColumnStatistics` 物件以及失敗原因的詳細資訊。

欄位

- `ColumnStatistics` – ColumnStatistics 物件。

  欄的 `ColumnStatistics`。

- `Error` – ErrorDetail 物件。

顯示作業失敗原因的錯誤訊息。

ColumnError 結構

封裝失敗的欄名稱和失敗原因。

欄位

- ColumnName – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  失敗欄的名稱。

- Error – ErrorDetail 物件。

  顯示作業失敗原因的錯誤訊息。

ColumnStatisticsData 結構

包含個別類型的欄統計資料。只有一個資料物件應該由 Type 屬性設定與指定。

欄位

- Type – 必要：UTF-8 字串 (有效值：BOOLEAN | DATE | DECIMAL | DOUBLE | LONG | STRING | BINARY).

  欄統計資料的類型。

- BooleanColumnStatisticsData – BooleanColumnStatisticsData 物件。

  布林資料欄統計資料。

- DateColumnStatisticsData – DateColumnStatisticsData 物件。

  日期欄統計資料。

- DecimalColumnStatisticsData – DecimalColumnStatisticsData 物件。

  十進制列統計數據。 UnscaledValues 其中是 Base64 編碼的二進制對象，存儲大端，這是十進制未縮放值的二進制補碼表示。

- DoubleColumnStatisticsData – DoubleColumnStatisticsData 物件。

雙欄統計資料。

- LongColumnStatisticsData – LongColumnStatisticsData 物件。

  長欄統計資料。

- StringColumnStatisticsData – StringColumnStatisticsData 物件。

  字串欄統計資料。

- BinaryColumnStatisticsData – BinaryColumnStatisticsData 物件。

  二進位資料行統計資料。

BooleanColumnStatisticsData 結構

定義布林資料欄支援的欄統計資料。

欄位

- NumberOfTrues – 必要：數字 (long)，不可大於 None (無)。

  欄中的 true 值數目。

- NumberOfFalses – 必要：數字 (long)，不可大於 None (無)。

  欄中的 false 值數目。

- NumberOfNulls – 必要：數字 (long)，不可大於 None (無)。

  欄中的 null 值數目。

DateColumnStatisticsData 結構

定義時間戳記資料欄支援的欄統計資料。

欄位

- MinimumValue – 時間戳記。

  欄中的最低值。

- MaximumValue – 時間戳記。

欄中的最高值。

- NumberOfNulls – 必要：數字 (long)，不可大於 None (無)。

  欄中的 null 值數目。

- NumberOfDistinctValues – 必要：數字 (long)，不可大於 None (無)。

  欄中相異值的數目。

DecimalColumnStatisticsData 結構

定義固定點數目資料欄支援的欄統計資料。

欄位

- MinimumValue – DecimalNumber 物件。

  欄中的最低值。

- MaximumValue – DecimalNumber 物件。

  欄中的最高值。

- NumberOfNulls – 必要：數字 (long)，不可大於 None (無)。

  欄中的 null 值數目。

- NumberOfDistinctValues – 必要：數字 (long)，不可大於 None (無)。

  欄中相異值的數目。

DoubleColumnStatisticsData 結構

定義浮點數資料欄支援的欄統計資料。

欄位

- MinimumValue – 數字 (雙位數)。

  欄中的最低值。

- MaximumValue – 數字 (雙位數)。

欄中的最高值。

- NumberOfNulls – 必要：數字 (long)，不可大於 None (無)。

  欄中的 null 值數目。

- NumberOfDistinctValues – 必要：數字 (long)，不可大於 None (無)。

  欄中相異值的數目。

LongColumnStatisticsData 結構

定義整數資料欄支援的欄統計資料。

欄位

- MinimumValue – 數字 (long)。

  欄中的最低值。

- MaximumValue – 數字 (long)。

  欄中的最高值。

- NumberOfNulls – 必要：數字 (long)，不可大於 None (無)。

  欄中的 null 值數目。

- NumberOfDistinctValues – 必要：數字 (long)，不可大於 None (無)。

  欄中相異值的數目。

StringColumnStatisticsData 結構

定義字元序列資料值支援的欄統計資料。

欄位

- MaximumLength – 必要：數字 (long)，不可大於 None (無)。

  欄中最長字串的大小。

- AverageLength – 必要：數字 (double)，不可大於 None (無)。

欄中的平均字串長度。

- NumberOfNulls – 必要：數字 (long)，不可大於 None (無)。

  欄中的 null 值數目。

- NumberOfDistinctValues – 必要：數字 (long)，不可大於 None (無)。

  欄中相異值的數目。

BinaryColumnStatisticsData 結構

定義位元序列資料值支援的欄統計資料。

欄位

- MaximumLength – 必要：數字 (long)，不可大於 None (無)。

  欄中最長位元序列的大小。

- AverageLength – 必要：數字 (double)，不可大於 None (無)。

  欄中的平均位元序列長度。

- NumberOfNulls – 必要：數字 (long)，不可大於 None (無)。

  欄中的 null 值數目。

字串模式

API 使用以下常規表達式來定義適用於各種字串參數和成員的有效內容：

- 單行字串模式 –「[\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\t]*」
- URI 位址多行字串模式 –「[\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\r\n\t]*」
- Logstash Grok 字串模式 –「[\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\r\t]*」
- 識別符字串模式 –「[A-Za-z_][A-Za-z0-9_]*」
- AWS IAM ARN 字串模式 –「arn:aws:iam::\d{12}:role/.*」
- 版本字串模式 –「^[a-zA-Z0-9-_]+$」

- 日誌群組字串模式 –「[\.\-_/#A-Za-z0-9]+」
- 日誌串流字串模式 –「[^:*]*」
- 自訂字串模式 #10 – "[^\r\n]"
- 自訂字串模式 #11 – "^arn:aws(-(cn|us-gov|iso(-[bef])?))?:secretsmanager:.*$"
- 自訂字串模式 #12 – "^(https?)://[-a-zA-Z0-9+@#/%?=~_|!:,.;]*[-a-zA-Z0-9+@#/%=~_|]"
- 自訂字串模式 #13 – "\S+"
- 自訂字串模式 #14 – "^(https?):\/\/[^\s/$.?#].[^\s]*$"
- 自訂字串模式 #15 – "^subnet-[a-z0-9]+$"
- 自訂字串模式 #16 – "[\p{L}\p{N}\p{P}]*"
- 自訂字串模式 #17 – "[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"
- 自訂字串模式 #18 – "[a-zA-Z0-9-_$#.]+"
- 自訂字串模式 #19 – "^\w+\.\w+\.\w+$"
- 自訂字串模式 #20 – "^\w+\.\w+$"
- 自訂字串模式 #21 – "^([2-3]|3[.]9)$"
- 自訂字串模式 #22 – "arn:(aws|aws-us-gov|aws-cn):glue:.*"
- 自訂字串模式 #23 – "(^arn:aws:iam::\w{12}:root)"
- 自訂字串模式 #24 – "^arn:aws(-(cn|us-gov|iso(-[bef])?))?:iam::[0-9]{12}:role/.+"
- 自訂字串模式 #25 – "arn:aws:kms:.*"
- 自訂字串模式 #26 – "arn:aws[^:]*:iam::[0-9]*:role/.+"
- 自訂字串模式 #27 – "[\.\-_A-Za-z0-9]+"
- 自訂字串模式 #28 – "^s3://([^/]+)/([^/]+/)*([^/]+)$"
- 自訂字串模式 #29 – ".*"
- 自訂字串模式 #30 – "^(Sun|Mon|Tue|Wed|Thu|Fri|Sat):([01]?[0-9]|2[0-3])$"
- 自訂字串模式 #31 – "[a-zA-Z0-9_.-]+"
- 自訂字串模式 #32 – ".*\S.*"
- 自訂字串模式 #33 – "[a-zA-Z0-9-=._/@]+"
- 自訂字串模式 #34 – "[1-9][0-9]*|[1-9][0-9]*-[1-9][0-9]*"
- 自訂字串模式 #35 – "[\s\S]*"

- 自訂字串模式 #36 — "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF]|[^\S\r\n"'= ;])*」
- 自訂字串模式 #37 — "[*A-Za-z0-9_-]*」
- 自訂字串模式 #38 — "([\u0020-\u007E\r\s\n])*」
- 自訂字串模式 #39 — "[A-Za-z0-9_-]*」
- 自訂字串模式 #40 — "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF]|[^\S\r\n"'])*」
- 自訂字串模式 #41 — "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF]|[^\S\r\n])*」
- 自訂字串模式 #42 — "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\s])*」
- 自訂字串模式 #43 — "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF]|[^\r\n])*」

例外狀況

本節說明可用來尋找問題來源並加以修正的 AWS Glue 例外狀況。如需機器學習相關例外狀況的 HTTP 錯誤碼和字串的詳細資訊，請參閱the section called “AWS Glue 機器學習例外狀況”。

AccessDeniedException 結構

存取資源遭拒。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

AlreadyExistsException 結構

要建立或新增的資源已存在。

欄位

- Message – UTF-8 字串。

說明問題的訊息。

ConcurrentModificationException 結構

同時有兩個程序嘗試修改資源。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

ConcurrentRunsExceededException 結構

正在同時執行太多任務。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

CrawlerNotRunningException 結構

指定的爬蟲程式未執行。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

CrawlerRunningException 結構

此操作無法執行，因為爬蟲程式已在執行中。

欄位

- Message – UTF-8 字串。

說明問題的訊息。

CrawlerStoppingException 結構

指定的爬蟲程式正在停用。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

EntityNotFoundException 結構

指定的實體不存在

欄位

- Message – UTF-8 字串。

  說明問題的訊息。
- FromFederationSource – 布林值。

  指出例外狀況是否與聯合來源相關。

FederationSourceException 結構

聯合來源失敗。

欄位

- FederationSourceErrorCode— UTF-8 字串 (有效值：AccessDeniedExceptionEntityNotFoundException| InvalidCredentialsException | InvalidInputException | InvalidResponseException | OperationTimeoutException | OperationNotSupportedException | InternalServiceException | PartialFailureException |ThrottlingException)。

  問題的錯誤碼。

- Message – UTF-8 字串。

  說明問題的訊息。

FederationSourceRetryableException 結構

聯合來源失敗，但可重試操作。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

GlueEncryptionException 結構

加密操作失敗。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

IdempotentParameterMismatchException 結構

相同的唯一識別符與兩個不同的記錄關聯。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

IllegalWorkflowStateException 結構

工作流程處於無效狀態，因而無法執行請求的操作。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

InternalServiceException 結構

發生內部服務錯誤。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

InvalidExecutionEngineException 結構

指定的執行引擎不明或無效。

欄位

- message – UTF-8 字串。

  說明問題的訊息。

InvalidInputException 結構

提供的輸入無效。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。
- FromFederationSource – 布林值。

  指出例外狀況是否與聯合來源相關。

InvalidStateException 結構

表示資料為無效狀態的錯誤。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

InvalidTaskStatusTransitionException 結構

從一個任務適當轉換到下一個任務失敗。

欄位

- message – UTF-8 字串。

  說明問題的訊息。

JobDefinitionErrorException 結構

無效的任務定義。

欄位

- message – UTF-8 字串。

  說明問題的訊息。

JobRunInTerminalStateException 結構

任務執行的結束狀態發出失敗訊號。

欄位

- message – UTF-8 字串。

  說明問題的訊息。

JobRunInvalidStateTransitionException 結構

任務執行從來源狀態轉換到目標狀態無效。

欄位

- jobRunId – UTF-8 字串，長度不可小於 1 個位元組，也不可以超過 255 個位元組，需符合Single-line string pattern。

  任務執行有問題的 ID。

- message – UTF-8 字串。

  說明問題的訊息。

- sourceState— UTF-8 字串 (有效值：STARTINGRUNNING| STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT | ERROR | WAITING |EXPIRED)。

  來源狀態。

- targetState— UTF-8 字串 (有效值：STARTINGRUNNING| STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT | ERROR | WAITING |EXPIRED)。

  目標狀態。

JobRunNotInTerminalStateException 結構

任務執行未處於結束狀態。

欄位

- message – UTF-8 字串。

  說明問題的訊息。

LateRunnerException 結構

任務執行器延遲。

欄位

- Message – UTF-8 字串。

說明問題的訊息。

NoScheduleException 結構

沒有適用的排程。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

OperationTimeoutException 結構

此操作逾時。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

ResourceNotReadyException 結構

資源尚未準備好進行交易。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

ResourceNumberLimitExceededException 結構

資源數值超出限制。

欄位

- Message – UTF-8 字串。

說明問題的訊息。

SchedulerNotRunningException 結構

指定的排程器未執行。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

SchedulerRunningException 結構

指定的排程器已在執行中。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

SchedulerTransitioningException 結構

指定的排程器正在轉換。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

UnrecognizedRunnerException 結構

任務執行器無法辨識。

欄位

- Message – UTF-8 字串。

說明問題的訊息。

ValidationException 結構

無法驗證值。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

VersionMismatchException 結構

發生版本衝突。

欄位

- Message – UTF-8 字串。

  說明問題的訊息。

AWS Glue 使用 AWS SDK 的 API 程式碼範例

下列程式碼範例顯示如何搭 AWS Glue 配 AWS 軟體開發套件 (SDK) 使用。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境和跨服務範例中查看內容中的動作。

Scenarios (案例) 是向您展示如何呼叫相同服務中的多個函數來完成特定任務的程式碼範例。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含入門相關資訊和舊版 SDK 的詳細資訊。

開始使用

你好 AWS Glue

下列程式碼範例示範如何開始使用 AWS Glue。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
namespace GlueActions;

public class HelloGlue
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Glue.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
```

```
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
 LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonGlue>()
                .AddTransient<GlueWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloGlue>();
        var glueClient = host.Services.GetRequiredService<IAmazonGlue>();

        var request = new ListJobsRequest();

        var jobNames = new List<string>();

        do
        {
            var response = await glueClient.ListJobsAsync(request);
            jobNames.AddRange(response.JobNames);
            request.NextToken = response.NextToken;
        }
        while (request.NextToken is not null);

        Console.Clear();
        Console.WriteLine("Hello, Glue. Let's list your existing Glue Jobs:");
        if (jobNames.Count == 0)
        {
            Console.WriteLine("You don't have any AWS Glue jobs.");
        }
        else
        {
            jobNames.ForEach(Console.WriteLine);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考ListJobs中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

C MakeLists .txt 的 CMake 文件的代碼。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS glue)

# Set this project's name.
project("hello_glue")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.
```

```
    # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                    # and set the proper subdirectory to the
 executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_glue.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_glue.cpp 來源檔案的程式碼。

```
#include <aws/core/Aws.h>
#include <aws/glue/GlueClient.h>
#include <aws/glue/model/ListJobsRequest.h>
#include <iostream>

/*
 *  A "Hello Glue" starter application which initializes an AWS Glue client and
 lists the
 *  AWS Glue job definitions.
 *
 *  main function
 *
 *  Usage: 'hello_glue'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
```

```
        // clientConfig.region = "us-east-1";

        Aws::Glue::GlueClient glueClient(clientConfig);

        std::vector<Aws::String> jobs;

        Aws::String nextToken;  // Used for pagination.
        do {
            Aws::Glue::Model::ListJobsRequest listJobsRequest;
            if (!nextToken.empty()) {
                listJobsRequest.SetNextToken(nextToken);
            }

            Aws::Glue::Model::ListJobsOutcome listRunsOutcome =
 glueClient.ListJobs(
                    listJobsRequest);

            if (listRunsOutcome.IsSuccess()) {
                const std::vector<Aws::String> &jobNames =
 listRunsOutcome.GetResult().GetJobNames();
                jobs.insert(jobs.end(), jobNames.begin(), jobNames.end());

                nextToken = listRunsOutcome.GetResult().GetNextToken();
            } else {
                std::cerr << "Error listing jobs. "
                          << listRunsOutcome.GetError().GetMessage()
                          << std::endl;
                result = 1;
                break;
            }
        } while (!nextToken.empty());

        std::cout << "Your account has " << jobs.size() << " jobs."
                  << std::endl;
        for (size_t i = 0; i < jobs.size(); ++i) {
            std::cout << "   " << i + 1 << ". " << jobs[i] << std::endl;
        }
    }
    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考ListJobs中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
package com.example.glue;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.glue.GlueClient;
import software.amazon.awssdk.services.glue.model.ListJobsRequest;
import software.amazon.awssdk.services.glue.model.ListJobsResponse;
import java.util.List;

public class HelloGlue {
    public static void main(String[] args) {
        GlueClient glueClient = GlueClient.builder()
                .region(Region.US_EAST_1)
                .build();

        listJobs(glueClient);
    }

    public static void listJobs(GlueClient glueClient) {
        ListJobsRequest request = ListJobsRequest.builder()
                .maxResults(10)
                .build();
        ListJobsResponse response = glueClient.listJobs(request);
        List<String> jobList = response.jobNames();
        jobList.forEach(job -> {
            System.out.println("Job Name: " + job);
        });
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考ListJobs中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import { ListJobsCommand, GlueClient } from "@aws-sdk/client-glue";

const client = new GlueClient({});

export const main = async () => {
  const command = new ListJobsCommand({});

  const { JobNames } = await client.send(command);
  const formattedJobNames = JobNames.join("\n");
  console.log("Job names: ");
  console.log(formattedJobNames);
  return JobNames;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考ListJobs中的。

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        let mut list_jobs = glue.list_jobs().into_paginator().send();
        while let Some(list_jobs_output) = list_jobs.next().await {
            match list_jobs_output {
```

```
                Ok(list_jobs) => {
                    let names = list_jobs.job_names();
                    info!(?names, "Found these jobs")
                }
                Err(err) => return Err(GlueMvpError::from_glue_sdk(err)),
            }
        }
```

- 如需 API 的詳細資訊，請參閱 AWS SDK ListJobs中的 Rust API 參考資料。

程式碼範例

- AWS Glue 使用 AWS SDK 的動作
  - 搭CreateCrawler配 AWS 開發套件或 CLI 使用
  - 搭CreateJob配 AWS 開發套件或 CLI 使用
  - 搭DeleteCrawler配 AWS 開發套件或 CLI 使用
  - 搭DeleteDatabase配 AWS 開發套件或 CLI 使用
  - 搭DeleteJob配 AWS 開發套件或 CLI 使用
  - 搭DeleteTable配 AWS 開發套件或 CLI 使用
  - 搭GetCrawler配 AWS 開發套件或 CLI 使用
  - 搭GetDatabase配 AWS 開發套件或 CLI 使用
  - 搭GetDatabases配 AWS 開發套件或 CLI 使用
  - 搭GetJob配 AWS 開發套件或 CLI 使用
  - 搭GetJobRun配 AWS 開發套件或 CLI 使用
  - 搭GetJobRuns配 AWS 開發套件或 CLI 使用
  - 搭GetTables配 AWS 開發套件或 CLI 使用
  - 搭ListJobs配 AWS 開發套件或 CLI 使用
  - 搭StartCrawler配 AWS 開發套件或 CLI 使用
  - 搭StartJobRun配 AWS 開發套件或 CLI 使用
- AWS Glue 使用 AWS SDK 的案例
  - 開始使用 SDK 執行 AWS Glue 搜尋器和工作 AWS

AWS Glue 使用 AWS SDK 的動作

下列程式碼範例示範如何使用 AWS SDK 執 AWS Glue 行個別動作。這些摘錄會呼叫 AWS Glue API，是來自必須在內容中執行的大型程式碼摘錄。每個範例都包含一個連結 GitHub，您可以在其中找到設定和執行程式碼的指示。

下列範例僅包含最常使用的動作。如需完整清單，請參閱《AWS Glue API 參考》。

範例

- 搭CreateCrawler配 AWS 開發套件或 CLI 使用
- 搭CreateJob配 AWS 開發套件或 CLI 使用
- 搭DeleteCrawler配 AWS 開發套件或 CLI 使用
- 搭DeleteDatabase配 AWS 開發套件或 CLI 使用
- 搭DeleteJob配 AWS 開發套件或 CLI 使用
- 搭DeleteTable配 AWS 開發套件或 CLI 使用
- 搭GetCrawler配 AWS 開發套件或 CLI 使用
- 搭GetDatabase配 AWS 開發套件或 CLI 使用
- 搭GetDatabases配 AWS 開發套件或 CLI 使用
- 搭GetJob配 AWS 開發套件或 CLI 使用
- 搭GetJobRun配 AWS 開發套件或 CLI 使用
- 搭GetJobRuns配 AWS 開發套件或 CLI 使用
- 搭GetTables配 AWS 開發套件或 CLI 使用
- 搭ListJobs配 AWS 開發套件或 CLI 使用
- 搭StartCrawler配 AWS 開發套件或 CLI 使用
- 搭StartJobRun配 AWS 開發套件或 CLI 使用

搭`CreateCrawler`配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用`CreateCrawler`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
    /// <summary>
    /// Create an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name for the crawler.</param>
    /// <param name="crawlerDescription">A description of the crawler.</param>
    /// <param name="role">The AWS Identity and Access Management (IAM) role to
    /// be assumed by the crawler.</param>
    /// <param name="schedule">The schedule on which the crawler will be
executed.</param>
    /// <param name="s3Path">The path to the Amazon Simple Storage Service
(Amazon S3)
    /// bucket where the Python script has been stored.</param>
    /// <param name="dbName">The name to use for the database that will be
    /// created by the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateCrawlerAsync(
        string crawlerName,
        string crawlerDescription,
        string role,
        string schedule,
        string s3Path,
        string dbName)
    {
        var s3Target = new S3Target
        {
            Path = s3Path,
        };

        var targetList = new List<S3Target>
        {
```

```
            s3Target,
        };

        var targets = new CrawlerTargets
        {
            S3Targets = targetList,
        };

        var crawlerRequest = new CreateCrawlerRequest
        {
            DatabaseName = dbName,
            Name = crawlerName,
            Description = crawlerDescription,
            Targets = targets,
            Role = role,
            Schedule = schedule,
        };

        var response = await _amazonGlue.CreateCrawlerAsync(crawlerRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考CreateCrawler中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);
```

```
        Aws::Glue::Model::S3Target s3Target;
        s3Target.SetPath("s3://crawler-public-us-east-1/flight/2016/csv");
        Aws::Glue::Model::CrawlerTargets crawlerTargets;
        crawlerTargets.AddS3Targets(s3Target);

        Aws::Glue::Model::CreateCrawlerRequest request;
        request.SetTargets(crawlerTargets);
        request.SetName(CRAWLER_NAME);
        request.SetDatabaseName(CRAWLER_DATABASE_NAME);
        request.SetTablePrefix(CRAWLER_DATABASE_PREFIX);
        request.SetRole(roleArn);

        Aws::Glue::Model::CreateCrawlerOutcome outcome =
client.CreateCrawler(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully created the crawler." << std::endl;
        }
        else {
            std::cerr << "Error creating a crawler. " <<
outcome.GetError().GetMessage()
                      << std::endl;
            deleteAssets("", CRAWLER_DATABASE_NAME, "", bucketName,
clientConfig);
            return false;
        }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考CreateCrawler中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.glue.GlueClient;
import software.amazon.awssdk.services.glue.model.CreateCrawlerRequest;
import software.amazon.awssdk.services.glue.model.CrawlerTargets;
import software.amazon.awssdk.services.glue.model.GlueException;
import software.amazon.awssdk.services.glue.model.S3Target;
import java.util.ArrayList;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateCrawler {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <IAM> <s3Path> <cron> <dbName> <crawlerName>

                Where:
                    IAM - The ARN of the IAM role that has AWS Glue and S3
 permissions.\s
                    s3Path - The Amazon Simple Storage Service (Amazon S3) target
 that contains data (for example, CSV data).
                    cron - A cron expression used to specify the schedule  (i.e.,
 cron(15 12 * * ? *).
                    dbName - The database name.\s
                    crawlerName - The name of the crawler.\s
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String iam = args[0];
        String s3Path = args[1];
        String cron = args[2];
        String dbName = args[3];
```

```
        String crawlerName = args[4];
        Region region = Region.US_EAST_1;
        GlueClient glueClient = GlueClient.builder()
                .region(region)
                .build();

        createGlueCrawler(glueClient, iam, s3Path, cron, dbName, crawlerName);
        glueClient.close();
    }

    public static void createGlueCrawler(GlueClient glueClient,
            String iam,
            String s3Path,
            String cron,
            String dbName,
            String crawlerName) {

        try {
            S3Target s3Target = S3Target.builder()
                    .path(s3Path)
                    .build();

            // Add the S3Target to a list.
            List<S3Target> targetList = new ArrayList<>();
            targetList.add(s3Target);

            CrawlerTargets targets = CrawlerTargets.builder()
                    .s3Targets(targetList)
                    .build();

            CreateCrawlerRequest crawlerRequest = CreateCrawlerRequest.builder()
                    .databaseName(dbName)
                    .name(crawlerName)
                    .description("Created by the AWS Glue Java API")
                    .targets(targets)
                    .role(iam)
                    .schedule(cron)
                    .build();

            glueClient.createCrawler(crawlerRequest);
            System.out.println(crawlerName + " was successfully created");

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
```

```
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考CreateCrawler中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考CreateCrawler中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun createGlueCrawler(
    iam: String?,
    s3Path: String?,
    cron: String?,
    dbName: String?,
    crawlerName: String,
) {
    val s3Target =
        S3Target {
            path = s3Path
        }

    // Add the S3Target to a list.
    val targetList = mutableListOf<S3Target>()
    targetList.add(s3Target)

    val targetOb =
        CrawlerTargets {
            s3Targets = targetList
        }

    val request =
        CreateCrawlerRequest {
            databaseName = dbName
            name = crawlerName
            description = "Created by the AWS Glue Kotlin API"
            targets = targetOb
            role = iam
            schedule = cron
        }

    GlueClient { region = "us-west-2" }.use { glueClient ->
```

```
        glueClient.createCrawler(request)
        println("$crawlerName was successfully created")
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK CreateCrawler中的 Kotlin API 參考。

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        $crawlerName = "example-crawler-test-" . $uniqid;

        $role = $iamService->getRole("AWSGlueServiceRole-DocExample");

        $path = 's3://crawler-public-us-east-1/flight/2016/csv';
        $glueService->createCrawler($crawlerName, $role['Role']['Arn'],
$databaseName, $path);

   public function createCrawler($crawlerName, $role, $databaseName, $path):
Result
   {
        return $this->customWaiter(function () use ($crawlerName, $role,
$databaseName, $path) {
            return $this->glueClient->createCrawler([
                'Name' => $crawlerName,
                'Role' => $role,
                'DatabaseName' => $databaseName,
                'Targets' => [
                    'S3Targets' =>
                        [[
                            'Path' => $path,
                        ]]
                ],
            ]);
```

```
        });
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考CreateCrawler中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def create_crawler(self, name, role_arn, db_name, db_prefix, s3_target):
        """
        Creates a crawler that can crawl the specified target and populate a
        database in your AWS Glue Data Catalog with metadata that describes the
 data
        in the target.

        :param name: The name of the crawler.
        :param role_arn: The Amazon Resource Name (ARN) of an AWS Identity and
 Access
                         Management (IAM) role that grants permission to let AWS
 Glue
                         access the resources it needs.
        :param db_name: The name to give the database that is created by the
 crawler.
```

```
        :param db_prefix: The prefix to give any database tables that are created
 by
                          the crawler.
        :param s3_target: The URL to an S3 bucket that contains data that is
                          the target of the crawler.
        """
        try:
            self.glue_client.create_crawler(
                Name=name,
                Role=role_arn,
                DatabaseName=db_name,
                TablePrefix=db_prefix,
                Targets={"S3Targets": [{"Path": s3_target}]},
            )
        except ClientError as err:
            logger.error(
                "Couldn't create crawler. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件CreateCrawler中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
```

```
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Creates a new crawler with the specified configuration.
  #
  # @param name [String] The name of the crawler.
  # @param role_arn [String] The ARN of the IAM role to be used by the crawler.
  # @param db_name [String] The name of the database where the crawler stores its
 metadata.
  # @param db_prefix [String] The prefix to be added to the names of tables that
 the crawler creates.
  # @param s3_target [String] The S3 path that the crawler will crawl.
  # @return [void]
  def create_crawler(name, role_arn, db_name, db_prefix, s3_target)
    @glue_client.create_crawler(
      name: name,
      role: role_arn,
      database_name: db_name,
      targets: {
        s3_targets: [
          {
            path: s3_target
          }
        ]
      }
    )
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not create crawler: \n#{e.message}")
    raise
  end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考CreateCrawler中的。

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        let create_crawler = glue
            .create_crawler()
            .name(self.crawler())
            .database_name(self.database())
            .role(self.iam_role.expose_secret())
            .targets(
                CrawlerTargets::builder()
                    .s3_targets(S3Target::builder().path(CRAWLER_TARGET).build())
                    .build(),
            )
            .send()
            .await;

        match create_crawler {
            Err(err) => {
                let glue_err: aws_sdk_glue::Error = err.into();
                match glue_err {
                    aws_sdk_glue::Error::AlreadyExistsException(_) => {
                        info!("Using existing crawler");
                        Ok(())
                    }
                    _ => Err(GlueMvpError::GlueSdk(glue_err)),
                }
            }
            Ok(_) => Ok(()),
        }?;
```

- 如需 API 的詳細資訊，請參閱 AWS SDK CreateCrawler中的 Rust API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭CreateJob配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用CreateJob。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
    /// <summary>
    /// Create an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="roleName">The name of the IAM role to be assumed by
    /// the job.</param>
    /// <param name="description">A description of the job.</param>
    /// <param name="scriptUrl">The URL to the script.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateJobAsync(string dbName, string tableName,
string bucketUrl, string jobName, string roleName, string description, string
scriptUrl)
    {
        var command = new JobCommand
        {
            PythonVersion = "3",
            Name = "glueetl",
            ScriptLocation = scriptUrl,
        };
```

```
        var arguments = new Dictionary<string, string>
        {
            { "--input_database", dbName },
            { "--input_table", tableName },
            { "--output_bucket_url", bucketUrl }
        };

        var request = new CreateJobRequest
        {
            Command = command,
            DefaultArguments = arguments,
            Description = description,
            GlueVersion = "3.0",
            Name = jobName,
            NumberOfWorkers = 10,
            Role = roleName,
            WorkerType = "G.1X"
        };

        var response = await _amazonGlue.CreateJobAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考CreateJob中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";
```

```
    Aws::Glue::GlueClient client(clientConfig);

        Aws::Glue::Model::CreateJobRequest request;
        request.SetName(JOB_NAME);
        request.SetRole(roleArn);
        request.SetGlueVersion(GLUE_VERSION);

        Aws::Glue::Model::JobCommand command;
        command.SetName(JOB_COMMAND_NAME);
        command.SetPythonVersion(JOB_PYTHON_VERSION);
        command.SetScriptLocation(
                Aws::String("s3://") + bucketName + "/" + PYTHON_SCRIPT);
        request.SetCommand(command);

        Aws::Glue::Model::CreateJobOutcome outcome = client.CreateJob(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully created the job." << std::endl;
        }
        else {
            std::cerr << "Error creating the job. " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                         clientConfig);
            return false;
        }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考CreateJob中的。

CLI

AWS CLI

建立工作以轉換資料

下列 `create-job` 範例會建立串流工作，它可執行存放在 S3 中的指令碼。

```
aws glue create-job \
    --name my-testing-job \
    --role AWSGlueServiceRoleDefault \
```

```
    --command '{ \
        "Name": "gluestreaming", \
        "ScriptLocation": "s3://DOC-EXAMPLE-BUCKET/folder/" \
    }' \
    --region us-east-1 \
    --output json \
    --default-arguments '{ \
        "--job-language":"scala", \
        "--class":"GlueApp" \
    }' \
    --profile my-profile \
    --endpoint https://glue.us-east-1.amazonaws.com
```

test_script.scala 的內容：

```
import com.amazonaws.services.glue.ChoiceOption
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.ResolveSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {
    def main(sysArgs: Array[String]) {
        val spark: SparkContext = new SparkContext()
        val glueContext: GlueContext = new GlueContext(spark)
        // @params: [JOB_NAME]
        val args = GlueArgParser.getResolvedOptions(sysArgs,
 Seq("JOB_NAME").toArray)
        Job.init(args("JOB_NAME"), glueContext, args.asJava)
        // @type: DataSource
        // @args: [database = "tempdb", table_name = "s3-source",
 transformation_ctx = "datasource0"]
        // @return: datasource0
        // @inputs: []
        val datasource0 = glueContext.getCatalogSource(database = "tempdb",
 tableName = "s3-source", redshiftTmpDir = "", transformationContext =
 "datasource0").getDynamicFrame()
        // @type: ApplyMapping
```

```
        // @args: [mapping = [("sensorid", "int", "sensorid", "int"),
 ("currenttemperature", "int", "currenttemperature", "int"), ("status", "string",
 "status", "string")], transformation_ctx = "applymapping1"]
        // @return: applymapping1
        // @inputs: [frame = datasource0]
        val applymapping1 = datasource0.applyMapping(mappings = Seq(("sensorid",
 "int", "sensorid", "int"), ("currenttemperature", "int", "currenttemperature",
 "int"), ("status", "string", "status", "string")), caseSensitive = false,
 transformationContext = "applymapping1")
        // @type: SelectFields
        // @args: [paths = ["sensorid", "currenttemperature", "status"],
 transformation_ctx = "selectfields2"]
        // @return: selectfields2
        // @inputs: [frame = applymapping1]
        val selectfields2 = applymapping1.selectFields(paths = Seq("sensorid",
 "currenttemperature", "status"), transformationContext = "selectfields2")
        // @type: ResolveChoice
        // @args: [choice = "MATCH_CATALOG", database = "tempdb", table_name =
 "my-s3-sink", transformation_ctx = "resolvechoice3"]
        // @return: resolvechoice3
        // @inputs: [frame = selectfields2]
        val resolvechoice3 = selectfields2.resolveChoice(choiceOption =
 Some(ChoiceOption("MATCH_CATALOG")), database = Some("tempdb"), tableName =
 Some("my-s3-sink"), transformationContext = "resolvechoice3")
        // @type: DataSink
        // @args: [database = "tempdb", table_name = "my-s3-sink",
 transformation_ctx = "datasink4"]
        // @return: datasink4
        // @inputs: [frame = resolvechoice3]
        val datasink4 = glueContext.getCatalogSink(database = "tempdb",
 tableName = "my-s3-sink", redshiftTmpDir = "", transformationContext =
 "datasink4").writeDynamicFrame(resolvechoice3)
        Job.commit()
    }
}
```

輸出：

```
{
    "Name": "my-testing-job"
}
```

如需詳細資訊，請參閱 AWS Glue 開發人員指南中的使用AWS Glue 編寫工作。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考CreateJob中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考CreateJob中的。

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        $role = $iamService->getRole("AWSGlueServiceRole-DocExample");

        $jobName = 'test-job-' . $uniqid;

        $scriptLocation = "s3://$bucketName/run_job.py";
        $job = $glueService->createJob($jobName, $role['Role']['Arn'],
$scriptLocation);

   public function createJob($jobName, $role, $scriptLocation, $pythonVersion =
'3', $glueVersion = '3.0'): Result
   {
       return $this->glueClient->createJob([
           'Name' => $jobName,
           'Role' => $role,
           'Command' => [
               'Name' => 'glueetl',
               'ScriptLocation' => $scriptLocation,
               'PythonVersion' => $pythonVersion,
           ],
           'GlueVersion' => $glueVersion,
       ]);
   }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考CreateJob中的。

PowerShell

適用的工具 PowerShell

範例 1：此範例會在 AWS Glue 中建立新工作。指令名稱值永遠為**glueetl**。 AWS Glue 支持運行寫在 Python 或斯卡拉作業腳本。在這個例子中，工作腳本（MyTestGlueJob.py）是用 Python 編寫的。Python 參數在**$DefArgs**變量中指定，然後傳遞給該 PowerShell 命令中的**DefaultArguments**參數，它接受一個哈希表。**$JobParams**變數中的參數來自 CreateJob API，該 API 記錄在 AWS Glue API 參考資料的工作 (https://docs.aws.amazon.com/glue/latest/dg/ aws-glue-api-jobs-job.html) 主題中。

```
$Command = New-Object Amazon.Glue.Model.JobCommand
$Command.Name = 'glueetl'
$Command.ScriptLocation = 's3://aws-glue-scripts-000000000000-us-west-2/admin/
MyTestGlueJob.py'
$Command

$Source = "source_test_table"
$Target = "target_test_table"
$Connections = $Source, $Target

$DefArgs = @{
     '--TempDir' = 's3://aws-glue-temporary-000000000000-us-west-2/admin'
     '--job-bookmark-option' = 'job-bookmark-disable'
     '--job-language' = 'python'
     }
$DefArgs

$ExecutionProp = New-Object Amazon.Glue.Model.ExecutionProperty
$ExecutionProp.MaxConcurrentRuns = 1
$ExecutionProp

$JobParams = @{
    "AllocatedCapacity"    = "5"
    "Command"              = $Command
    "Connections_Connection" = $Connections
    "DefaultArguments"  = $DefArgs
    "Description"       = "This is a test"
    "ExecutionProperty" = $ExecutionProp
    "MaxRetries"        = "1"
    "Name"              = "MyOregonTestGlueJob"
    "Role"              = "Amazon-GlueServiceRoleForSSM"
    "Timeout"           = "20"
```

```
    }

New-GlueJob @JobParams
```

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 指令程CreateJob式參考中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def create_job(self, name, description, role_arn, script_location):
        """
        Creates a job definition for an extract, transform, and load (ETL) job
 that can
        be run by AWS Glue.

        :param name: The name of the job definition.
        :param description: The description of the job definition.
        :param role_arn: The ARN of an IAM role that grants AWS Glue the
 permissions
                         it requires to run the job.
        :param script_location: The Amazon S3 URL of a Python ETL script that is
 run as
                                part of the job. The script defines how the data
 is
                                transformed.
```

```
        """
        try:
            self.glue_client.create_job(
                Name=name,
                Description=description,
                Role=role_arn,
                Command={
                    "Name": "glueetl",
                    "ScriptLocation": script_location,
                    "PythonVersion": "3",
                },
                GlueVersion="3.0",
            )
        except ClientError as err:
            logger.error(
                "Couldn't create job %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件CreateJob中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
```

```
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Creates a new job with the specified configuration.
  #
  # @param name [String] The name of the job.
  # @param description [String] The description of the job.
  # @param role_arn [String] The ARN of the IAM role to be used by the job.
  # @param script_location [String] The location of the ETL script for the job.
  # @return [void]
  def create_job(name, description, role_arn, script_location)
    @glue_client.create_job(
      name: name,
      description: description,
      role: role_arn,
      command: {
        name: "glueetl",
        script_location: script_location,
        python_version: "3"
      },
      glue_version: "3.0"
    )
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not create job #{name}: \n#{e.message}")
    raise
  end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考CreateJob中的。

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        let create_job = glue
            .create_job()
            .name(self.job())
            .role(self.iam_role.expose_secret())
            .command(
                JobCommand::builder()
                    .name("glueetl")
                    .python_version("3")
                    .script_location(format!("s3://{}/job.py", self.bucket()))
                    .build(),
            )
            .glue_version("3.0")
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        let job_name = create_job.name().ok_or_else(|| {
            GlueMvpError::Unknown("Did not get job name after creating
job".into())
        })?;
```

- 如需 API 的詳細資訊，請參閱 AWS SDK CreateJob中的 Rust API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭DeleteCrawler配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用DeleteCrawler。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
    /// <summary>
    /// Delete an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteCrawlerAsync(string crawlerName)
    {
        var response = await _amazonGlue.DeleteCrawlerAsync(new
DeleteCrawlerRequest { Name = crawlerName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考DeleteCrawler中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
 (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);

        Aws::Glue::Model::DeleteCrawlerRequest request;
        request.SetName(crawler);

        Aws::Glue::Model::DeleteCrawlerOutcome outcome =
 client.DeleteCrawler(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully deleted the crawler." << std::endl;
        }
        else {
            std::cerr << "Error deleting the crawler. "
                      << outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考DeleteCrawler中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });
```

```
  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考DeleteCrawler中的。

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        echo "Delete the crawler.\n";
        $glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);

    public function deleteCrawler($crawlerName)
    {
        return $this->glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考DeleteCrawler中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def delete_crawler(self, name):
        """
        Deletes a crawler.

        :param name: The name of the crawler to delete.
        """
        try:
            self.glue_client.delete_crawler(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't delete crawler %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件DeleteCrawler中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Deletes a crawler with the specified name.
  #
  # @param name [String] The name of the crawler to delete.
  # @return [void]
  def delete_crawler(name)
    @glue_client.delete_crawler(name: name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete crawler #{name}: \n#{e.message}")
    raise
  end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考DeleteCrawler中的。

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        glue.delete_crawler()
            .name(self.crawler())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
```

- 如需 API 的詳細資訊，請參閱 AWS SDK DeleteCrawler中的 Rust API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭DeleteDatabase配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用DeleteDatabase。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
    /// <summary>
    /// Delete the AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteDatabaseAsync(string dbName)
    {
        var response = await _amazonGlue.DeleteDatabaseAsync(new
DeleteDatabaseRequest { Name = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考DeleteDatabase中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);

        Aws::Glue::Model::DeleteDatabaseRequest request;
        request.SetName(database);

        Aws::Glue::Model::DeleteDatabaseOutcome outcome = client.DeleteDatabase(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully deleted the database." << std::endl;
```

```
        }
        else {
            std::cerr << "Error deleting database. " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考DeleteDatabase中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考DeleteDatabase中的。

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        echo "Delete the databases.\n";
        $glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);

    public function deleteDatabase($databaseName)
    {
        return $this->glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考DeleteDatabase中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
```

```
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def delete_database(self, name):
        """
        Deletes a metadata database from your Data Catalog.

        :param name: The name of the database to delete.
        """
        try:
            self.glue_client.delete_database(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't delete database %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件DeleteDatabase中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
```

```
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Removes a specified database from a Data Catalog.
  #
  # @param database_name [String] The name of the database to delete.
  # @return [void]
  def delete_database(database_name)
    @glue_client.delete_database(name: database_name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete database: \n#{e.message}")
  end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考DeleteDatabase中的。

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        glue.delete_database()
            .name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
```

- 如需 API 的詳細資訊，請參閱 AWS SDK DeleteDatabase中的 Rust API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭DeleteJob配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用DeleteJob。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
    /// <summary>
    /// Delete an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteJobAsync(string jobName)
    {
        var response = await _amazonGlue.DeleteJobAsync(new DeleteJobRequest
{ JobName = jobName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考DeleteJob中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);

        Aws::Glue::Model::DeleteJobRequest request;
        request.SetJobName(job);

        Aws::Glue::Model::DeleteJobOutcome outcome = client.DeleteJob(request);


        if (outcome.IsSuccess()) {
            std::cout << "Successfully deleted the job." << std::endl;
        }
        else {
            std::cerr << "Error deleting the job. " <<
outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考DeleteJob中的。

CLI

AWS CLI

若要刪除工作

下列 delete-job 範例會刪除不再需要的工作。

```
aws glue delete-job \
    --job-name my-testing-job
```

輸出：

```
{
    "JobName": "my-testing-job"
}
```

如需詳細資訊，請參閱 Glue 開發人員指南中的在 AWS Glu AWS e 主控台上使用工作。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考DeleteJob中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考DeleteJob中的。

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        echo "Delete the job.\n";
        $glueClient->deleteJob([
            'JobName' => $job['Name'],
        ]);

    public function deleteJob($jobName)
    {
        return $this->glueClient->deleteJob([
            'JobName' => $jobName,
        ]);
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考DeleteJob中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
```

```
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def delete_job(self, job_name):
        """
        Deletes a job definition. This also deletes data about all runs that are
        associated with this job definition.

        :param job_name: The name of the job definition to delete.
        """
        try:
            self.glue_client.delete_job(JobName=job_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete job %s. Here's why: %s: %s",
                job_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件DeleteJob中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
```

```
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Deletes a job with the specified name.
  #
  # @param job_name [String] The name of the job to delete.
  # @return [void]
  def delete_job(job_name)
    @glue_client.delete_job(job_name: job_name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete job: \n#{e.message}")
  end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考DeleteJob中的。

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        glue.delete_job()
            .job_name(self.job())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
```

- 如需 API 的詳細資訊，請參閱 AWS SDK DeleteJob中的 Rust API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭DeleteTable配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用DeleteTable。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
    /// <summary>
    /// Delete a table from an AWS Glue database.
    /// </summary>
    /// <param name="tableName">The table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTableAsync(string dbName, string tableName)
    {
        var response = await _amazonGlue.DeleteTableAsync(new DeleteTableRequest
{ Name = tableName, DatabaseName = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考DeleteTable中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考DeleteTable中的。

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        echo "Delete the tables.\n";
        foreach ($tables['TableList'] as $table) {
            $glueService->deleteTable($table['Name'], $databaseName);
        }

    public function deleteTable($tableName, $databaseName)
```

```
    {
        return $this->glueClient->deleteTable([
            'DatabaseName' => $databaseName,
            'Name' => $tableName,
        ]);
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考DeleteTable中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def delete_table(self, db_name, table_name):
        """
        Deletes a table from a metadata database.

        :param db_name: The name of the database that contains the table.
        :param table_name: The name of the table to delete.
        """
        try:
            self.glue_client.delete_table(DatabaseName=db_name, Name=table_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete table %s. Here's why: %s: %s",
```

```
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件DeleteTable中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Deletes a table with the specified name.
  #
  # @param database_name [String] The name of the catalog database in which the
 table resides.
  # @param table_name [String] The name of the table to be deleted.
  # @return [void]
  def delete_table(database_name, table_name)
    @glue_client.delete_table(database_name: database_name, name: table_name)
  rescue Aws::Glue::Errors::ServiceError => e
```

```
    @logger.error("Glue could not delete job: \n#{e.message}")
  end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考DeleteTable中的。

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        for t in &self.tables {
            glue.delete_table()
                .name(t.name())
                .database_name(self.database())
                .send()
                .await
                .map_err(GlueMvpError::from_glue_sdk)?;
        }
```

- 如需 API 的詳細資訊，請參閱 AWS SDK DeleteTable中的 Rust API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭GetCrawler配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用GetCrawler。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
    /// <summary>
    /// Get information about an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Crawler object describing the crawler.</returns>
    public async Task<Crawler?> GetCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new GetCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.GetCrawlerAsync(crawlerRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            var databaseName = response.Crawler.DatabaseName;
            Console.WriteLine($"{crawlerName} has the database {databaseName}");
            return response.Crawler;
        }

        Console.WriteLine($"No information regarding {crawlerName} could be
found.");
        return null;
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考GetCrawler中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Glue::GlueClient client(clientConfig);

       Aws::Glue::Model::GetCrawlerRequest request;
       request.SetName(CRAWLER_NAME);

       Aws::Glue::Model::GetCrawlerOutcome outcome = client.GetCrawler(request);

       if (outcome.IsSuccess()) {
           Aws::Glue::Model::CrawlerState crawlerState =
outcome.GetResult().GetCrawler().GetState();
           std::cout << "Retrieved crawler with state " <<

Aws::Glue::Model::CrawlerStateMapper::GetNameForCrawlerState(
                           crawlerState)
                   << "." << std::endl;
       }
       else {
           std::cerr << "Error retrieving a crawler.  "
                     << outcome.GetError().GetMessage() << std::endl;
           deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                        clientConfig);
           return false;
       }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考GetCrawler中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.glue.GlueClient;
import software.amazon.awssdk.services.glue.model.GetCrawlerRequest;
import software.amazon.awssdk.services.glue.model.GetCrawlerResponse;
import software.amazon.awssdk.services.glue.model.GlueException;
import java.time.Instant;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;
import java.util.Locale;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class GetCrawler {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <crawlerName>

                Where:
                    crawlerName - The name of the crawler.\s
                """;

        if (args.length != 1) {
```

```
            System.out.println(usage);
            System.exit(1);
        }

        String crawlerName = args[0];
        Region region = Region.US_EAST_1;
        GlueClient glueClient = GlueClient.builder()
                .region(region)
                .build();

        getSpecificCrawler(glueClient, crawlerName);
        glueClient.close();
    }

    public static void getSpecificCrawler(GlueClient glueClient, String
 crawlerName) {
        try {
            GetCrawlerRequest crawlerRequest = GetCrawlerRequest.builder()
                    .name(crawlerName)
                    .build();

            GetCrawlerResponse response = glueClient.getCrawler(crawlerRequest);
            Instant createDate = response.crawler().creationTime();

            // Convert the Instant to readable date
            DateTimeFormatter formatter =
 DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
                    .withLocale(Locale.US)
                    .withZone(ZoneId.systemDefault());

            formatter.format(createDate);
            System.out.println("The create date of the Crawler is " +
 createDate);

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考GetCrawler中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考GetCrawler中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun getSpecificCrawler(crawlerName: String?) {
    val request =
        GetCrawlerRequest {
            name = crawlerName
        }
    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getCrawler(request)
```

```
        val role = response.crawler?.role
        println("The role associated with this crawler is $role")
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK GetCrawler中的 Kotlin API 參考。

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        echo "Waiting for crawler";
        do {
            $crawler = $glueService->getCrawler($crawlerName);
            echo ".";
            sleep(10);
        } while ($crawler['Crawler']['State'] != "READY");
        echo "\n";

    public function getCrawler($crawlerName)
    {
        return $this->customWaiter(function () use ($crawlerName) {
            return $this->glueClient->getCrawler([
                'Name' => $crawlerName,
            ]);
        });
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考GetCrawler中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def get_crawler(self, name):
        """
        Gets information about a crawler.

        :param name: The name of the crawler to look up.
        :return: Data about the crawler.
        """
        crawler = None
        try:
            response = self.glue_client.get_crawler(Name=name)
            crawler = response["Crawler"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "EntityNotFoundException":
                logger.info("Crawler %s doesn't exist.", name)
            else:
                logger.error(
                    "Couldn't get crawler %s. Here's why: %s: %s",
                    name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
```

```
        return crawler
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件GetCrawler中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves information about a specific crawler.
  #
  # @param name [String] The name of the crawler to retrieve information about.
  # @return [Aws::Glue::Types::Crawler, nil] The crawler object if found, or nil
 if not found.
  def get_crawler(name)
    @glue_client.get_crawler(name: name)
  rescue Aws::Glue::Errors::EntityNotFoundException
    @logger.info("Crawler #{name} doesn't exist.")
    false
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get crawler #{name}: \n#{e.message}")
    raise
```

```
end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考GetCrawler中的。

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        let tmp_crawler = glue
            .get_crawler()
            .name(self.crawler())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
```

- 如需 API 的詳細資訊，請參閱 AWS SDK GetCrawler中的 Rust API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭GetDatabase配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用GetDatabase。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
    /// <summary>
    /// Get information about an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Database object containing information about the database.</
returns>
    public async Task<Database> GetDatabaseAsync(string dbName)
    {
        var databasesRequest = new GetDatabaseRequest
        {
            Name = dbName,
        };

        var response = await _amazonGlue.GetDatabaseAsync(databasesRequest);
        return response.Database;
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考GetDatabase中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);

        Aws::Glue::Model::GetDatabaseRequest request;
        request.SetName(CRAWLER_DATABASE_NAME);

        Aws::Glue::Model::GetDatabaseOutcome outcome =
client.GetDatabase(request);

        if (outcome.IsSuccess()) {
            const Aws::Glue::Model::Database &database =
outcome.GetResult().GetDatabase();

            std::cout << "Successfully retrieve the database\n" <<
                      database.Jsonize().View().WriteReadable() << "'." <<
std::endl;
        }
        else {
            std::cerr << "Error getting the database.  "
                      << outcome.GetError().GetMessage() << std::endl;
            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                         clientConfig);
            return false;
        }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考GetDatabase中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.glue.GlueClient;
import software.amazon.awssdk.services.glue.model.GetDatabaseRequest;
import software.amazon.awssdk.services.glue.model.GetDatabaseResponse;
import software.amazon.awssdk.services.glue.model.GlueException;
import java.time.Instant;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;
import java.util.Locale;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class GetDatabase {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <databaseName>

                Where:
                    databaseName - The name of the database.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String databaseName = args[0];
        Region region = Region.US_EAST_1;
        GlueClient glueClient = GlueClient.builder()
                .region(region)
                .build();

        getSpecificDatabase(glueClient, databaseName);
```

```
        glueClient.close();
    }

    public static void getSpecificDatabase(GlueClient glueClient, String
 databaseName) {
        try {
            GetDatabaseRequest databasesRequest = GetDatabaseRequest.builder()
                    .name(databaseName)
                    .build();

            GetDatabaseResponse response =
 glueClient.getDatabase(databasesRequest);
            Instant createDate = response.database().createTime();

            // Convert the Instant to readable date.
            DateTimeFormatter formatter =
 DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
                    .withLocale(Locale.US)
                    .withZone(ZoneId.systemDefault());

            formatter.format(createDate);
            System.out.println("The create date of the database is " +
 createDate);

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考GetDatabase中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考GetDatabase中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun getSpecificDatabase(databaseName: String?) {
    val request =
        GetDatabaseRequest {
            name = databaseName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getDatabase(request)
        val dbDesc = response.database?.description
        println("The database description is $dbDesc")
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK GetDatabase中的 Kotlin API 參考。

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        $databaseName = "doc-example-database-$uniqid";

        $database = $glueService->getDatabase($databaseName);
        echo "Found a database named " . $database['Database']['Name'] . "\n";

    public function getDatabase(string $databaseName): Result
    {
        return $this->customWaiter(function () use ($databaseName) {
            return $this->glueClient->getDatabase([
                'Name' => $databaseName,
            ]);
        });
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考GetDatabase中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
```

```
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def get_database(self, name):
        """
        Gets information about a database in your Data Catalog.

        :param name: The name of the database to look up.
        :return: Information about the database.
        """
        try:
            response = self.glue_client.get_database(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't get database %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Database"]
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件GetDatabase中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves information about a specific database.
  #
  # @param name [String] The name of the database to retrieve information about.
  # @return [Aws::Glue::Types::Database, nil] The database object if found, or
 nil if not found.
  def get_database(name)
    response = @glue_client.get_database(name: name)
    response.database
rescue Aws::Glue::Errors::GlueException => e
  @logger.error("Glue could not get database #{name}: \n#{e.message}")
  raise
  end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考GetDatabase中的。

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        let database = glue
            .get_database()
            .name(self.database())
            .send()
```

```
            .await
            .map_err(GlueMvpError::from_glue_sdk)?
            .to_owned();
        let database = database
            .database()
            .ok_or_else(|| GlueMvpError::Unknown("Could not find
 database".into()))?;
```

- 如需 API 的詳細資訊，請參閱 AWS SDK GetDatabase中的 Rust API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭GetDatabases配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用GetDatabases。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

CLI

AWS CLI

列出 AWS Glue 資料型錄中部分或所有資料庫的定義

下列 get-databases 範例會傳回 Data Catalog 中的資料庫相關資訊。

```
aws glue get-databases
```

輸出：

```
{
    "DatabaseList": [
        {
            "Name": "default",
            "Description": "Default Hive database",
            "LocationUri": "file:/spark-warehouse",
```

```
            "CreateTime": 1602084052.0,
            "CreateTableDefaultPermissions": [
                {
                    "Principal": {
                        "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
                    },
                    "Permissions": [
                        "ALL"
                    ]
                }
            ],
            "CatalogId": "111122223333"
        },
        {
            "Name": "flights-db",
            "CreateTime": 1587072847.0,
            "CreateTableDefaultPermissions": [
                {
                    "Principal": {
                        "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
                    },
                    "Permissions": [
                        "ALL"
                    ]
                }
            ],
            "CatalogId": "111122223333"
        },
        {
            "Name": "legislators",
            "CreateTime": 1601415625.0,
            "CreateTableDefaultPermissions": [
                {
                    "Principal": {
                        "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
                    },
                    "Permissions": [
                        "ALL"
                    ]
                }
            ],
            "CatalogId": "111122223333"
        },
        {
```

```
            "Name": "tempdb",
            "CreateTime": 1601498566.0,
            "CreateTableDefaultPermissions": [
                {
                    "Principal": {
                        "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
                    },
                    "Permissions": [
                        "ALL"
                    ]
                }
            ],
            "CatalogId": "111122223333"
        }
    ]
}
```

如需詳細資訊，請參閱《AWS Glue 開發人員指南》中的在 Data Catalog 中定義資料庫。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考GetDatabases中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});

  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考GetDatabases中的。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭GetJob配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用GetJob。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

CLI

AWS CLI

擷取工作相關資訊

以下 get-job 範例會擷取工作相關資訊。

```
aws glue get-job \
    --job-name my-testing-job
```

輸出：

```
{
    "Job": {
        "Name": "my-testing-job",
        "Role": "Glue_DefaultRole",
        "CreatedOn": 1602805698.167,
        "LastModifiedOn": 1602805698.167,
        "ExecutionProperty": {
            "MaxConcurrentRuns": 1
        },
        "Command": {
            "Name": "gluestreaming",
            "ScriptLocation": "s3://janetst-bucket-01/Scripts/test_script.scala",
            "PythonVersion": "2"
        },
        "DefaultArguments": {
            "--class": "GlueApp",
            "--job-language": "scala"
```

```
        },
        "MaxRetries": 0,
        "AllocatedCapacity": 10,
        "MaxCapacity": 10.0,
        "GlueVersion": "1.0"
    }
}
```

如需詳細資訊，請參閱《AWS Glue 開發人員指南》中的工作。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考GetJob中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const getJob = (jobName) => {
  const client = new GlueClient({});

  const command = new GetJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考GetJob中的。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭GetJobRun配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用GetJobRun。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
    /// <summary>
    /// Get information about a specific AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="jobRunId">The Id of the job run.</param>
    /// <returns>A JobRun object with information about the job run.</returns>
    public async Task<JobRun> GetJobRunAsync(string jobName, string jobRunId)
    {
        var response = await _amazonGlue.GetJobRunAsync(new GetJobRunRequest
{ JobName = jobName, RunId = jobRunId });
        return response.JobRun;
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考GetJobRun中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Glue::GlueClient client(clientConfig);

       Aws::Glue::Model::GetJobRunRequest jobRunRequest;
       jobRunRequest.SetJobName(jobName);
       jobRunRequest.SetRunId(jobRunID);

       Aws::Glue::Model::GetJobRunOutcome jobRunOutcome = client.GetJobRun(
               jobRunRequest);

       if (jobRunOutcome.IsSuccess()) {
           std::cout << "Displaying the job run JSON description." << std::endl;
           std::cout
                   <<
jobRunOutcome.GetResult().GetJobRun().Jsonize().View().WriteReadable()
                   << std::endl;
       }
       else {
           std::cerr << "Error get a job run. "
                     << jobRunOutcome.GetError().GetMessage()
                     << std::endl;
       }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考GetJobRun中的。

CLI

AWS CLI

取得工作執行的相關資訊

以下 get-job-run 範例會擷取工作執行的相關資訊。

```
aws glue get-job-run \
    --job-name "Combine legistators data" \
    --run-id jr_012e176506505074d94d761755e5c62538ee1aad6f17d39f527e9140cf0c9a5e
```

輸出：

```
{
    "JobRun": {
        "Id":
 "jr_012e176506505074d94d761755e5c62538ee1aad6f17d39f527e9140cf0c9a5e",
        "Attempt": 0,
        "JobName": "Combine legistators data",
        "StartedOn": 1602873931.255,
        "LastModifiedOn": 1602874075.985,
        "CompletedOn": 1602874075.985,
        "JobRunState": "SUCCEEDED",
        "Arguments": {
            "--enable-continuous-cloudwatch-log": "true",
            "--enable-metrics": "",
            "--enable-spark-ui": "true",
            "--job-bookmark-option": "job-bookmark-enable",
            "--spark-event-logs-path": "s3://aws-glue-assets-111122223333-us-
east-1/sparkHistoryLogs/"
        },
        "PredecessorRuns": [],
        "AllocatedCapacity": 10,
        "ExecutionTime": 117,
        "Timeout": 2880,
        "MaxCapacity": 10.0,
        "WorkerType": "G.1X",
        "NumberOfWorkers": 10,
        "LogGroupName": "/aws-glue/jobs",
        "GlueVersion": "2.0"
    }
}
```

如需詳細資訊，請參閱《AWS Glue 開發人員指南》中的工作執行。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考GetJobRun中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考GetJobRun中的。

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        $jobName = 'test-job-' . $uniqid;

        $outputBucketUrl = "s3://$bucketName";
```

```
        $runId = $glueService->startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl)['JobRunId'];

        echo "waiting for job";
        do {
            $jobRun = $glueService->getJobRun($jobName, $runId);
            echo ".";
            sleep(10);
        } while (!array_intersect([$jobRun['JobRun']['JobRunState']],
['SUCCEEDED', 'STOPPED', 'FAILED', 'TIMEOUT']));
        echo "\n";

    public function getJobRun($jobName, $runId, $predecessorsIncluded = false):
Result
    {
        return $this->glueClient->getJobRun([
            'JobName' => $jobName,
            'RunId' => $runId,
            'PredecessorsIncluded' => $predecessorsIncluded,
        ]);
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考GetJobRun中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
```

```
        self.glue_client = glue_client


    def get_job_run(self, name, run_id):
        """
        Gets information about a single job run.

        :param name: The name of the job definition for the run.
        :param run_id: The ID of the run.
        :return: Information about the run.
        """
        try:
            response = self.glue_client.get_job_run(JobName=name, RunId=run_id)
        except ClientError as err:
            logger.error(
                "Couldn't get job run %s/%s. Here's why: %s: %s",
                name,
                run_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobRun"]
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件GetJobRun中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
```

```
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves data for a specific job run.
  #
  # @param job_name [String] The name of the job run to retrieve data for.
  # @return [Glue::Types::GetJobRunResponse]
  def get_job_run(job_name, run_id)
    @glue_client.get_job_run(job_name: job_name, run_id: run_id)
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get job runs: \n#{e.message}")
  end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考GetJobRun中的。

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        let get_job_run = || async {
            Ok::<JobRun, GlueMvpError>(
                glue.get_job_run()
                    .job_name(self.job())
                    .run_id(job_run_id.to_string())
                    .send()
                    .await
                    .map_err(GlueMvpError::from_glue_sdk)?
                    .job_run()
```

```
                    .ok_or_else(|| GlueMvpError::Unknown("Failed to get
job_run".into()))?
                    .to_owned(),
            )
        };

        let mut job_run = get_job_run().await?;
        let mut state =
job_run.job_run_state().unwrap_or(&unknown_state).to_owned();

        while matches!(
            state,
            JobRunState::Starting | JobRunState::Stopping | JobRunState::Running
        ) {
            info!(?state, "Waiting for job to finish");
            tokio::time::sleep(self.wait_delay).await;

            job_run = get_job_run().await?;
            state = job_run.job_run_state().unwrap_or(&unknown_state).to_owned();
        }
```

- 如需 API 的詳細資訊，請參閱 AWS SDK GetJobRun中的 Rust API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭GetJobRuns配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用GetJobRuns。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
    /// <summary>
    /// Get information about all AWS Glue runs of a specific job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A list of JobRun objects.</returns>
    public async Task<List<JobRun>> GetJobRunsAsync(string jobName)
    {
        var jobRuns = new List<JobRun>();

        var request = new GetJobRunsRequest
        {
            JobName = jobName,
        };

        // No need to loop to get all the log groups--the SDK does it for us
behind the scenes
        var paginatorForJobRuns =
            _amazonGlue.Paginators.GetJobRuns(request);

        await foreach (var response in paginatorForJobRuns.Responses)
        {
            response.JobRuns.ForEach(jobRun =>
            {
                jobRuns.Add(jobRun);
            });
        }

        return jobRuns;
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考GetJobRuns中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Glue::GlueClient client(clientConfig);

       Aws::Glue::Model::GetJobRunsRequest getJobRunsRequest;
       getJobRunsRequest.SetJobName(jobName);

       Aws::String nextToken; // Used for pagination.
       std::vector<Aws::Glue::Model::JobRun> allJobRuns;
       do {
           if (!nextToken.empty()) {
               getJobRunsRequest.SetNextToken(nextToken);
           }
           Aws::Glue::Model::GetJobRunsOutcome jobRunsOutcome =
client.GetJobRuns(
                   getJobRunsRequest);

           if (jobRunsOutcome.IsSuccess()) {
               const std::vector<Aws::Glue::Model::JobRun> &jobRuns =
jobRunsOutcome.GetResult().GetJobRuns();
               allJobRuns.insert(allJobRuns.end(), jobRuns.begin(),
jobRuns.end());

               nextToken = jobRunsOutcome.GetResult().GetNextToken();
           }
           else {
               std::cerr << "Error getting job runs. "
```

```
                            << jobRunsOutcome.GetError().GetMessage()
                            << std::endl;
                    break;
                }
            } while (!nextToken.empty());
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考GetJobRuns中的。

CLI

AWS CLI

取得工作的全部工作執行相關資訊

以下 get-job-runs 範例會擷取工作的工作執行相關資訊。

```
aws glue get-job-runs \
    --job-name "my-testing-job"
```

輸出：

```
{
    "JobRuns": [
        {
            "Id":
 "jr_012e176506505074d94d761755e5c62538ee1aad6f17d39f527e9140cf0c9a5e",
            "Attempt": 0,
            "JobName": "my-testing-job",
            "StartedOn": 1602873931.255,
            "LastModifiedOn": 1602874075.985,
            "CompletedOn": 1602874075.985,
            "JobRunState": "SUCCEEDED",
            "Arguments": {
                "--enable-continuous-cloudwatch-log": "true",
                "--enable-metrics": "",
                "--enable-spark-ui": "true",
                "--job-bookmark-option": "job-bookmark-enable",
                "--spark-event-logs-path": "s3://aws-glue-assets-111122223333-us-
east-1/sparkHistoryLogs/"
            },
            "PredecessorRuns": [],
            "AllocatedCapacity": 10,
```

```
            "ExecutionTime": 117,
            "Timeout": 2880,
            "MaxCapacity": 10.0,
            "WorkerType": "G.1X",
            "NumberOfWorkers": 10,
            "LogGroupName": "/aws-glue/jobs",
            "GlueVersion": "2.0"
        },
        {
            "Id":
 "jr_03cc19ddab11c4e244d3f735567de74ff93b0b3ef468a713ffe73e53d1aec08f_attempt_2",
            "Attempt": 2,
            "PreviousRunId":
 "jr_03cc19ddab11c4e244d3f735567de74ff93b0b3ef468a713ffe73e53d1aec08f_attempt_1",
            "JobName": "my-testing-job",
            "StartedOn": 1602811168.496,
            "LastModifiedOn": 1602811282.39,
            "CompletedOn": 1602811282.39,
            "JobRunState": "FAILED",
            "ErrorMessage": "An error occurred while calling
 o122.pyWriteDynamicFrame.
                Access Denied (Service: Amazon S3; Status Code: 403; Error Code:
 AccessDenied;
                Request ID: 021AAB703DB20A2D;
                S3 Extended Request ID: teZk24Y09TkXzBvMPG502L5VJBhe9DJuWA9/
TXtuGOqfByajkfL/Tlqt5JBGdEGpigAqzdMDM/U=)",
            "PredecessorRuns": [],
            "AllocatedCapacity": 10,
            "ExecutionTime": 110,
            "Timeout": 2880,
            "MaxCapacity": 10.0,
            "WorkerType": "G.1X",
            "NumberOfWorkers": 10,
            "LogGroupName": "/aws-glue/jobs",
            "GlueVersion": "2.0"
        },
        {
            "Id":
 "jr_03cc19ddab11c4e244d3f735567de74ff93b0b3ef468a713ffe73e53d1aec08f_attempt_1",
            "Attempt": 1,
            "PreviousRunId":
 "jr_03cc19ddab11c4e244d3f735567de74ff93b0b3ef468a713ffe73e53d1aec08f",
            "JobName": "my-testing-job",
            "StartedOn": 1602811020.518,
```

```
            "LastModifiedOn": 1602811138.364,
            "CompletedOn": 1602811138.364,
            "JobRunState": "FAILED",
            "ErrorMessage": "An error occurred while calling
 o122.pyWriteDynamicFrame.
                Access Denied (Service: Amazon S3; Status Code: 403; Error Code:
 AccessDenied;
                Request ID: 2671D37856AE7ABB;
                S3 Extended Request ID: RLJCJw20brV
+PpC6GpORahyF2fp9flB5SSb2bTGPnUSPVizLXRl1PN3QZldb+v1o9qRVktNYbW8=)",
            "PredecessorRuns": [],
            "AllocatedCapacity": 10,
            "ExecutionTime": 113,
            "Timeout": 2880,
            "MaxCapacity": 10.0,
            "WorkerType": "G.1X",
            "NumberOfWorkers": 10,
            "LogGroupName": "/aws-glue/jobs",
            "GlueVersion": "2.0"
        }
    ]
}
```

如需詳細資訊，請參閱《AWS Glue 開發人員指南》中的工作執行。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考GetJobRuns中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });
```

```
  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考GetJobRuns中的。

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        $jobName = 'test-job-' . $uniqid;

        $jobRuns = $glueService->getJobRuns($jobName);

    public function getJobRuns($jobName, $maxResults = 0, $nextToken = ''):
Result
    {
        $arguments = ['JobName' => $jobName];
        if ($maxResults) {
            $arguments['MaxResults'] = $maxResults;
        }
        if ($nextToken) {
            $arguments['NextToken'] = $nextToken;
        }
        return $this->glueClient->getJobRuns($arguments);
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考GetJobRuns中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def get_job_runs(self, job_name):
        """
        Gets information about runs that have been performed for a specific job
        definition.

        :param job_name: The name of the job definition to look up.
        :return: The list of job runs.
        """
        try:
            response = self.glue_client.get_job_runs(JobName=job_name)
        except ClientError as err:
            logger.error(
                "Couldn't get job runs for %s. Here's why: %s: %s",
                job_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobRuns"]
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件GetJobRuns中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves a list of job runs for the specified job.
  #
  # @param job_name [String] The name of the job to retrieve job runs for.
  # @return [Array<Aws::Glue::Types::JobRun>]
  def get_job_runs(job_name)
    response = @glue_client.get_job_runs(job_name: job_name)
    response.job_runs
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get job runs: \n#{e.message}")
  end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考GetJobRuns中的。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭GetTables配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用GetTables。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
/// <summary>
/// Get a list of tables for an AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A list of Table objects.</returns>
public async Task<List<Table>> GetTablesAsync(string dbName)
{
    var request = new GetTablesRequest { DatabaseName = dbName };
    var tables = new List<Table>();

    // Get a paginator for listing the tables.
    var tablePaginator = _amazonGlue.Paginators.GetTables(request);

    await foreach (var response in tablePaginator.Responses)
    {
        tables.AddRange(response.TableList);
    }

    return tables;
```

```
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考GetTables中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Glue::GlueClient client(clientConfig);

       Aws::Glue::Model::GetTablesRequest request;
       request.SetDatabaseName(CRAWLER_DATABASE_NAME);
       std::vector<Aws::Glue::Model::Table> all_tables;
       Aws::String nextToken; // Used for pagination.
       do {
           Aws::Glue::Model::GetTablesOutcome outcome =
client.GetTables(request);

           if (outcome.IsSuccess()) {
               const std::vector<Aws::Glue::Model::Table> &tables =
outcome.GetResult().GetTableList();
               all_tables.insert(all_tables.end(), tables.begin(),
tables.end());
               nextToken = outcome.GetResult().GetNextToken();
           }
           else {
               std::cerr << "Error getting the tables. "
                         << outcome.GetError().GetMessage()
                         << std::endl;
```

```
                deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                             clientConfig);
                return false;
            }
        } while (!nextToken.empty());

        std::cout << "The database contains " << all_tables.size()
                  << (all_tables.size() == 1 ?
                      " table." : "tables.") << std::endl;
        std::cout << "Here is a list of the tables in the database.";
        for (size_t index = 0; index < all_tables.size(); ++index) {
            std::cout << "    " << index + 1 << ":  " <<
all_tables[index].GetName()
                      << std::endl;
        }

        if (!all_tables.empty()) {
            int tableIndex = askQuestionForIntRange(
                    "Enter an index to display the database detail ",
                    1, static_cast<int>(all_tables.size()));
            std::cout << all_tables[tableIndex -
1].Jsonize().View().WriteReadable()
                      << std::endl;

            tableName = all_tables[tableIndex - 1].GetName();
        }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考[GetTables](#)中的。

CLI

AWS CLI

列出指定資料庫中部分或全部資料表的定義

下列 `get-tables` 範例會傳回指定資料庫中資料表的相關資訊。

```
aws glue get-tables --database-name 'tempdb'
```

輸出：

```
{
```

```
    "TableList": [
        {
            "Name": "my-s3-sink",
            "DatabaseName": "tempdb",
            "CreateTime": 1602730539.0,
            "UpdateTime": 1602730539.0,
            "Retention": 0,
            "StorageDescriptor": {
                "Columns": [
                    {
                        "Name": "sensorid",
                        "Type": "int"
                    },
                    {
                        "Name": "currenttemperature",
                        "Type": "int"
                    },
                    {
                        "Name": "status",
                        "Type": "string"
                    }
                ],
                "Location": "s3://janetst-bucket-01/test-s3-output/",
                "Compressed": false,
                "NumberOfBuckets": 0,
                "SerdeInfo": {
                    "SerializationLibrary": "org.openx.data.jsonserde.JsonSerDe"
                },
                "SortColumns": [],
                "StoredAsSubDirectories": false
            },
            "Parameters": {
                "classification": "json"
            },
            "CreatedBy": "arn:aws:iam::007436865787:user/JRSTERN",
            "IsRegisteredWithLakeFormation": false,
            "CatalogId": "007436865787"
        },
        {
            "Name": "s3-source",
            "DatabaseName": "tempdb",
            "CreateTime": 1602730658.0,
            "UpdateTime": 1602730658.0,
            "Retention": 0,
```

```
            "StorageDescriptor": {
                "Columns": [
                    {
                        "Name": "sensorid",
                        "Type": "int"
                    },
                    {
                        "Name": "currenttemperature",
                        "Type": "int"
                    },
                    {
                        "Name": "status",
                        "Type": "string"
                    }
                ],
                "Location": "s3://janetst-bucket-01/",
                "Compressed": false,
                "NumberOfBuckets": 0,
                "SortColumns": [],
                "StoredAsSubDirectories": false
            },
            "Parameters": {
                "classification": "json"
            },
            "CreatedBy": "arn:aws:iam::007436865787:user/JRSTERN",
            "IsRegisteredWithLakeFormation": false,
            "CatalogId": "007436865787"
        },
        {
            "Name": "test-kinesis-input",
            "DatabaseName": "tempdb",
            "CreateTime": 1601507001.0,
            "UpdateTime": 1601507001.0,
            "Retention": 0,
            "StorageDescriptor": {
                "Columns": [
                    {
                        "Name": "sensorid",
                        "Type": "int"
                    },
                    {
                        "Name": "currenttemperature",
                        "Type": "int"
                    },
```

```
                    {
                        "Name": "status",
                        "Type": "string"
                    }
                ],
                "Location": "my-testing-stream",
                "Compressed": false,
                "NumberOfBuckets": 0,
                "SerdeInfo": {
                    "SerializationLibrary": "org.openx.data.jsonserde.JsonSerDe"
                },
                "SortColumns": [],
                "Parameters": {
                    "kinesisUrl": "https://kinesis.us-east-1.amazonaws.com",
                    "streamName": "my-testing-stream",
                    "typeOfData": "kinesis"
                },
                "StoredAsSubDirectories": false
            },
            "Parameters": {
                "classification": "json"
            },
            "CreatedBy": "arn:aws:iam::007436865787:user/JRSTERN",
            "IsRegisteredWithLakeFormation": false,
            "CatalogId": "007436865787"
        }
    ]
}
```

如需詳細資訊，請參閱 Glue 開發人員指南中的 AWS Glue 資料型錄中AWS 的定義表格。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考GetTables中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.glue.GlueClient;
import software.amazon.awssdk.services.glue.model.GetTableRequest;
import software.amazon.awssdk.services.glue.model.GetTableResponse;
import software.amazon.awssdk.services.glue.model.GlueException;
import java.time.Instant;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;
import java.util.Locale;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class GetTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <dbName> <tableName>

                Where:
                    dbName - The database name.\s
                    tableName - The name of the table.\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String dbName = args[0];
        String tableName = args[1];
        Region region = Region.US_EAST_1;
        GlueClient glueClient = GlueClient.builder()
                .region(region)
                .build();
```

```
        getGlueTable(glueClient, dbName, tableName);
        glueClient.close();
    }

    public static void getGlueTable(GlueClient glueClient, String dbName, String
 tableName) {
        try {
            GetTableRequest tableRequest = GetTableRequest.builder()
                    .databaseName(dbName)
                    .name(tableName)
                    .build();

            GetTableResponse tableResponse = glueClient.getTable(tableRequest);
            Instant createDate = tableResponse.table().createTime();

            // Convert the Instant to readable date.
            DateTimeFormatter formatter =
 DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
                    .withLocale(Locale.US)
                    .withZone(ZoneId.systemDefault());

            formatter.format(createDate);
            System.out.println("The create date of the table is " + createDate);

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考GetTables中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考GetTables中的。

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        $databaseName = "doc-example-database-$uniqid";

        $tables = $glueService->getTables($databaseName);

    public function getTables($databaseName): Result
    {
        return $this->glueClient->getTables([
```

```
            'DatabaseName' => $databaseName,
        ]);
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考GetTables中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def get_tables(self, db_name):
        """
        Gets a list of tables in a Data Catalog database.

        :param db_name: The name of the database to query.
        :return: The list of tables in the database.
        """
        try:
            response = self.glue_client.get_tables(DatabaseName=db_name)
        except ClientError as err:
            logger.error(
                "Couldn't get tables %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
```

```
            )
            raise
        else:
            return response["TableList"]
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件GetTables中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves a list of tables in the specified database.
  #
  # @param db_name [String] The name of the database to retrieve tables from.
  # @return [Array<Aws::Glue::Types::Table>]
  def get_tables(db_name)
    response = @glue_client.get_tables(database_name: db_name)
    response.table_list
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get tables #{db_name}: \n#{e.message}")
    raise
```

```
end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考GetTables中的。

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        let tables = glue
            .get_tables()
            .database_name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        let tables = tables.table_list();
```

- 如需 API 的詳細資訊，請參閱 AWS SDK GetTables中的 Rust API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭`ListJobs`配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用`ListJobs`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
    /// <summary>
    /// List AWS Glue jobs using a paginator.
    /// </summary>
    /// <returns>A list of AWS Glue job names.</returns>
    public async Task<List<string>> ListJobsAsync()
    {
        var jobNames = new List<string>();

        var listJobsPaginator = _amazonGlue.Paginators.ListJobs(new
ListJobsRequest { MaxResults = 10 });
        await foreach (var response in listJobsPaginator.Responses)
        {
            jobNames.AddRange(response.JobNames);
        }

        return jobNames;
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考ListJobs中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Glue::GlueClient client(clientConfig);

       Aws::Glue::Model::ListJobsRequest listJobsRequest;

       Aws::String nextToken;
       std::vector<Aws::String> allJobNames;

       do {
           if (!nextToken.empty()) {
               listJobsRequest.SetNextToken(nextToken);
           }
           Aws::Glue::Model::ListJobsOutcome listRunsOutcome = client.ListJobs(
                   listJobsRequest);

           if (listRunsOutcome.IsSuccess()) {
               const std::vector<Aws::String> &jobNames =
listRunsOutcome.GetResult().GetJobNames();
               allJobNames.insert(allJobNames.end(), jobNames.begin(),
jobNames.end());
               nextToken = listRunsOutcome.GetResult().GetNextToken();
           }
           else {
               std::cerr << "Error listing jobs. "
                         << listRunsOutcome.GetError().GetMessage()
                         << std::endl;
           }
       } while (!nextToken.empty());
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考ListJobs中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考ListJobs中的。

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        $jobs = $glueService->listJobs();
        echo "Current jobs:\n";
        foreach ($jobs['JobNames'] as $jobsName) {
            echo "{$jobsName}\n";
        }

    public function listJobs($maxResults = null, $nextToken = null, $tags = []):
 Result
    {
```

```
        $arguments = [];
        if ($maxResults) {
            $arguments['MaxResults'] = $maxResults;
        }
        if ($nextToken) {
            $arguments['NextToken'] = $nextToken;
        }
        if (!empty($tags)) {
            $arguments['Tags'] = $tags;
        }
        return $this->glueClient->listJobs($arguments);
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考ListJobs中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def list_jobs(self):
        """
        Lists the names of job definitions in your account.

        :return: The list of job definition names.
        """
```

```
        try:
            response = self.glue_client.list_jobs()
        except ClientError as err:
            logger.error(
                "Couldn't list jobs. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobNames"]
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件ListJobs中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves a list of jobs in AWS Glue.
  #
  # @return [Aws::Glue::Types::ListJobsResponse]
```

```
def list_jobs
  @glue_client.list_jobs
rescue Aws::Glue::Errors::GlueException => e
  @logger.error("Glue could not list jobs: \n#{e.message}")
  raise
end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考ListJobs中的。

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        let mut list_jobs = glue.list_jobs().into_paginator().send();
        while let Some(list_jobs_output) = list_jobs.next().await {
            match list_jobs_output {
                Ok(list_jobs) => {
                    let names = list_jobs.job_names();
                    info!(?names, "Found these jobs")
                }
                Err(err) => return Err(GlueMvpError::from_glue_sdk(err)),
            }
        }
```

- 如需 API 的詳細資訊，請參閱 AWS SDK ListJobs中的 Rust API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭`StartCrawler`配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用`StartCrawler`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
/// <summary>
/// Start an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> StartCrawlerAsync(string crawlerName)
{
    var crawlerRequest = new StartCrawlerRequest
    {
        Name = crawlerName,
    };

    var response = await _amazonGlue.StartCrawlerAsync(crawlerRequest);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考StartCrawler中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Glue::GlueClient client(clientConfig);

       Aws::Glue::Model::StartCrawlerRequest request;
       request.SetName(CRAWLER_NAME);

       Aws::Glue::Model::StartCrawlerOutcome outcome =
client.StartCrawler(request);


       if (outcome.IsSuccess() || (Aws::Glue::GlueErrors::CRAWLER_RUNNING ==
                                   outcome.GetError().GetErrorType())) {
           if (!outcome.IsSuccess()) {
               std::cout << "Crawler was already started." << std::endl;
           }
           else {
               std::cout << "Successfully started crawler." << std::endl;
           }

           std::cout << "This may take a while to run." << std::endl;

           Aws::Glue::Model::CrawlerState crawlerState =
Aws::Glue::Model::CrawlerState::NOT_SET;
           int iterations = 0;
           while (Aws::Glue::Model::CrawlerState::READY != crawlerState) {
               std::this_thread::sleep_for(std::chrono::seconds(1));
               ++iterations;
               if ((iterations % 10) == 0) { // Log status every 10 seconds.
```

```
                    std::cout << "Crawler status " <<

Aws::Glue::Model::CrawlerStateMapper::GetNameForCrawlerState(
                                       crawlerState)
                              << ". After " << iterations
                              << " seconds elapsed."
                              << std::endl;
                }
                Aws::Glue::Model::GetCrawlerRequest getCrawlerRequest;
                getCrawlerRequest.SetName(CRAWLER_NAME);

                Aws::Glue::Model::GetCrawlerOutcome getCrawlerOutcome =
client.GetCrawler(
                        getCrawlerRequest);

                if (getCrawlerOutcome.IsSuccess()) {
                    crawlerState =
getCrawlerOutcome.GetResult().GetCrawler().GetState();
                }
                else {
                    std::cerr << "Error getting crawler.  "
                              << getCrawlerOutcome.GetError().GetMessage() <<
std::endl;
                    break;
                }
            }

            if (Aws::Glue::Model::CrawlerState::READY == crawlerState) {
                std::cout << "Crawler finished running after " << iterations
                          << " seconds."
                          << std::endl;
            }
        }
        else {
            std::cerr << "Error starting a crawler.  "
                      << outcome.GetError().GetMessage()
                      << std::endl;

            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                         clientConfig);
            return false;
        }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考StartCrawler中的。

CLI

AWS CLI

啟動爬蟲程式

以下 `start-crawler` 範例會啟動爬蟲程式。

```
aws glue start-crawler --name my-crawler
```

輸出：

```
None
```

如需詳細資訊，請參閱《AWS Glue 開發人員指南》中的定義爬蟲程式。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考StartCrawler中的。

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.glue.GlueClient;
import software.amazon.awssdk.services.glue.model.GlueException;
import software.amazon.awssdk.services.glue.model.StartCrawlerRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class StartCrawler {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <crawlerName>

                Where:
                    crawlerName - The name of the crawler.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String crawlerName = args[0];
        Region region = Region.US_EAST_1;
        GlueClient glueClient = GlueClient.builder()
                .region(region)
                .build();

        startSpecificCrawler(glueClient, crawlerName);
        glueClient.close();
    }

    public static void startSpecificCrawler(GlueClient glueClient, String
 crawlerName) {
        try {
            StartCrawlerRequest crawlerRequest = StartCrawlerRequest.builder()
                    .name(crawlerName)
                    .build();

            glueClient.startCrawler(crawlerRequest);

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考StartCrawler中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考StartCrawler中的。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun startSpecificCrawler(crawlerName: String?) {
```

```
    val request =
        StartCrawlerRequest {
            name = crawlerName
        }

    GlueClient { region = "us-west-2" }.use { glueClient ->
        glueClient.startCrawler(request)
        println("$crawlerName was successfully started.")
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK StartCrawler中的 Kotlin API 參考。

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        $crawlerName = "example-crawler-test-" . $uniqid;

        $databaseName = "doc-example-database-$uniqid";

        $glueService->startCrawler($crawlerName);

    public function startCrawler($crawlerName): Result
    {
        return $this->glueClient->startCrawler([
            'Name' => $crawlerName,
        ]);
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考StartCrawler中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def start_crawler(self, name):
        """
        Starts a crawler. The crawler crawls its configured target and creates
        metadata that describes the data it finds in the target data source.

        :param name: The name of the crawler to start.
        """
        try:
            self.glue_client.start_crawler(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't start crawler %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件StartCrawler中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Starts a crawler with the specified name.
  #
  # @param name [String] The name of the crawler to start.
  # @return [void]
  def start_crawler(name)
    @glue_client.start_crawler(name: name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not start crawler #{name}: \n#{e.message}")
    raise
  end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考StartCrawler中的。

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        let start_crawler =
glue.start_crawler().name(self.crawler()).send().await;

        match start_crawler {
            Ok(_) => Ok(()),
            Err(err) => {
                let glue_err: aws_sdk_glue::Error = err.into();
                match glue_err {
                    aws_sdk_glue::Error::CrawlerRunningException(_) => Ok(()),
                    _ => Err(GlueMvpError::GlueSdk(glue_err)),
                }
            }
        }?;
```

- 如需 API 的詳細資訊，請參閱 AWS SDK StartCrawler中的 Rust API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭`StartJobRun`配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用`StartJobRun`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 開始使用爬蟲程式和任務

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
/// <summary>
/// Start an AWS Glue job run.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A string representing the job run Id.</returns>
public async Task<string> StartJobRunAsync(
    string jobName,
    string inputDatabase,
    string inputTable,
    string bucketName)
{
    var request = new StartJobRunRequest
    {
        JobName = jobName,
        Arguments = new Dictionary<string, string>
        {
            {"--input_database", inputDatabase},
            {"--input_table", inputTable},
            {"--output_bucket_url", $"s3://{bucketName}/"}
        }
    };

    var response = await _amazonGlue.StartJobRunAsync(request);
    return response.JobRunId;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考StartJobRun中的。

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Glue::GlueClient client(clientConfig);

       Aws::Glue::Model::StartJobRunRequest request;
       request.SetJobName(JOB_NAME);

       Aws::Map<Aws::String, Aws::String> arguments;
       arguments["--input_database"] = CRAWLER_DATABASE_NAME;
       arguments["--input_table"] = tableName;
       arguments["--output_bucket_url"] = Aws::String("s3://") + bucketName +
"/";
       request.SetArguments(arguments);

       Aws::Glue::Model::StartJobRunOutcome outcome =
client.StartJobRun(request);

       if (outcome.IsSuccess()) {
           std::cout << "Successfully started the job." << std::endl;

           Aws::String jobRunId = outcome.GetResult().GetJobRunId();

           int iterator = 0;
           bool done = false;
           while (!done) {
               ++iterator;
               std::this_thread::sleep_for(std::chrono::seconds(1));
               Aws::Glue::Model::GetJobRunRequest jobRunRequest;
               jobRunRequest.SetJobName(JOB_NAME);
```

```
                jobRunRequest.SetRunId(jobRunId);

                Aws::Glue::Model::GetJobRunOutcome jobRunOutcome =
client.GetJobRun(
                        jobRunRequest);

                if (jobRunOutcome.IsSuccess()) {
                    const Aws::Glue::Model::JobRun &jobRun =
jobRunOutcome.GetResult().GetJobRun();
                    Aws::Glue::Model::JobRunState jobRunState =
jobRun.GetJobRunState();

                    if ((jobRunState == Aws::Glue::Model::JobRunState::STOPPED)
||
                        (jobRunState == Aws::Glue::Model::JobRunState::FAILED) ||
                        (jobRunState == Aws::Glue::Model::JobRunState::TIMEOUT))
{
                        std::cerr << "Error running job. "
                                  << jobRun.GetErrorMessage()
                                  << std::endl;
                        deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME,
JOB_NAME,
                                     bucketName,
                                     clientConfig);
                        return false;
                    }
                    else if (jobRunState ==
                             Aws::Glue::Model::JobRunState::SUCCEEDED) {
                        std::cout << "Job run succeeded after  " << iterator <<
                                  " seconds elapsed." << std::endl;
                        done = true;
                    }
                    else if ((iterator % 10) == 0) { // Log status every 10
seconds.
                        std::cout << "Job run status " <<

Aws::Glue::Model::JobRunStateMapper::GetNameForJobRunState(
                                         jobRunState) <<
                                  ". " << iterator <<
                                  " seconds elapsed." << std::endl;
                    }
                }
                else {
                    std::cerr << "Error retrieving job run state. "
```

```
                            << jobRunOutcome.GetError().GetMessage()
                            << std::endl;
                    deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
                                 bucketName, clientConfig);
                    return false;
                }
            }
        }
        else {
            std::cerr << "Error starting a job. " <<
outcome.GetError().GetMessage()
                      << std::endl;
            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
bucketName,
                         clientConfig);
            return false;
        }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考StartJobRun中的。

CLI

AWS CLI

開始執行工作

以下 `start-job-run` 範例會啟動工作。

```
aws glue start-job-run \
    --job-name my-job
```

輸出：

```
{
    "JobRunId":
 "jr_22208b1f44eb5376a60569d4b21dd20fcb8621e1a366b4e7b2494af764b82ded"
}
```

如需詳細資訊，請參閱《AWS Glue 開發人員指南》中的授權工作。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考StartJobRun中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考StartJobRun中的。

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        $jobName = 'test-job-' . $uniqid;
```

```
        $databaseName = "doc-example-database-$uniqid";

        $tables = $glueService->getTables($databaseName);

        $outputBucketUrl = "s3://$bucketName";
        $runId = $glueService->startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl)['JobRunId'];

    public function startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl): Result
    {
        return $this->glueClient->startJobRun([
            'JobName' => $jobName,
            'Arguments' => [
                'input_database' => $databaseName,
                'input_table' => $tables['TableList'][0]['Name'],
                'output_bucket_url' => $outputBucketUrl,
                '--input_database' => $databaseName,
                '--input_table' => $tables['TableList'][0]['Name'],
                '--output_bucket_url' => $outputBucketUrl,
            ],
        ]);
    }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考StartJobRun中的。

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
```

```
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def start_job_run(self, name, input_database, input_table,
output_bucket_name):
        """
        Starts a job run. A job run extracts data from the source, transforms it,
        and loads it to the output bucket.

        :param name: The name of the job definition.
        :param input_database: The name of the metadata database that contains
tables
                               that describe the source data. This is typically
created
                               by a crawler.
        :param input_table: The name of the table in the metadata database that
                            describes the source data.
        :param output_bucket_name: The S3 bucket where the output is written.
        :return: The ID of the job run.
        """
        try:
            # The custom Arguments that are passed to this function are used by
the
            # Python ETL script to determine the location of input and output
data.
            response = self.glue_client.start_job_run(
                JobName=name,
                Arguments={
                    "--input_database": input_database,
                    "--input_table": input_table,
                    "--output_bucket_url": f"s3://{output_bucket_name}/",
                },
            )
        except ClientError as err:
            logger.error(
                "Couldn't start job run %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
```

```
            return response["JobRunId"]
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件StartJobRun中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Starts a job run for the specified job.
  #
  # @param name [String] The name of the job to start the run for.
  # @param input_database [String] The name of the input database for the job.
  # @param input_table [String] The name of the input table for the job.
  # @param output_bucket_name [String] The name of the output S3 bucket for the
 job.
  # @return [String] The ID of the started job run.
  def start_job_run(name, input_database, input_table, output_bucket_name)
    response = @glue_client.start_job_run(
      job_name: name,
      arguments: {
        '--input_database': input_database,
```

```
        '--input_table': input_table,
        '--output_bucket_url': "s3://#{output_bucket_name}/"
      }
    )
    response.job_run_id
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not start job run #{name}: \n#{e.message}")
    raise
  end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考StartJobRun中的。

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
        let job_run_output = glue
            .start_job_run()
            .job_name(self.job())
            .arguments("--input_database", self.database())
            .arguments(
                "--input_table",
                self.tables
                    .first()
                    .ok_or_else(|| GlueMvpError::Unknown("Missing crawler
table".into()))?
                    .name(),
            )
            .arguments("--output_bucket_url", self.bucket())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        let job = job_run_output
            .job_run_id()
```

```
            .ok_or_else(|| GlueMvpError::Unknown("Missing run id from just
started job".into()))?
            .to_string();
```

- 如需 API 的詳細資訊，請參閱 AWS SDK StartJobRun中的 Rust API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

AWS Glue 使用 AWS SDK 的案例

下列程式碼範例說明如何在 AWS SDK 中 AWS Glue 實作常見案例。這些案例會示範如何透過在其中呼叫多個函式來完成特定工作 AWS Glue。每個案例都包含一個連結 GitHub，您可以在其中找到如何設定和執行程式碼的指示。

範例

- 開始使用 SDK 執行 AWS Glue 搜尋器和工作 AWS

開始使用 SDK 執行 AWS Glue 搜尋器和工作 AWS

下列程式碼範例示範如何：

- 建立網路爬取公有 Amazon S3 儲存貯體的爬蟲程式，以及產生 CSV 格式中繼資料的資料庫。
- 列出有關 AWS Glue Data Catalog.
- 建立從 S3 儲存貯體中擷取 CSV 資料的任務、轉換資料，以及將 JSON 格式的輸出載入至另一個 S3 儲存貯體。
- 列出任務執行的相關資訊、檢視已轉換的資料以及清除資源。

如需詳細資訊，請參閱教學課程：開始使用 AWS Glue Studio。

.NET

AWS SDK for .NET

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

建立包裝案例中使用之 AWS Glue 函數的類別。

```
using System.Net;

namespace GlueActions;

public class GlueWrapper
{
    private readonly IAmazonGlue _amazonGlue;

    /// <summary>
    /// Constructor for the AWS Glue actions wrapper.
    /// </summary>
    /// <param name="amazonGlue"></param>
    public GlueWrapper(IAmazonGlue amazonGlue)
    {
        _amazonGlue = amazonGlue;
    }

    /// <summary>
    /// Create an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name for the crawler.</param>
    /// <param name="crawlerDescription">A description of the crawler.</param>
    /// <param name="role">The AWS Identity and Access Management (IAM) role to
    /// be assumed by the crawler.</param>
    /// <param name="schedule">The schedule on which the crawler will be
 executed.</param>
    /// <param name="s3Path">The path to the Amazon Simple Storage Service
 (Amazon S3)
    /// bucket where the Python script has been stored.</param>
    /// <param name="dbName">The name to use for the database that will be
```

```
    /// created by the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateCrawlerAsync(
        string crawlerName,
        string crawlerDescription,
        string role,
        string schedule,
        string s3Path,
        string dbName)
    {
        var s3Target = new S3Target
        {
            Path = s3Path,
        };

        var targetList = new List<S3Target>
        {
            s3Target,
        };

        var targets = new CrawlerTargets
        {
            S3Targets = targetList,
        };

        var crawlerRequest = new CreateCrawlerRequest
        {
            DatabaseName = dbName,
            Name = crawlerName,
            Description = crawlerDescription,
            Targets = targets,
            Role = role,
            Schedule = schedule,
        };

        var response = await _amazonGlue.CreateCrawlerAsync(crawlerRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Create an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
```

```
    /// <param name="roleName">The name of the IAM role to be assumed by
    /// the job.</param>
    /// <param name="description">A description of the job.</param>
    /// <param name="scriptUrl">The URL to the script.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateJobAsync(string dbName, string tableName,
string bucketUrl, string jobName, string roleName, string description, string
scriptUrl)
    {
        var command = new JobCommand
        {
            PythonVersion = "3",
            Name = "glueetl",
            ScriptLocation = scriptUrl,
        };

        var arguments = new Dictionary<string, string>
        {
            { "--input_database", dbName },
            { "--input_table", tableName },
            { "--output_bucket_url", bucketUrl }
        };

        var request = new CreateJobRequest
        {
            Command = command,
            DefaultArguments = arguments,
            Description = description,
            GlueVersion = "3.0",
            Name = jobName,
            NumberOfWorkers = 10,
            Role = roleName,
            WorkerType = "G.1X"
        };

        var response = await _amazonGlue.CreateJobAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
```

```
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteCrawlerAsync(string crawlerName)
    {
        var response = await _amazonGlue.DeleteCrawlerAsync(new
DeleteCrawlerRequest { Name = crawlerName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete the AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteDatabaseAsync(string dbName)
    {
        var response = await _amazonGlue.DeleteDatabaseAsync(new
DeleteDatabaseRequest { Name = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteJobAsync(string jobName)
    {
        var response = await _amazonGlue.DeleteJobAsync(new DeleteJobRequest
{ JobName = jobName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete a table from an AWS Glue database.
    /// </summary>
    /// <param name="tableName">The table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTableAsync(string dbName, string tableName)
    {
        var response = await _amazonGlue.DeleteTableAsync(new DeleteTableRequest
{ Name = tableName, DatabaseName = dbName });
```

```
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Get information about an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Crawler object describing the crawler.</returns>
    public async Task<Crawler?> GetCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new GetCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.GetCrawlerAsync(crawlerRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            var databaseName = response.Crawler.DatabaseName;
            Console.WriteLine($"{crawlerName} has the database {databaseName}");
            return response.Crawler;
        }

        Console.WriteLine($"No information regarding {crawlerName} could be
found.");
        return null;
    }


    /// <summary>
    /// Get information about the state of an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A value describing the state of the crawler.</returns>
    public async Task<CrawlerState> GetCrawlerStateAsync(string crawlerName)
    {
        var response = await _amazonGlue.GetCrawlerAsync(
            new GetCrawlerRequest { Name = crawlerName });
        return response.Crawler.State;
    }


    /// <summary>
```

```
    /// Get information about an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Database object containing information about the database.</
returns>
    public async Task<Database> GetDatabaseAsync(string dbName)
    {
        var databasesRequest = new GetDatabaseRequest
        {
            Name = dbName,
        };

        var response = await _amazonGlue.GetDatabaseAsync(databasesRequest);
        return response.Database;
    }


    /// <summary>
    /// Get information about a specific AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="jobRunId">The Id of the job run.</param>
    /// <returns>A JobRun object with information about the job run.</returns>
    public async Task<JobRun> GetJobRunAsync(string jobName, string jobRunId)
    {
        var response = await _amazonGlue.GetJobRunAsync(new GetJobRunRequest
 { JobName = jobName, RunId = jobRunId });
        return response.JobRun;
    }


    /// <summary>
    /// Get information about all AWS Glue runs of a specific job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A list of JobRun objects.</returns>
    public async Task<List<JobRun>> GetJobRunsAsync(string jobName)
    {
        var jobRuns = new List<JobRun>();

        var request = new GetJobRunsRequest
        {
            JobName = jobName,
        };
```

```
        // No need to loop to get all the log groups--the SDK does it for us
behind the scenes
        var paginatorForJobRuns =
            _amazonGlue.Paginators.GetJobRuns(request);

        await foreach (var response in paginatorForJobRuns.Responses)
        {
            response.JobRuns.ForEach(jobRun =>
            {
                jobRuns.Add(jobRun);
            });
        }

        return jobRuns;
    }


    /// <summary>
    /// Get a list of tables for an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A list of Table objects.</returns>
    public async Task<List<Table>> GetTablesAsync(string dbName)
    {
        var request = new GetTablesRequest { DatabaseName = dbName };
        var tables = new List<Table>();

        // Get a paginator for listing the tables.
        var tablePaginator = _amazonGlue.Paginators.GetTables(request);

        await foreach (var response in tablePaginator.Responses)
        {
            tables.AddRange(response.TableList);
        }

        return tables;
    }


    /// <summary>
    /// List AWS Glue jobs using a paginator.
    /// </summary>
    /// <returns>A list of AWS Glue job names.</returns>
```

```
    public async Task<List<string>> ListJobsAsync()
    {
        var jobNames = new List<string>();

        var listJobsPaginator = _amazonGlue.Paginators.ListJobs(new
ListJobsRequest { MaxResults = 10 });
        await foreach (var response in listJobsPaginator.Responses)
        {
            jobNames.AddRange(response.JobNames);
        }

        return jobNames;
    }


    /// <summary>
    /// Start an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> StartCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new StartCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.StartCrawlerAsync(crawlerRequest);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Start an AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A string representing the job run Id.</returns>
    public async Task<string> StartJobRunAsync(
        string jobName,
        string inputDatabase,
        string inputTable,
        string bucketName)
    {
```

```
        var request = new StartJobRunRequest
        {
            JobName = jobName,
            Arguments = new Dictionary<string, string>
            {
                {"--input_database", inputDatabase},
                {"--input_table", inputTable},
                {"--output_bucket_url", $"s3://{bucketName}/"}
            }
        };

        var response = await _amazonGlue.StartJobRunAsync(request);
        return response.JobRunId;
    }

}
```

建立可執行案例的類別。

```
global using Amazon.Glue;
global using GlueActions;
global using Microsoft.Extensions.Configuration;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;



using Amazon.Glue.Model;
using Amazon.S3;
using Amazon.S3.Model;

namespace GlueBasics;

public class GlueBasics
{
    private static ILogger logger = null!;
    private static IConfiguration _configuration = null!;
```

```
   static async Task Main(string[] args)
   {
       // Set up dependency injection for AWS Glue.
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
           .ConfigureServices((_, services) =>
           services.AddAWSService<IAmazonGlue>()
           .AddTransient<GlueWrapper>()
           .AddTransient<UiWrapper>()
           )
           .Build();

       logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
       .CreateLogger<GlueBasics>();

       _configuration = new ConfigurationBuilder()
           .SetBasePath(Directory.GetCurrentDirectory())
           .AddJsonFile("settings.json") // Load settings from .json file.
           .AddJsonFile("settings.local.json",
               true) // Optionally load local settings.
           .Build();

       // These values are stored in settings.json
       // Once you have run the CDK script to deploy the resources,
       // edit the file to set "BucketName", "RoleName", and "ScriptURL"
       // to the appropriate values. Also set "CrawlerName" to the name
       // you want to give the crawler when it is created.
       string bucketName = _configuration["BucketName"]!;
       string bucketUrl = _configuration["BucketUrl"]!;
       string crawlerName = _configuration["CrawlerName"]!;
       string roleName = _configuration["RoleName"]!;
       string sourceData = _configuration["SourceData"]!;
       string dbName = _configuration["DbName"]!;
       string cron = _configuration["Cron"]!;
       string scriptUrl = _configuration["ScriptURL"]!;
       string jobName = _configuration["JobName"]!;

       var wrapper = host.Services.GetRequiredService<GlueWrapper>();
       var uiWrapper = host.Services.GetRequiredService<UiWrapper>();
```

```
        uiWrapper.DisplayOverview();
        uiWrapper.PressEnter();

        // Create the crawler and wait for it to be ready.
        uiWrapper.DisplayTitle("Create AWS Glue crawler");
        Console.WriteLine("Let's begin by creating the AWS Glue crawler.");

        var crawlerDescription = "Crawler created for the AWS Glue Basics
scenario.";
        var crawlerCreated = await wrapper.CreateCrawlerAsync(crawlerName,
crawlerDescription, roleName, cron, sourceData, dbName);
        if (crawlerCreated)
        {
            Console.WriteLine($"The crawler: {crawlerName} has been created. Now
let's wait until it's ready.");
            CrawlerState crawlerState;
            do
            {
                crawlerState = await wrapper.GetCrawlerStateAsync(crawlerName);
            }
            while (crawlerState != "READY");
            Console.WriteLine($"The crawler {crawlerName} is now ready for
use.");
        }
        else
        {
            Console.WriteLine($"Couldn't create crawler {crawlerName}.");
            return; // Exit the application.
        }

        uiWrapper.DisplayTitle("Start AWS Glue crawler");
        Console.WriteLine("Now let's wait until the crawler has successfully
started.");
        var crawlerStarted = await wrapper.StartCrawlerAsync(crawlerName);
        if (crawlerStarted)
        {
            CrawlerState crawlerState;
            do
            {
                crawlerState = await wrapper.GetCrawlerStateAsync(crawlerName);
            }
            while (crawlerState != "READY");
```

```
            Console.WriteLine($"The crawler {crawlerName} is now ready for
use.");
        }
        else
        {
            Console.WriteLine($"Couldn't start the crawler {crawlerName}.");
            return; // Exit the application.
        }

        uiWrapper.PressEnter();

        Console.WriteLine($"\nLet's take a look at the database: {dbName}");
        var database = await wrapper.GetDatabaseAsync(dbName);

        if (database != null)
        {
            uiWrapper.DisplayTitle($"{database.Name} Details");
            Console.WriteLine($"{database.Name} created on
{database.CreateTime}");
            Console.WriteLine(database.Description);
        }

        uiWrapper.PressEnter();

        var tables = await wrapper.GetTablesAsync(dbName);
        if (tables.Count > 0)
        {
            tables.ForEach(table =>
            {
                Console.WriteLine($"{table.Name}\tCreated:
{table.CreateTime}\tUpdated: {table.UpdateTime}");
            });
        }

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Create AWS Glue job");
        Console.WriteLine("Creating a new AWS Glue job.");
        var description = "An AWS Glue job created using the AWS SDK for .NET";
        await wrapper.CreateJobAsync(dbName, tables[0].Name, bucketUrl, jobName,
roleName, description, scriptUrl);

        uiWrapper.PressEnter();
```

```
        uiWrapper.DisplayTitle("Starting AWS Glue job");
        Console.WriteLine("Starting the new AWS Glue job...");
        var jobRunId = await wrapper.StartJobRunAsync(jobName, dbName,
tables[0].Name, bucketName);
        var jobRunComplete = false;
        var jobRun = new JobRun();
        do
        {
            jobRun = await wrapper.GetJobRunAsync(jobName, jobRunId);
            if (jobRun.JobRunState == "SUCCEEDED" || jobRun.JobRunState ==
"STOPPED" ||
                jobRun.JobRunState == "FAILED" || jobRun.JobRunState ==
"TIMEOUT")
            {
                jobRunComplete = true;
            }
        } while (!jobRunComplete);

        uiWrapper.DisplayTitle($"Data in {bucketName}");

        // Get the list of data stored in the S3 bucket.
        var s3Client = new AmazonS3Client();

        var response = await s3Client.ListObjectsAsync(new ListObjectsRequest
{ BucketName = bucketName });
        response.S3Objects.ForEach(s3Object =>
        {
            Console.WriteLine(s3Object.Key);
        });

        uiWrapper.DisplayTitle("AWS Glue jobs");
        var jobNames = await wrapper.ListJobsAsync();
        jobNames.ForEach(jobName =>
        {
            Console.WriteLine(jobName);
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Get AWS Glue job run information");
        Console.WriteLine("Getting information about the AWS Glue job.");
        var jobRuns = await wrapper.GetJobRunsAsync(jobName);

        jobRuns.ForEach(jobRun =>
```

```
        {

 Console.WriteLine($"{jobRun.JobName}\t{jobRun.JobRunState}\t{jobRun.CompletedOn}");
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Deleting resources");
        Console.WriteLine("Deleting the AWS Glue job used by the example.");
        await wrapper.DeleteJobAsync(jobName);

        Console.WriteLine("Deleting the tables from the database.");
        tables.ForEach(async table =>
        {
            await wrapper.DeleteTableAsync(dbName, table.Name);
        });

        Console.WriteLine("Deleting the database.");
        await wrapper.DeleteDatabaseAsync(dbName);

        Console.WriteLine("Deleting the AWS Glue crawler.");
        await wrapper.DeleteCrawlerAsync(crawlerName);

        Console.WriteLine("The AWS Glue scenario has completed.");
        uiWrapper.PressEnter();
    }
}


namespace GlueBasics;

public class UiWrapper
{
    public readonly string SepBar = new string('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the scenario.
    /// </summary>
    public void DisplayOverview()
    {
        Console.Clear();
        DisplayTitle("Amazon Glue: get started with crawlers and jobs");

        Console.WriteLine("This example application does the following:");
```

```
        Console.WriteLine("\t 1. Create a crawler, pass it the IAM role and the
URL to the public S3 bucket that contains the source data");
        Console.WriteLine("\t 2. Start the crawler.");
        Console.WriteLine("\t 3. Get the database created by the crawler and the
tables in the database.");
        Console.WriteLine("\t 4. Create a job.");
        Console.WriteLine("\t 5. Start a job run.");
        Console.WriteLine("\t 6. Wait for the job run to complete.");
        Console.WriteLine("\t 7. Show the data stored in the bucket.");
        Console.WriteLine("\t 8. List jobs for the account.");
        Console.WriteLine("\t 9. Get job run details for the job that was run.");
        Console.WriteLine("\t10. Delete the demo job.");
        Console.WriteLine("\t11. Delete the database and tables created for the
demo.");
        Console.WriteLine("\t12. Delete the crawler.");
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPlease press <Enter> to continue. ");
        _ = Console.ReadLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to center on the screen.</param>
    /// <returns>The string padded to make it center on the screen.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
```

```
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for .NET API 參考》中的下列主題。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

C++

適用於 C++ 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
//! Scenario which demonstrates using AWS Glue to add a crawler and run a job.
/*!
 \\sa runGettingStartedWithGlueScenario()
 \param bucketName: An S3 bucket created in the setup.
 \param roleName: An AWS Identity and Access Management (IAM) role created in the
 setup.
 \param clientConfig: AWS client configuration.
 \return bool: Successful completion.
 */

bool AwsDoc::Glue::runGettingStartedWithGlueScenario(const Aws::String
 &bucketName,
                                                      const Aws::String &roleName,
                                                      const
 Aws::Client::ClientConfiguration &clientConfig) {
    Aws::Glue::GlueClient client(clientConfig);

    Aws::String roleArn;
    if (!getRoleArn(roleName, roleArn, clientConfig)) {
        std::cerr << "Error getting role ARN for role." << std::endl;
        return false;
    }

    // 1. Upload the job script to the S3 bucket.
    {
        std::cout << "Uploading the job script '"
                  << AwsDoc::Glue::PYTHON_SCRIPT
                  << "'." << std::endl;

        if (!AwsDoc::Glue::uploadFile(bucketName,
                                      AwsDoc::Glue::PYTHON_SCRIPT_PATH,
                                      AwsDoc::Glue::PYTHON_SCRIPT,
```

```
                                        clientConfig)) {
            std::cerr << "Error uploading the job file." << std::endl;
            return false;
        }
    }

    // 2. Create a crawler.
    {
        Aws::Glue::Model::S3Target s3Target;
        s3Target.SetPath("s3://crawler-public-us-east-1/flight/2016/csv");
        Aws::Glue::Model::CrawlerTargets crawlerTargets;
        crawlerTargets.AddS3Targets(s3Target);

        Aws::Glue::Model::CreateCrawlerRequest request;
        request.SetTargets(crawlerTargets);
        request.SetName(CRAWLER_NAME);
        request.SetDatabaseName(CRAWLER_DATABASE_NAME);
        request.SetTablePrefix(CRAWLER_DATABASE_PREFIX);
        request.SetRole(roleArn);

        Aws::Glue::Model::CreateCrawlerOutcome outcome =
client.CreateCrawler(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully created the crawler." << std::endl;
        }
        else {
            std::cerr << "Error creating a crawler. " <<
outcome.GetError().GetMessage()
                      << std::endl;
            deleteAssets("", CRAWLER_DATABASE_NAME, "", bucketName,
clientConfig);
            return false;
        }
    }

    // 3. Get a crawler.
    {
        Aws::Glue::Model::GetCrawlerRequest request;
        request.SetName(CRAWLER_NAME);

        Aws::Glue::Model::GetCrawlerOutcome outcome = client.GetCrawler(request);

        if (outcome.IsSuccess()) {
```

```
            Aws::Glue::Model::CrawlerState crawlerState =
outcome.GetResult().GetCrawler().GetState();
            std::cout << "Retrieved crawler with state " <<

Aws::Glue::Model::CrawlerStateMapper::GetNameForCrawlerState(
                              crawlerState)
                      << "." << std::endl;
        }
        else {
            std::cerr << "Error retrieving a crawler.  "
                      << outcome.GetError().GetMessage() << std::endl;
            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                         clientConfig);
            return false;
        }
    }

    // 4. Start a crawler.
    {
        Aws::Glue::Model::StartCrawlerRequest request;
        request.SetName(CRAWLER_NAME);

        Aws::Glue::Model::StartCrawlerOutcome outcome =
client.StartCrawler(request);


        if (outcome.IsSuccess() || (Aws::Glue::GlueErrors::CRAWLER_RUNNING ==
                                    outcome.GetError().GetErrorType())) {
            if (!outcome.IsSuccess()) {
                std::cout << "Crawler was already started." << std::endl;
            }
            else {
                std::cout << "Successfully started crawler." << std::endl;
            }

            std::cout << "This may take a while to run." << std::endl;

            Aws::Glue::Model::CrawlerState crawlerState =
Aws::Glue::Model::CrawlerState::NOT_SET;
            int iterations = 0;
            while (Aws::Glue::Model::CrawlerState::READY != crawlerState) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
                ++iterations;
                if ((iterations % 10) == 0) { // Log status every 10 seconds.
```

```
                    std::cout << "Crawler status " <<

Aws::Glue::Model::CrawlerStateMapper::GetNameForCrawlerState(
                                      crawlerState)
                              << ". After " << iterations
                              << " seconds elapsed."
                              << std::endl;
                }
                Aws::Glue::Model::GetCrawlerRequest getCrawlerRequest;
                getCrawlerRequest.SetName(CRAWLER_NAME);

                Aws::Glue::Model::GetCrawlerOutcome getCrawlerOutcome =
client.GetCrawler(
                        getCrawlerRequest);

                if (getCrawlerOutcome.IsSuccess()) {
                    crawlerState =
getCrawlerOutcome.GetResult().GetCrawler().GetState();
                }
                else {
                    std::cerr << "Error getting crawler.  "
                              << getCrawlerOutcome.GetError().GetMessage() <<
std::endl;
                    break;
                }
            }

            if (Aws::Glue::Model::CrawlerState::READY == crawlerState) {
                std::cout << "Crawler finished running after " << iterations
                          << " seconds."
                          << std::endl;
            }
        }
        else {
            std::cerr << "Error starting a crawler.  "
                      << outcome.GetError().GetMessage()
                      << std::endl;

            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                         clientConfig);
            return false;
        }
    }
```

```
   // 5. Get a database.
   {
       Aws::Glue::Model::GetDatabaseRequest request;
       request.SetName(CRAWLER_DATABASE_NAME);

       Aws::Glue::Model::GetDatabaseOutcome outcome =
client.GetDatabase(request);

       if (outcome.IsSuccess()) {
           const Aws::Glue::Model::Database &database =
outcome.GetResult().GetDatabase();

           std::cout << "Successfully retrieve the database\n" <<
                     database.Jsonize().View().WriteReadable() << "'." <<
std::endl;
       }
       else {
           std::cerr << "Error getting the database.  "
                     << outcome.GetError().GetMessage() << std::endl;
           deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                        clientConfig);
           return false;
       }
   }

   // 6. Get tables.
   Aws::String tableName;
   {
       Aws::Glue::Model::GetTablesRequest request;
       request.SetDatabaseName(CRAWLER_DATABASE_NAME);
       std::vector<Aws::Glue::Model::Table> all_tables;
       Aws::String nextToken; // Used for pagination.
       do {
           Aws::Glue::Model::GetTablesOutcome outcome =
client.GetTables(request);

           if (outcome.IsSuccess()) {
               const std::vector<Aws::Glue::Model::Table> &tables =
outcome.GetResult().GetTableList();
               all_tables.insert(all_tables.end(), tables.begin(),
tables.end());
               nextToken = outcome.GetResult().GetNextToken();
           }
           else {
```

```
                std::cerr << "Error getting the tables. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                             clientConfig);
                return false;
            }
        } while (!nextToken.empty());

        std::cout << "The database contains " << all_tables.size()
                  << (all_tables.size() == 1 ?
                      " table." : "tables.") << std::endl;
        std::cout << "Here is a list of the tables in the database.";
        for (size_t index = 0; index < all_tables.size(); ++index) {
            std::cout << "    " << index + 1 << ":  " <<
all_tables[index].GetName()
                      << std::endl;
        }

        if (!all_tables.empty()) {
            int tableIndex = askQuestionForIntRange(
                    "Enter an index to display the database detail ",
                    1, static_cast<int>(all_tables.size()));
            std::cout << all_tables[tableIndex -
1].Jsonize().View().WriteReadable()
                      << std::endl;

            tableName = all_tables[tableIndex - 1].GetName();
        }
    }

    // 7. Create a job.
    {
        Aws::Glue::Model::CreateJobRequest request;
        request.SetName(JOB_NAME);
        request.SetRole(roleArn);
        request.SetGlueVersion(GLUE_VERSION);

        Aws::Glue::Model::JobCommand command;
        command.SetName(JOB_COMMAND_NAME);
        command.SetPythonVersion(JOB_PYTHON_VERSION);
        command.SetScriptLocation(
                Aws::String("s3://") + bucketName + "/" + PYTHON_SCRIPT);
        request.SetCommand(command);
```

```
        Aws::Glue::Model::CreateJobOutcome outcome = client.CreateJob(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully created the job." << std::endl;
        }
        else {
            std::cerr << "Error creating the job. " <<
outcome.GetError().GetMessage()
                      << std::endl;
            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                         clientConfig);
            return false;
        }
    }

    // 8. Start a job run.
    {
        Aws::Glue::Model::StartJobRunRequest request;
        request.SetJobName(JOB_NAME);

        Aws::Map<Aws::String, Aws::String> arguments;
        arguments["--input_database"] = CRAWLER_DATABASE_NAME;
        arguments["--input_table"] = tableName;
        arguments["--output_bucket_url"] = Aws::String("s3://") + bucketName +
"/";
        request.SetArguments(arguments);

        Aws::Glue::Model::StartJobRunOutcome outcome =
client.StartJobRun(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully started the job." << std::endl;

            Aws::String jobRunId = outcome.GetResult().GetJobRunId();

            int iterator = 0;
            bool done = false;
            while (!done) {
                ++iterator;
                std::this_thread::sleep_for(std::chrono::seconds(1));
                Aws::Glue::Model::GetJobRunRequest jobRunRequest;
                jobRunRequest.SetJobName(JOB_NAME);
                jobRunRequest.SetRunId(jobRunId);
```

```
                Aws::Glue::Model::GetJobRunOutcome jobRunOutcome =
client.GetJobRun(
                        jobRunRequest);

                if (jobRunOutcome.IsSuccess()) {
                    const Aws::Glue::Model::JobRun &jobRun =
jobRunOutcome.GetResult().GetJobRun();
                    Aws::Glue::Model::JobRunState jobRunState =
jobRun.GetJobRunState();

                    if ((jobRunState == Aws::Glue::Model::JobRunState::STOPPED)
||
                        (jobRunState == Aws::Glue::Model::JobRunState::FAILED) ||
                        (jobRunState == Aws::Glue::Model::JobRunState::TIMEOUT))
{
                        std::cerr << "Error running job. "
                                  << jobRun.GetErrorMessage()
                                  << std::endl;
                        deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME,
JOB_NAME,
                                     bucketName,
                                     clientConfig);
                        return false;
                    }
                    else if (jobRunState ==
                             Aws::Glue::Model::JobRunState::SUCCEEDED) {
                        std::cout << "Job run succeeded after  " << iterator <<
                                  " seconds elapsed." << std::endl;
                        done = true;
                    }
                    else if ((iterator % 10) == 0) { // Log status every 10
seconds.
                        std::cout << "Job run status " <<

Aws::Glue::Model::JobRunStateMapper::GetNameForJobRunState(
                                         jobRunState) <<
                                  ". " << iterator <<
                                  " seconds elapsed." << std::endl;
                    }
                }
                else {
                    std::cerr << "Error retrieving job run state. "
                              << jobRunOutcome.GetError().GetMessage()
```

```
                            << std::endl;
                    deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
                                 bucketName, clientConfig);
                    return false;
                }
            }
        }
        else {
            std::cerr << "Error starting a job. " <<
outcome.GetError().GetMessage()
                      << std::endl;
            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
bucketName,
                         clientConfig);
            return false;
        }
    }

    // 9. List the output data stored in the S3 bucket.
    {
        Aws::S3::S3Client s3Client;
        Aws::S3::Model::ListObjectsV2Request request;
        request.SetBucket(bucketName);
        request.SetPrefix(OUTPUT_FILE_PREFIX);

        Aws::String continuationToken; // Used for pagination.
        std::vector<Aws::S3::Model::Object> allObjects;
        do {
            if (!continuationToken.empty()) {
                request.SetContinuationToken(continuationToken);
            }
            Aws::S3::Model::ListObjectsV2Outcome outcome =
s3Client.ListObjectsV2(
                    request);

            if (outcome.IsSuccess()) {
                const std::vector<Aws::S3::Model::Object> &objects =
                        outcome.GetResult().GetContents();
                allObjects.insert(allObjects.end(), objects.begin(),
objects.end());
                continuationToken =
outcome.GetResult().GetNextContinuationToken();
            }
            else {
```

```
                std::cerr << "Error listing objects. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
        } while (!continuationToken.empty());

        std::cout << "Data from your job is in " << allObjects.size() <<
                  " files in the S3 bucket, " << bucketName << "." << std::endl;

        for (size_t i = 0; i < allObjects.size(); ++i) {
            std::cout << "    " << i + 1 << ". " << allObjects[i].GetKey()
                      << std::endl;
        }

        int objectIndex = askQuestionForIntRange(
                std::string(
                        "Enter the number of a block to download it and see the
 first ") +
                std::to_string(LINES_OF_RUN_FILE_TO_DISPLAY) +
                " lines of JSON output in the block: ", 1,
                static_cast<int>(allObjects.size()));

        Aws::String objectKey = allObjects[objectIndex - 1].GetKey();

        std::stringstream stringStream;
        if (getObjectFromBucket(bucketName, objectKey, stringStream,
                                clientConfig)) {
            for (int i = 0; i < LINES_OF_RUN_FILE_TO_DISPLAY && stringStream; +
+i) {
                std::string line;
                std::getline(stringStream, line);
                std::cout << "    " << line << std::endl;
            }
        }
        else {
            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
 bucketName,
                         clientConfig);
            return false;
        }
    }

    // 10. List all the jobs.
```

```
    Aws::String jobName;
    {
        Aws::Glue::Model::ListJobsRequest listJobsRequest;

        Aws::String nextToken;
        std::vector<Aws::String> allJobNames;

        do {
            if (!nextToken.empty()) {
                listJobsRequest.SetNextToken(nextToken);
            }
            Aws::Glue::Model::ListJobsOutcome listRunsOutcome = client.ListJobs(
                    listJobsRequest);

            if (listRunsOutcome.IsSuccess()) {
                const std::vector<Aws::String> &jobNames =
listRunsOutcome.GetResult().GetJobNames();
                allJobNames.insert(allJobNames.end(), jobNames.begin(),
jobNames.end());
                nextToken = listRunsOutcome.GetResult().GetNextToken();
            }
            else {
                std::cerr << "Error listing jobs. "
                          << listRunsOutcome.GetError().GetMessage()
                          << std::endl;
            }
        } while (!nextToken.empty());
        std::cout << "Your account has " << allJobNames.size() << " jobs."
                  << std::endl;
        for (size_t i = 0; i < allJobNames.size(); ++i) {
            std::cout << "   " << i + 1 << ". " << allJobNames[i] << std::endl;
        }
        int jobIndex = askQuestionForIntRange(
                Aws::String("Enter a number between 1 and ") +
                std::to_string(allJobNames.size()) +
                " to see the list of runs for a job: ",
                1, static_cast<int>(allJobNames.size()));

        jobName = allJobNames[jobIndex - 1];
    }

    // 11. Get the job runs for a job.
    Aws::String jobRunID;
    if (!jobName.empty()) {
```

```
        Aws::Glue::Model::GetJobRunsRequest getJobRunsRequest;
        getJobRunsRequest.SetJobName(jobName);

        Aws::String nextToken; // Used for pagination.
        std::vector<Aws::Glue::Model::JobRun> allJobRuns;
        do {
            if (!nextToken.empty()) {
                getJobRunsRequest.SetNextToken(nextToken);
            }
            Aws::Glue::Model::GetJobRunsOutcome jobRunsOutcome =
client.GetJobRuns(
                    getJobRunsRequest);

            if (jobRunsOutcome.IsSuccess()) {
                const std::vector<Aws::Glue::Model::JobRun> &jobRuns =
jobRunsOutcome.GetResult().GetJobRuns();
                allJobRuns.insert(allJobRuns.end(), jobRuns.begin(),
jobRuns.end());

                nextToken = jobRunsOutcome.GetResult().GetNextToken();
            }
            else {
                std::cerr << "Error getting job runs. "
                          << jobRunsOutcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
        } while (!nextToken.empty());

        std::cout << "There are " << allJobRuns.size() << " runs in the job '"
                  <<
                  jobName << "'." << std::endl;

        for (size_t i = 0; i < allJobRuns.size(); ++i) {
            std::cout << "   " << i + 1 << ". " << allJobRuns[i].GetJobName()
                      << std::endl;
        }

        int runIndex = askQuestionForIntRange(
                Aws::String("Enter a number between 1 and ") +
                std::to_string(allJobRuns.size()) +
                " to see details for a run: ",
                1, static_cast<int>(allJobRuns.size()));
        jobRunID = allJobRuns[runIndex - 1].GetId();
```

```
    }

    // 12. Get a single job run.
    if (!jobRunID.empty()) {
        Aws::Glue::Model::GetJobRunRequest jobRunRequest;
        jobRunRequest.SetJobName(jobName);
        jobRunRequest.SetRunId(jobRunID);

        Aws::Glue::Model::GetJobRunOutcome jobRunOutcome = client.GetJobRun(
                jobRunRequest);

        if (jobRunOutcome.IsSuccess()) {
            std::cout << "Displaying the job run JSON description." << std::endl;
            std::cout
                    <<
 jobRunOutcome.GetResult().GetJobRun().Jsonize().View().WriteReadable()
                    << std::endl;
        }
        else {
            std::cerr << "Error get a job run. "
                      << jobRunOutcome.GetError().GetMessage()
                      << std::endl;
        }
    }

    return deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
 bucketName,
                        clientConfig);
}

//! Cleanup routine to delete created assets.
/*!
 \\sa deleteAssets()
 \param crawler: Name of an AWS Glue crawler.
 \param database: The name of an AWS Glue database.
 \param job: The name of an AWS Glue job.
 \param bucketName: The name of an S3 bucket.
 \param clientConfig: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::Glue::deleteAssets(const Aws::String &crawler, const Aws::String
 &database,
                                const Aws::String &job, const Aws::String
 &bucketName,
```

```
                                const Aws::Client::ClientConfiguration
&clientConfig) {
   const Aws::Glue::GlueClient client(clientConfig);
   bool result = true;

   // 13. Delete a job.
   if (!job.empty()) {
       Aws::Glue::Model::DeleteJobRequest request;
       request.SetJobName(job);

       Aws::Glue::Model::DeleteJobOutcome outcome = client.DeleteJob(request);


       if (outcome.IsSuccess()) {
           std::cout << "Successfully deleted the job." << std::endl;
       }
       else {
           std::cerr << "Error deleting the job. " <<
outcome.GetError().GetMessage()
                     << std::endl;
           result = false;
       }
   }

   // 14. Delete a database.
   if (!database.empty()) {
       Aws::Glue::Model::DeleteDatabaseRequest request;
       request.SetName(database);

       Aws::Glue::Model::DeleteDatabaseOutcome outcome = client.DeleteDatabase(
               request);

       if (outcome.IsSuccess()) {
           std::cout << "Successfully deleted the database." << std::endl;
       }
       else {
           std::cerr << "Error deleting database. " <<
outcome.GetError().GetMessage()
                     << std::endl;
           result = false;
       }
   }

   // 15. Delete a crawler.
```

```
    if (!crawler.empty()) {
        Aws::Glue::Model::DeleteCrawlerRequest request;
        request.SetName(crawler);

        Aws::Glue::Model::DeleteCrawlerOutcome outcome =
 client.DeleteCrawler(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully deleted the crawler." << std::endl;
        }
        else {
            std::cerr << "Error deleting the crawler. "
                      << outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
    }

    // 16. Delete the job script and run data from the S3 bucket.
    result &= AwsDoc::Glue::deleteAllObjectsInS3Bucket(bucketName,
                                                       clientConfig);
    return result;
}

//! Routine which uploads a file to an S3 bucket.
/*!
 \\sa uploadFile()
 \param bucketName: An S3 bucket created in the setup.
 \param filePath: The path of the file to upload.
 \param fileName The name for the uploaded file.
 \param clientConfig: AWS client configuration.
 \return bool: Successful completion.
 */
bool
AwsDoc::Glue::uploadFile(const Aws::String &bucketName,
                         const Aws::String &filePath,
                         const Aws::String &fileName,
                         const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::S3::S3Client s3_client(clientConfig);

    Aws::S3::Model::PutObjectRequest request;
    request.SetBucket(bucketName);
    request.SetKey(fileName);

    std::shared_ptr<Aws::IOStream> inputData =
```

```
            Aws::MakeShared<Aws::FStream>("SampleAllocationTag",
                                          filePath.c_str(),
                                          std::ios_base::in |
 std::ios_base::binary);

    if (!*inputData) {
        std::cerr << "Error unable to read file " << filePath << std::endl;
        return false;
    }

    request.SetBody(inputData);

    Aws::S3::Model::PutObjectOutcome outcome =
            s3_client.PutObject(request);

    if (!outcome.IsSuccess()) {
        std::cerr << "Error: PutObject: " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Added object '" << filePath << "' to bucket '"
                  << bucketName << "'." << std::endl;
    }

    return outcome.IsSuccess();
}

//! Routine which deletes all objects in an S3 bucket.
/*!
 \\sa deleteAllObjectsInS3Bucket()
 \param bucketName: The S3 bucket name.
 \param clientConfig: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::Glue::deleteAllObjectsInS3Bucket(const Aws::String &bucketName,
                                              const
 Aws::Client::ClientConfiguration &clientConfig) {
    Aws::S3::S3Client client(clientConfig);
    Aws::S3::Model::ListObjectsV2Request listObjectsRequest;
    listObjectsRequest.SetBucket(bucketName);

    Aws::String continuationToken; // Used for pagination.
    bool result = true;
    do {
```

```
        if (!continuationToken.empty()) {
            listObjectsRequest.SetContinuationToken(continuationToken);
        }

        Aws::S3::Model::ListObjectsV2Outcome listObjectsOutcome =
client.ListObjectsV2(
                listObjectsRequest);

        if (listObjectsOutcome.IsSuccess()) {
            const std::vector<Aws::S3::Model::Object> &objects =
listObjectsOutcome.GetResult().GetContents();
            if (!objects.empty()) {
                Aws::S3::Model::DeleteObjectsRequest deleteObjectsRequest;
                deleteObjectsRequest.SetBucket(bucketName);

                std::vector<Aws::S3::Model::ObjectIdentifier> objectIdentifiers;
                for (const Aws::S3::Model::Object &object: objects) {
                    objectIdentifiers.push_back(
                            Aws::S3::Model::ObjectIdentifier().WithKey(
                                    object.GetKey()));
                }
                Aws::S3::Model::Delete objectsDelete;
                objectsDelete.SetObjects(objectIdentifiers);
                objectsDelete.SetQuiet(true);
                deleteObjectsRequest.SetDelete(objectsDelete);

                Aws::S3::Model::DeleteObjectsOutcome deleteObjectsOutcome =
                        client.DeleteObjects(deleteObjectsRequest);

                if (!deleteObjectsOutcome.IsSuccess()) {
                    std::cerr << "Error deleting objects. " <<
                              deleteObjectsOutcome.GetError().GetMessage() <<
std::endl;
                    result = false;
                    break;
                }
                else {
                    std::cout << "Successfully deleted the objects." <<
std::endl;

                }
            }
            else {
                std::cout << "No objects to delete in '" << bucketName << "'."
```

```
                          << std::endl;
            }

            continuationToken =
 listObjectsOutcome.GetResult().GetNextContinuationToken();
        }
        else {
            std::cerr << "Error listing objects. "
                      << listObjectsOutcome.GetError().GetMessage() << std::endl;
            result = false;
            break;
        }
    } while (!continuationToken.empty());

    return result;
}

//! Routine which retrieves an object from an S3 bucket.
/*!
 \\sa getObjectFromBucket()
 \param bucketName: The S3 bucket name.
 \param objectKey: The object's name.
 \param objectStream: A stream to receive the retrieved data.
 \param clientConfig: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::Glue::getObjectFromBucket(const Aws::String &bucketName,
                                       const Aws::String &objectKey,
                                       std::ostream &objectStream,
                                       const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::S3::S3Client client(clientConfig);
    Aws::S3::Model::GetObjectRequest request;
    request.SetBucket(bucketName);
    request.SetKey(objectKey);

    Aws::S3::Model::GetObjectOutcome outcome = client.GetObject(request);


    if (outcome.IsSuccess()) {
        std::cout << "Successfully retrieved '" << objectKey << "'." <<
 std::endl;
        auto &body = outcome.GetResult().GetBody();
        objectStream << body.rdbuf();
```

```
    }
    else {
        std::cerr << "Error retrieving object. " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for C++ API 參考》中的下列主題。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

Java

適用於 Java 2.x 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
/**
 *
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To set up the resources, see this documentation topic:
 *
 * https://docs.aws.amazon.com/glue/latest/ug/tutorial-add-crawler.html
 *
 * This example performs the following tasks:
 *
 * 1. Create a database.
 * 2. Create a crawler.
 * 3. Get a crawler.
 * 4. Start a crawler.
 * 5. Get a database.
 * 6. Get tables.
 * 7. Create a job.
 * 8. Start a job run.
 * 9. List all jobs.
 * 10. Get job runs.
 * 11. Delete a job.
 * 12. Delete a database.
 * 13. Delete a crawler.
 */

public class GlueScenario {
```

```
   public static final String DASHES = new String(new char[80]).replace("\0",
"-");

   public static void main(String[] args) throws InterruptedException {
       final String usage = """

               Usage:
                   <iam> <s3Path> <cron> <dbName> <crawlerName> <jobName>\s

               Where:
                   iam - The ARN of the IAM role that has AWS Glue and S3
permissions.\s
                   s3Path - The Amazon Simple Storage Service (Amazon S3) target
that contains data (for example, CSV data).
                   cron - A cron expression used to specify the schedule  (i.e.,
cron(15 12 * * ? *).
                   dbName - The database name.\s
                   crawlerName - The name of the crawler.\s
                   jobName - The name you assign to this job definition.
                   scriptLocation - The Amazon S3 path to a script that runs a
job.
                   locationUri - The location of the database
                   bucketNameSc - The Amazon S3 bucket name used when creating a
job
                   """;

       if (args.length != 9) {
           System.out.println(usage);
           System.exit(1);
       }

       String iam = args[0];
       String s3Path = args[1];
       String cron = args[2];
       String dbName = args[3];
       String crawlerName = args[4];
       String jobName = args[5];
       String scriptLocation = args[6];
       String locationUri = args[7];
       String bucketNameSc = args[8];

       Region region = Region.US_EAST_1;
       GlueClient glueClient = GlueClient.builder()
               .region(region)
```

```
                .build();
        System.out.println(DASHES);
        System.out.println("Welcome to the AWS Glue scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("1. Create a database.");
        createDatabase(glueClient, dbName, locationUri);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Create a crawler.");
        createGlueCrawler(glueClient, iam, s3Path, cron, dbName, crawlerName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Get a crawler.");
        getSpecificCrawler(glueClient, crawlerName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Start a crawler.");
        startSpecificCrawler(glueClient, crawlerName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Get a database.");
        getSpecificDatabase(glueClient, dbName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("*** Wait 5 min for the tables to become available");
        TimeUnit.MINUTES.sleep(5);
        System.out.println("6. Get tables.");
        String myTableName = getGlueTables(glueClient, dbName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Create a job.");
        createJob(glueClient, jobName, iam, scriptLocation);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Start a Job run.");
```

```
        startJob(glueClient, jobName, dbName, myTableName, bucketNameSc);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. List all jobs.");
        getAllJobs(glueClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Get job runs.");
        getJobRuns(glueClient, jobName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Delete a job.");
        deleteJob(glueClient, jobName);
        System.out.println("*** Wait 5 MIN for the " + crawlerName + " to stop");
        TimeUnit.MINUTES.sleep(5);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Delete a database.");
        deleteDatabase(glueClient, dbName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("Delete a crawler.");
        deleteSpecificCrawler(glueClient, crawlerName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("Successfully completed the AWS Glue Scenario");
        System.out.println(DASHES);
    }

    public static void createDatabase(GlueClient glueClient, String dbName,
String locationUri) {
        try {
            DatabaseInput input = DatabaseInput.builder()
                    .description("Built with the AWS SDK for Java V2")
                    .name(dbName)
                    .locationUri(locationUri)
                    .build();
```

```
            CreateDatabaseRequest request = CreateDatabaseRequest.builder()
                    .databaseInput(input)
                    .build();

            glueClient.createDatabase(request);
            System.out.println(dbName + " was successfully created");

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void createGlueCrawler(GlueClient glueClient,
            String iam,
            String s3Path,
            String cron,
            String dbName,
            String crawlerName) {

        try {
            S3Target s3Target = S3Target.builder()
                    .path(s3Path)
                    .build();

            List<S3Target> targetList = new ArrayList<>();
            targetList.add(s3Target);
            CrawlerTargets targets = CrawlerTargets.builder()
                    .s3Targets(targetList)
                    .build();

            CreateCrawlerRequest crawlerRequest = CreateCrawlerRequest.builder()
                    .databaseName(dbName)
                    .name(crawlerName)
                    .description("Created by the AWS Glue Java API")
                    .targets(targets)
                    .role(iam)
                    .schedule(cron)
                    .build();

            glueClient.createCrawler(crawlerRequest);
            System.out.println(crawlerName + " was successfully created");

        } catch (GlueException e) {
```

```
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void getSpecificCrawler(GlueClient glueClient, String
crawlerName) {
        try {
            GetCrawlerRequest crawlerRequest = GetCrawlerRequest.builder()
                    .name(crawlerName)
                    .build();

            boolean ready = false;
            while (!ready) {
                GetCrawlerResponse response =
glueClient.getCrawler(crawlerRequest);
                String status = response.crawler().stateAsString();
                if (status.compareTo("READY") == 0) {
                    ready = true;
                }
                Thread.sleep(3000);
            }

            System.out.println("The crawler is now ready");

        } catch (GlueException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void startSpecificCrawler(GlueClient glueClient, String
crawlerName) {
        try {
            StartCrawlerRequest crawlerRequest = StartCrawlerRequest.builder()
                    .name(crawlerName)
                    .build();

            glueClient.startCrawler(crawlerRequest);
            System.out.println(crawlerName + " was successfully started!");

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
```

```
        }
    }

    public static void getSpecificDatabase(GlueClient glueClient, String
databaseName) {
        try {
            GetDatabaseRequest databasesRequest = GetDatabaseRequest.builder()
                    .name(databaseName)
                    .build();

            GetDatabaseResponse response =
glueClient.getDatabase(databasesRequest);
            Instant createDate = response.database().createTime();

            // Convert the Instant to readable date.
            DateTimeFormatter formatter =
DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
                    .withLocale(Locale.US)
                    .withZone(ZoneId.systemDefault());

            formatter.format(createDate);
            System.out.println("The create date of the database is " +
createDate);

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static String getGlueTables(GlueClient glueClient, String dbName) {
        String myTableName = "";
        try {
            GetTablesRequest tableRequest = GetTablesRequest.builder()
                    .databaseName(dbName)
                    .build();

            GetTablesResponse response = glueClient.getTables(tableRequest);
            List<Table> tables = response.tableList();
            if (tables.isEmpty()) {
                System.out.println("No tables were returned");
            } else {
                for (Table table : tables) {
                    myTableName = table.name();
```

```
                    System.out.println("Table name is: " + myTableName);
                }
            }

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return myTableName;
    }

    public static void startJob(GlueClient glueClient, String jobName, String
inputDatabase, String inputTable,
            String outBucket) {
        try {
            Map<String, String> myMap = new HashMap<>();
            myMap.put("--input_database", inputDatabase);
            myMap.put("--input_table", inputTable);
            myMap.put("--output_bucket_url", outBucket);

            StartJobRunRequest runRequest = StartJobRunRequest.builder()
                    .workerType(WorkerType.G_1_X)
                    .numberOfWorkers(10)
                    .arguments(myMap)
                    .jobName(jobName)
                    .build();

            StartJobRunResponse response = glueClient.startJobRun(runRequest);
            System.out.println("The request Id of the job is " +
response.responseMetadata().requestId());

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void createJob(GlueClient glueClient, String jobName, String
iam, String scriptLocation) {
        try {
            JobCommand command = JobCommand.builder()
                    .pythonVersion("3")
                    .name("glueetl")
                    .scriptLocation(scriptLocation)
```

```
                    .build();

            CreateJobRequest jobRequest = CreateJobRequest.builder()
                    .description("A Job created by using the AWS SDK for Java
V2")
                    .glueVersion("2.0")
                    .workerType(WorkerType.G_1_X)
                    .numberOfWorkers(10)
                    .name(jobName)
                    .role(iam)
                    .command(command)
                    .build();

            glueClient.createJob(jobRequest);
            System.out.println(jobName + " was successfully created.");

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void getAllJobs(GlueClient glueClient) {
        try {
            GetJobsRequest jobsRequest = GetJobsRequest.builder()
                    .maxResults(10)
                    .build();

            GetJobsResponse jobsResponse = glueClient.getJobs(jobsRequest);
            List<Job> jobs = jobsResponse.jobs();
            for (Job job : jobs) {
                System.out.println("Job name is : " + job.name());
                System.out.println("The job worker type is : " +
job.workerType().name());
            }

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void getJobRuns(GlueClient glueClient, String jobName) {
        try {
```

```
            GetJobRunsRequest runsRequest = GetJobRunsRequest.builder()
                    .jobName(jobName)
                    .maxResults(20)
                    .build();

            boolean jobDone = false;
            while (!jobDone) {
                GetJobRunsResponse response = glueClient.getJobRuns(runsRequest);
                List<JobRun> jobRuns = response.jobRuns();
                for (JobRun jobRun : jobRuns) {
                    String jobState = jobRun.jobRunState().name();
                    if (jobState.compareTo("SUCCEEDED") == 0) {
                        System.out.println(jobName + " has succeeded");
                        jobDone = true;

                    } else if (jobState.compareTo("STOPPED") == 0) {
                        System.out.println("Job run has stopped");
                        jobDone = true;

                    } else if (jobState.compareTo("FAILED") == 0) {
                        System.out.println("Job run has failed");
                        jobDone = true;

                    } else if (jobState.compareTo("TIMEOUT") == 0) {
                        System.out.println("Job run has timed out");
                        jobDone = true;

                    } else {
                        System.out.println("*** Job run state is " +
jobRun.jobRunState().name());
                        System.out.println("Job run Id is " + jobRun.id());
                        System.out.println("The Glue version is " +
jobRun.glueVersion());
                    }
                    TimeUnit.SECONDS.sleep(5);
                }
            }

        } catch (GlueException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

```
    public static void deleteJob(GlueClient glueClient, String jobName) {
        try {
            DeleteJobRequest jobRequest = DeleteJobRequest.builder()
                    .jobName(jobName)
                    .build();

            glueClient.deleteJob(jobRequest);
            System.out.println(jobName + " was successfully deleted");

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteDatabase(GlueClient glueClient, String databaseName)
{
        try {
            DeleteDatabaseRequest request = DeleteDatabaseRequest.builder()
                    .name(databaseName)
                    .build();

            glueClient.deleteDatabase(request);
            System.out.println(databaseName + " was successfully deleted");

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteSpecificCrawler(GlueClient glueClient, String
crawlerName) {
        try {
            DeleteCrawlerRequest deleteCrawlerRequest =
DeleteCrawlerRequest.builder()
                    .name(crawlerName)
                    .build();

            glueClient.deleteCrawler(deleteCrawlerRequest);
            System.out.println(crawlerName + " was deleted");

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
```

```
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

JavaScript

適用於 JavaScript (v3) 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

建立並執行可網路爬取公有 Amazon Simple Storage Service (Amazon S3) 儲存貯體的爬蟲程式，並產生描述其所尋找 CSV 格式資料的中繼資料的資料庫。

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};

const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const crawlerExists = async ({ getCrawler }, crawlerName) => {
  try {
    await getCrawler(crawlerName);
    return true;
```

```
  } catch {
    return false;
  }
};

/**
 * @param {{ createCrawler: import('../../../actions/create-
crawler.js').createCrawler}} actions
 */
const makeCreateCrawlerStep = (actions) => async (context) => {
  if (await crawlerExists(actions, process.env.CRAWLER_NAME)) {
    log("Crawler already exists. Skipping creation.");
  } else {
    await actions.createCrawler(
      process.env.CRAWLER_NAME,
      process.env.ROLE_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_PREFIX,
      process.env.S3_TARGET_PATH,
    );

    log("Crawler created successfully.", { type: "success" });
  }

  return { ...context };
};

/**
 * @param {(name: string) => Promise<import('@aws-sdk/client-
glue').GetCrawlerCommandOutput>} getCrawler
 * @param {string} crawlerName
 */
const waitForCrawler = async (getCrawler, crawlerName) => {
  const waitTimeInSeconds = 30;
  const { Crawler } = await getCrawler(crawlerName);

  if (!Crawler) {
    throw new Error(`Crawler with name ${crawlerName} not found.`);
  }

  if (Crawler.State === "READY") {
    return;
  }
```

```
  log(`Crawler is ${Crawler.State}. Waiting ${waitTimeInSeconds} seconds...`);
  await wait(waitTimeInSeconds);
  return waitForCrawler(getCrawler, crawlerName);
};

const makeStartCrawlerStep =
  ({ startCrawler, getCrawler }) =>
  async (context) => {
    log("Starting crawler.");
    await startCrawler(process.env.CRAWLER_NAME);
    log("Crawler started.", { type: "success" });

    log("Waiting for crawler to finish running. This can take a while.");
    await waitForCrawler(getCrawler, process.env.CRAWLER_NAME);
    log("Crawler ready.", { type: "success" });

    return { ...context };
  };
```

列出有關 AWS Glue Data Catalog.

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};

const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};

const makeGetDatabaseStep =
```

```
  ({ getDatabase }) =>
  async (context) => {
    const {
      Database: { Name },
    } = await getDatabase(process.env.DATABASE_NAME);
    log(`Database: ${Name}`);
    return { ...context };
  };

/**
 * @param {{ getTables: () => Promise<import('@aws-sdk/client-
glue').GetTablesCommandOutput}} config
 */
const makeGetTablesStep =
  ({ getTables }) =>
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME);
    log("Tables:");
    log(TableList.map((table) => `  • ${table.Name}\n`));
    return { ...context };
  };
```

建立並執行從來源 Amazon S3 儲存貯體中擷取 CSV 資料的任務、透過移除和重新命名欄位進行轉換，以及將 JSON 格式的輸出載入另一個 Amazon S3 儲存貯體。

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};
```

```
const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};

const makeCreateJobStep =
  ({ createJob }) =>
  async (context) => {
    log("Creating Job.");
    await createJob(
      process.env.JOB_NAME,
      process.env.ROLE_NAME,
      process.env.BUCKET_NAME,
      process.env.PYTHON_SCRIPT_KEY,
    );
    log("Job created.", { type: "success" });

    return { ...context };
  };

/**
 * @param {(name: string, runId: string) => Promise<import('@aws-sdk/client-
glue').GetJobRunCommandOutput> }  getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const waitForJobRun = async (getJobRun, jobName, jobRunId) => {
  const waitTimeInSeconds = 30;
  const { JobRun } = await getJobRun(jobName, jobRunId);

  if (!JobRun) {
    throw new Error(`Job run with id ${jobRunId} not found.`);
  }
```

```
  switch (JobRun.JobRunState) {
    case "FAILED":
    case "TIMEOUT":
    case "STOPPED":
      throw new Error(
        `Job ${JobRun.JobRunState}. Error: ${JobRun.ErrorMessage}`,
      );
    case "RUNNING":
      break;
    case "SUCCEEDED":
      return;
    default:
      throw new Error(`Unknown job run state: ${JobRun.JobRunState}`);
  }

  log(
    `Job ${JobRun.JobRunState}. Waiting ${waitTimeInSeconds} more seconds...`,
  );
  await wait(waitTimeInSeconds);
  return waitForJobRun(getJobRun, jobName, jobRunId);
};

/**
 * @param {{ prompter: { prompt: () => Promise<{ shouldOpen: boolean }>} }}
 context
 */
const promptToOpen = async (context) => {
  const { shouldOpen } = await context.prompter.prompt({
    name: "shouldOpen",
    type: "confirm",
    message: "Open the output bucket in your browser?",
  });

  if (shouldOpen) {
    return open(
      `https://s3.console.aws.amazon.com/s3/buckets/${process.env.BUCKET_NAME} to
 view the output.`,
    );
  }
};

const makeStartJobRunStep =
  ({ startJobRun, getJobRun }) =>
  async (context) => {
```

```
    log("Starting job.");
    const { JobRunId } = await startJobRun(
      process.env.JOB_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_NAME,
      process.env.BUCKET_NAME,
    );
    log("Job started.", { type: "success" });

    log("Waiting for job to finish running. This can take a while.");
    await waitForJobRun(getJobRun, process.env.JOB_NAME, JobRunId);
    log("Job run succeeded.", { type: "success" });

    await promptToOpen(context);

    return { ...context };
  };
```

列出任務執行的相關資訊，並檢視部分轉換的資料。

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};

const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};

/**
 * @typedef {{ prompter: { prompt: () => Promise<{jobName: string}> } }} Context
 */
```

```
/**
 * @typedef {() => Promise<import('@aws-sdk/client-
glue').GetJobRunCommandOutput>} getJobRun
 */

/**
 * @typedef {() => Promise<import('@aws-sdk/client-
glue').GetJobRunsCommandOutput} getJobRuns
 */

/**
 *
 * @param {getJobRun} getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const logJobRunDetails = async (getJobRun, jobName, jobRunId) => {
  const { JobRun } = await getJobRun(jobName, jobRunId);
  log(JobRun, { type: "object" });
};

/**
 *
 * @param {{getJobRuns: getJobRuns, getJobRun: getJobRun }} funcs
 */
const makePickJobRunStep =
  ({ getJobRuns, getJobRun }) =>
  async (/** @type { Context } */ context) => {
    if (context.selectedJobName) {
      const { JobRuns } = await getJobRuns(context.selectedJobName);

      const { jobRunId } = await context.prompter.prompt({
        name: "jobRunId",
        type: "list",
        message: "Select a job run to see details.",
        choices: JobRuns.map((run) => run.Id),
      });

      logJobRunDetails(getJobRun, context.selectedJobName, jobRunId);
    }

    return { ...context };
  };
```

刪除透過示範建立的所有資源。

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};

const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};

const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};

const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
```

```
};

/**
 *
 * @param {import('../../../actions/delete-job.js').deleteJob} deleteJobFn
 * @param {string[]} jobNames
 * @param {{ prompter: { prompt: () => Promise<any> }}} context
 */
const handleDeleteJobs = async (deleteJobFn, jobNames, context) => {
  /**
   * @type {{ selectedJobNames: string[] }}
   */
  const { selectedJobNames } = await context.prompter.prompt({
    name: "selectedJobNames",
    type: "checkbox",
    message: "Let's clean up jobs. Select jobs to delete.",
    choices: jobNames,
  });

  if (selectedJobNames.length === 0) {
    log("No jobs selected.");
  } else {
    log("Deleting jobs.");
    await Promise.all(
      selectedJobNames.map((n) => deleteJobFn(n).catch(console.error)),
    );
    log("Jobs deleted.", { type: "success" });
  }
};

/**
 * @param {{
 *   listJobs: import('../../../actions/list-jobs.js').listJobs,
 *   deleteJob: import('../../../actions/delete-job.js').deleteJob
 * }} config
 */
const makeCleanUpJobsStep =
  ({ listJobs, deleteJob }) =>
  async (context) => {
    const { JobNames } = await listJobs();
    if (JobNames.length > 0) {
      await handleDeleteJobs(deleteJob, JobNames, context);
    }
```

```
    return { ...context };
  };

/**
 * @param {import('../../../actions/delete-table.js').deleteTable} deleteTable
 * @param {string} databaseName
 * @param {string[]} tableNames
 */
const deleteTables = (deleteTable, databaseName, tableNames) =>
  Promise.all(
    tableNames.map((tableName) =>
      deleteTable(databaseName, tableName).catch(console.error),
    ),
  );

/**
 * @param {{
 *  getTables: import('../../../actions/get-tables.js').getTables,
 *  deleteTable: import('../../../actions/delete-table.js').deleteTable
 * }} config
 */
const makeCleanUpTablesStep =
  ({ getTables, deleteTable }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any>}}} context
   */
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME).catch(
      () => ({ TableList: null }),
    );

    if (TableList && TableList.length > 0) {
      /**
       * @type {{ tableNames: string[] }}
       */
      const { tableNames } = await context.prompter.prompt({
        name: "tableNames",
        type: "checkbox",
        message: "Let's clean up tables. Select tables to delete.",
        choices: TableList.map((t) => t.Name),
      });

      if (tableNames.length === 0) {
        log("No tables selected.");
```

```
      } else {
        log("Deleting tables.");
        await deleteTables(deleteTable, process.env.DATABASE_NAME, tableNames);
        log("Tables deleted.", { type: "success" });
      }
    }

    return { ...context };
  };

/**
 * @param {import('../../../actions/delete-database.js').deleteDatabase}
 deleteDatabase
 * @param {string[]} databaseNames
 */
const deleteDatabases = (deleteDatabase, databaseNames) =>
  Promise.all(
    databaseNames.map((dbName) => deleteDatabase(dbName).catch(console.error)),
  );

/**
 * @param {{
 *   getDatabases: import('../../../actions/get-databases.js').getDatabases
 *   deleteDatabase: import('../../../actions/delete-database.js').deleteDatabase
 * }} config
 */
const makeCleanUpDatabasesStep =
  ({ getDatabases, deleteDatabase }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any>}} context
   */
  async (context) => {
    const { DatabaseList } = await getDatabases();

    if (DatabaseList.length > 0) {
      /** @type {{ dbNames: string[] }} */
      const { dbNames } = await context.prompter.prompt({
        name: "dbNames",
        type: "checkbox",
        message: "Let's clean up databases. Select databases to delete.",
        choices: DatabaseList.map((db) => db.Name),
      });

      if (dbNames.length === 0) {
```

```
        log("No databases selected.");
      } else {
        log("Deleting databases.");
        await deleteDatabases(deleteDatabase, dbNames);
        log("Databases deleted.", { type: "success" });
      }
    }

    return { ...context };
  };

const cleanUpCrawlerStep = async (context) => {
  log(`Deleting crawler.`);

  try {
    await deleteCrawler(process.env.CRAWLER_NAME);
    log("Crawler deleted.", { type: "success" });
  } catch (err) {
    if (err.name === "EntityNotFoundException") {
      log(`Crawler is already deleted.`);
    } else {
      throw err;
    }
  }

  return { ...context };
};
```

- 如需 API 詳細資訊，請參閱《AWS SDK for JavaScript API 參考》中的下列主題。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases

- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <iam> <s3Path> <cron> <dbName> <crawlerName> <jobName>
 <scriptLocation> <locationUri>

        Where:
            iam - The Amazon Resource Name (ARN) of the AWS Identity and Access
 Management (IAM) role that has AWS Glue and Amazon Simple Storage Service
 (Amazon S3) permissions.
            s3Path - The Amazon Simple Storage Service (Amazon S3) target that
 contains data (for example, CSV data).
            cron - A cron expression used to specify the schedule (for example,
 cron(15 12 * * ? *).
            dbName - The database name.
            crawlerName - The name of the crawler.
            jobName - The name you assign to this job definition.
            scriptLocation - Specifies the Amazon S3 path to a script that runs a
 job.
            locationUri - Specifies the location of the database
        """
```

```
    if (args.size != 8) {
        println(usage)
        exitProcess(1)
    }

    val iam = args[0]
    val s3Path = args[1]
    val cron = args[2]
    val dbName = args[3]
    val crawlerName = args[4]
    val jobName = args[5]
    val scriptLocation = args[6]
    val locationUri = args[7]

    println("About to start the AWS Glue Scenario")
    createDatabase(dbName, locationUri)
    createCrawler(iam, s3Path, cron, dbName, crawlerName)
    getCrawler(crawlerName)
    startCrawler(crawlerName)
    getDatabase(dbName)
    getGlueTables(dbName)
    createJob(jobName, iam, scriptLocation)
    startJob(jobName)
    getJobs()
    getJobRuns(jobName)
    deleteJob(jobName)
    println("*** Wait for 5 MIN so the $crawlerName is ready to be deleted")
    TimeUnit.MINUTES.sleep(5)
    deleteMyDatabase(dbName)
    deleteCrawler(crawlerName)
}

suspend fun createDatabase(
    dbName: String?,
    locationUriVal: String?,
) {
    val input =
        DatabaseInput {
            description = "Built with the AWS SDK for Kotlin"
            name = dbName
            locationUri = locationUriVal
        }
```

```
    val request =
        CreateDatabaseRequest {
            databaseInput = input
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.createDatabase(request)
        println("The database was successfully created")
    }
}

suspend fun createCrawler(
    iam: String?,
    s3Path: String?,
    cron: String?,
    dbName: String?,
    crawlerName: String,
) {
    val s3Target =
        S3Target {
            path = s3Path
        }

    val targetList = ArrayList<S3Target>()
    targetList.add(s3Target)

    val targetOb =
        CrawlerTargets {
            s3Targets = targetList
        }

    val crawlerRequest =
        CreateCrawlerRequest {
            databaseName = dbName
            name = crawlerName
            description = "Created by the AWS Glue Java API"
            targets = targetOb
            role = iam
            schedule = cron
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.createCrawler(crawlerRequest)
        println("$crawlerName was successfully created")
```

```
    }
}

suspend fun getCrawler(crawlerName: String?) {
    val request =
        GetCrawlerRequest {
            name = crawlerName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getCrawler(request)
        val role = response.crawler?.role
        println("The role associated with this crawler is $role")
    }
}

suspend fun startCrawler(crawlerName: String) {
    val crawlerRequest =
        StartCrawlerRequest {
            name = crawlerName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.startCrawler(crawlerRequest)
        println("$crawlerName was successfully started.")
    }
}

suspend fun getDatabase(databaseName: String?) {
    val request =
        GetDatabaseRequest {
            name = databaseName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getDatabase(request)
        val dbDesc = response.database?.description
        println("The database description is $dbDesc")
    }
}

suspend fun getGlueTables(dbName: String?) {
    val tableRequest =
        GetTablesRequest {
```

```
            databaseName = dbName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getTables(tableRequest)
        response.tableList?.forEach { tableName ->
            println("Table name is ${tableName.name}")
        }
    }
}

suspend fun startJob(jobNameVal: String?) {
    val runRequest =
        StartJobRunRequest {
            workerType = WorkerType.G1X
            numberOfWorkers = 10
            jobName = jobNameVal
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.startJobRun(runRequest)
        println("The job run Id is ${response.jobRunId}")
    }
}

suspend fun createJob(
    jobName: String,
    iam: String?,
    scriptLocationVal: String?,
) {
    val commandOb =
        JobCommand {
            pythonVersion = "3"
            name = "MyJob1"
            scriptLocation = scriptLocationVal
        }

    val jobRequest =
        CreateJobRequest {
            description = "A Job created by using the AWS SDK for Java V2"
            glueVersion = "2.0"
            workerType = WorkerType.G1X
            numberOfWorkers = 10
            name = jobName
```

```
            role = iam
            command = commandOb
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.createJob(jobRequest)
        println("$jobName was successfully created.")
    }
}

suspend fun getJobs() {
    val request =
        GetJobsRequest {
            maxResults = 10
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getJobs(request)
        response.jobs?.forEach { job ->
            println("Job name is ${job.name}")
        }
    }
}

suspend fun getJobRuns(jobNameVal: String?) {
    val request =
        GetJobRunsRequest {
            jobName = jobNameVal
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getJobRuns(request)
        response.jobRuns?.forEach { job ->
            println("Job name is ${job.jobName}")
        }
    }
}

suspend fun deleteJob(jobNameVal: String) {
    val jobRequest =
        DeleteJobRequest {
            jobName = jobNameVal
        }
```

```
    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.deleteJob(jobRequest)
        println("$jobNameVal was successfully deleted")
    }
}

suspend fun deleteMyDatabase(databaseName: String) {
    val request =
        DeleteDatabaseRequest {
            name = databaseName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.deleteDatabase(request)
        println("$databaseName was successfully deleted")
    }
}

suspend fun deleteCrawler(crawlerName: String) {
    val request =
        DeleteCrawlerRequest {
            name = crawlerName
        }
    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.deleteCrawler(request)
        println("$crawlerName was deleted")
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS 適用於 Kotlin 的 SDK API 參考》中的下列主題。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases

- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

PHP

適用於 PHP 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

```
namespace Glue;

use Aws\Glue\GlueClient;
use Aws\S3\S3Client;
use AwsUtilities\AWSServiceClass;
use GuzzleHttp\Psr7\Stream;
use Iam\IAMService;

class GettingStartedWithGlue
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the AWS Glue getting started demo using PHP!\n");
        echo("--------------------------------------\n");

        $clientArgs = [
            'region' => 'us-west-2',
            'version' => 'latest',
            'profile' => 'default',
```

```
        ];
        $uniqid = uniqid();

        $glueClient = new GlueClient($clientArgs);
        $glueService = new GlueService($glueClient);
        $iamService = new IAMService();
        $crawlerName = "example-crawler-test-" . $uniqid;

        AWSServiceClass::$waitTime = 5;
        AWSServiceClass::$maxWaitAttempts = 20;

        $role = $iamService->getRole("AWSGlueServiceRole-DocExample");

        $databaseName = "doc-example-database-$uniqid";
        $path = 's3://crawler-public-us-east-1/flight/2016/csv';
        $glueService->createCrawler($crawlerName, $role['Role']['Arn'],
$databaseName, $path);
        $glueService->startCrawler($crawlerName);

        echo "Waiting for crawler";
        do {
            $crawler = $glueService->getCrawler($crawlerName);
            echo ".";
            sleep(10);
        } while ($crawler['Crawler']['State'] != "READY");
        echo "\n";

        $database = $glueService->getDatabase($databaseName);
        echo "Found a database named " . $database['Database']['Name'] . "\n";

        //Upload job script
        $s3client = new S3Client($clientArgs);
        $bucketName = "test-glue-bucket-" . $uniqid;
        $s3client->createBucket([
            'Bucket' => $bucketName,
            'CreateBucketConfiguration' => ['LocationConstraint' => 'us-west-2'],
        ]);

        $s3client->putObject([
            'Bucket' => $bucketName,
            'Key' => 'run_job.py',
            'SourceFile' => __DIR__ . '/flight_etl_job_script.py'
        ]);
        $s3client->putObject([
```

```
            'Bucket' => $bucketName,
            'Key' => 'setup_scenario_getting_started.yaml',
            'SourceFile' => __DIR__ . '/setup_scenario_getting_started.yaml'
        ]);

        $tables = $glueService->getTables($databaseName);

        $jobName = 'test-job-' . $uniqid;
        $scriptLocation = "s3://$bucketName/run_job.py";
        $job = $glueService->createJob($jobName, $role['Role']['Arn'],
$scriptLocation);

        $outputBucketUrl = "s3://$bucketName";
        $runId = $glueService->startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl)['JobRunId'];

        echo "waiting for job";
        do {
            $jobRun = $glueService->getJobRun($jobName, $runId);
            echo ".";
            sleep(10);
        } while (!array_intersect([$jobRun['JobRun']['JobRunState']],
['SUCCEEDED', 'STOPPED', 'FAILED', 'TIMEOUT']));
        echo "\n";

        $jobRuns = $glueService->getJobRuns($jobName);

        $objects = $s3client->listObjects([
            'Bucket' => $bucketName,
        ])['Contents'];

        foreach ($objects as $object) {
            echo $object['Key'] . "\n";
        }

        echo "Downloading " . $objects[1]['Key'] . "\n";
        /** @var Stream $downloadObject */
        $downloadObject = $s3client->getObject([
            'Bucket' => $bucketName,
            'Key' => $objects[1]['Key'],
        ])['Body']->getContents();
        echo "Here is the first 1000 characters in the object.";
        echo substr($downloadObject, 0, 1000);
```

```
        $jobs = $glueService->listJobs();
        echo "Current jobs:\n";
        foreach ($jobs['JobNames'] as $jobsName) {
            echo "{$jobsName}\n";
        }

        echo "Delete the job.\n";
        $glueClient->deleteJob([
            'JobName' => $job['Name'],
        ]);

        echo "Delete the tables.\n";
        foreach ($tables['TableList'] as $table) {
            $glueService->deleteTable($table['Name'], $databaseName);
        }

        echo "Delete the databases.\n";
        $glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);

        echo "Delete the crawler.\n";
        $glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);

        $deleteObjects = $s3client->listObjectsV2([
            'Bucket' => $bucketName,
        ]);
        echo "Delete all objects in the bucket.\n";
        $deleteObjects = $s3client->deleteObjects([
            'Bucket' => $bucketName,
            'Delete' => [
                'Objects' => $deleteObjects['Contents'],
            ]
        ]);
        echo "Delete the bucket.\n";
        $s3client->deleteBucket(['Bucket' => $bucketName]);

        echo "This job was brought to you by the number $uniqid\n";
    }
}

namespace Glue;
```

```
use Aws\Glue\GlueClient;
use Aws\Result;

use function PHPUnit\Framework\isEmpty;

class GlueService extends \AwsUtilities\AWSServiceClass
{
    protected GlueClient $glueClient;

    public function __construct($glueClient)
    {
        $this->glueClient = $glueClient;
    }

    public function getCrawler($crawlerName)
    {
        return $this->customWaiter(function () use ($crawlerName) {
            return $this->glueClient->getCrawler([
                'Name' => $crawlerName,
            ]);
        });
    }

    public function createCrawler($crawlerName, $role, $databaseName, $path):
 Result
    {
        return $this->customWaiter(function () use ($crawlerName, $role,
 $databaseName, $path) {
            return $this->glueClient->createCrawler([
                'Name' => $crawlerName,
                'Role' => $role,
                'DatabaseName' => $databaseName,
                'Targets' => [
                    'S3Targets' =>
                        [[
                            'Path' => $path,
                        ]]
                ],
            ]);
        });
    }

    public function startCrawler($crawlerName): Result
```

```
    {
        return $this->glueClient->startCrawler([
            'Name' => $crawlerName,
        ]);
    }

    public function getDatabase(string $databaseName): Result
    {
        return $this->customWaiter(function () use ($databaseName) {
            return $this->glueClient->getDatabase([
                'Name' => $databaseName,
            ]);
        });
    }

    public function getTables($databaseName): Result
    {
        return $this->glueClient->getTables([
            'DatabaseName' => $databaseName,
        ]);
    }

    public function createJob($jobName, $role, $scriptLocation, $pythonVersion =
'3', $glueVersion = '3.0'): Result
    {
        return $this->glueClient->createJob([
            'Name' => $jobName,
            'Role' => $role,
            'Command' => [
                'Name' => 'glueetl',
                'ScriptLocation' => $scriptLocation,
                'PythonVersion' => $pythonVersion,
            ],
            'GlueVersion' => $glueVersion,
        ]);
    }

    public function startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl): Result
    {
        return $this->glueClient->startJobRun([
            'JobName' => $jobName,
            'Arguments' => [
                'input_database' => $databaseName,
```

```
                'input_table' => $tables['TableList'][0]['Name'],
                'output_bucket_url' => $outputBucketUrl,
                '--input_database' => $databaseName,
                '--input_table' => $tables['TableList'][0]['Name'],
                '--output_bucket_url' => $outputBucketUrl,
            ],
        ]);
    }

    public function listJobs($maxResults = null, $nextToken = null, $tags = []):
Result
    {
        $arguments = [];
        if ($maxResults) {
            $arguments['MaxResults'] = $maxResults;
        }
        if ($nextToken) {
            $arguments['NextToken'] = $nextToken;
        }
        if (!empty($tags)) {
            $arguments['Tags'] = $tags;
        }
        return $this->glueClient->listJobs($arguments);
    }

    public function getJobRuns($jobName, $maxResults = 0, $nextToken = ''):
Result
    {
        $arguments = ['JobName' => $jobName];
        if ($maxResults) {
            $arguments['MaxResults'] = $maxResults;
        }
        if ($nextToken) {
            $arguments['NextToken'] = $nextToken;
        }
        return $this->glueClient->getJobRuns($arguments);
    }

    public function getJobRun($jobName, $runId, $predecessorsIncluded = false):
Result
    {
        return $this->glueClient->getJobRun([
            'JobName' => $jobName,
            'RunId' => $runId,
```

```
            'PredecessorsIncluded' => $predecessorsIncluded,
        ]);
    }

    public function deleteJob($jobName)
    {
        return $this->glueClient->deleteJob([
            'JobName' => $jobName,
        ]);
    }

    public function deleteTable($tableName, $databaseName)
    {
        return $this->glueClient->deleteTable([
            'DatabaseName' => $databaseName,
            'Name' => $tableName,
        ]);
    }

    public function deleteDatabase($databaseName)
    {
        return $this->glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);
    }

    public function deleteCrawler($crawlerName)
    {
        return $this->glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for PHP API 參考》中的下列主題。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob

- DeleteTable
- GetCrawler
- GetDatabase
- GetDatabases
- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

Python

適用於 Python (Boto3) 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

創建一個包裝在場景中使用的 AWS Glue 函數的類。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def get_crawler(self, name):
        """
        Gets information about a crawler.
```

```
        :param name: The name of the crawler to look up.
        :return: Data about the crawler.
        """
        crawler = None
        try:
            response = self.glue_client.get_crawler(Name=name)
            crawler = response["Crawler"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "EntityNotFoundException":
                logger.info("Crawler %s doesn't exist.", name)
            else:
                logger.error(
                    "Couldn't get crawler %s. Here's why: %s: %s",
                    name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return crawler


    def create_crawler(self, name, role_arn, db_name, db_prefix, s3_target):
        """
        Creates a crawler that can crawl the specified target and populate a
        database in your AWS Glue Data Catalog with metadata that describes the
data
        in the target.

        :param name: The name of the crawler.
        :param role_arn: The Amazon Resource Name (ARN) of an AWS Identity and
Access
                         Management (IAM) role that grants permission to let AWS
Glue
                         access the resources it needs.
        :param db_name: The name to give the database that is created by the
crawler.
        :param db_prefix: The prefix to give any database tables that are created
by
                          the crawler.
        :param s3_target: The URL to an S3 bucket that contains data that is
                          the target of the crawler.
        """
        try:
```

```
            self.glue_client.create_crawler(
                Name=name,
                Role=role_arn,
                DatabaseName=db_name,
                TablePrefix=db_prefix,
                Targets={"S3Targets": [{"Path": s3_target}]},
            )
        except ClientError as err:
            logger.error(
                "Couldn't create crawler. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def start_crawler(self, name):
        """
        Starts a crawler. The crawler crawls its configured target and creates
        metadata that describes the data it finds in the target data source.

        :param name: The name of the crawler to start.
        """
        try:
            self.glue_client.start_crawler(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't start crawler %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def get_database(self, name):
        """
        Gets information about a database in your Data Catalog.

        :param name: The name of the database to look up.
        :return: Information about the database.
        """
        try:
            response = self.glue_client.get_database(Name=name)
```

```
        except ClientError as err:
            logger.error(
                "Couldn't get database %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Database"]


    def get_tables(self, db_name):
        """
        Gets a list of tables in a Data Catalog database.

        :param db_name: The name of the database to query.
        :return: The list of tables in the database.
        """
        try:
            response = self.glue_client.get_tables(DatabaseName=db_name)
        except ClientError as err:
            logger.error(
                "Couldn't get tables %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["TableList"]


    def create_job(self, name, description, role_arn, script_location):
        """
        Creates a job definition for an extract, transform, and load (ETL) job
that can
        be run by AWS Glue.

        :param name: The name of the job definition.
        :param description: The description of the job definition.
        :param role_arn: The ARN of an IAM role that grants AWS Glue the
permissions
                         it requires to run the job.
```

```
        :param script_location: The Amazon S3 URL of a Python ETL script that is
run as
                                part of the job. The script defines how the data
is
                                transformed.
        """
        try:
            self.glue_client.create_job(
                Name=name,
                Description=description,
                Role=role_arn,
                Command={
                    "Name": "glueetl",
                    "ScriptLocation": script_location,
                    "PythonVersion": "3",
                },
                GlueVersion="3.0",
            )
        except ClientError as err:
            logger.error(
                "Couldn't create job %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def start_job_run(self, name, input_database, input_table,
output_bucket_name):
        """
        Starts a job run. A job run extracts data from the source, transforms it,
        and loads it to the output bucket.

        :param name: The name of the job definition.
        :param input_database: The name of the metadata database that contains
tables
                               that describe the source data. This is typically
created
                               by a crawler.
        :param input_table: The name of the table in the metadata database that
                            describes the source data.
        :param output_bucket_name: The S3 bucket where the output is written.
        :return: The ID of the job run.
```

```
        """
        try:
            # The custom Arguments that are passed to this function are used by
the
            # Python ETL script to determine the location of input and output
data.
            response = self.glue_client.start_job_run(
                JobName=name,
                Arguments={
                    "--input_database": input_database,
                    "--input_table": input_table,
                    "--output_bucket_url": f"s3://{output_bucket_name}/",
                },
            )
        except ClientError as err:
            logger.error(
                "Couldn't start job run %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobRunId"]


    def list_jobs(self):
        """
        Lists the names of job definitions in your account.

        :return: The list of job definition names.
        """
        try:
            response = self.glue_client.list_jobs()
        except ClientError as err:
            logger.error(
                "Couldn't list jobs. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobNames"]
```

```
    def get_job_runs(self, job_name):
        """
        Gets information about runs that have been performed for a specific job
        definition.

        :param job_name: The name of the job definition to look up.
        :return: The list of job runs.
        """
        try:
            response = self.glue_client.get_job_runs(JobName=job_name)
        except ClientError as err:
            logger.error(
                "Couldn't get job runs for %s. Here's why: %s: %s",
                job_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobRuns"]


    def get_job_run(self, name, run_id):
        """
        Gets information about a single job run.

        :param name: The name of the job definition for the run.
        :param run_id: The ID of the run.
        :return: Information about the run.
        """
        try:
            response = self.glue_client.get_job_run(JobName=name, RunId=run_id)
        except ClientError as err:
            logger.error(
                "Couldn't get job run %s/%s. Here's why: %s: %s",
                name,
                run_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobRun"]
```

```
    def delete_job(self, job_name):
        """
        Deletes a job definition. This also deletes data about all runs that are
        associated with this job definition.

        :param job_name: The name of the job definition to delete.
        """
        try:
            self.glue_client.delete_job(JobName=job_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete job %s. Here's why: %s: %s",
                job_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_table(self, db_name, table_name):
        """
        Deletes a table from a metadata database.

        :param db_name: The name of the database that contains the table.
        :param table_name: The name of the table to delete.
        """
        try:
            self.glue_client.delete_table(DatabaseName=db_name, Name=table_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_database(self, name):
        """
        Deletes a metadata database from your Data Catalog.
```

```
        :param name: The name of the database to delete.
        """
        try:
            self.glue_client.delete_database(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't delete database %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_crawler(self, name):
        """
        Deletes a crawler.

        :param name: The name of the crawler to delete.
        """
        try:
            self.glue_client.delete_crawler(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't delete crawler %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

建立可執行案例的類別。

```
class GlueCrawlerJobScenario:
    """
    Encapsulates a scenario that shows how to create an AWS Glue crawler and job
 and use
    them to transform data from CSV to JSON format.
    """
```

```
    def __init__(self, glue_client, glue_service_role, glue_bucket):
        """
        :param glue_client: A Boto3 AWS Glue client.
        :param glue_service_role: An AWS Identity and Access Management (IAM)
role
                                  that AWS Glue can assume to gain access to the
                                  resources it requires.
        :param glue_bucket: An S3 bucket that can hold a job script and output
data
                            from AWS Glue job runs.
        """
        self.glue_client = glue_client
        self.glue_service_role = glue_service_role
        self.glue_bucket = glue_bucket

    @staticmethod
    def wait(seconds, tick=12):
        """
        Waits for a specified number of seconds, while also displaying an
animated
        spinner.

        :param seconds: The number of seconds to wait.
        :param tick: The number of frames per second used to animate the spinner.
        """
        progress = "|/-\\"
        waited = 0
        while waited < seconds:
            for frame in range(tick):
                sys.stdout.write(f"\r{progress[frame % len(progress)]}")
                sys.stdout.flush()
                time.sleep(1 / tick)
            waited += 1

    def upload_job_script(self, job_script):
        """
        Uploads a Python ETL script to an S3 bucket. The script is used by the
AWS Glue
        job to transform data.

        :param job_script: The relative path to the job script.
        """
        try:
```

```
            self.glue_bucket.upload_file(Filename=job_script, Key=job_script)
            print(f"Uploaded job script '{job_script}' to the example bucket.")
        except S3UploadFailedError as err:
            logger.error("Couldn't upload job script. Here's why: %s", err)
            raise

    def run(self, crawler_name, db_name, db_prefix, data_source, job_script,
job_name):
        """
        Runs the scenario. This is an interactive experience that runs at a
command
        prompt and asks you for input throughout.

        :param crawler_name: The name of the crawler used in the scenario. If the
                             crawler does not exist, it is created.
        :param db_name: The name to give the metadata database created by the
crawler.
        :param db_prefix: The prefix to give tables added to the database by the
                          crawler.
        :param data_source: The location of the data source that is targeted by
the
                            crawler and extracted during job runs.
        :param job_script: The job script that is used to transform data during
job
                           runs.
        :param job_name: The name to give the job definition that is created
during the
                         scenario.
        """
        wrapper = GlueWrapper(self.glue_client)
        print(f"Checking for crawler {crawler_name}.")
        crawler = wrapper.get_crawler(crawler_name)
        if crawler is None:
            print(f"Creating crawler {crawler_name}.")
            wrapper.create_crawler(
                crawler_name,
                self.glue_service_role.arn,
                db_name,
                db_prefix,
                data_source,
            )
            print(f"Created crawler {crawler_name}.")
            crawler = wrapper.get_crawler(crawler_name)
        pprint(crawler)
```

```
        print("-" * 88)

        print(
            f"When you run the crawler, it crawls data stored in {data_source}
and "
            f"creates a metadata database in the AWS Glue Data Catalog that
describes "
            f"the data in the data source."
        )
        print("In this example, the source data is in CSV format.")
        ready = False
        while not ready:
            ready = Question.ask_question(
                "Ready to start the crawler? (y/n) ", Question.is_yesno
            )
        wrapper.start_crawler(crawler_name)
        print("Let's wait for the crawler to run. This typically takes a few
minutes.")
        crawler_state = None
        while crawler_state != "READY":
            self.wait(10)
            crawler = wrapper.get_crawler(crawler_name)
            crawler_state = crawler["State"]
            print(f"Crawler is {crawler['State']}.")
        print("-" * 88)

        database = wrapper.get_database(db_name)
        print(f"The crawler created database {db_name}:")
        pprint(database)
        print(f"The database contains these tables:")
        tables = wrapper.get_tables(db_name)
        for index, table in enumerate(tables):
            print(f"\t{index + 1}. {table['Name']}")
        table_index = Question.ask_question(
            f"Enter the number of a table to see more detail: ",
            Question.is_int,
            Question.in_range(1, len(tables)),
        )
        pprint(tables[table_index - 1])
        print("-" * 88)

        print(f"Creating job definition {job_name}.")
        wrapper.create_job(
            job_name,
```

```
            "Getting started example job.",
            self.glue_service_role.arn,
            f"s3://{self.glue_bucket.name}/{job_script}",
        )
        print("Created job definition.")
        print(
            f"When you run the job, it extracts data from {data_source},
transforms it "
            f"by using the {job_script} script, and loads the output into "
            f"S3 bucket {self.glue_bucket.name}."
        )
        print(
            "In this example, the data is transformed from CSV to JSON, and only
a few "
            "fields are included in the output."
        )
        job_run_status = None
        if Question.ask_question(f"Ready to run? (y/n) ", Question.is_yesno):
            job_run_id = wrapper.start_job_run(
                job_name, db_name, tables[0]["Name"], self.glue_bucket.name
            )
            print(f"Job {job_name} started. Let's wait for it to run.")
            while job_run_status not in ["SUCCEEDED", "STOPPED", "FAILED",
"TIMEOUT"]:
                self.wait(10)
                job_run = wrapper.get_job_run(job_name, job_run_id)
                job_run_status = job_run["JobRunState"]
                print(f"Job {job_name}/{job_run_id} is {job_run_status}.")
        print("-" * 88)

        if job_run_status == "SUCCEEDED":
            print(
                f"Data from your job run is stored in your S3 bucket
'{self.glue_bucket.name}':"
            )
            try:
                keys = [
                    obj.key for obj in
self.glue_bucket.objects.filter(Prefix="run-")
                ]
                for index, key in enumerate(keys):
                    print(f"\t{index + 1}: {key}")
                lines = 4
                key_index = Question.ask_question(
```

```
                    f"Enter the number of a block to download it and see the
first {lines} "
                    f"lines of JSON output in the block: ",
                    Question.is_int,
                    Question.in_range(1, len(keys)),
                )
                job_data = io.BytesIO()
                self.glue_bucket.download_fileobj(keys[key_index - 1], job_data)
                job_data.seek(0)
                for _ in range(lines):
                    print(job_data.readline().decode("utf-8"))
            except ClientError as err:
                logger.error(
                    "Couldn't get job run data. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
            print("-" * 88)

        job_names = wrapper.list_jobs()
        if job_names:
            print(f"Your account has {len(job_names)} jobs defined:")
            for index, job_name in enumerate(job_names):
                print(f"\t{index + 1}. {job_name}")
            job_index = Question.ask_question(
                f"Enter a number between 1 and {len(job_names)} to see the list
of runs for "
                f"a job: ",
                Question.is_int,
                Question.in_range(1, len(job_names)),
            )
            job_runs = wrapper.get_job_runs(job_names[job_index - 1])
            if job_runs:
                print(f"Found {len(job_runs)} runs for job {job_names[job_index -
1]}:")
                for index, job_run in enumerate(job_runs):
                    print(
                        f"\t{index + 1}. {job_run['JobRunState']} on "
                        f"{job_run['CompletedOn']:%Y-%m-%d %H:%M:%S}"
                    )
                run_index = Question.ask_question(
                    f"Enter a number between 1 and {len(job_runs)} to see details
for a run: ",
```

```
                    Question.is_int,
                    Question.in_range(1, len(job_runs)),
                )
                pprint(job_runs[run_index - 1])
            else:
                print(f"No runs found for job {job_names[job_index - 1]}")
        else:
            print("Your account doesn't have any jobs defined.")
        print("-" * 88)

        print(
            f"Let's clean up. During this example we created job definition
'{job_name}'."
        )
        if Question.ask_question(
            "Do you want to delete the definition and all runs? (y/n) ",
            Question.is_yesno,
        ):
            wrapper.delete_job(job_name)
            print(f"Job definition '{job_name}' deleted.")
        tables = wrapper.get_tables(db_name)
        print(f"We also created database '{db_name}' that contains these
tables:")
        for table in tables:
            print(f"\t{table['Name']}")
        if Question.ask_question(
            "Do you want to delete the tables and the database? (y/n) ",
            Question.is_yesno,
        ):
            for table in tables:
                wrapper.delete_table(db_name, table["Name"])
                print(f"Deleted table {table['Name']}.")
            wrapper.delete_database(db_name)
            print(f"Deleted database {db_name}.")
        print(f"We also created crawler '{crawler_name}'.")
        if Question.ask_question(
            "Do you want to delete the crawler? (y/n) ", Question.is_yesno
        ):
            wrapper.delete_crawler(crawler_name)
            print(f"Deleted crawler {crawler_name}.")
        print("-" * 88)


def parse_args(args):
```

```
    """
    Parse command line arguments.

    :param args: The command line arguments.
    :return: The parsed arguments.
    """
    parser = argparse.ArgumentParser(
        description="Runs the AWS Glue getting started with crawlers and jobs
 scenario. "
        "Before you run this scenario, set up scaffold resources by running "
        "'python scaffold.py deploy'."
    )
    parser.add_argument(
        "role_name",
        help="The name of an IAM role that AWS Glue can assume. This role must
 grant access "
        "to Amazon S3 and to the permissions granted by the AWSGlueServiceRole "
        "managed policy.",
    )
    parser.add_argument(
        "bucket_name",
        help="The name of an S3 bucket that AWS Glue can access to get the job
 script and "
        "put job results.",
    )
    parser.add_argument(
        "--job_script",
        default="flight_etl_job_script.py",
        help="The name of the job script file that is used in the scenario.",
    )
    return parser.parse_args(args)


def main():
    args = parse_args(sys.argv[1:])
    try:
        print("-" * 88)
        print(
            "Welcome to the AWS Glue getting started with crawlers and jobs
 scenario."
        )
        print("-" * 88)
        scenario = GlueCrawlerJobScenario(
            boto3.client("glue"),
```

```
            boto3.resource("iam").Role(args.role_name),
            boto3.resource("s3").Bucket(args.bucket_name),
        )
        scenario.upload_job_script(args.job_script)
        scenario.run(
            "doc-example-crawler",
            "doc-example-database",
            "doc-example-",
            "s3://crawler-public-us-east-1/flight/2016/csv",
            args.job_script,
            "doc-example-job",
        )
        print("-" * 88)
        print(
            "To destroy scaffold resources, including the IAM role and S3 bucket
 "
            "used in this scenario, run 'python scaffold.py destroy'."
        )
        print("\nThanks for watching!")
        print("-" * 88)
    except Exception:
        logging.exception("Something went wrong with the example.")
```

建立 ETL 指令碼，AWS Glue 以便在工作執行期間擷取、轉換和載入資料。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

"""
These custom arguments must be passed as Arguments to the StartJobRun request.
    --input_database    The name of a metadata database that is contained in
 your
                        AWS Glue Data Catalog and that contains tables that
 describe
                        the data to be processed.
```

```
    --input_table       The name of a table in the database that describes the
 data to
                        be processed.
    --output_bucket_url An S3 bucket that receives the transformed output data.
"""
args = getResolvedOptions(
    sys.argv, ["JOB_NAME", "input_database", "input_table", "output_bucket_url"]
)
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node S3 Flight Data.
S3FlightData_node1 = glueContext.create_dynamic_frame.from_catalog(
    database=args["input_database"],
    table_name=args["input_table"],
    transformation_ctx="S3FlightData_node1",
)

# This mapping performs two main functions:
# 1. It simplifies the output by removing most of the fields from the data.
# 2. It renames some fields. For example, `fl_date` is renamed to `flight_date`.
ApplyMapping_node2 = ApplyMapping.apply(
    frame=S3FlightData_node1,
    mappings=[
        ("year", "long", "year", "long"),
        ("month", "long", "month", "tinyint"),
        ("day_of_month", "long", "day", "tinyint"),
        ("fl_date", "string", "flight_date", "string"),
        ("carrier", "string", "carrier", "string"),
        ("fl_num", "long", "flight_num", "long"),
        ("origin_city_name", "string", "origin_city_name", "string"),
        ("origin_state_abr", "string", "origin_state_abr", "string"),
        ("dest_city_name", "string", "dest_city_name", "string"),
        ("dest_state_abr", "string", "dest_state_abr", "string"),
        ("dep_time", "long", "departure_time", "long"),
        ("wheels_off", "long", "wheels_off", "long"),
        ("wheels_on", "long", "wheels_on", "long"),
        ("arr_time", "long", "arrival_time", "long"),
        ("mon", "string", "mon", "string"),
    ],
    transformation_ctx="ApplyMapping_node2",
```

```
)

# Script generated for node Revised Flight Data.
RevisedFlightData_node3 = glueContext.write_dynamic_frame.from_options(
    frame=ApplyMapping_node2,
    connection_type="s3",
    format="json",
    connection_options={"path": args["output_bucket_url"], "partitionKeys": []},
    transformation_ctx="RevisedFlightData_node3",
)

job.commit()
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考資料》中的下列主題。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

Ruby

適用於 Ruby 的開發套件

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

創建一個包裝在場景中使用的 AWS Glue 函數的類。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves information about a specific crawler.
  #
  # @param name [String] The name of the crawler to retrieve information about.
  # @return [Aws::Glue::Types::Crawler, nil] The crawler object if found, or nil
 if not found.
  def get_crawler(name)
    @glue_client.get_crawler(name: name)
  rescue Aws::Glue::Errors::EntityNotFoundException
    @logger.info("Crawler #{name} doesn't exist.")
    false
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get crawler #{name}: \n#{e.message}")
    raise
  end

  # Creates a new crawler with the specified configuration.
  #
  # @param name [String] The name of the crawler.
```

```
 # @param role_arn [String] The ARN of the IAM role to be used by the crawler.
 # @param db_name [String] The name of the database where the crawler stores its
metadata.
 # @param db_prefix [String] The prefix to be added to the names of tables that
the crawler creates.
 # @param s3_target [String] The S3 path that the crawler will crawl.
 # @return [void]
 def create_crawler(name, role_arn, db_name, db_prefix, s3_target)
   @glue_client.create_crawler(
     name: name,
     role: role_arn,
     database_name: db_name,
     targets: {
       s3_targets: [
         {
           path: s3_target
         }
       ]
     }
   )
 rescue Aws::Glue::Errors::GlueException => e
   @logger.error("Glue could not create crawler: \n#{e.message}")
   raise
 end

 # Starts a crawler with the specified name.
 #
 # @param name [String] The name of the crawler to start.
 # @return [void]
 def start_crawler(name)
   @glue_client.start_crawler(name: name)
 rescue Aws::Glue::Errors::ServiceError => e
   @logger.error("Glue could not start crawler #{name}: \n#{e.message}")
   raise
 end

 # Deletes a crawler with the specified name.
 #
 # @param name [String] The name of the crawler to delete.
 # @return [void]
 def delete_crawler(name)
   @glue_client.delete_crawler(name: name)
 rescue Aws::Glue::Errors::ServiceError => e
   @logger.error("Glue could not delete crawler #{name}: \n#{e.message}")
```

```
    raise
  end

  # Retrieves information about a specific database.
  #
  # @param name [String] The name of the database to retrieve information about.
  # @return [Aws::Glue::Types::Database, nil] The database object if found, or
 nil if not found.
  def get_database(name)
    response = @glue_client.get_database(name: name)
    response.database
rescue Aws::Glue::Errors::GlueException => e
  @logger.error("Glue could not get database #{name}: \n#{e.message}")
  raise
  end

  # Retrieves a list of tables in the specified database.
  #
  # @param db_name [String] The name of the database to retrieve tables from.
  # @return [Array<Aws::Glue::Types::Table>]
  def get_tables(db_name)
    response = @glue_client.get_tables(database_name: db_name)
    response.table_list
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get tables #{db_name}: \n#{e.message}")
    raise
  end

  # Creates a new job with the specified configuration.
  #
  # @param name [String] The name of the job.
  # @param description [String] The description of the job.
  # @param role_arn [String] The ARN of the IAM role to be used by the job.
  # @param script_location [String] The location of the ETL script for the job.
  # @return [void]
  def create_job(name, description, role_arn, script_location)
    @glue_client.create_job(
      name: name,
      description: description,
      role: role_arn,
      command: {
        name: "glueetl",
        script_location: script_location,
        python_version: "3"
```

```
      },
      glue_version: "3.0"
    )
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not create job #{name}: \n#{e.message}")
    raise
  end

  # Starts a job run for the specified job.
  #
  # @param name [String] The name of the job to start the run for.
  # @param input_database [String] The name of the input database for the job.
  # @param input_table [String] The name of the input table for the job.
  # @param output_bucket_name [String] The name of the output S3 bucket for the
job.
  # @return [String] The ID of the started job run.
  def start_job_run(name, input_database, input_table, output_bucket_name)
    response = @glue_client.start_job_run(
      job_name: name,
      arguments: {
        '--input_database': input_database,
        '--input_table': input_table,
        '--output_bucket_url': "s3://#{output_bucket_name}/"
      }
    )
    response.job_run_id
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not start job run #{name}: \n#{e.message}")
    raise
  end

  # Retrieves a list of jobs in AWS Glue.
  #
  # @return [Aws::Glue::Types::ListJobsResponse]
  def list_jobs
    @glue_client.list_jobs
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not list jobs: \n#{e.message}")
    raise
  end

  # Retrieves a list of job runs for the specified job.
  #
  # @param job_name [String] The name of the job to retrieve job runs for.
```

```
  # @return [Array<Aws::Glue::Types::JobRun>]
  def get_job_runs(job_name)
    response = @glue_client.get_job_runs(job_name: job_name)
    response.job_runs
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get job runs: \n#{e.message}")
  end

  # Retrieves data for a specific job run.
  #
  # @param job_name [String] The name of the job run to retrieve data for.
  # @return [Glue::Types::GetJobRunResponse]
  def get_job_run(job_name, run_id)
    @glue_client.get_job_run(job_name: job_name, run_id: run_id)
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get job runs: \n#{e.message}")
  end

  # Deletes a job with the specified name.
  #
  # @param job_name [String] The name of the job to delete.
  # @return [void]
  def delete_job(job_name)
    @glue_client.delete_job(job_name: job_name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete job: \n#{e.message}")
  end

  # Deletes a table with the specified name.
  #
  # @param database_name [String] The name of the catalog database in which the
table resides.
  # @param table_name [String] The name of the table to be deleted.
  # @return [void]
  def delete_table(database_name, table_name)
    @glue_client.delete_table(database_name: database_name, name: table_name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete job: \n#{e.message}")
  end

  # Removes a specified database from a Data Catalog.
  #
  # @param database_name [String] The name of the database to delete.
  # @return [void]
```

```
  def delete_database(database_name)
    @glue_client.delete_database(name: database_name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete database: \n#{e.message}")
  end

  # Uploads a job script file to an S3 bucket.
  #
  # @param file_path [String] The local path of the job script file.
  # @param bucket_resource [Aws::S3::Bucket] The S3 bucket resource to upload the
 file to.
  # @return [void]
  def upload_job_script(file_path, bucket_resource)
    File.open(file_path) do |file|
      bucket_resource.client.put_object({
       body: file,
       bucket: bucket_resource.name,
       key: file_path
     })
    end
  rescue Aws::S3::Errors::S3UploadFailedError => e
    @logger.error("S3 could not upload job script: \n#{e.message}")
    raise
  end

end
```

建立可執行案例的類別。

```
class GlueCrawlerJobScenario
  def initialize(glue_client, glue_service_role, glue_bucket, logger)
    @glue_client = glue_client
    @glue_service_role = glue_service_role
    @glue_bucket = glue_bucket
    @logger = logger
  end

  def run(crawler_name, db_name, db_prefix, data_source, job_script, job_name)
    wrapper = GlueWrapper.new(@glue_client, @logger)

    new_step(1, "Create a crawler")
    puts "Checking for crawler #{crawler_name}."
```

```
    crawler = wrapper.get_crawler(crawler_name)
    if crawler == false
      puts "Creating crawler #{crawler_name}."
      wrapper.create_crawler(crawler_name, @glue_service_role.arn, db_name,
db_prefix, data_source)
      puts "Successfully created #{crawler_name}:"
      crawler = wrapper.get_crawler(crawler_name)
      puts JSON.pretty_generate(crawler).yellow
    end
    print "\nDone!\n".green

    new_step(2, "Run a crawler to output a database.")
    puts "Location of input data analyzed by crawler: #{data_source}"
    puts "Outputs: a Data Catalog database in CSV format containing metadata on
input."
    wrapper.start_crawler(crawler_name)
    puts "Starting crawler... (this typically takes a few minutes)"
    crawler_state = nil
    while crawler_state != "READY"
      custom_wait(15)
      crawler = wrapper.get_crawler(crawler_name)
      crawler_state = crawler[0]["state"]
      print "Status check: #{crawler_state}.".yellow
    end
    print "\nDone!\n".green

    new_step(3, "Query the database.")
    database = wrapper.get_database(db_name)
    puts "The crawler created database #{db_name}:"
    print "#{database}".yellow
    puts "\nThe database contains these tables:"
    tables = wrapper.get_tables(db_name)
    tables.each_with_index do |table, index|
      print "\t#{index + 1}. #{table['name']}".yellow
    end
    print "\nDone!\n".green

    new_step(4, "Create a job definition that runs an ETL script.")
    puts "Uploading Python ETL script to S3..."
    wrapper.upload_job_script(job_script, @glue_bucket)
    puts "Creating job definition #{job_name}:\n"
    response = wrapper.create_job(job_name, "Getting started example job.",
@glue_service_role.arn, "s3://#{@glue_bucket.name}/#{job_script}")
    puts JSON.pretty_generate(response).yellow
```

```
    print "\nDone!\n".green

    new_step(5, "Start a new job")
    job_run_status = nil
    job_run_id = wrapper.start_job_run(
      job_name,
      db_name,
      tables[0]["name"],
      @glue_bucket.name
    )
    puts "Job #{job_name} started. Let's wait for it to run."
    until ["SUCCEEDED", "STOPPED", "FAILED", "TIMEOUT"].include?(job_run_status)
      custom_wait(10)
      job_run = wrapper.get_job_runs(job_name)
      job_run_status = job_run[0]["job_run_state"]
      print "Status check: #{job_name}/#{job_run_id} - #{job_run_status}.".yellow
    end
    print "\nDone!\n".green

    new_step(6, "View results from a successful job run.")
    if job_run_status == "SUCCEEDED"
      puts "Data from your job run is stored in your S3 bucket
'#{@glue_bucket.name}'. Files include:"
      begin

        # Print the key name of each object in the bucket.
        @glue_bucket.objects.each do |object_summary|
          if object_summary.key.include?("run-")
            print "#{object_summary.key}".yellow
          end
        end

        # Print the first 256 bytes of a run file
        desired_sample_objects = 1
        @glue_bucket.objects.each do |object_summary|
          if object_summary.key.include?("run-")
            if desired_sample_objects > 0
              sample_object = @glue_bucket.object(object_summary.key)
              sample = sample_object.get(range: "bytes=0-255").body.read
              puts "\nSample run file contents:"
              print "#{sample}".yellow
              desired_sample_objects -= 1
            end
          end
```

```
        end
      rescue Aws::S3::Errors::ServiceError => e
        logger.error(
          "Couldn't get job run data. Here's why: %s: %s",
          e.response.error.code, e.response.error.message
        )
        raise
      end
    end
    print "\nDone!\n".green

    new_step(7, "Delete job definition and crawler.")
    wrapper.delete_job(job_name)
    puts "Job deleted: #{job_name}."
    wrapper.delete_crawler(crawler_name)
    puts "Crawler deleted: #{crawler_name}."
    wrapper.delete_table(db_name, tables[0]["name"])
    puts "Table deleted: #{tables[0]["name"]} in #{db_name}."
    wrapper.delete_database(db_name)
    puts "Database deleted: #{db_name}."
    print "\nDone!\n".green
  end
end

def main

  banner("../../helpers/banner.txt")
  puts
 "#########################################################################################
  puts "#
                            #".yellow
  puts "#                                          EXAMPLE CODE DEMO:
                            #".yellow
  puts "#                                              AWS Glue
                            #".yellow
  puts "#
                            #".yellow
  puts
 "#########################################################################################
  puts ""
  puts "You have launched a demo of AWS Glue using the AWS for Ruby v3 SDK. Over
 the next 60 seconds, it will"
  puts "do the following:"
  puts "    1. Create a crawler."
```

```
puts "    2. Run a crawler to output a database."
puts "    3. Query the database."
puts "    4. Create a job definition that runs an ETL script."
puts "    5. Start a new job."
puts "    6. View results from a successful job run."
puts "    7. Delete job definition and crawler."
puts ""

confirm_begin
billing
security
puts "\e[H\e[2J"

# Set input file names
job_script_filepath = "job_script.py"
resource_names = YAML.load_file("resource_names.yaml")

# Instantiate existing IAM role.
iam = Aws::IAM::Resource.new(region: "us-east-1")
iam_role_name = resource_names["glue_service_role"]
iam_role = iam.role(iam_role_name)

# Instantiate existing S3 bucket.
s3 = Aws::S3::Resource.new(region: "us-east-1")
s3_bucket_name = resource_names["glue_bucket"]
s3_bucket = s3.bucket(s3_bucket_name)

scenario = GlueCrawlerJobScenario.new(
  Aws::Glue::Client.new(region: "us-east-1"),
  iam_role,
  s3_bucket,
  @logger
)

random_int = rand(10 ** 4)
scenario.run(
  "doc-example-crawler-#{random_int}",
  "doc-example-database-#{random_int}",
  "doc-example-#{random_int}-",
  "s3://crawler-public-us-east-1/flight/2016/csv",
  job_script_filepath,
  "doc-example-job-#{random_int}"
)
```

```
  puts "-" * 88
  puts "You have reached the end of this tour of AWS Glue."
  puts "To destroy CDK-created resources, run:\n       cdk destroy"
  puts "-" * 88

end
```

建立 ETL 指令碼，AWS Glue 以便在工作執行期間擷取、轉換和載入資料。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

"""
These custom arguments must be passed as Arguments to the StartJobRun request.
    --input_database    The name of a metadata database that is contained in
 your
                        AWS Glue Data Catalog and that contains tables that
 describe
                        the data to be processed.
    --input_table       The name of a table in the database that describes the
 data to
                        be processed.
    --output_bucket_url An S3 bucket that receives the transformed output data.
"""
args = getResolvedOptions(
    sys.argv, ["JOB_NAME", "input_database", "input_table", "output_bucket_url"]
)
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node S3 Flight Data.
S3FlightData_node1 = glueContext.create_dynamic_frame.from_catalog(
    database=args["input_database"],
    table_name=args["input_table"],
    transformation_ctx="S3FlightData_node1",
```

```
)

# This mapping performs two main functions:
# 1. It simplifies the output by removing most of the fields from the data.
# 2. It renames some fields. For example, `fl_date` is renamed to `flight_date`.
ApplyMapping_node2 = ApplyMapping.apply(
    frame=S3FlightData_node1,
    mappings=[
        ("year", "long", "year", "long"),
        ("month", "long", "month", "tinyint"),
        ("day_of_month", "long", "day", "tinyint"),
        ("fl_date", "string", "flight_date", "string"),
        ("carrier", "string", "carrier", "string"),
        ("fl_num", "long", "flight_num", "long"),
        ("origin_city_name", "string", "origin_city_name", "string"),
        ("origin_state_abr", "string", "origin_state_abr", "string"),
        ("dest_city_name", "string", "dest_city_name", "string"),
        ("dest_state_abr", "string", "dest_state_abr", "string"),
        ("dep_time", "long", "departure_time", "long"),
        ("wheels_off", "long", "wheels_off", "long"),
        ("wheels_on", "long", "wheels_on", "long"),
        ("arr_time", "long", "arrival_time", "long"),
        ("mon", "string", "mon", "string"),
    ],
    transformation_ctx="ApplyMapping_node2",
)

# Script generated for node Revised Flight Data.
RevisedFlightData_node3 = glueContext.write_dynamic_frame.from_options(
    frame=ApplyMapping_node2,
    connection_type="s3",
    format="json",
    connection_options={"path": args["output_bucket_url"], "partitionKeys": []},
    transformation_ctx="RevisedFlightData_node3",
)

job.commit()
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Ruby API 參考》中的下列主題。
  - CreateCrawler
  - CreateJob

- DeleteCrawler
- DeleteDatabase
- DeleteJob
- DeleteTable
- GetCrawler
- GetDatabase
- GetDatabases
- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

Rust

適用於 Rust 的 SDK

> **Note**
>
> 還有更多關於 GitHub。尋找完整範例，並了解如何在AWS 設定和執行程式碼範例儲存庫。

建立並執行可網路爬取公有 Amazon Simple Storage Service (Amazon S3) 儲存貯體的爬蟲程式，並產生描述其所尋找 CSV 格式資料的中繼資料的資料庫。

```
        let create_crawler = glue
            .create_crawler()
            .name(self.crawler())
            .database_name(self.database())
            .role(self.iam_role.expose_secret())
            .targets(
                CrawlerTargets::builder()
                    .s3_targets(S3Target::builder().path(CRAWLER_TARGET).build())
```

```
                    .build(),
            )
            .send()
            .await;

        match create_crawler {
            Err(err) => {
                let glue_err: aws_sdk_glue::Error = err.into();
                match glue_err {
                    aws_sdk_glue::Error::AlreadyExistsException(_) => {
                        info!("Using existing crawler");
                        Ok(())
                    }
                    _ => Err(GlueMvpError::GlueSdk(glue_err)),
                }
            }
            Ok(_) => Ok(()),
        }?;

        let start_crawler =
glue.start_crawler().name(self.crawler()).send().await;

        match start_crawler {
            Ok(_) => Ok(()),
            Err(err) => {
                let glue_err: aws_sdk_glue::Error = err.into();
                match glue_err {
                    aws_sdk_glue::Error::CrawlerRunningException(_) => Ok(()),
                    _ => Err(GlueMvpError::GlueSdk(glue_err)),
                }
            }
        }?;
```

列出有關 AWS Glue Data Catalog.

```
        let database = glue
            .get_database()
            .name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?
            .to_owned();
```

```
        let database = database
            .database()
            .ok_or_else(|| GlueMvpError::Unknown("Could not find
database".into()))?;

        let tables = glue
            .get_tables()
            .database_name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        let tables = tables.table_list();
```

建立並執行從來源 Amazon S3 儲存貯體中擷取 CSV 資料的任務、透過移除和重新命名欄位進行轉換，以及將 JSON 格式的輸出載入另一個 Amazon S3 儲存貯體。

```
        let create_job = glue
            .create_job()
            .name(self.job())
            .role(self.iam_role.expose_secret())
            .command(
                JobCommand::builder()
                    .name("glueetl")
                    .python_version("3")
                    .script_location(format!("s3://{}/job.py", self.bucket()))
                    .build(),
            )
            .glue_version("3.0")
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        let job_name = create_job.name().ok_or_else(|| {
            GlueMvpError::Unknown("Did not get job name after creating
job".into())
        })?;

        let job_run_output = glue
            .start_job_run()
            .job_name(self.job())
            .arguments("--input_database", self.database())
```

```
            .arguments(
                "--input_table",
                self.tables
                    .first()
                    .ok_or_else(|| GlueMvpError::Unknown("Missing crawler
table".into()))?
                    .name(),
            )
            .arguments("--output_bucket_url", self.bucket())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        let job = job_run_output
            .job_run_id()
            .ok_or_else(|| GlueMvpError::Unknown("Missing run id from just
started job".into()))?
            .to_string();
```

刪除透過示範建立的所有資源。

```
        glue.delete_job()
            .job_name(self.job())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        for t in &self.tables {
            glue.delete_table()
                .name(t.name())
                .database_name(self.database())
                .send()
                .await
                .map_err(GlueMvpError::from_glue_sdk)?;
        }

        glue.delete_database()
            .name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
```

```
        glue.delete_crawler()
            .name(self.crawler())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的下列主題。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

AWS Glue 中的安全性

雲端安全是 AWS 最重視的一環。身為 AWS 客戶的您，將能從資料中心和網路架構的建置中獲益，以滿足組織最為敏感的安全要求。

安全是 AWS 與您共同的責任。共同的責任模型 將此描述為雲端 本身 的安全和雲端內部的安全：

- 雲端本身的安全：AWS 負責保護在 AWS Cloud 中執行 AWS 服務的基礎設施。AWS 也提供您可安全使用的服務。在 AWS 合規計劃中，第三方稽核員會定期測試並驗證我們的安全功效。若要進一步瞭解適用於 AWS Glue 的合規計劃，請參閱 合規計劃範圍內的 AWS 服務。
- 雲端內部的安全：您的責任取決於所使用的 AWS 服務。您也必須對其他因素負責，包括資料的機密性、您公司的要求和適用法律和法規。

本文件有助於您了解如何在使用 AWS Glue 時套用共同責任模型。下列主題說明如何將 AWS Glue 設定為達到您的安全及合規目標。您也會了解如何使用其他 AWS 服務來協助監控並保護 AWS Glue 資源。

主題

- AWS Glue 中的資料保護
- AWS Glue 的身分識別與存取管理
- AWS Glue 中的記錄和監控
- 符合性驗證 AWS Glue
- 韌性 AWS Glue
- AWS Glue 中的基礎設施安全

AWS Glue 中的資料保護

AWS Glue 提供數種功能，旨在協助保護您的資料。

主題

- 靜態加密
- 傳輸中加密
- FIPS 合規
- 金鑰管理

- AWS Glue 對其他 AWS 服務的相依性
- 開發端點

靜態加密

AWS Glue 支援 使用 AWS Glue Studio 建立視覺化 ETL 任務 和 使用開發端點來開發指令碼 的待用資料加密。您可以設定擷取、轉換和載入 (ETL) 任務和開發端點，以使用 AWS Key Management Service (AWS KMS) 金鑰來寫入待用加密資料。您也可以使用您透過 AWS KMS 管理的金鑰來加密存放在AWS Glue Data Catalog中的中繼資料。此外，您可以使用 AWS KMS 金鑰來加密爬取程式和 ETL 任務產生的任務書籤和日誌。

除了依任務、搜尋器和開發端點寫入 Amazon 簡單儲存服務 (Amazon S3) 和 Amazon CloudWatch 日誌的資料外，您還可以AWS Glue Data Catalog在中加密中繼資料物件。在 AWS Glue 中建立任務、爬蟲程式和開發端點時，您可以透過連接安全組態來提供加密設定。安全性組態包含由 Amazon S3 管理的伺服器端加密金鑰 (SSE-S3) 或儲存在 AWS KMS (SSE-KMS) 中的客戶主金鑰 (CMK)。您可以使用 AWS Glue 主控台來建立安全性組態。

您也可以開啟帳戶中整個 Data Catalog 的加密。您可以藉由指定儲存在 AWS KMS 中的 CMK 來執行此操作。

> **Important**
>
> AWS Glue僅支援對稱的客戶管理金鑰。如需詳細資訊，請參閱AWS Key Management Service開發人員指南中的客戶管理金鑰 (CMK)。

在加密已開啟的狀況下，在您新增 Data Catalog 物件、執行爬蟲程式、執行任務或開始開發端點時，系統會使用 SSE-S3 或 SSE-KMS 金鑰來寫入靜態資料。此外，您可以將 AWS Glue 設定為僅透過信任的 Transport Layer Security (TLS) 通訊協定來存取 Java Database Connectivity (JDBC) 資料存放區。

您可以在 AWS Glue 中存取以下位置中的加密設定：

- Data Catalog 的設定。
- 您建立的安全組態。
- 伺服器端加密設定 (SSE-S3 或 SSE-KMS)，以參數形式將其傳遞給您的 AWS Glue ETL (擷取、轉換和載入) 任務。

有關設定加密的詳細資訊，請參閱設定 AWS Glue 中的加密。

主題

- 加密您的資料目錄
- 加密連線密碼
- 對 AWS Glue 寫入的資料加密

加密您的資料目錄

AWS Glue Data Catalog加密可為您的敏感資料提供增強的安全性。 AWS Glue與 AWS Key Management Service (AWS KMS) 整合以加密儲存在資料目錄中的中繼資料。您可以使用AWS Glue主控台或啟用資料目錄中資源的加密設定AWS CLI。

當您為資料目錄啟用加密時，您建立的所有新物件都會加密。停用加密時，您建立的新物件將不會加密，但現有的加密物件將保持加密狀態。

您可以使用AWS受管加密金鑰或客戶管理的加密金鑰來加密整個資料目錄。如需有關金鑰類型和狀態的詳細資訊，請參閱AWS Key Management Service開發人員指南中的AWS Key Management Service概念。

AWS 受管金鑰

AWS受管理金鑰是您帳戶中的 KMS 金鑰，由整合的AWS服務代表您建立、管理和使用AWS KMS。您可以檢視帳戶中的AWS受管理金鑰、檢視其金鑰策略，以及稽核其在AWS CloudTrail記錄中的使用情況。但是，您無法管理這些金鑰或變更其權限。

靜態加密會自動整AWS KMS合，以管理用於加密中繼資料的AWS受管理金鑰。AWS Glue如果您啟用中繼資料加密時，AWS受管理金鑰不存在，AWS KMS會自動為您建立新的金鑰。

如需詳細資訊，請參閱 AWS 受管金鑰。

客戶受管金鑰

客戶受管金鑰是您在 AWS 帳戶 中建立、擁有和管理的 KMS 金鑰。您可以完全控制這些 KMS 金鑰。您可以：

- 建立和維護其關鍵政策、IAM 政策和授權
- 啟用和禁用它們
- 旋轉其密碼材料

- 新增標籤
- 創建引用它們的別名
- 安排它們進行刪除

如需有關管理客戶管理金鑰權限的詳細資訊，請參閱客戶受管金鑰。

> **Important**
>
> AWS Glue僅支援對稱的客戶管理金鑰。KMS 金鑰清單只會顯示對稱金鑰。不過，如果您選取 [選擇 KMS 金鑰 ARN]，則主控台可讓您輸入任何金鑰類型的 ARN。請確定您僅輸入對稱金鑰的 ARN。
>
> 若要建立對稱的客戶管理金鑰，請遵循AWS Key Management Service開發人員指南中建立對稱客戶管理金鑰的步驟。

當您啟用靜態資料目錄加密時，會使用 KMS 金鑰加密下列資源類型：

- 資料庫
- 資料表
- 資料分割
- 檔案版本
- 資料欄統計資料
- 使用者定義的函數
- 資料目錄檢視

AWS Glue 加密內容

加密內容是選用的一組鍵值對，可包含資料的其他內容資訊。AWS KMS 使用加密內容作為其他身分驗證資料，以支援身分驗證加密。在加密資料的請求中包含加密內容時，AWS KMS 會將加密內容繫結至加密的資料。若要解密資料，請在要求中包含相同的加密內容。 AWS Glue在所有密AWS KMS碼編譯作業中使用相同的加密內容，其中金鑰所在`glue_catalog_id`且值為。`catalogId`

```
"encryptionContext": {
    "glue_catalog_id": "111122223333"
}
```

當您使用AWS受管金鑰或對稱的客戶受管金鑰來加密資料目錄時，您也可以在稽核記錄和記錄中使用加密內容來識別金鑰的使用方式。加密內容也會顯示在AWS CloudTrail或記錄檔所產生的Amazon CloudWatch記錄中。

啟用加密

您可以在AWS Glue主控台的「資料目錄」設定中或使用啟用AWS Glue Data Catalog物件的加密AWS CLI。

Console

使用主控台啟用加密

1. 登入 AWS Management Console，並前往 https://console.aws.amazon.com/glue/ 開啟 AWS Glue 主控台。
2. 在導覽窗格中選擇 [資料目錄]。
3. 在 [資料目錄設定] 頁面上，選取 [中繼資料加密] 核取方塊，然後選擇AWS KMS金鑰。

   啟用加密時，如果未指定客戶受管金鑰，則加密設定會使用受AWS管 KMS 金鑰。
4. (選用) 當您使用客戶受管金鑰加密資料目錄時，資料目錄會提供註冊 IAM 角色以加密和解密資源的選項。您需要授予可代表您假設的 IAM 角色許AWS Glue可。這包括加密和解密資料的AWS KMS權限。

   在資料目錄中建立新資源時，AWS Glue會假設提供用於加密資料的 IAM 角色。同樣地，當取用者存取資源時，AWS Glue會假設 IAM 角色來解密資料。如果您使用所需許可註冊 IAM 角色，則呼叫主體不再需要存取金鑰和解密資料的權限。

> **Important**
>
> 只有在使用客戶受管金鑰加密資料目錄資源時，才能將 KMS 作業委派給 IAM 角色。KMS 角色委派功能目前不支援使用AWS受管金鑰加密資料目錄資源。

> **Warning**
>
> 當您啟用 IAM 角色委派 KMS 作業時，您無法再存取先前使用AWS受管金鑰加密的資料目錄資源。

a. 若要啟用AWS Glue可代表您加密和解密資料的 IAM 角色，請選取將 KMS 操作委派給 IAM 角色選項。

b. 接下來，選擇 IAM 角色。

若要建立 IAM 角色，請參閱建立 AWS Glue IAM 角色。

AWS Glue假設存取資料目錄的 IAM 角色必須具有加密和解密資料目錄中繼資料的許可。您可以建立 IAM 角色，並附加下列內嵌政策：

- 新增下列原則以包含加密和解密資料目錄的AWS KMS權限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:Encrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:<region>:<account-id>:key/<key-id>"
        }
    ]
}
```

- 接下來，將下列信任政策新增至AWS Glue服務角色，以擔任 IAM 角色。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
```

```
}
```

- 接下來，將iam:PassRole權限新增至 IAM 角色。

```
                    {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::<account-id>:role/<encryption-role-name>"
            ]
        }
    ]
}
```

啟用加密時，如果尚未指定AWS Glue要假設的 IAM 角色，則存取資料目錄的主體必須具有執行下列 API 作業的權限：

- kms:Decrypt
- kms:Encrypt
- kms:GenerateDataKey

AWS CLI

使用軟體開發套件或 AWS CLI 啟用加密

- 使用 PutDataCatalogEncryptionSettings API 操作。如果未指定金鑰，則AWS Glue會使用客戶帳戶的AWS受管理加密金鑰來加密資料目錄。

```
aws glue put-data-catalog-encryption-settings \
  --data-catalog-encryption-settings '{
      "EncryptionAtRest": {
      "CatalogEncryptionMode": "SSE-KMS-WITH-SERVICE-ROLE",
      "SseAwsKmsKeyId": "arn:aws:kms:<region>:<account-id>:key/<key-id>",
```

```
      "CatalogEncryptionServiceRole":"arn:aws:iam::<account-
id>:role/<encryption-role-name>"
    }

  }'
```

啟用加密時，您在資料目錄物件中建立的所有物件都會加密。如果清除此設定，您在資料目錄中建立的物件將不再加密。您可以使用必要的 KMS 權限繼續存取資料目錄中現有的加密物件。

> **Important**
>
> AWS KMS 金鑰必須在 AWS KMS 金鑰存放區保持可供任何物件使用的狀態，這些物件是在 Data Catalog 中使用該金鑰而加密的。如果您移除金鑰，就無法再解密該物件。在某些情況下，您可能要防止使用者存取 Data Catalog 中繼資料。

監控您的 KMS 金鑰 AWS Glue

將 KMS 金鑰與資料目錄資源搭配使用時，您可以使用AWS CloudTrail或Amazon CloudWatch記錄來追蹤AWS Glue傳送至的要求AWS KMS。 AWS CloudTrail監控和記錄AWS Glue呼叫以存取由 KMS 金鑰加密之資料的 KMS 作業。

下列範例是`Decrypt`和`GenerateDataKey`作業的AWS CloudTrail事件。

Decrypt

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAXPHTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAXPHTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
```

```
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-01-10T14:33:56Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "glue.amazonaws.com"
    },
    "eventTime": "2024-01-10T15:18:11Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "eu-west-2",
    "sourceIPAddress": "glue.amazonaws.com",
    "userAgent": "glue.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "glue_catalog_id": "111122223333"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "43b019aa-34b8-4798-9b98-ee968b2d63df",
    "eventID": "d7614763-d3fe-4f84-a1e1-3ca4d2a5bbd5",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:<region>:111122223333:key/<key-id>"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "sessionCredentialFromConsole": "true"
}
```

GenerateDataKey


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId":
 "AROAXPHTESTANDEXAMPLE:V_00_GLUE_KMS_GENERATE_DATA_KEY_111122223333",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/
V_00_GLUE_KMS_GENERATE_DATA_KEY_111122223333",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAXPHTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "AKIAIOSFODNN7EXAMPLE",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-01-05T21:15:47Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "glue.amazonaws.com"
    },
    "eventTime": "2024-01-05T21:15:47Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "eu-west-2",
    "sourceIPAddress": "glue.amazonaws.com",
    "userAgent": "glue.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:eu-west-2:AKIAIOSFODNN7EXAMPLE:key/
AKIAIOSFODNN7EXAMPLE",
        "encryptionContext": {
            "glue_catalog_id": "111122223333"
        },
        "keySpec": "AES_256"
    },
    "responseElements": null,
    "requestID": "64d1783a-4b62-44ba-b0ab-388b50188070",
```

```
    "eventID": "1c73689b-2ef2-443b-aed7-8c126585ca5e",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-west-2:111122223333:key/AKIAIOSFODNN7EXAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

加密連線密碼

您可以使用 `GetConnection` 與 `GetConnections` API 操作，擷取 AWS Glue Data Catalog 中的連線密碼。 這些密碼存放在 Data Catalog 連線，並在 AWS Glue 連線至 Java Database Connectivity (JDBC) 資料存放區時使用密碼。建立連線或密碼時， Data Catalog 設定中的選項決定是否加密密碼，以及加密時指定哪項 AWS Key Management Service (AWS KMS) 金鑰。

您可在 AWS Glue 主控台的 Data catalog settings (Data Catalog 設定) 頁面上開啟此選項。

加密連線密碼

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在導覽窗格中選擇 Settings (設定)。
3. 在 Data catalog settings (Data Catalog 設定) 頁面，選擇 Encrypt connection passwords (加密連線密碼)，並選擇 AWS KMS 金鑰。

> **Important**
>
> AWS Glue 僅支援對稱客戶主金鑰 (CMK)。AWS KMS key (KMS 金鑰) 清單僅會顯示對稱金鑰。不過，如果您選取 Choose a AWS KMS key ARN (選擇 KMS 金鑰 ARN)，主控台可讓您輸入任何金鑰類型的 ARN。請確定您僅輸入對稱金鑰的 ARN。

如需更多詳細資訊，請參閱 資料目錄設定 。

對 AWS Glue 寫入的資料加密

安全組態是一組安全屬性，AWS Glue 會使用這組屬性。您可以使用安全組態來加密靜態資料。以下案例說明您可以使用安全組態的一些方式。

- 將安全組態附加到AWS Glue爬蟲以寫入加密的 Amazon CloudWatch 日誌。如需有關將安全性組態附加至編目器的詳細資訊，請參閱the section called “步驟 3：設定安全設定”。
- 將安全組態附加到擷取、轉換和載入 (ETL) 任務，以寫入加密的 Amazon Simple Storage Service (Amazon S3) 目標和加密 CloudWatch 日誌。
- 將安全組態附加到 ETL 任務來將其任務書籤寫入做為加密的 Amazon S3 資料。
- 將安全組態附加至開發端點來寫入加密的 Amazon S3 目標。

> **Important**
>
> 目前，安全組態會覆寫任何伺服器端加密 (SSE-S3) 設定，此設定會以 ETL 任務參數的形式進行傳遞。因此，此兩個安全組態和 SSE-S3 參數會與任務相關聯，且 SSE-S3 參數會遭到忽略。

如需有關安全組態的詳細資訊，請參閱在 AWS Glue 主控台上使用安全組態。

主題

- 設定 AWS Glue 來使用安全組態
- 建立路由至 AWS KMS 供 VPC 任務和爬蟲程式使用
- 在 AWS Glue 主控台上使用安全組態

設定 AWS Glue 來使用安全組態

遵循這些步驟來設定 AWS Glue 環境，即可使用安全組態。

1. 建立或更新您的 AWS Key Management Service (AWS KMS) 金鑰，將AWS KMS權限授與傳遞給AWS Glue檢索器和工作以加密 CloudWatch 記錄的 IAM 角色。如需詳細資訊，請參閱 Amazon CloudWatch 日誌使用者指南中的使用加密 CloudWatch 日誌AWS KMS中的日誌資料。

   在下列範例中，*"role1"*、*"role2"* 和 *"role3"* 為 IAM 角色，這些角色會傳送到爬蟲程式和任務。

   ```
   {
          "Effect": "Allow",
          "Principal": { "Service": "logs.region.amazonaws.com",
          "AWS": [
                   "role1",
                   "role2",
                   "role3"
                ] },
                      "Action": [
                             "kms:Encrypt*",
                             "kms:Decrypt*",
                             "kms:ReEncrypt*",
                             "kms:GenerateDataKey*",
                             "kms:Describe*"
                      ],
                      "Resource": "*"
   }
   ```

   如果您使用金鑰加密 CloudWatch 記錄檔`"Service": "logs.region.amazonaws.com"`，則需要顯示為`Service`陳述式。

2. 在使用 AWS KMS 金鑰前，確保此金鑰就是 ENABLED。

> **Note**
>
> 若您使用 Iceberg 作為資料湖架構，Iceberg 資料表會透過其專屬機制啟用伺服器端加密。除了 AWS Glue 的安全組態之外，您還應啟用這些組態。若要在 Iceberg 資料表上啟用伺服器端加密，請檢閱 Iceberg 文件中的指引。

建立路由至 AWS KMS 供 VPC 任務和爬蟲程式使用

您可以透過在 Virtual Private Cloud (VPC) 內的私有端點直接連接至 AWS KMS，而無需連接至網際網路。使用 VPC 端點時，VPC 和 AWS KMS 之間的通訊會在整個 AWS 網路中執行。

您可以在 VPC 中建立 AWS KMS VPC 端點。少了這個步驟，任務和爬蟲程式可能會失敗，而在任務上出現 `kms timeout` 或在爬蟲程式上出現 `internal service exception`。如需詳細說明，請參閱 AWS Key Management Service 開發人員指南的透過 VPC 端點連接到 AWS KMS。

當您在 VPC 主控台上遵循這些指示時，您必須執行以下：

- 選取 Enable Private DNS name (啟用私人 DNS 名稱)。
- 選擇用於存取 Java Database Connectivity (JDBC) 的任務或爬蟲程式的 Security group (安全群組) (含自我參考規則)。如需有關 AWS Glue 連線的詳細資訊，請參閱 連線至資料。

當您新增安全組態到存取 JDBC 資料存放區的爬蟲程式或任務時，AWS Glue 必須有連接 AWS KMS 端點的路由。您可以透過網路位址轉譯 (NAT) 閘道或 AWS KMS VPC 端點提供路由。若要建立 NAT 閘道，請參閱 Amazon VPC 使用者指南中的 NAT 閘道。

在 AWS Glue 主控台上使用安全組態

> **Warning**
>
> Ray 任務中目前不支援 AWS Glue 安全組態。

AWS Glue 中安全組態的包含在您寫入加密資料時所需的屬性。您將在 AWS Glue 主控台建立安全組態，以提供爬蟲程式、任務以及開發端點所使用的加密屬性。

若要查看您已建立的所有安全組態清單，請開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台，然後在導覽窗格中選擇 Security configurations (安全組態) 標籤。

安全組態清單顯示有關各個組態的下列屬性：

Name

在建立組態時提供的唯一獨特名稱。名稱可包含字母 (A-Z)、數字 (0-9)、連字號 (-) 或底線 (_)，並且長度上限為 255 個字元。

啟用 Amazon S3 加密

如果開啟，則會針對資料型錄中的中繼資料存放區啟用 SSE-KMS 或 SSE-S3 等 Amazon Simple Storage Service (Amazon S3) 等加密模式。

啟用 Amazon CloudWatch Logs 加密

如果開啟，則會在將日誌寫入 Amazon CloudWatch 時啟用 SSE-KMS 等 Amazon S3 加密模式。

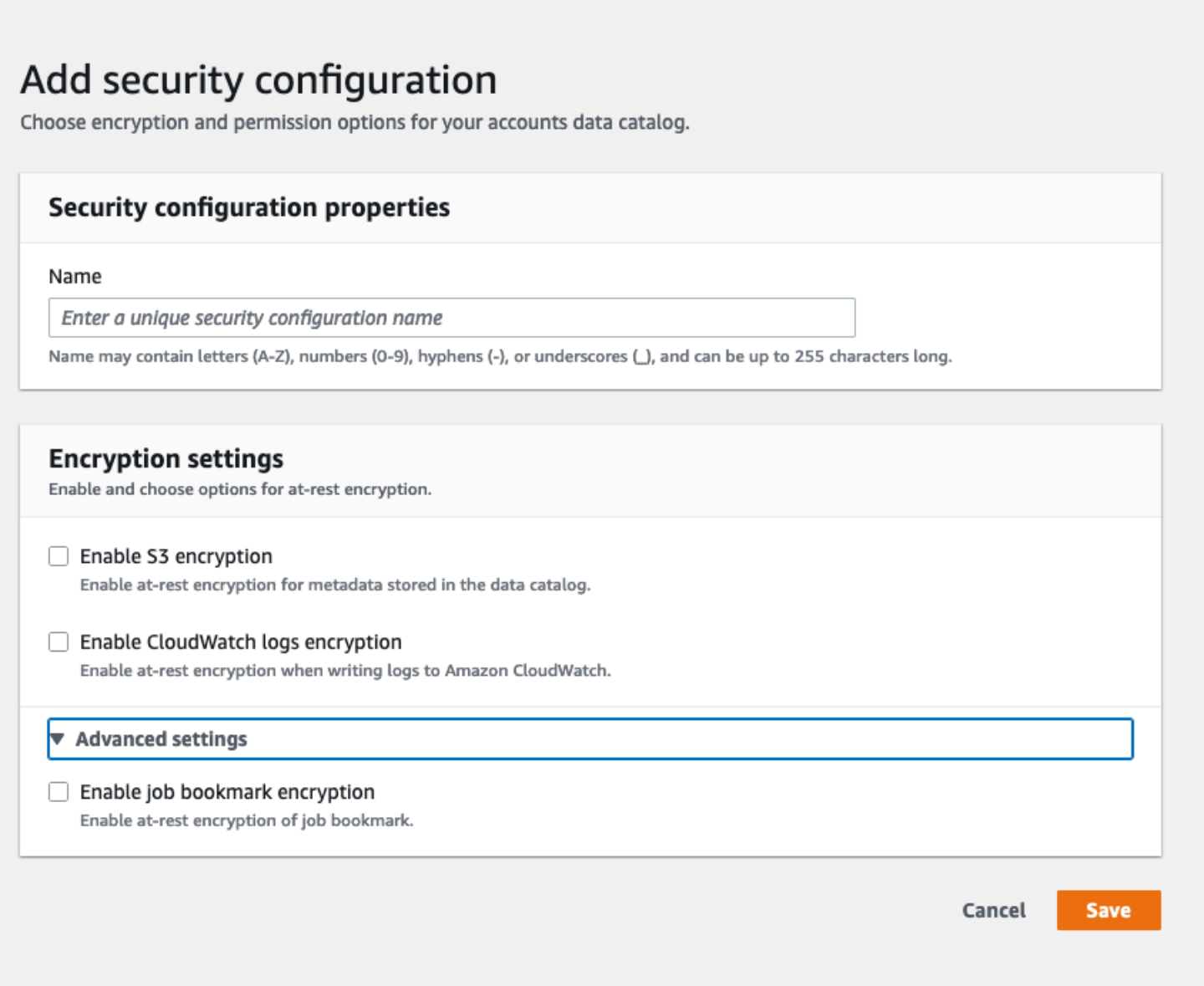
進階設定：啟用任務書籤加密

如果開啟，則會在為任務加上書籤時啟用 CSE-KMS 等 Amazon S3 加密模式。

您可以在主控台的 Security configurations (安全組態) 部分中新增或刪除組態。若要查看組態詳細資訊，請在清單中選擇組態的名稱。詳細資訊包含您在建立組態時所定義的資訊。

新增安全組態

若要使用 AWS Glue 主控台新增安全組態，請在 Security configurations (安全組態) 頁面，選擇 Add security configuration (新增安全性設定)。



安全性組態屬性

輸入不重複的安全性組態名稱。名稱可包含字母 (A-Z)、數字 (0-9)、連字號 (-) 或底線 (_)，並且長度上限為 255 個字元。

加密設定

您可以針對存放在 Amazon S3 資料型錄中的中繼資料和 Amazon CloudWatch 中的日誌啟用靜態加密。若要在 AWS Glue 主控台上使用 AWS Key Management Service (AWS KMS) 金鑰設定資料與中繼資料加密，請新增政策至主控台使用者。此政策必須指定允許的資源做為金鑰 Amazon Resource Name (ARN) (用於加密 Amazon S3 資料存放區)，如以下範例所示。

```
{
```

```
"Version": "2012-10-17",
  "Statement": {
  "Effect": "Allow",
  "Action": [
    "kms:GenerateDataKey",
    "kms:Decrypt",
    "kms:Encrypt"],
  "Resource": "arn:aws:kms:region:account-id:key/key-id"}
}
```

> **Important**
>
> 當安全組態連接至爬蟲程式或任務時，傳遞的 IAM 角色必須擁有 AWS KMS 許可。如需詳細資訊，請參閱 對 AWS Glue 寫入的資料加密。

當您定義組態時，您可為下列屬性提供值：

啟用 S3 加密

當您寫入 Amazon S3 資料時，您將使用 Amazon S3 受管金鑰 (SSE-S3) 或 AWS KMS 受管金鑰 (SSE-KMS) 的伺服器端加密。此欄位為選用欄位。若要允許存取 Amazon S3，請選擇 AWS KMS 金鑰或選擇 Enter a key ARN (輸入金鑰 ARN)，並提供該金鑰的 ARN。以 `arn:aws:kms:region:account-id:key/key-id` 的形式輸入 ARN。您也可以提供 ARN 做為金鑰別名，例如 `arn:aws:kms:region:account-id:alias/alias-name`。

如果您為任務啟用 Spark UI，則上傳到 Amazon S3 的 Spark UI 日誌檔案將採用相同的加密方式。

> **Important**
>
> AWS Glue 僅支援對稱客戶主金鑰 (CMK)。AWS KMS key (KMS 金鑰) 清單僅會顯示對稱金鑰。不過，如果您選取 Choose a AWS KMS key ARN (選擇 KMS 金鑰 ARN)，主控台可讓您輸入任何金鑰類型的 ARN。請確定您僅輸入對稱金鑰的 ARN。

啟用 CloudWatch Logs 加密

伺服器端 (SSE-KMS) 加密是用來加密 CloudWatch Logs。此欄位為選用欄位。選擇 AWS KMS 金鑰或選擇 Enter a key ARN (輸入金鑰 ARN)，並提供該金鑰的 ARN 來開啟它。以

arn:aws:kms:*region*:*account-id*:key/*key-id* 的形式輸入 ARN。您也可以提供 ARN 做為金鑰別名，例如 arn:aws:kms:*region*:*account-id*:alias/*alias-name*。

進階設定：任務書籤加密

用戶端 (CSE-KMS) 加密是用於加密任務書籤。此欄位為選用欄位。書籤資料會先加密後才傳送至 Amazon S3 儲存。選擇 AWS KMS 金鑰或選擇 Enter a key ARN (輸入金鑰 ARN)，並提供該金鑰的 ARN 來開啟它。以 arn:aws:kms:*region*:*account-id*:key/*key-id* 的形式輸入 ARN。您也可以提供 ARN 做為金鑰別名，例如 arn:aws:kms:*region*:*account-id*:alias/*alias-name*。

如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的下列主題：

- 如需關於 SSE-S3 的詳細資訊，請參閱使用伺服器端加密與 Amazon S3 受管加密金鑰 (SSE-S3) 保護資料。
- 如需有關 SSE-KMS 的詳細資訊，請參閱 Protecting Data Using Server-Side Encryption with AWS KMS keys。
- 如需有關 CSE-KMS 的資訊，請參閱 Using a KMS key stored in AWS KMS。

傳輸中加密

AWS 為傳輸中的資料提供 Transport Layer Security (TLS) 加密。您可以使用AWS Glue 中的安全組態設定爬蟲程式、ETL 任務和開發端點的加密設定。您可以透過 Data Catalog 的設定開啟 AWS Glue Data Catalog 加密。

截至 2018 年 9 月 4 日，支援適用於 AWS Glue ETL 的 AWS KMS (使用自有金鑰和伺服器端加密) 和 AWS Glue Data Catalog。

FIPS 合規

如果您在透過命令列介面或 API 存取 AWS 時，需要 FIPS 140-2 驗證的加密模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的詳細資訊，請參閱聯邦資訊處理標準 (FIPS) 140-2 概觀。

金鑰管理

您可以搭配 AWS Identity and Access Management (IAM) 使用 AWS Glue 來定義使用者、AWS 資源、群組、角色和與存取、拒絕等相關的細部政策。

您可以使用資源類型政策和身分識別型政策定義對中繼資料的存取權限，取決於您組織的需求。資源類型政策會列出允許或拒絕存取您資源的委託人，讓您可以設定像是跨帳戶存取等政策。身分政策則是特別連接到 IAM 內使用者、群組和角色的政策。

如需逐步範例，請參閱 AWS Big Data 部落格上的使用資源層級 IAM 許可和資源類型政策限制對您 AWS Glue Data Catalog 的存取。

政策的細部存取部分則會定義在 `Resource` 子句中。這部分會定義可在其上執行動作的 AWS Glue Data Catalog 物件，以及該操作會傳回什麼結果物件。

「開發端點」是一種環境，可讓您用於開發及測試 AWS Glue 指令碼` 您可以新增、刪除或輪換開發端點的 SSH 金鑰。

截至 2018 年 9 月 4 日，支援適用於 AWS KMS ETL 的 AWS Glue (使用自有金鑰和伺服器端加密) 和 AWS Glue Data Catalog。

AWS Glue 對其他 AWS 服務的相依性

使用者若要使用 AWS Glue 主控台，必須擁有一組最基本的許可，以使用其 AWS 帳戶的 AWS Glue 資源。除了這些 AWS Glue 許可之外，主控台還需要以下服務的許可：

- 顯示日誌的 Amazon CloudWatch Logs 許可。
- 列出並傳遞角色的 AWS Identity and Access Management (IAM) 許可。
- 使用堆疊的 AWS CloudFormation 許可。
- 列出虛擬私有雲端 (VPC)、子網路、安全群組、執行個體和其他物件的 Amazon Elastic Compute Cloud (Amazon EC2) 許可 (以在執行任務、爬取程式和建立開發端點時設定像是 VPC 等 Amazon EC2 項目)。
- 列出儲存貯體和物件，以及擷取和儲存指令碼的 Amazon Simple Storage Service (Amazon S3) 許可。
- 使用叢集的 Amazon Redshift 許可
- 列出執行個體的 Amazon Relational Database Service (Amazon RDS) 許可。

開發端點

開發端點是一種環境，可讓您用於開發及測試 AWS Glue 指令碼。您可以使用 AWS Glue 來建立、編輯和刪除開發端點。您可以列出所有已建立的開發端點。您可以新增、刪除或輪換開發端點的 SSH 金鑰。您也可以建立使用開發端點的筆記本。

您可以提供組態值來佈建開發環境。這些值會告訴 AWS Glue 如何設定網路，讓您安全地存取您的開發端點，同時讓您的端點可以存取您的資料存放區。然後，您便可以建立筆記本，連線到開發端點。您可以使用您的筆記本來編寫和測試 ETL 指令碼。

使用具備與您用來執行 AWS Glue ETL 任務 IAM 角色相似許可的 AWS Identity and Access Management (IAM) 角色。使用虛擬私有雲端 (VPC)、子網路和安全群組來建立可安全地連線到您資料資源的開發端點。您可以產生 SSH 金鑰對來使用 SSH 連線到開發環境。

您可以為 Amazon S3 資料，以及在您可以透過 JDBC 存取資料集的 VPC 中建立開發端點。

您可以在本機電腦上安裝 Jupyter 筆記本，並在開發端點上將其用於偵錯和測試 ETL 指令碼。或者，您可使用 Sagemaker 筆記本在 AWS 上的 JupyterLab 中撰寫 ETL 指令碼。請參閱搭配開發端點使用 SageMaker 筆記本。

AWS Glue 會使用附帶 `aws-glue-dev-endpoint` 字首的名稱，為 Amazon EC2 執行個體加上標籤。

您可以在開發端點上設定筆記本伺服器，以使用 AWS Glue 延伸模組執行 PySpark。

AWS Glue 的身分識別與存取管理

AWS Identity and Access Management (IAM) 可協助系統管理員安全地控制 AWS 資源存取權。 AWS 服務 IAM 管理員控制哪些人可以驗證 (登入) 和授權 (具有權限) 以使用 AWS Glue 資源。IAM 是您可以使用的 AWS 服務 ，無需額外付費。

> **Note**
>
> 您可以使用其中一AWS Glue種方法或授權，授與 AWS Glue 資料型錄中資料的存取 AWS Lake Formation 權。您可以使用 AWS Identity and Access Management (IAM) 政策，透過AWS Glue方法設定精細的存取控制。Lake Formation 使用更簡單的 `GRANT/REVOKE` 許可模型，這類似於關聯式資料庫系統中的 `GRANT/REVOKE` 命令。
> 本節包含如何使用 AWS Glue 方法的相關資訊。如需使用 Lake Formation 授予的詳細資訊，請參閱AWS Lake Formation 開發人員指南中的授予 Lake Formation 許可。

主題

- 物件
- 使用身分驗證

- 使用政策管理存取權
- AWS Glue 如何與 IAM 搭配使用
- 設定 AWS Glue 的 IAM 許可
- AWS Glue 存取控制政策範例
- AWSAWS Glue 的受管理原則
- 指定 AWS Glue 資源 ARN
- 授予跨帳戶存取權
- 疑難排解 AWS Glue 身分識別

物件

根據您在 AWS Glue 中執行的工作，使用方式 AWS Identity and Access Management (IAM) 會有所不同。

服務使用者 — 如果您使用 AWS Glue 服務執行工作，則管理員會為您提供所需的認證和權限。當您使用更多 AWS Glue 功能完成工作時，您可能需要額外的權限。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。若您無法存取 AWS Glue 中的某項功能，請參閱 疑難排解 AWS Glue 身分識別。

服務管理員 — 如果您負責公司的 AWS Glue 資源，您可能擁有 AWS Glue 的完整存取權。決定您的服務使用者應該存取哪些 AWS Glue 功能和資源是您的工作。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要深入瞭解貴公司如何搭配 AWS Glue 使用 IAM，請參閱AWS Glue 如何與 IAM 搭配使用。

IAM 管理員 — 如果您是 IAM 管理員，可能需要瞭解如何撰寫政策以管理 AWS Glue 存取權的詳細資訊。若要檢視可在 IAM 中使用的 AWS Glue 身分型政策範例，請參閱。AWS Glue 的身分型政策範例

使用身分驗證

驗證是您 AWS 使用身分認證登入的方式。您必須以 IAM 使用者身分或假設 IAM 角色進行驗證 (登入 AWS)。 AWS 帳戶根使用者

您可以使用透過 AWS 身分識別來源提供的認證，以聯合身分識別身分登入。 AWS IAM Identity Center (IAM 身分中心) 使用者、貴公司的單一登入身分驗證，以及您的 Google 或 Facebook 登入資料都是聯合身分識別的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使 AWS 用同盟存取時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需有關登入的詳細資訊 AWS，請參閱《AWS 登入 使用指南》 AWS 帳戶中的如何登入您的。

如果您 AWS 以程式設計方式存取，請 AWS 提供軟體開發套件 (SDK) 和命令列介面 (CLI)，以使用您的認證以加密方式簽署您的要求。如果您不使用 AWS 工具，則必須自行簽署要求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱 IAM 使用者指南中的簽署 AWS API 請求。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如， AWS 建議您使用多重要素驗證 (MFA) 來增加帳戶的安全性。如需更多資訊，請參閱 AWS IAM Identity Center 使用者指南中的多重要素驗證和 IAM 使用者指南中的在 AWS中使用多重要素驗證 (MFA)。

AWS 帳戶 根使用者

當您建立時 AWS 帳戶，您會從一個登入身分開始，該身分可完整存取該帳戶中的所有資源 AWS 服務和資源。此身分稱為 AWS 帳戶 root 使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常任務。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱 IAM 使用者指南中的需要根使用者憑證的任務。

聯合身分

最佳作法是要求人類使用者 (包括需要系統管理員存取權的使用者) 使用與身分識別提供者的同盟，才能使用臨時認證 AWS 服務 來存取。

聯合身分識別是來自企業使用者目錄的使用者、Web 身分識別提供者、Identi ty Center 目錄，或使用透過身分識別來源提供的認證進行存取 AWS 服務 的任何使用者。 AWS Directory Service同盟身分存取時 AWS 帳戶，他們會假設角色，而角色則提供臨時認證。

對於集中式存取權管理，我們建議您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中建立使用者和群組，也可以連線並同步到自己身分識別來源中的一組使用者和群組，以便在所有應用程式 AWS 帳戶 和應用程式中使用。如需 IAM Identity Center 的詳細資訊，請參閱 AWS IAM Identity Center 使用者指南中的什麼是 IAM Identity Center？。

IAM 使用者和群組

IAM 使用者是您內部的身分，具 AWS 帳戶 有單一人員或應用程式的特定許可。建議您盡可能依賴暫時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需更多資訊，請參閱 IAM 使用者指南中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分簽署。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的程序變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供暫時憑證。如需進一步了解，請參閱IAM 使用者指南中的建立 IAM 使用者 (而非角色) 的時機。

IAM 角色

IAM 角色是您 AWS 帳戶 內部具有特定許可的身分。它類似 IAM 使用者，但不與特定的人員相關聯。您可以切換角色，在中暫時擔任 IAM 角色。 AWS Management Console 您可以透過呼叫 AWS CLI 或 AWS API 作業或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱 IAM 使用者指南中的使用 IAM 角色。

使用暫時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需有關聯合角色的相關資訊，請參閱 IAM 使用者指南中的為第三方身分提供者建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的許可集。
- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取權：您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的主體) 存取您帳戶的資源。角色是授予跨帳戶存取權的主要方式。但是，對於某些策略 AWS 服務，您可以將策略直接附加到資源（而不是使用角色作為代理）。若要了解跨帳戶存取角色和以資源為基礎的政策之間的差異，請參閱 IAM 使用者指南中的 IAM 中的跨帳戶資源存取。
- 跨服務訪問 — 有些 AWS 服務 使用其他 AWS 服務功能。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉寄存取工作階段 (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 會使用主體呼叫的權限 AWS 服務，並結合要求 AWS 服務 向下游服務發出要求。只有當服務收到需要與其 AWS 服務 他資源互動才能完成的請求時，才會發出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱《轉發存取工作階段》。
  - 服務角色 – 服務角色是服務擔任的 IAM 角色，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱 IAM 使用者指南中的建立角色以委派許可給 AWS 服務服務。

- 服務連結角色 — 服務連結角色是連結至. 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 且屬於服務所有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon EC2 上執行的應用程式 — 您可以使用 IAM 角色來管理在 EC2 執行個體上執行的應用程式以及發出 AWS CLI 或 AWS API 請求的臨時登入資料。這是在 EC2 執行個體內儲存存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體並提供給其所有應用程式，請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該角色，並且可讓 EC2 執行個體上執行的程式取得暫時憑證。如需詳細資訊，請參閱 IAM 使用者指南中的利用 IAM 角色來授予許可給 Amazon EC2 執行個體上執行的應用程式。

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

使用政策管理存取權

您可以透 AWS 過建立原則並將其附加至 AWS 身分識別或資源來控制中的存取。原則是一個物件 AWS ，當與身分識別或資源相關聯時，會定義其權限。 AWS 當主參與者 (使用者、root 使用者或角色工作階段) 提出要求時，評估這些原則。政策中的許可決定是否允許或拒絕請求。大多數原則會 AWS 以 JSON 文件的形式儲存在中。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哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該原則的使用者可以從 AWS Management Console AWS CLI、或 AWS API 取得角色資訊。

身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分類型政策，請參閱 IAM 使用者指南中的建立 IAM 政策。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管理的策略是獨立策略，您可以將其附加到您的 AWS 帳戶. 受管政策包括 AWS 受管政策和客戶管理

的策略。如需了解如何在受管政策及內嵌政策間選擇，請參閱 IAM 使用者指南中的在受管政策和內嵌政策間選擇。

資源型政策

資源型政策是連接到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。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。主參與者可以包括帳戶、使用者、角色、同盟使用者或。 AWS 服務

資源型政策是位於該服務中的內嵌政策。您無法在以資源為基礎的政策中使用 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 和 Amazon VPC 是支援 ACL 的服務範例。 AWS WAF如需進一步了解 ACL，請參閱 Amazon Simple Storage Service 開發人員指南中的存取控制清單 (ACL) 概觀。

其他政策類型

AWS 支援其他較不常見的原則類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可界限的詳細資訊，請參閱 IAM 使用者指南中的 IAM 實體許可界限。
- 服務控制策略（SCP）— SCP 是 JSON 策略，用於指定中組織或組織單位（OU）的最大權限。AWS Organizations AWS Organizations 是一種用於分組和集中管理您企業擁 AWS 帳戶 有的多個服務。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 限制成員帳戶中實體的權限，包括每個 AWS 帳戶根使用者帳戶。如需 Organizations 和 SCP 的詳細資訊，請參閱 AWS Organizations 使用者指南中的 SCP 運作方式。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合使用者的暫時工作階段時，作為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作

階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，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。要了解如何在涉及多個政策類型時 AWS 確定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

AWS Glue 如何與 IAM 搭配使用

在您使用 IAM 管理 AWS Glue 的存取權限之前，請先了解哪些 IAM 功能可與 AWS Glue 搭配使用。

您可以搭配 AWS Glue 使用的 IAM 功能

IAM 功能	AWS Glue 支持
身分型政策	是
資源型政策	部分
政策動作	是
政策資源	是
政策條件索引鍵 (服務特定)	是
ACL	否
ABAC(政策中的標籤)	部分
臨時憑證	是
主體許可	否
服務角色	是
服務連結角色	否

若要深入瞭解 AWS Glue 和其他 AWS 服務如何搭配大多數 IAM 功能使用，請參閱 IAM 使用者指南中的搭配 IAM 使用的AWS 服務。

Glue 的身分識別原則 AWS

支援身分型政策	是

身分型政策是可以連接到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分類型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策。

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。您無法在身分型政策中指定主體，因為這會套用至連接的使用者或角色。如要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素參考。

AWS Glue 支援所有 AWS Glue 操作的身分識別型政策 (IAM 政策)。連接政策，即可授予建立、存取或修改 AWS Glue 資源 (例如 AWS Glue Data Catalog 中的資料表) 的許可。

Glue 的身分識別原則範例 AWS

若要檢視 AWS Glue 身分型原則的範例，請參閱。AWS Glue 的身分型政策範例

Glue 中 AWS 基於資源的政策

支援以資源基礎的政策	部分

資源型政策是附加到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。主參與者可以包括帳戶、使用者、角色、同盟使用者或。 AWS 服務

如需啟用跨帳戶存取權，您可以指定在其他帳戶內的所有帳戶或 IAM 實體，作為資源型政策的主體。新增跨帳戶主體至資源型政策，只是建立信任關係的一半。當主體和資源位於不同時 AWS 帳戶，受信任帳戶中的 IAM 管理員也必須授與主體實體 (使用者或角色) 權限，才能存取資源。其透過將身分型政策連接到實體來授與許可。不過，如果資源型政策會為相同帳戶中的主體授予存取，這時就不需要額外的身分型政策。如需詳細資訊，請參閱 IAM 使用者指南中的 IAM 中的跨帳戶資源存取。

> **Note**
>
> 您只能使用 AWS Glue 資源政策來管理 Data Catalog 資源的許可。您無法將它連接到任何其他 AWS Glue 資源 (例如任務、觸發、開發端點、爬蟲程式或分類器)。
> 每個目錄只允許一項資源政策，而且其大小限制為 10 KB。

在 AWS Glue 中，資源原則會附加至目錄，目錄是前面提到之各種資料目錄資源的虛擬容器。每個 AWS 帳戶在 AWS 區域中擁有一個目錄，其目錄 ID 與 AWS 帳戶 ID 相同。無法刪除或修改目錄。

針對目錄的所有 API 呼叫評估資源政策，而在此目錄中，發起人委託人包含在政策文件的 `"Principal"` 區塊中。

若要檢視以 AWS Glue 資源為基礎的原則範例，請參閱AWS Glue 以資源型為基礎的範例。

AWS Glue 的政策行動

支援政策動作	是

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 `Action` 元素描述您可以用來允許或拒絕政策中存取的動作。原則動作通常與關聯的 AWS API 作業具有相同的名稱。有一些例外狀況，例如沒有相符的 API 操作的僅限許可動作。也有一些作業需要政策中的多個動作。這些額外的動作稱為相依動作。

政策會使用動作來授予執行相關聯動作的許可。

若要查看 AWS Glue 動作清單，請參閱服務授權參考中由 AWS Glue 定義的動作。

AWS Glue 中的策略操作在操作之前使用以下前綴：

```
glue
```

如需在單一陳述式中指定多個動作，請用逗號分隔。

```
"Action": [
```

```
      "glue:action1",
      "glue: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 Get 文字的所有動作，請包含以下動作：

```
"Action": "glue:Get*"
```

若要檢視範例原則，請參閱AWS Glue 存取控制政策範例。

AWS Glue 的政策資源

支援政策資源	是

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Resource JSON 政策元素可指定要套用動作的物件。陳述式必須包含 Resource 或 NotResource 元素。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出操作)，請使用萬用字元 (*) 來表示陳述式適用於所有資源。

```
"Resource": "*"
```

如需有關如何使用 ARN 控制 AWS Glue 資源存取的詳細資訊，請參閱指定 AWS Glue 資源 ARN。

若要查看 AWS Glue 資源類型及其 ARN 的清單，請參閱服務授權參考資料中由 AWS Glue 定義的資源。若要瞭解您可以使用哪些動作來指定每個資源的 ARN，請參閱 AWS Glue 定義的動作。

AWS Glue 的政策條件金鑰

支援服務特定政策條件金鑰	是

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Condition 元素 (或 Condition 區塊) 可讓您指定使陳述式生效的條件。Condition 元素是選用項目。您可以建立使用條件運算子的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。

若您在陳述式中指定多個 Condition 元素，或是在單一 Condition 元素中指定多個索引鍵， AWS 會使用邏輯 AND 操作評估他們。如果您為單一條件索引鍵指定多個值，請使用邏輯OR運算來 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，您可以只在使用者使用其 IAM 使用者名稱標記時，將存取資源的許可授予該 IAM 使用者。如需更多資訊，請參閱 IAM 使用者指南中的 IAM 政策元素：變數和標籤。

AWS 支援全域條件金鑰和服務特定條件金鑰。若要查看所有 AWS 全域條件金鑰，請參閱《IAM 使用者指南》中的AWS 全域條件內容金鑰。

若要查看 AWS Glue 條件金鑰清單，請參閱服務授權參考資料中的 AWS Glue 條件金鑰。若要瞭解您可以使用條件金鑰的動作和資源，請參閱 AWS Glue 定義的動作。

若要檢視範例原則，請參閱使用條件索引鍵或內容索引鍵來控制設定。

Glue 中的 AWS ACL

支援 ACL	否

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

用 Glue AWS

支援 ABAC (政策中的標籤)	部分

屬性型存取控制 (ABAC) 是一種授權策略，可根據屬性來定義許可。在中 AWS，這些屬性稱為標籤。您可以將標籤附加到 IAM 實體 (使用者或角色) 和許多 AWS 資源。為實體和資源加上標籤是 ABAC 的第一步。您接著要設計 ABAC 政策，允許在主體的標籤與其嘗試存取的資源標籤相符時操作。

ABAC 在成長快速的環境中相當有幫助，並能在政策管理變得繁瑣時提供協助。

如需根據標籤控制存取，請使用 aws:ResourceTag/*key-name*、aws:RequestTag/*key-name* 或 aws:TagKeys 條件索引鍵，在政策的條件元素中，提供標籤資訊。

如果服務支援每個資源類型的全部三個條件金鑰，則對該服務而言，值為 Yes。如果服務僅支援某些資源類型的全部三個條件金鑰，則值為 Partial。

如需 ABAC 的詳細資訊，請參閱 IAM 使用者指南中的什麼是 ABAC?。如要查看含有設定 ABAC 步驟的教學課程，請參閱 IAM 使用者指南中的使用屬性型存取控制 (ABAC)。

> **Important**
>
> 條件內容金鑰只會套用在爬蟲程式、任務、觸發條件和開發端點上的 AWS Glue API 動作。如需有關哪些 API 作業受到影響的詳細資訊，請參閱 AWS Glue 的條件金鑰。
> AWS Glue Data Catalog API 操作目前不支援 aws:referer 和 aws:UserAgent 全域條件內容索引鍵。

若要檢視身分型政策範例，以根據該資源上的標籤來限制存取資源，請參閱使用標籤授權存取。

使用臨時憑證與 AWS Glue

支援臨時憑證	是

當您使用臨時憑據登錄時，某些 AWS 服務 不起作用。如需其他資訊，包括哪些 AWS 服務 與臨時登入資料搭配AWS 服務 使用，請參閱 IAM 使用者指南中的 IAM。

如果您使用除了使用者名稱和密碼以外的任何方法登入，則您正在 AWS Management Console 使用臨時認證。例如，當您 AWS 使用公司的單一登入 (SSO) 連結存取時，該程序會自動建立暫時認證。當您以使用者身分登入主控台，然後切換角色時，也會自動建立臨時憑證。如需切換角色的詳細資訊，請參閱 IAM 使用者指南中的切換至角色 (主控台)。

您可以使用 AWS CLI 或 AWS API 手動建立臨時登入資料。然後，您可以使用這些臨時登入資料來存取 AWS。 AWS 建議您動態產生臨時登入資料，而不是使用長期存取金鑰。如需詳細資訊，請參閱 IAM 中的暫時性安全憑證。

Glue 的 AWS 跨服務主體權限

支援轉寄存取工作階段 (FAS)	否

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 會使用主體呼叫的權限 AWS 服務，並結合要求 AWS 服務 向下游服務發出要求。只有當服務收到需要與其 AWS 服務 他資源互動才能完成的請求時，才會發出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱《轉發存取工作階段》。

AWS Glue 的服務角色

支援服務角色	是

服務角色是服務擔任的 IAM 角色，可代您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱 IAM 使用者指南中的建立角色以委派許可給 AWS 服務服務。

> **Warning**
>
> 變更服務角色的權限可能會中斷 AWS Glue 功能。只有在 AWS Glue 提供指引時才編輯服務角色。

如需建立 AWS Glue 服務角色的詳細說明，請參閱步驟 1：建立適用於 AWS Glue 服務的 IAM 政策和步驟 2：為 AWS Glue 建立 IAM 角色。

Glue 的 AWS 服務連結角色

支援服務連結角色。	否

服務連結角色是一種連結至. 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 且屬於服務所有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

如需建立或管理服務連結角色的詳細資訊，請參閱可搭配 IAM 運作的AWS 服務。在表格中尋找服務，其中包含服務連結角色欄中的 Yes。選擇是連結，以檢視該服務的服務連結角色文件。

設定 AWS Glue 的 IAM 許可

您使用 AWS Identity and Access Management (IAM) 來定義 AWS Glue 在存取資源時所用的政策和角色。下列步驟會引導您選擇各種選項，以為 AWS Glue 設定許可。根據您的商業需求，您可能需要新增或降低資源存取。

> **Note**
>
> 若要改為開始使用 AWS Glue 的基本 IAM 許可，請參閱 設定下列項目的 IAM 許可 AWS Glue。

1. 建立適用於 AWS Glue 服務的 IAM 政策：建立允許存取 AWS Glue 資源的服務政策。
2. 為 AWS Glue 建立 IAM 角色：建立 IAM 角色，並連接 AWS Glue 服務政策和 AWS Glue 使用的 Amazon Simple Storage Service (Amazon S3) 資源的政策。
3. 連接政策到存取 AWS Glue 的使用者或群組：連接政策到登入 AWS Glue 主控台的任何使用者或群組。
4. 建立適用於筆記本的 IAM 政策：建立筆記本伺服器政策，以在開發端點上建立筆記本伺服器時使用。
5. 建立適用於筆記本的 IAM 角色：建立 IAM 角色，並連接筆記本伺服器政策。
6. 建立 Amazon SageMaker 筆記本的 IAM 政策：建立 IAM 政策，以在開發端點上建立 Amazon SageMaker 筆記本時使用。
7. 為 Amazon SageMaker 筆記本建立 IAM 角色：建立 IAM 角色，並連接政策，以在開發端點上建立 Amazon SageMaker 筆記本時授予許可。

步驟 1：建立適用於 AWS Glue 服務的 IAM 政策

對於存取其他 AWS 資源上的資料的任何操作，例如存取 Amazon S3 中的物件，AWS Glue 需要能代為存取資源的許可。您可用 AWS Identity and Access Management (IAM) 來提供這些許可。

> **Note**
>
> 如果您使用 AWS 受管政策 **`AWSGlueServiceRole`**，可跳過此步驟。

在本步驟中，您將建立一個類似 `AWSGlueServiceRole` 的政策。最新版本的 `AWSGlueServiceRole` 可在 IAM 主控台找到。

建立適用於 AWS Glue 的 IAM 政策

此政策授予部分 Amazon S3 動作的許可，在 AWS Glue 擔任使用此政策的角色時可用於管理其所需的帳戶內資源。此政策指定的部分資源參照 AWS Glue 用於 Amazon S3 儲存貯體、Amazon S3 ETL 指

令碼、CloudWatch Logs 和 Amazon EC2 資源的預設名稱。為了簡化，AWS Glue 寫入部分 Amazon S3 物件到您預設字首為 `aws-glue-*` 之帳戶的儲存貯體內。

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在左側導覽窗格中選擇 Policies (政策)。
3. 選擇 建立政策 。
4. 在 Create Policy (建立政策) 畫面上導覽至索引標籤，以編輯 JSON。使用下列 JSON 陳述式建立政策文件，然後選擇 Review policy (檢閱政策)。

> **Note**
>
> 新增 Amazon S3 資源所需的任何許可。您可能希望將存取政策的資源部分範圍限縮在需要的資源。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:*",
                "s3:GetBucketLocation",
                "s3:ListBucket",
                "s3:ListAllMyBuckets",
                "s3:GetBucketAcl",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeRouteTables",
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "iam:ListRolePolicies",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "cloudwatch:PutMetricData"
            ],
```

```
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutBucketPublicAccessBlock"
            ],
            "Resource": [
                "arn:aws:s3:::aws-glue-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::aws-glue-*/*",
                "arn:aws:s3:::*/*aws-glue-*/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::crawler-public*",
                "arn:aws:s3:::aws-glue-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:AssociateKmsKey"
            ],
```

```
            "Resource": [
                "arn:aws:logs:*:*:log-group:/aws-glue/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags",
                "ec2:DeleteTags"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": [
                        "aws-glue-service-resource"
                    ]
                }
            },
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:instance/*"
            ]
        }
    ]
}
```

下表說明此政策授予的許可。

Action	Resource	Description (描述)
`"glue:*"`	`"*"`	授予執行所有 AWS Glue API 操作的許可。
`"s3:GetBucketLocation", "s3:ListBucket", "s3:ListAllMyBuckets", "s3:GetBucketAcl",`	`"*"`	允許列出爬蟲程式、任務、開發端點和筆記本伺服器的 Amazon S3 儲存貯體。

Action	Resource	Description (描述)
`"ec2:DescribeVpcEndpoints", "ec2:DescribeRouteTables", "ec2:CreateNetworkInterface", "ec2:DeleteNetworkInterface", "ec2:DescribeNetworkInterfaces", "ec2:DescribeSecurityGroups", "ec2:DescribeSubnets", "ec2:DescribeVpcAttribute",`	`"*"`	允許執行任務、爬蟲程式和開發端點時設定 Amazon EC2 網路項目，例如 Virtual Private Cloud (VPC)。
`"iam:ListRolePolicies", "iam:GetRole", "iam:GetRolePolicy"`	`"*"`	允許列出爬蟲程式、工作、開發端點和筆記本伺服器的 IAM 角色。
`"cloudwatch:PutMetricData"`	`"*"`	允許寫入任務的 CloudWatch 指標。
`"s3:CreateBucket", "s3:PutBucketPublicAccessBlock"`	`"arn:aws:s3:::aws-glue-*"`	允許從工作和筆記本伺服器在您的帳戶中建立 Amazon S3 儲存貯體。 命名慣例：使用名為 aws-glue- 的 Amazon S3 資料夾。 讓 AWS Glue 建立封鎖公開存取的儲存貯體。

Action	Resource	Description (描述)
"s3:GetObject", "s3:PutObject", "s3:DeleteObject"	"arn:aws:s3:::aws-glue-*/*", "arn:aws:s3:::*/*aws-glue-*/*"	允許存放 ETL 指令碼和筆記本伺服器位置等物件時在您的帳戶中取得、放入和刪除 Amazon S3 物件。 命名慣例：授予許可至名稱字首為 aws-glue- 的 Amazon S3 儲存貯體或資料夾。
"s3:GetObject"	"arn:aws:s3:::crawler-public*", "arn:aws:s3:::aws-glue-*"	允許從爬蟲程式和任務取得範例和教學所用的 Amazon S3 物件。 命名慣例：名稱開頭為 crawler-public 和 aws-glue- 的 Amazon S3 儲存貯體。
"logs:CreateLogGroup", "logs:CreateLogStream", "logs:PutLogEvents"	"arn:aws:logs:*:*:log-group:/aws-glue/*"	允許將日誌寫入至 CloudWatch Logs。 命名慣例：AWS Glue 寫入日誌到名稱開頭為 aws-glue 的日誌群組。
"ec2:CreateTags", "ec2:DeleteTags"	"arn:aws:ec2:*:*:network-interface/*", "arn:aws:ec2:*:*:security-group/*", "arn:aws:ec2:*:*:instance/*"	允許標記專為開發端點建立的 Amazon EC2 資源。 命名慣例：AWS Glue 會在 Amazon EC2 網路介面、安全群組和執行個體加上 aws-glue-service-resource 標籤。

5. 在 Review Policy (檢閱政策) 畫面上，輸入 Policy Name (政策名稱)，例如 GlueServiceRolePolicy。輸入選用的說明，當您對政策滿意時，即可選擇 Create policy (建立政策)。

步驟 2：為 AWS Glue 建立 IAM 角色

您需要授予 AWS Glue 在代為呼叫其他服務時可擔任之 IAM 角色的許可。這包括存取 Amazon S3 的任何來源、目標、指令碼和使用於 AWS Glue 的臨時目錄。爬蟲程式、工作和開發端點皆需要許可。

您可用 AWS Identity and Access Management (IAM) 來提供這些許可。新增政策至傳送到 AWS Glue 的 IAM 角色。

為 AWS Glue 建立 IAM 角色

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 選擇建立角色。
4. 在角色類型中選擇 AWS Service (AWS 服務)，找出並選擇 Glue，然後選擇 Next: Permissions (下一步：許可)。
5. 在 Attach permissions policy (連接許可政策) 頁面上，選擇包含必要許可的政策；例如，適用於一般 AWS Glue 許可的 AWS 受管政策 `AWSGlueServiceRole`，和用於存取 Amazon S3 資源的 AWS 受管政策 AmazonS3FullAccess。接著選擇 Next: Review (下一步：檢閱)。

   > **Note**
   >
   > 確定此角色其中一個政策授予 Amazon S3 來源和目標的許可。您可能想要提供自己的政策來存取特定的 Amazon S3 資源。資料來源需要 `s3:ListBucket` 和 `s3:GetObject` 許可。資料目標需要 `s3:ListBucket`、`s3:PutObject` 和 `s3:DeleteObject` 許可。如需為您的資源建立 Amazon S3 政策的詳細資訊，請參閱在政策中指定資源。如需範例 Amazon S3 政策，請參閱編寫 IAM 政策：如何授予 Amazon S3 儲存貯體的存取。
   > 如果您計劃存取使用 SSE-KMS 加密的 Amazon S3 來源和目標，請連接允許 AWS Glue 爬蟲程式、任務和開發端點解密資料的政策。如需詳細資訊，請參閱搭配使用伺服器端加密與 AWS KMS 受管金鑰 (SSE-KMS) 來保護資料。
   > 以下是範例。

   ```
   {
      "Version":"2012-10-17",
      "Statement":[
         {
            "Effect":"Allow",
            "Action":[
               "kms:Decrypt"
   ```

```
            ],
            "Resource":[
               "arn:aws:kms:*:account-id-without-hyphens:key/key-id"
            ]
         }
      ]
}
```

6. 在 Role name (角色名稱) 中輸入角色名稱，例如 AWSGlueServiceRoleDefault。建立名稱字首為 AWSGlueServiceRole 字串的角色，以允許角色從主控台使用者傳送到服務。AWS Glue 提供的政策預期 IAM 服務角色的開頭應為 AWSGlueServiceRole。否則，您必須新增政策，允許您的使用者取得 iam:PassRole 許可，使 IAM 角色符合您的命名慣例。選擇建立角色。

> **Note**
>
> 在您使用角色建立筆記本時，系統會將該角色傳遞至互動式工作階段，以便在兩個位置皆可使用同一個角色。因此，iam:PassRole 許可需要成為角色政策的一部分。
> 使用下列範例為角色建立新政策。用您的帳號與角色名稱取代範例中的值。
>
> ```
> {
>     "Version": "2012-10-17",
>     "Statement": [
>         {
>             "Effect": "Allow",
>             "Action": "iam:PassRole",
>             "Resource": "arn:aws:iam::090000000210:role/<role_name>"
>         }
>     ]
> }
> ```

步驟 3：連接政策到存取 AWS Glue 的使用者或群組

管理員必須使用 AWS Glue 主控台或 AWS Command Line Interface (AWS CLI)，將許可指派給任何使用者、群組或角色。您可透過政策用AWS Identity and Access Management (IAM) 提供這些許可。此步驟說明如何將許可指派給使用者或群組。

完成此步驟時，您的使用者或群組將連接以下政策：

- AWS 受管政策 AWSGlueConsoleFullAccess 或自訂政策 GlueConsoleAccessPolicy
- **AWSGlueConsoleSageMakerNotebookFullAccess**
- **CloudWatchLogsReadOnlyAccess**
- **AWSCloudFormationReadOnlyAccess**
- **AmazonAthenaFullAccess**

連接內嵌政策並內嵌於使用者或群組

您可以將 AWS 受管政策或內嵌政策連接到使用者或群組，以允許其存取 AWS Glue 主控台。此政策指定的部分資源參照 AWS Glue 用於 Amazon S3 儲存貯體、Amazon S3 ETL 指令碼、CloudWatch Logs、AWS CloudFormation 和 Amazon EC2 資源的預設名稱。為了簡化，AWS Glue 寫入部分 Amazon S3 物件到您預設字首為 aws-glue-* 之帳戶的儲存貯體內。

> **Note**
>
> 如果您使用 AWS 受管政策 **AWSGlueConsoleFullAccess**，可跳過此步驟。

> **Important**
>
> AWS Glue 需要獲得許可才能擔任代為執行任務的角色。為了達成這個目標，您新增 **iam:PassRole** 許可至 AWS Glue 使用者或群組。此政策授予許可給 AWS Glue 服務角色開頭為 AWSGlueServiceRole 的角色，和建立筆記本伺服器時需要之角色的 AWSGlueServiceNotebookRole。您也可以依照您的命名慣例建立自己的 iam:PassRole 許可政策。
>
> 根據安全性最佳實務，建議透過收緊政策來限制存取，進一步限制對 Amazon S3 儲存貯體和 Amazon CloudWatch 日誌群組的存取。如需範例 Amazon S3 政策，請參閱編寫 IAM 政策：如何授予 Amazon S3 儲存貯體的存取。

在本步驟中，您將建立一個類似 AWSGlueConsoleFullAccess 的政策。最新版本的 AWSGlueConsoleFullAccess 可在 IAM 主控台找到。

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在導覽窗格中，選擇群組或使用者。

3. 在清單中，選擇要內嵌政策的使用者或群組名稱。
4. 選擇 Permissions (許可) 索引標籤，並在必要時，展開 Permissions policies (許可政策) 部分。
5. 選擇 Add Inline policy (新增內嵌政策) 連結。
6. 在 Create Policy (建立政策) 畫面上導覽至索引標籤，以編輯 JSON。使用下列 JSON 陳述式建立政策文件，然後選擇 Review policy (檢閱政策)。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:*",
                "redshift:DescribeClusters",
                "redshift:DescribeClusterSubnetGroups",
                "iam:ListRoles",
                "iam:ListUsers",
                "iam:ListGroups",
                "iam:ListRolePolicies",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListAttachedRolePolicies",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeRouteTables",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeInstances",
                "rds:DescribeDBInstances",
                "rds:DescribeDBClusters",
                "rds:DescribeDBSubnetGroups",
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "cloudformation:DescribeStacks",
                "cloudformation:GetTemplateSummary",
                "dynamodb:ListTables",
                "kms:ListAliases",
                "kms:DescribeKey",
```

```
                "cloudwatch:GetMetricData",
                "cloudwatch:ListDashboards"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::*/*aws-glue-*/*",
                "arn:aws:s3:::aws-glue-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "tag:GetResources"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutBucketPublicAccessBlock"           ],
            "Resource": [
                "arn:aws:s3:::aws-glue-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:GetLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:/aws-glue/*"
            ]
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack"
            ],
            "Resource": "arn:aws:cloudformation:*:*:stack/aws-glue*/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*",
                "arn:aws:ec2:*:*:key-pair/*",
                "arn:aws:ec2:*:*:image/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*:*:subnet/*",
                "arn:aws:ec2:*:*:volume/*"
            ]
        },
        {
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iam::*:role/AWSGlueServiceRole*",
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "glue.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iam::*:role/AWSGlueServiceNotebookRole*",
```

```
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "ec2.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iam::*:role/service-role/AWSGlueServiceRole*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "glue.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

下表說明此政策授予的許可。

Action	Resource	Description (描述)
"glue:*"	"*"	授予執行所有 AWS Glue API 操作的許可。 如果您之前已建立不含 "glue:*" 動作的政策，則您必須將下列個別許可指派至您的政策： • "glue:ListCrawlers" • "glue:BatchGetCrawlers" • "glue:ListTriggers" • "glue:BatchGetTriggers" • "glue:ListDevEndpoints" • "glue:BatchGetDevEndpoints" • "glue:ListJobs" • "glue:BatchGetJobs"
"redshift:DescribeClusters", "redshift:DescribeClusterSubnetGroups"	"*"	允許建立 Amazon Redshift 的連線。
"iam:ListRoles", "iam:ListRolePolicies", "iam:GetRole", "iam:GetRolePolicy", "iam:ListAttachedRolePolicies"	"*"	允許使用爬蟲程式、工作、開發端點和筆記本伺服器時列出 IAM 角色。

Action	Resource	Description (描述)
"ec2:DescribeSecurityGroups", "ec2:DescribeSubnets", "ec2:DescribeVpcs", "ec2:DescribeVpcEndpoints", "ec2:DescribeRouteTables", "ec2:DescribeVpcAttribute", "ec2:DescribeKeyPairs", "ec2:DescribeInstances"	"*"	允許執行任務、爬蟲程式和開發端點時設定 Amazon EC2 網路項目，例如 VPC。
"rds:DescribeDBInstances"	"*"	允許建立 Amazon RDS 的連線。
"s3:ListAllMyBuckets", "s3:ListBucket", "s3:GetBucketAcl", "s3:GetBucketLocation"	"*"	允許使用爬蟲程式、任務、開發端點和筆記本伺服器時列出 Amazon S3 儲存貯體。
"dynamodb:ListTables"	"*"	允許列出 DynamoDB 資料表。
"kms:ListAliases", "kms:DescribeKey"	"*"	允許使用 KMS 金鑰。
"cloudwatch:GetMetricData", "cloudwatch:ListDashboards"	"*"	允許使用 CloudWatch 指標。

Action	Resource	Description (描述)
"s3:GetObject", "s3:PutObject"	"arn:aws:s3::: aws-glue-*/*", "arn:aws:s3::: */*aws-glue-*/*", "arn:aws:s3::: aws-glue-*"	允許存放 ETL 指令碼和筆記本伺服器位置等物件時在您的帳戶中取得和放入 Amazon S3 物件。 命名慣例：授予許可至名稱字首為 aws-glue- 的 Amazon S3 儲存貯體或資料夾。
"tag:GetResources"	"*"	允許擷取 AWS 標籤。
"s3:CreateBucket", "s3:PutBucketPublicAccessBlock"	"arn:aws:s3::: aws-glue-*"	允許存放 ETL 指令碼和筆記本伺服器位置等物件時在您的帳戶中建立 Amazon S3 儲存貯體。 命名慣例：授予許可至名稱字首為 aws-glue- 的 Amazon S3 儲存貯體。 讓 AWS Glue 建立封鎖公開存取的儲存貯體。
"logs:GetLogEvents"	"arn:aws:logs:*:*: /aws-glue/*"	允許擷取 CloudWatch Logs。 命名慣例：AWS Glue 寫入日誌到名稱開頭為 aws-glue- 的日誌群組。

Action	Resource	Description (描述)
"cloudformation:CreateStack", "cloudformation:DeleteStack"	"arn:aws:cloudformation:*:*:stack/aws-glue*/*"	允許使用筆記本伺服器時管理 AWS CloudFormation 堆疊。 命名慣例：AWS Glue 建立名稱開頭為 aws-glue 的堆疊。
"ec2:RunInstances"	"arn:aws:ec2:*:*:instance/*", "arn:aws:ec2:*:*:key-pair/*", "arn:aws:ec2:*:*:image/*", "arn:aws:ec2:*:*:security-group/*", "arn:aws:ec2:*:*:network-interface/*", "arn:aws:ec2:*:*:subnet/*", "arn:aws:ec2:*:*:volume/*"	允許開發端點和筆記本伺服器的執行。
"iam:PassRole"	"arn:aws:iam::*:role/ AWSGlueServiceRole*"	允許 AWS Glue 擔任名稱開頭為 AWSGlueServiceRole 之角色的 PassRole 許可。
"iam:PassRole"	"arn:aws:iam::*:role/ AWSGlueServiceNotebookRole*"	允許 Amazon EC2 擔任名稱開頭為 AWSGlueServiceNotebookRole 之角色的 PassRole 許可。

Action	Resource	Description (描述)
`"iam:PassRole"`	`"arn:aws:iam::*:role/service-role/AWSGlueServiceRole*"`	允許 AWS Glue 擔任名稱開頭為 `service-role/AWSGlueServiceRole` 之角色的 `PassRole` 許可。

7. 在 Review policy (檢閱政策) 頁面上，輸入政策名稱 (例如 GlueConsoleAccessPolicy)。當您滿意時，選擇 Create policy (建立政策)。確認畫面頂端的紅色方塊未出現任何錯誤。如出現任何錯誤，請加以修正。

> **Note**
>
> 如果選取 Use autoformatting (使用自動格式化)，每當您開啟政策或選擇 Validate Policy (驗證政策) 時，政策都會重新格式化。

附加 AWSGlueConsoleFullAccess 受管政策

您可連接 `AWSGlueConsoleFullAccess` 政策，以提供 AWS Glue 主控台使用者所需要的許可。

> **Note**
>
> 如果您已自行為 AWS Glue 主控台存取建立政策，可跳過此步驟。

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在導覽窗格中，選擇 政策 。
3. 在政策清單中，選取 AWSGlueConsoleFullAccess 旁的核取方塊。您可用 Filter (篩選) 功能表和搜尋方塊來篩選政策清單。
4. 選擇 Policy actions (政策動作)，再選擇 Attach (附加)。
5. 選擇要附加政策的使用者。您可用 Filter (篩選) 功能表和搜尋方塊來篩選主體實體清單。選擇要附加的使用者後，選擇 Attach policy (附加政策)。

連接 **AWSGlueConsoleSageMakerNotebookFullAccess** 受管政策

您可以將 AWSGlueConsoleSageMakerNotebookFullAccess 政策連接至使用者，以管理在 AWS Glue 主控台中建立的 SageMaker 筆記本。除了其他必要的 AWS Glue 主控台許可之外，此政策會授予管理 SageMaker 筆記本所需的資源存取權。

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在導覽窗格中，選擇 政策 。
3. 在政策清單中，選取 AWSGlueConsoleSageMakerNotebookFullAccess 旁的核取方塊。您可用 Filter (篩選) 功能表和搜尋方塊來篩選政策清單。
4. 選擇 Policy actions (政策動作)，再選擇 Attach (附加)。
5. 選擇要附加政策的使用者。您可用 Filter (篩選) 功能表和搜尋方塊來篩選主體實體清單。選擇要附加的使用者後，選擇 Attach policy (附加政策)。

附加 CloudWatchLogsReadOnlyAccess 受管政策

您可以將 CloudWatchLogsReadOnlyAccess 政策連接到使用者，以在 CloudWatch Logs 主控台上檢視 AWS Glue 建立的日誌。

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在導覽窗格中，選擇 政策 。
3. 在政策清單中，選取 CloudWatchLogsReadOnlyAccess 旁的核取方塊。您可用 Filter (篩選) 功能表和搜尋方塊來篩選政策清單。
4. 選擇 Policy actions (政策動作)，再選擇 Attach (附加)。
5. 選擇要附加政策的使用者。您可用 Filter (篩選) 功能表和搜尋方塊來篩選主體實體清單。選擇要附加的使用者後，選擇 Attach policy (附加政策)。

附加 AWSCloudFormationReadOnlyAccess 受管政策

您可以將 AWSCloudFormationReadOnlyAccess 政策附加到使用者，在 AWS CloudFormation 主控台上檢視 AWS Glue 所用的 AWS CloudFormation 堆疊。

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。

2. 在導覽窗格中，選擇 政策 。
3. 在政策清單中，選取 AWSCloudFormationReadOnlyAccess 旁的核取方塊。您可用 Filter (篩選) 功能表和搜尋方塊來篩選政策清單。
4. 選擇 Policy actions (政策動作)，再選擇 Attach (附加)。
5. 選擇要附加政策的使用者。您可用 Filter (篩選) 功能表和搜尋方塊來篩選主體實體清單。選擇要附加的使用者後，選擇 Attach policy (附加政策)。

連接 AmazonAthenaFullAccess 受管政策

您可以將 AmazonAthenaFullAccess 政策連接到使用者，以在 Athena 主控台中檢視 Amazon S3 資料。

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在導覽窗格中，選擇 政策 。
3. 在政策清單中，選取 AmazonAthenaFullAccess 旁的核取方塊。您可用 Filter (篩選) 功能表和搜尋方塊來篩選政策清單。
4. 選擇 Policy actions (政策動作)，再選擇 Attach (附加)。
5. 選擇要附加政策的使用者。您可用 Filter (篩選) 功能表和搜尋方塊來篩選主體實體清單。選擇要附加的使用者後，選擇 Attach policy (附加政策)。

步驟 4：建立適用於筆記本伺服器的 IAM 政策

如果您計劃使用筆記本搭配開發端點，您必須指定建立筆記本伺服器時的許可。您可用 AWS Identity and Access Management (IAM) 來提供這些許可。

此政策授予部分 Amazon S3 動作的許可，在 AWS Glue 擔任使用此政策的角色時可用於管理其所需的帳戶內資源。此政策指定的部分資源參照 AWS Glue 用於 Amazon S3 儲存貯體、Amazon S3 ETL 指令碼和 Amazon EC2 資源的預設名稱。為了簡化，AWS Glue 預設寫入部分 Amazon S3 物件到您預設字首為 `aws-glue-*` 之帳戶的儲存貯體內。

> **Note**
>
> 如果您使用 AWS 受管政策 **`AWSGlueServiceNotebookRole`**，可跳過此步驟。

在本步驟中，您將建立一個類似 AWSGlueServiceNotebookRole 的政策。最新版本的 AWSGlueServiceNotebookRole 可在 IAM 主控台找到。

建立適用於筆記本的 IAM 政策

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在左側導覽窗格中選擇 Policies (政策)。
3. 選擇 建立政策 。
4. 在 Create Policy (建立政策) 畫面上導覽至索引標籤，以編輯 JSON。使用下列 JSON 陳述式建立政策文件，然後選擇 Review policy (檢閱政策)。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "glue:CreateDatabase",
            "glue:CreatePartition",
            "glue:CreateTable",
            "glue:DeleteDatabase",
            "glue:DeletePartition",
            "glue:DeleteTable",
            "glue:GetDatabase",
            "glue:GetDatabases",
            "glue:GetPartition",
            "glue:GetPartitions",
            "glue:GetTable",
            "glue:GetTableVersions",
            "glue:GetTables",
            "glue:UpdateDatabase",
            "glue:UpdatePartition",
            "glue:UpdateTable",
            "glue:GetJobBookmark",
            "glue:ResetJobBookmark",
            "glue:CreateConnection",
            "glue:CreateJob",
            "glue:DeleteConnection",
            "glue:DeleteJob",
            "glue:GetConnection",
            "glue:GetConnections",
```

```
            "glue:GetDevEndpoint",
            "glue:GetDevEndpoints",
            "glue:GetJob",
            "glue:GetJobs",
            "glue:UpdateJob",
            "glue:BatchDeleteConnection",
            "glue:UpdateConnection",
            "glue:GetUserDefinedFunction",
            "glue:UpdateUserDefinedFunction",
            "glue:GetUserDefinedFunctions",
            "glue:DeleteUserDefinedFunction",
            "glue:CreateUserDefinedFunction",
            "glue:BatchGetPartition",
            "glue:BatchDeletePartition",
            "glue:BatchCreatePartition",
            "glue:BatchDeleteTable",
            "glue:UpdateDevEndpoint",
            "s3:GetBucketLocation",
            "s3:ListBucket",
            "s3:ListAllMyBuckets",
            "s3:GetBucketAcl"
         ],
         "Resource":[
            "*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject"
         ],
         "Resource":[
            "arn:aws:s3:::crawler-public*",
            "arn:aws:s3:::aws-glue*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:PutObject",
            "s3:DeleteObject"
         ],
         "Resource":[
            "arn:aws:s3:::aws-glue*"
```

```
            ]
         },
         {
            "Effect":"Allow",
            "Action":[
               "ec2:CreateTags",
               "ec2:DeleteTags"
            ],
            "Condition":{
               "ForAllValues:StringEquals":{
                  "aws:TagKeys":[
                     "aws-glue-service-resource"
                  ]
               }
            },
            "Resource":[
               "arn:aws:ec2:*:*:network-interface/*",
               "arn:aws:ec2:*:*:security-group/*",
               "arn:aws:ec2:*:*:instance/*"
            ]
         }
      ]
}
```

下表說明此政策授予的許可。

Action	Resource	Description (描述)
`"glue:*"`	`"*"`	授予執行所有 AWS Glue API 操作的許可。
`"s3:GetBucketLocation", "s3:ListBucket", "s3:ListAllMyBuckets", "s3:GetBucketAcl"`	`"*"`	允許從筆記本伺服器列出 Amazon S3 儲存貯體。

Action	Resource	Description (描述)
`"s3:GetObject"`	`"arn:aws:s3:::crawler-public*", "arn:aws:s3:::aws-glue-*"`	允許從筆記本取得範例和教學所用的 Amazon S3 物件。 命名慣例：名稱開頭為 crawler-public 和 aws-glue- 的 Amazon S3 儲存貯體。
`"s3:PutObject", "s3:DeleteObject"`	`"arn:aws:s3:::aws-glue*"`	允許從筆記本在您的帳戶內放入和刪除 Amazon S3 物件。 命名慣例：使用名為 aws-glue- 的 Amazon S3 資料夾。
`"ec2:CreateTags", "ec2:DeleteTags"`	`"arn:aws:ec2:*:*:network-interface/*", "arn:aws:ec2:*:*:security-group/*", "arn:aws:ec2:*:*:instance/*"`	允許標記專為筆記本伺服器建立的 Amazon EC2 資源。 命名慣例：AWS Glue 會在 Amazon EC2 執行個體加上 aws-glue-service-resource 標籤。

5. 在 Review Policy (檢閱政策) 畫面上，輸入 Policy Name (政策名稱)，例如 GlueServiceNotebookPolicyDefault。輸入選用的說明，當您對政策滿意時，即可選擇 Create policy (建立政策)。

步驟 5：建立適用於筆記本伺服器的 IAM 角色

如果您計劃使用筆記本搭配開發端點，您需要授予 IAM 角色許可。您可透過 IAM 角色使用 AWS Identity and Access Management IAM 來提供這些許可。

> **Note**
>
> 使用 IAM 主控台建立 IAM 角色時，主控台會自動建立執行個體描述檔，並將其命名為與對應角色相同的名稱。

建立適用於筆記本的 IAM 角色

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 選擇 建立角色。
4. 在角色類型中選擇 AWS Service (AWS 服務)，找出並選擇 EC2，接著選擇 EC2 使用案例，然後選擇 Next: Permissions (下一步：許可)。
5. 在 Attach permissions policy (連接許可政策) 頁面上，選擇包含必要許可的政策；例如，適用於一般 AWS Glue 許可的 AWSGlueServiceNotebookRole，和用於存取 Amazon S3 資源的 AWS 受管政策 AmazonS3FullAccess。接著選擇 Next: Review (下一步：檢閱)。

> **Note**
>
> 確定此角色其中一個政策授予 Amazon S3 來源和目標的許可。同時確認您的政策允許在建立筆記本伺服器時完整存取筆記本存放位置。您可能想要提供自己的政策來存取特定的 Amazon S3 資源。如需為您的資源建立 Amazon S3 政策的詳細資訊，請參閱在政策中指定資源。
> 如果您計劃存取使用 SSE-KMS 加密的 Amazon S3 來源和目標，請連接允許筆記本解密資料的政策。如需詳細資訊，請參閱搭配使用伺服器端加密與 AWS KMS 受管金鑰 (SSE-KMS) 來保護資料。
> 以下是範例。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "kms:Decrypt"
         ],
         "Resource":[
```

```
            "arn:aws:kms:*:account-id-without-hyphens:key/key-id"
        ]
    }
  ]
}
```

6. 針對 Role name (角色名稱)，輸入您的角色名稱。建立名稱字首為 `AWSGlueServiceNotebookRole` 字串的角色，以允許角色從主控台使用者傳送到筆記本伺服器。AWS Glue 提供的政策預期 IAM 服務角色的開頭應為 `AWSGlueServiceNotebookRole`。否則，您必須為使用者新增政策，允許其取得 `iam:PassRole` 許可，使 IAM 角色符合您的命名慣例。例如，輸入 `AWSGlueServiceNotebookRoleDefault`。然後選擇 Create role (建立角色)。

步驟 6：建立適用於 SageMaker 筆記本的 IAM 政策

如果您計劃使用 SageMaker 筆記本搭配開發端點，您必須在建立筆記本時指定許可。您可用 AWS Identity and Access Management (IAM) 來提供這些許可。

建立適用於 SageMaker 筆記本的 IAM 政策

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在左側導覽窗格中選擇 Policies (政策)。
3. 選擇 建立政策 。
4. 在 Create Policy (建立政策) 頁面上，導覽至索引標籤，以編輯 JSON。使用以下 JSON 陳述式建立政策文件。編輯您環境的 *bucket-name*、*region-code* 和 *account-id*。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:ListBucket"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::bucket-name"
            ]
        },
```

```
        {
            "Action": [
                "s3:GetObject"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::bucket-name*"
            ]
        },
        {
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:logs:region-code:account-id:log-group:/aws/sagemaker/*",
                "arn:aws:logs:region-code:account-id:log-group:/aws/sagemaker/
*:log-stream:aws-glue-*"
            ]
        },
        {
            "Action": [
                "glue:UpdateDevEndpoint",
                "glue:GetDevEndpoint",
                "glue:GetDevEndpoints"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:glue:region-code:account-id:devEndpoint/*"
            ]
        },
        {
            "Action": [
                "sagemaker:ListTags"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:sagemaker:region-code:account-id:notebook-instance/*"
            ]
         }
    ]
```

```
}
```

接著選擇 Review policy (檢閱政策)。

下表說明此政策授予的許可。

Action	Resource	Description (描述)
"s3:ListBucket*"	"arn:aws:s3::: *bucket-name* "	列出 Amazon S3 儲存貯體的許可。
"s3:GetObject"	"arn:aws:s3::: *bucket-name* *"	授予取得 SageMaker 筆記本所使用之 Amazon S3 物件的許可。
"logs:CreateLogStream", "logs:DescribeLogStreams", "logs:PutLogEvents", "logs:CreateLogGroup"	"arn:aws:logs: *region-code* :*account-id* :log-group:/aws/sagemaker/*", "arn:aws:logs: *region-code* :*account-id* :log-group:/aws/sagemaker/*:log-stream:aws-glue-*"	授予將日誌從筆記本寫入 Amazon CloudWatch Logs 的許可。 命名慣例：寫入名稱開頭為 aws-glue 的日誌群組。
"glue:UpdateDevEndpoint", "glue:GetDevEndpoint", "glue:GetDevEndpoints"	"arn:aws:glue: *region-code* :*account-id* :devEndpoint/*"	授予從 SageMaker 筆記本使用開發端點的許可。
"sagemaker:ListTags"	"arn:aws:sagemaker : *region-code* :*account-id* :notebook-instance /*"	授予傳回 SageMaker 資源之標籤的許可。SageMaker 筆記本需使用 aws-glue-dev-endpoint 標籤，才能將筆記本連線到開發端點。

5. 在 Review Policy (檢閱政策) 畫面上，輸入 Policy Name (政策名稱)，例如 `AWSGlueSageMakerNotebook`。輸入選用的說明，當您對政策滿意時，即可選擇 Create policy (建立政策)。

步驟 7：建立適用於 SageMaker 筆記本的 IAM 角色

如果您計劃使用 SageMaker 筆記本搭配開發端點，您需要授予 IAM 角色許可。您可透過 IAM 角色使用 AWS Identity and Access Management (IAM) 來提供這些許可。

建立適用於 SageMaker 筆記本的 IAM 角色

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 選擇 建立角色。
4. 針對角色類型，選擇 AWS Service (AWS 服務)，尋找並選擇 SageMaker，接著選擇 SageMaker - Execution (SageMaker - 執行) 使用案例。然後選擇 Next: Permissions (下一步：許可)。
5. 在 Attach permissions policy (連接許可政策) 頁面上，選擇包含所需許可的政策；例如 AmazonSageMakerFullAccess。選擇 下一步：檢閱 。

   如果您計劃存取使用 SSE-KMS 加密的 Amazon S3 來源和目標，請連接允許筆記本解密資料的政策，如以下範例所示。如需詳細資訊，請參閱搭配使用伺服器端加密與 AWS KMS 受管金鑰 (SSE-KMS) 來保護資料。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "kms:Decrypt"
         ],
         "Resource":[
            "arn:aws:kms:*:account-id-without-hyphens:key/key-id"
         ]
      }
   ]
}
```

6. 針對 Role name (角色名稱)，輸入您的角色名稱。若要允許從主控台使用者將角色傳送到 SageMaker，請使用字首為 `AWSGlueServiceSageMakerNotebookRole` 字串的名稱。AWS Glue 提供的政策預期 IAM 角色的開頭應為 `AWSGlueServiceSageMakerNotebookRole`。否則，您必須為使用者新增政策，允許其取得 `iam:PassRole` 許可，使 IAM 角色符合您的命名慣例。

   例如，輸入 `AWSGlueServiceSageMakerNotebookRole-Default`，然後選擇 Create role (建立角色)。

7. 在您建立角色之後，即可連接政策以允許從 AWS Glue 建立 SageMaker 筆記本所需的額外許可。

   開啟您剛才建立的角色 `AWSGlueServiceSageMakerNotebookRole-Default`，然後選擇 Attach policies (連接政策)。將您建立的名為 `AWSGlueSageMakerNotebook` 的政策連接至角色。

AWS Glue 存取控制政策範例

本節包含身分識別型 (IAM) 存取控制政策和 AWS Glue 資源政策的範例。

內容

AWS Glue 的身分型政策範例

根據預設，使用者和角色不具備建立或修改 AWS Glue 資源的權限。他們也無法使用 AWS Management Console、AWS Command Line Interface (AWS CLI) 或 AWS API 執行任務。若要授與使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

若要了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策。

如需 AWS Glue 所定義之動作和資源類型的詳細資訊，包括每種資源類型的 ARN 格式，請參閱《服務授權參考》中的 AWS Glue 適用的動作、資源和條件索引鍵。

> **Note**
>
> 本節提供的範例全部使用 `us-west-2` 區域。您可以將您想要使用的 AWS 區域取代此項目。

主題

- 搭配使用標籤與清單和批次 API 操作
- 使用條件索引鍵或內容索引鍵來控制設定
- 拒絕給予身分建立資料預覽工作階段的能力

政策最佳實務

身分型政策會判斷您帳戶中的某個人員是否可以建立、存取或刪除 AWS Glue 資源。這些動作可能會讓您的 AWS 帳戶 產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並朝向最低權限許可的目標邁進：如需開始授予許可給使用者和工作負載，請使用 AWS 受管政策，這些政策會授予許可給許多常用案例。它們可在您的 AWS 帳戶 中使用。我們建議您定義特定於使用案例的 AWS 客戶管理政策，以便進一步減少許可。如需更多資訊，請參閱 IAM 使用者指南中的 AWS 受管政策或任務職能的 AWS 受管政策。
- 套用最低許可許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的權限。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱 IAM 使用者指南中的 IAM 中的政策和許可。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。您也可以使用條件來授予對服務動作的存取權，前提是透過特定 AWS 服務（例如 AWS CloudFormation）使用條件。如需更多資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您編寫安全且實用的政策。如需更多資訊，請參閱 IAM 使用者指南中的 IAM Access Analyzer 政策驗證。
- 需要多重要素驗證 (MFA)：如果存在需要 AWS 帳戶中 IAM 使用者或根使用者的情況，請開啟 MFA 提供額外的安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需更多資訊，請參閱 IAM 使用者指南中的設定 MFA 保護的 API 存取。

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

資源層級許可僅適用特定的 AWS Glue 物件

您只能在 AWS Glue 中定義特定物件的精細控制。因此，您必須編寫您用戶端的 IAM 政策，讓允許 `Resource` 陳述式 Amazon Resource Name (ARN) 的 API 操作，不會與不允許 ARN 的 API 操作混合。

例如，以下 IAM 政策允許 `GetClassifier` 和 `GetJobRun` 的 API 操作。它將 `Resource` 定義為 `*`，因為 AWS Glue 不允許分類器和任務回合的 ARN。因為允許特定 API 操作的 ARN (例如 `GetDatabase` 和 `GetTable`)，所以可以在政策的下半部指定 ARN。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetClassifier*",
                "glue:GetJobRun*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:Get*"
            ],
            "Resource": [
                "arn:aws:glue:us-east-1:123456789012:catalog",
                "arn:aws:glue:us-east-1:123456789012:database/default",
                "arn:aws:glue:us-east-1:123456789012:table/default/e*1*",
                "arn:aws:glue:us-east-1:123456789012:connection/connection2"
            ]
        }
    ]
}
```

如需允許 ARN 的 AWS Glue 物件清單，請參閱資源 ARN

使用 AWS Glue 主控台

若要存取 AWS Glue 主控台，您必須擁有最低的一組許可。這些許可必須允許您列出和檢視您 AWS 帳戶 中 AWS Glue 資源的詳細資訊。如果您建立比最基本必要許可更嚴格的身分型政策，則對於具有該政策的實體 (使用者或角色) 而言，主控台就無法如預期運作。

對於僅呼叫 AWS CLI 或 AWS API 的使用者，您不需要允許其最基本主控台許可。反之，只需允許存取符合他們嘗試執行之 API 操作的動作就可以了。

為確保使用者和角色仍可使用 AWS Glue 主控台，還請將 AWS Glue *`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》中的新增許可到使用者。

使用者若要使用 AWS Glue 主控台，必須擁有一組最基本的許可，以使用其 AWS 帳戶的 AWS Glue 資源。除了這些 AWS Glue 許可之外，主控台還需要以下服務的許可：

- 顯示日誌的 Amazon CloudWatch Logs 許可。
- 列出並傳遞角色的 AWS Identity and Access Management (IAM) 許可。
- 使用堆疊的 AWS CloudFormation 許可。
- 列出 VPC、子網路、安全群組、執行個體和其他物件的 Amazon Elastic Compute Cloud (Amazon EC2) 許可。
- 列出儲存貯體和物件，以及擷取和儲存指令碼的 Amazon Simple Storage Service (Amazon S3) 許可。
- 使用叢集的 Amazon Redshift 許可
- 列出執行個體的 Amazon Relational Database Service (Amazon RDS) 許可。

如需使用者檢視和使用 AWS Glue 主控台所需許可的詳細資訊，請參閱步驟 3：連接政策到存取 AWS Glue 的使用者或群組。

如果您建立比最基本必要許可更嚴格的 IAM 政策，則對於採取該 IAM 政策的使用者而言， 主控台就無法如預期運作。為確保這些使用者仍可使用 AWS Glue 主控台，也請連接 AWS 的管理 (預先定義) 策略 AWS Glue 中所述的 `AWSGlueConsoleFullAccess` 受管政策。

允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視連接到他們使用者身分的內嵌及受管政策。此政策包含在主控台上，或是使用 AWS CLI 或 AWS API 透過編寫程式的方式完成此動作的許可。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
```

```
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

授予資料表的唯讀許可

下列政策可授予 db1 資料庫中 books 資料表的唯讀許可。如需資源 Amazon Resource Name (ARN) 的詳細資訊，請參閱Data Catalog ARN。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GetTablesActionOnBooks",
      "Effect": "Allow",
      "Action": [
        "glue:GetTables",
        "glue:GetTable"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:123456789012:catalog",
        "arn:aws:glue:us-west-2:123456789012:database/db1",
        "arn:aws:glue:us-west-2:123456789012:table/db1/books"
      ]
    }
  ]
}
```

此政策會授予名為 db1 之資料庫中名為 books 之資料表的唯讀許可。若要將 Get 許可授予資料表，則也需要目錄和資料庫資源的該許可。

下列政策會授予在 db1 資料庫中建立 tb1 資料表的最低必要許可：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:CreateTable"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:123456789012:table/db1/tbl1",
        "arn:aws:glue:us-west-2:123456789012:database/db1",
        "arn:aws:glue:us-west-2:123456789012:catalog"
      ]
    }
  ]
}
```

依據 GetTables 許可篩選資料表

假設有三個資料表 customers、stores 以及 store_sales 在 db1 資料庫中。下列政策會授予 GetTables 許可給 stores 和 store_sales，但不會授予給 customers。當您使用此政策呼叫 GetTables 時，結果只會包含兩個授權的資料表 (不會傳回 customers 資料表)。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesExample",
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/db1",
                "arn:aws:glue:us-west-2:123456789012:table/db1/store_sales",
                "arn:aws:glue:us-west-2:123456789012:table/db1/stores"
```

```
            ]
        }
    ]
}
```

您可以使用 store* 來比對任何開頭為 store 的資料表名稱，藉此簡化上述政策。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesExample2",
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/db1",
                "arn:aws:glue:us-west-2:123456789012:table/db1/store*"
            ]
        }
    ]
}
```

同樣地，使用 /db1/* 比對 db1 中所有資料表時，下列政策可將 GetTables 存取授予 db1 中所有資料表。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesReturnAll",
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/db1",
                "arn:aws:glue:us-west-2:123456789012:table/db1/*"
            ]
        }
```

```
    ]
}
```

如未提供任何資料表 ARN，則可成功呼叫 GetTables，但會傳回空白清單。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesEmptyResults",
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/db1"
            ]
        }
    ]
}
```

如果政策中缺少資料庫 ARN，則呼叫 GetTables 會失敗，並顯示 AccessDeniedException。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesAccessDeny",
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:table/db1/*"
            ]
        }
    ]
}
```

授予完整存取資料表和所有分割區的許可

下列政策可授予 db1 資料庫中名稱為 books 之資料表的所有許可。這包括資料表本身、其存檔版本和其所有分割區的讀取和寫入許可。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "FullAccessOnTable",
            "Effect": "Allow",
            "Action": [
                "glue:CreateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:UpdateTable",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:GetTableVersion",
                "glue:GetTableVersions",
                "glue:DeleteTableVersion",
                "glue:BatchDeleteTableVersion",
                "glue:CreatePartition",
                "glue:BatchCreatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:UpdatePartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition"
            ],
            "Resource": [
                        "arn:aws:glue:us-west-2:123456789012:catalog",
                        "arn:aws:glue:us-west-2:123456789012:database/db1",
                        "arn:aws:glue:us-west-2:123456789012:table/db1/books"
                        ]
        }
    ]
}
```

上述政策在實務中可予以簡化。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "FullAccessOnTable",
            "Effect": "Allow",
            "Action": [
                "glue:*Table*",
                "glue:*Partition*"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/db1",
                "arn:aws:glue:us-west-2:123456789012:table/db1/books"
            ]
        }
    ]
}
```

請注意，精細定義存取控制的最低精細程度是資料表層級。這表示您無法授予使用者存取資料表中某些分割區而非其他分割區，或是存取某些資料表欄位而非其他欄位。使用者可以存取整個資料表，不然就是完全不能存取資料表。

透過名稱字首和明確拒絕的存取控制

在此範例中，假設您 AWS Glue Data Catalog 中的資料庫和資料表是使用名稱字首來整理。開發階段中的資料庫名稱字首為 dev-，而生產階段中的資料庫名稱字首為 prod-。您可以使用下列政策授予開發人員完整存取含 dev- 字首的所有資料庫、資料表、UDF 等。但是，若是含 prod- 字首的所有項目，您只授予唯讀存取。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DevAndProdFullAccess",
            "Effect": "Allow",
            "Action": [
                "glue:*Database*",
                "glue:*Table*",
                "glue:*Partition*",
                "glue:*UserDefinedFunction*",
                "glue:*Connection*"
            ],
```

```
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/dev-*",
                "arn:aws:glue:us-west-2:123456789012:database/prod-*",
                "arn:aws:glue:us-west-2:123456789012:table/dev-*/*",
                "arn:aws:glue:us-west-2:123456789012:table/*/dev-*",
                "arn:aws:glue:us-west-2:123456789012:table/prod-*/*",
                "arn:aws:glue:us-west-2:123456789012:table/*/prod-*",
                "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/dev-*/*",
                "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/*/dev-*",
                "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/prod-*/*",
                "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/*/prod-*",
                "arn:aws:glue:us-west-2:123456789012:connection/dev-*",
                "arn:aws:glue:us-west-2:123456789012:connection/prod-*"
            ]
        },
        {
            "Sid": "ProdWriteDeny",
            "Effect": "Deny",
            "Action": [
                "glue:*Create*",
                "glue:*Update*",
                "glue:*Delete*"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:database/prod-*",
                "arn:aws:glue:us-west-2:123456789012:table/prod-*/*",
                "arn:aws:glue:us-west-2:123456789012:table/*/prod-*",
                "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/prod-*/*",
                "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/*/prod-*",
                "arn:aws:glue:us-west-2:123456789012:connection/prod-*"
            ]
        }
    ]
}
```

上述政策中的第二個陳述式會使用明確 deny。您可以使用明確 deny 來覆寫已授予委託人的任何 allow 許可。這可讓您鎖定關鍵資源的存取權，並防止其他政策意外授予這些資源的存取權。

在上述範例中，即使第一個陳述式授予 prod- 資源的完整存取，第二個陳述式仍明確撤銷其寫入存取，而只保留 prod- 資源的唯讀取存取。

使用標籤授權存取

例如，假設您想要限定只有您帳戶下名為 Tom 的特定使用者，才能存取觸發 t2。所有其他使用者，包括 Sam，都可以存取該觸發條件 t1。觸發條件 t1 和 t2 具備下列屬性。

```
aws glue get-triggers
{
    "Triggers": [
        {
            "State": "CREATED",
            "Type": "SCHEDULED",
            "Name": "t1",
            "Actions": [
                {
                    "JobName": "j1"
                }
            ],
            "Schedule": "cron(0 0/1 * * ? *)"
        },
        {
            "State": "CREATED",
            "Type": "SCHEDULED",
            "Name": "t2",
            "Actions": [
                {
                    "JobName": "j1"
                }
            ],
            "Schedule": "cron(0 0/1 * * ? *)"
        }
    ]
}
```

AWS Glue 管理員已將標籤值 Tom (aws:ResourceTag/Name": "Tom") 連接到觸發條件 t2。AWS Glue 管理員也對 Tom 實施搭配以標籤為基礎之條件陳述式的 IAM 政策。因此，Tom 能使用的 AWS Glue 操作，將僅限於操作標籤值 Tom 的資源。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
         "Effect": "Allow",
         "Action": "glue:*",
```

```
          "Resource": "*",
          "Condition": {
              "StringEquals": {
                 "aws:ResourceTag/Name": "Tom"
               }
           }
          }
    ]
}
```

當 Tom 嘗試存取觸發 t1 時，他會收到拒絕存取訊息。同時，他可以成功擷取觸發條件 t2。

```
aws glue get-trigger --name t1

An error occurred (AccessDeniedException) when calling the GetTrigger operation:
 User: Tom is not authorized to perform: glue:GetTrigger on resource: arn:aws:glue:us-
east-1:123456789012:trigger/t1

aws glue get-trigger --name t2
{
    "Trigger": {
        "State": "CREATED",
        "Type": "SCHEDULED",
        "Name": "t2",
        "Actions": [
            {
                "JobName": "j1"
            }
        ],
        "Schedule": "cron(0 0/1 * * ? *)"
    }
}
```

Tom 無法使用複數 GetTriggers API 操作來列出觸發，因為此作業不支援篩選標籤。

為讓 Tom 存取 GetTriggers，AWS Glue 管理員要建立一項政策，將這些許可分為兩部分。其中一區可讓 Tom 使用 GetTriggers API 操作，存取所有觸發條件。第二區則讓 Tom 存取已標記值 Tom 的 API 操作。透過此政策，便可允許 Tom 同時擁有觸發條件 t2 的 GetTriggers 和 GetTrigger 存取權。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "glue:GetTriggers",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "glue:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Name": "Tom"
                }
            }
        }
    ]
}
```

使用標籤拒絕存取

另一種資源政策方法就是明確拒絕資源存取。

> **Important**
>
> 明確拒絕政策不適用於 `GetTriggers` 等複數 API 操作。

在下列範例政策中，允許所有 AWS Glue 任務操作。但是，第二個 `Effect` 陳述式明確拒絕存取標記為 `Team` 索引鍵和 `Special` 值的任務。

當管理員將以下政策連接至身分時，身分能存取除標記為 `Team` 索引鍵和 `Special` 值以外的所有任務。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "glue:*",
      "Resource": "arn:aws:glue:us-east-1:123456789012:job/*"
    },
```

```
    {
      "Effect": "Deny",
      "Action": "glue:*",
      "Resource": "arn:aws:glue:us-east-1:123456789012:job/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Team": "Special"
        }
      }
    }
  ]
}
```

搭配使用標籤與清單和批次 API 操作

撰寫資源政策的第三個方法，就是使用 List API 操作來列出符合標籤值的資源清單，進而允許資源的存取。然後，使用對應的 Batch API 操作，允許存取特定資源的詳細資訊。使用此方法時，管理員並不需要允許存取複數的 GetCrawlers、GetDevEndpoints、GetJobs 或 GetTriggers API 操作。相反地，您可以搭配下列 API 操作，允許列出資源的操作：

- ListCrawlers
- ListDevEndpoints
- ListJobs
- ListTriggers

此外，您可以搭配下列 API 操作，允許取得個別資源詳細資訊的操作：

- BatchGetCrawlers
- BatchGetDevEndpoints
- BatchGetJobs
- BatchGetTriggers

身為管理員，您可以使用此方法執行下列操作：

1. 新增標籤到您的爬蟲程式、開發端點、任務和觸發條件。
2. 拒絕使用者存取 Get API 操作，例如 GetCrawlers、GetDevEndponts、GetJobs 和 GetTriggers。

3. 為了讓使用者能夠找出其具備存取權的標記資源，這時要允許使用者存取 List API 操作，例如 ListCrawlers、ListDevEndponts、ListJobs 和 ListTriggers。
4. 拒絕使用者存取 AWS Glue 標記 API，例如 TagResource 和 UntagResource。
5. 允許使用者搭配 BatchGet API 操作來存取資源詳細資訊，例如 BatchGetCrawlers、BatchGetDevEndponts、BatchGetJobs 和 BatchGetTriggers。

例如，呼叫 ListCrawlers 操作時，提供標籤值以比對使用者名稱。然後，會得到符合所提供標籤值的爬蟲程式清單結果。將名稱清單提供給 BatchGetCrawlers，取得每個具備指定標籤之爬蟲程式的詳細資訊。

例如，如果 Tom 應該只能擷取已標記 Tom 之觸發條件的詳細資訊，則管理員可以將標籤新增至 Tom 的觸發條件，向所有使用者拒絕 GetTriggers API 操作的存取，而允許所有使用者對於 ListTriggers 和 BatchGetTriggers 的存取。

下面是 AWS Glue 管理員授予 Tom 的資源政策。在政策的第一區中，GetTriggers 已拒絕用於 AWS Glue API 操作。在政策的第二區中，允許 ListTriggers 用於所有資源。但在第三區中，標記 Tom 的這些資源就會允許 BatchGetTriggers 存取權進行存取。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "glue:GetTriggers",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListTriggers"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:BatchGetTriggers"
            ],
```

```
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Name": "Tom"
                }
            }
        }
    ]
}
```

使用與前例相同的觸發時，Tom 可以存取觸發 t2，但不觸發 t1。下面範例示範當 Tom 嘗試搭配 BatchGetTriggers，存取 t1 和 t2 時的結果。

```
aws glue batch-get-triggers --trigger-names t2
{
    "Triggers": {
        "State": "CREATED",
        "Type": "SCHEDULED",
        "Name": "t2",
        "Actions": [
            {
                "JobName": "j2"
            }
        ],
        "Schedule": "cron(0 0/1 * * ? *)"
    }
}

aws glue batch-get-triggers  --trigger-names t1

An error occurred (AccessDeniedException) when calling the BatchGetTriggers operation:
 No access to any requested resource.
```

下面範例示範當 Tom 嘗試使用同的 BatchGetTriggers 呼叫，同時存取觸發條件 t2 和觸發條件 t3 (其實並不存在) 時的結果。請注意，因為 Tom 具備觸發條件 t2 的存取權，且只有當 t2 傳回時才具備此存取權限。雖然 Tom 獲允存取觸發條件 t3，但觸發條件 t3 並不存在，因此回應時傳回的 t3 會列在 "TriggersNotFound": [] 清單中。

```
aws glue batch-get-triggers --trigger-names t2 t3
{
```

```
    "Triggers": {
        "State": "CREATED",
        "Type": "SCHEDULED",
        "Name": "t2",
        "Actions": [
            {
                "JobName": "j2"
            }
        ],
        "TriggersNotFound": ["t3"],
        "Schedule": "cron(0 0/1 * * ? *)"
    }
}
```

使用條件索引鍵或內容索引鍵來控制設定

授予建立和更新任務的許可時，您可以使用條件索引鍵或內容索引鍵。以下章節討論索引鍵：

- 控制使用條件索引鍵來控制設定的政策
- 控制使用內容索引鍵來控制設定的政策

控制使用條件索引鍵來控制設定的政策

AWS Glue 提供三個 IAM 條件索引鍵：`glue:VpcIds`、`glue:SubnetIds` 和 `glue:SecurityGroupIds`。授予建立和更新任務的許可時，您可以在 IAM 政策中使用條件索引鍵。您可以使用此設定來確保未建立 (或更新) 任務或工作階段，以在所需的 VPC 環境之外執行。VPC 設定資訊不是直接從 `CreateJob` 請求輸入，而是從任務「連線」欄位中推斷，該欄位指向 AWS Glue 連線。

範例使用方式

建立名為 "traffic-monitored-connection" 的 AWS Glue 網路類型連線，其具有所需的 VpcId "vpc-id1234"、SubnetIds 和 SecurityGroupIds。

為 IAM 政策中的 `CreateJob` 和 `UpdateJob` 動作指定條件索引鍵。

```
{
  "Effect": "Allow",
  "Action": [
    "glue:CreateJob",
    "glue:UpdateJob"
```

```
  ],
  "Resource": [
    "*"
  ],
  "Condition": {
    "ForAnyValue:StringLike": {
      "glue:VpcIds": [
        "vpc-id1234"
      ]
    }
  }
}
```

您可以建立類似的 IAM 政策，以禁止建立 AWS Glue 任務，而不指定連線資訊。

限制 VPC 上的工作階段

若要強制建立的工作階段在指定的 VPC 內執行，您可以透過在 `glue:CreateSession` 動作上新增 Deny 效果來限制角色許可，條件是 glue:vpc-id 不等於 vpc-<123>。例如：

```
"Effect": "Deny",
"Action": [
    "glue:CreateSession"
 ],
"Condition": {
    "StringNotEquals" : {"glue:VpcIds" : ["vpc-123"]}
}
```

您也可以透過在 `glue:CreateSession` 動作上新增 Deny 效果來強制建立的工作階段在 VPC 內執行，條件是 `glue:vpc-id` 為 Null。例如：

```
{
    "Effect": "Deny",
    "Action": [
        "glue:CreateSession"
    ],
      "Condition": {
        "Null": {"glue:VpcIds": true}
    }
},
{
```

```
    "Effect": "Allow",
    "Action": [
        "glue:CreateSession"
    ],
    "Resource": ["*"]
}
```

控制使用內容索引鍵來控制設定的政策

AWS Glue 為每個角色工作階段提供一個內容索引鍵 (glue:CredentialIssuingService=glue.amazonaws.com)，AWS Glue 將此索引鍵提供給任務和開發人員端點。這可讓您為 AWS Glue 指令碼執行的動作實作安全控制。AWS Glue 為每個角色工作階段提供另一個內容索引鍵 (glue:RoleAssumedBy=glue.amazonaws.com)，其中 AWS Glue 代表客戶呼叫另一個 AWS 服務 (不是由任務/開發人員端點，而是直接由 AWS Glue 服務)。

範例使用方式

在 IAM 政策中指定條件性許可，並將其連接至 AWS Glue 任務使用的角色。根據角色工作階段是否用於 AWS Glue 任務執行時間環境，這可確保某些動作被允許/拒絕。

```
{
    "Effect": "Allow",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::confidential-bucket/*",
    "Condition": {
        "StringEquals": {
            "glue:CredentialIssuingService": "glue.amazonaws.com"
        }
    }
}
```

拒絕給予身分建立資料預覽工作階段的能力

本節包含的 IAM 政策範例可用於拒絕給予身分建立資料預覽工作階段的能力。將此政策連接至與資料預覽工作階段在執行期間所使用的角色不同的身分。

```
{
    "Sid": "DatapreviewDeny",
    "Effect": "Deny",
     "Action": [
           "glue:CreateSession"
```

```
    ],
    "Resource": [
        "arn:aws:glue:*:*:session/glue-studio-datapreview*"
     ]
}
```

AWS Glue 以資源型為基礎的範例

本節包含以資源為基礎的範例政策，包括授予跨帳戶存取權的政策。

範例使用 AWS Command Line Interface (AWS CLI) 與 AWS Glue 服務 API 作業互動。您可以在 AWS Glue 主控台上執行相同的操作或使用其中一種 AWS 開發套件。

> **Important**
>
> 變更 AWS Glue 資源政策時，您可能會不小心撤銷您帳戶中現有 AWS Glue 使用者的許可，並造成意外中斷。請僅在開發或測試帳戶中試用這些範例，並先確保您所做的變更不會破壞現有任務流程之後再變更。

主題

- 搭配 AWS Glue 使用以資源為基礎的政策之考量
- 使用資源政策來控制相同帳戶的存取

搭配 AWS Glue 使用以資源為基礎的政策之考量

> **Note**
>
> IAM 政策和 AWS Glue 資源政策都需要幾秒鐘的傳播時間。連接新政策之後到新政策在整個系統傳播之前，您可能會發現舊政策仍然有效。

您可以使用以 JSON 格式撰寫的政策文件來建立或修改資源政策。政策語法與以身分為基礎的 IAM 政策的政策語法相同 (請參閱 IAM JSON 政策參考)，但有下列例外狀況：

- 每個政策陳述式都需要 `"Principal"` 或 `"NotPrincipal"` 區塊。
- `"Principal"` 或 `"NotPrincipal"` 必須識別有效的現有主體。不允許萬用字元模式 (例如 `arn:aws:iam::*account-id*:user/*`)。

- 政策中的 "Resource" 區塊需要所有資源 ARN 符合下列規則表達式語法 (其中，第一個 %s 是 *region*，而第二個 %s 是 *account-id*)：

  ```
  *arn:aws:glue:%s:%s:(\*|[a-zA-Z\*]+\/?.*)
  ```

  例如，允許 arn:aws:glue:us-west-2:*account-id*:* 和 arn:aws:glue:us-west-2:*account-id*:database/default，但不允許 *。
- 與以身分為基礎的政策不同，AWS Glue 資源政策只能包含屬於政策連接目錄之資源的 Amazon Resource Name (ARN)。這類 ARN 的開頭一律為 arn:aws:glue:。
- 政策不能鎖定建立它的身分，使其無法執行進一步建立或修改。
- 資源政策 JSON 文件的大小不能超過 10 KB。

使用資源政策來控制相同帳戶的存取

在此範例中，帳戶 A 的管理使用者建立了一個資源政策，以授予帳戶 A 中 IAM 使用者 Alice 對目錄的完整存取權。Alice 未連接任何 IAM 政策。

若要執行此操作，管理員使用者要執行下列 AWS CLI 命令。

```
# Run as admin of Account A
$ aws glue put-resource-policy --profile administrator-name --region us-west-2 --
policy-in-json '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Principal": {
        "AWS": [
          "arn:aws:iam::account-A-id:user/Alice"
        ]
      },
      "Effect": "Allow",
      "Action": [
        "glue:*"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:account-A-id:*"
      ]
    }
  ]
}'
```

與其在 AWS CLI 命令中輸入 JSON 政策文件，您可以將政策文件儲存在檔案中，並在 AWS CLI 命令中以 `file://` 字首參考檔案路徑。以下範例示範如何執行上述作業。

```
$ echo '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Principal": {
        "AWS": [
          "arn:aws:iam::account-A-id:user/Alice"
        ]
      },
      "Effect": "Allow",
      "Action": [
        "glue:*"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:account-A-id:*"
      ]
    }
  ]
}' > /temp/policy.json

$ aws glue put-resource-policy --profile admin1 \
    --region us-west-2 --policy-in-json file:///temp/policy.json
```

傳播這項資源政策之後，Alice 即可存取帳戶 A 中的所有 AWS Glue 資源，如下所示。

```
# Run as user Alice
$ aws glue create-database --profile alice --region us-west-2 --database-input '{
    "Name": "new_database",
    "Description": "A new database created by Alice",
    "LocationUri": "s3://my-bucket"
}'

$ aws glue get-table --profile alice --region us-west-2 --database-name "default" --
table-name "tbl1"}
```

AWS Glue 服務會傳回下列項目，以回應 Alice 的 `get-table` 呼叫。

```
{
  "Table": {
```

```
    "Name": "tbl1",
    "PartitionKeys": [],
    "StorageDescriptor": {
        ......
    },
    ......
  }
}
```

AWSAWS Glue 的受管理原則

受 AWS 管理的策略是由建立和管理的獨立策略 AWS。 AWS 受管理的策略旨在為許多常見使用案例提供權限，以便您可以開始將權限指派給使用者、群組和角色。

請記住， AWS 受管理的政策可能不會為您的特定使用案例授與最低權限權限，因為這些權限可供所有 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。

您無法變更受 AWS 管理策略中定義的權限。如果 AWS 更新 AWS 受管理原則中定義的權限，則此更新會影響附加原則的所有主體識別 (使用者、群組和角色)。 AWS 當新的啟動或新 AWS 服務 的 API 操作可用於現有服務時，最有可能更新 AWS 受管理策略。

如需詳細資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策。

AWS 的管理 (預先定義) 策略 AWS Glue

AWS 透過提供由建立和管理的獨立 IAM 政策來解決許多常見使用案例 AWS。這些 AWS 受管理的政策會為常見使用案例授與必要的權限，因此您可以避免調查需要哪些權限。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 受管理的原則 (您可以附加至帳戶中的身分識別) 屬於使用案例，AWS Glue並依使用案例分組：

- AWSGlueConsoleFullAccess— 當附加原則的身分使用時，授與AWS Glue資源的完整存取權 AWS Management Console。如果您依照此政策中指定的資源命名慣例，使用者就能擁有完整的主控台功能。此政策通常會連接至 AWS Glue 主控台的使用者。
- AWSGlueServiceRole— 授予對代表您執行各種AWS Glue程序所需之資源的存取權。這些資源包括 AWS Glue Amazon S3、IAM、 CloudWatch 日誌和 Amazon EC2。如果您依照此政策中指定的資源命名慣例，AWS Glue 程序就能擁有必要的許可。此政策通常會連接至定義爬蟲程式、工作和開發端點時所指定的角色。

- AwsGlueSessionUserRestrictedServiceRole— 提供對工作階段以外的所有AWS Glue資源的完整存取權。其允許使用者建立並僅使用與使用者相關聯的互動式工作階段。此原則包含管理其他 AWS 服務中AWS Glue資源所需的其他權限。AWS Glue該策略還允許向其他 AWS 服務中的AWS Glue資源添加標籤。

  > **Note**
  >
  > 若要達到完整的安全性優點，請勿將此政策授予已指派 `AWSGlueServiceRole`、`AWSGlueConsoleFullAccess` 或 `AWSGlueConsoleSageMakerNotebookFullAccess` 政策的使用者。

- AwsGlueSessionUserRestrictedPolicy— 僅在提供與受指派人使用者 ID 相符的標籤金鑰和值時，才能存取使用 `CreateSession` API 作業建立AWS Glue互動式工作階段。 AWS 此身分政策連接至叫用 `CreateSession` API 操作的 IAM 使用者。此原則也允許工作負責人與 AWS 使用者 ID 相符的「owner」標籤和值建立的AWS Glue互動式工作階段資源互動。建立工作階段之後，此政策拒絕變更或移除 AWS Glue 工作階段資源中「擁有者」標籤的許可。

  > **Note**
  >
  > 若要達到完整的安全性優點，請勿將此政策授予已指派 `AWSGlueServiceRole`、`AWSGlueConsoleFullAccess` 或 `AWSGlueConsoleSageMakerNotebookFullAccess` 政策的使用者。

- AwsGlueSessionUserRestrictedNotebookServiceRole— 提供對AWS Glue Studio筆記本工作階段的足夠存取權，以便與特定的AWS Glue互動式工作階段資源 這些是 AWS 使用「owner」標籤值建立的資源，符合建立筆記本之主體 (IAM 使用者或角色) 的使用者 ID。如需這些標籤的詳細資訊，請參閱《IAM 使用者指南》中的主體鍵值圖表。

  此服務角色政策會連接至筆記本內使用神奇代碼命令指定的角色，或作為角色傳遞給 `CreateSession` API 操作。只有當標籤鍵「owner」和值符合主參與者的使用者識別碼時，此原則也允許主參與 AWS 者從AWS Glue Studio筆記本介面建立AWS Glue互動式工作階段。建立工作階段之後，此政策拒絕變更或移除 AWS Glue 工作階段資源中「擁有者」標籤的許可。此政策還包括從 Amazon S3 儲存貯體寫入和讀取、撰寫 CloudWatch 日誌，以及為所使用之 Amazon EC2 資源建立和刪除標籤的許可AWS Glue。

> **Note**
>
> 若要達到完整的安全性優點，請勿將此政策授予已指派 `AWSGlueServiceRole`、`AWSGlueConsoleFullAccess` 或 `AWSGlueConsoleSageMakerNotebookFullAccess` 政策的角色。

- AwsGlueSessionUserRestrictedNotebookPolicy— 僅當有標籤金鑰「owner」和值符合使用者 ID 建立AWS Glue Studio筆記本的主體 (IAM AWS 使用者或角色) 時，才提供從筆記本介面建立AWS Glue互動式工作階段的存取權。如需這些標籤的詳細資訊，請參閱《IAM 使用者指南》中的主體鍵值圖表。

  此政策會連接至從 AWS Glue Studio 筆記本介面建立工作階段的主體 (IAM 使用者或角色)。此政策還允許足夠的 AWS Glue Studio 筆記本存取權，以便與特定的 AWS Glue 互動式工作階段資源互動。這些資源是使用符合主體的 AWS 使用者識別碼的「owner」標籤值所建立的資源。建立工作階段之後，此政策拒絕變更或移除 AWS Glue 工作階段資源中「擁有者」標籤的許可。

- AWSGlueServiceNotebookRole— 授予對在AWS Glue Studio筆記本中啟動之AWS Glue工作階段的存取權。此原則允許列出並取得所有工作階段的工作階段資訊，但只允許使用者建立和使用標記為其使用 AWS 者 ID 的工作階段。此原則拒絕從使用其 AWS ID 標記的AWS Glue工作階段資源中變更或移除「owner」標籤的權限。

  將此原則指派給使用中的筆記本介面建立工作的使用 AWS 者AWS Glue Studio。

- AWSGlueConsoleSageMakerNotebookFullAccess— 當附加原則的身分使用時，授與 AWS Glue 和 SageMaker 資源的完整存取權 AWS Management Console。如果您依照此政策中指定的資源命名慣例，使用者就能擁有完整的主控台功能。此原則通常會附加至管理 SageMaker 筆記本電腦的AWS Glue主控台使用者。

- AWSGlueSchemaRegistryFullAccess— 當附加原則的身分使用 AWS Management Console 或時，授與對AWS Glue結構描述登錄資源的完整存取權 AWS CLI。如果您依照此政策中指定的資源命名慣例，使用者就能擁有完整的主控台功能。此原則通常會附加至AWS Glue主控台的使用 AWS CLI 者或管理AWS Glue結構描述登錄的使用者。

- AWSGlueSchemaRegistryReadonlyAccess— 當附加原則的身分使用 AWS Management Console 或時，授與對AWS Glue結構描述登錄資源的唯讀存取權 AWS CLI。如果您依照此政策中指定的資源命名慣例，使用者就能擁有完整的主控台功能。此原則通常會附加至AWS Glue主控台的使用 AWS CLI 者或使用AWS Glue結構描述登錄的使用者。

> **Note**
>
> 您可以登入 IAM 主控台並在該處搜尋特定政策，來檢閱這些許可政策。

您也可以建立自己的自訂 IAM 政策，以允許 AWS Glue 動作與資源的許可。您可以將這些自訂政策連接至需要這些許可的 IAM 使用者或群組。

AWS 將 AWS 受管理政策的更新

檢視 AWS Glue AWS 受管理原則更新的詳細資料，因為此服務開始追蹤這些變更。如需有關此頁面變更的自動警示，請訂閱「 AWS Glue 合文件歷史記錄」頁面上的 RSS 摘要。

變更	描述	日期
AwsGlueSessionUserRestrictedPolicy — 對現有策略的次要更新。	將 `glue:StartCompletion` 和 `glue:GetCompletion` 新增至政策。必須在 AWS Glue 中進行 Amazon Q 資料整合。	二二二四年四月三十日
AwsGlueSessionUserRestrictedNotebookServiceRole — 對現有策略的次要更新。	將 `glue:StartCompletion` 和 `glue:GetCompletion` 新增至政策。必須在 AWS Glue 中進行 Amazon Q 資料整合。	二二二四年四月三十日
AwsGlueSessionUserRestrictedServiceRole — 對現有策略的次要更新。	將 `glue:StartCompletion` 和 `glue:GetCompletion` 新增至政策。必須在 AWS Glue 中進行 Amazon Q 資料整合。	二二二四年四月三十日
AWSGlueServiceNotebookRole — 對現有策略的次要更新。	將 `glue:StartCompletion` 和 `glue:GetCompletion` 新增至政策。	2024年1月30日

變更	描述	日期
	必須在 AWS Glue 中進行 Amazon Q 資料整合。	
AwsGlueSessionUserRestrictedNotebookPolicy — 對現有策略的次要更新。	將 `glue:StartCompletion` 和 `glue:GetCompletion` 新增至政策。必須在 AWS Glue 中進行 Amazon Q 資料整合。	2023 年 11 月 29 日
AWSGlueServiceNotebookRole — 對現有策略的次要更新。	將 `codewhisperer:GenerateRecommendations` 新增至政策。 AWS Glue 產生 CodeWhisperer 建議的新功能是必需的。	2023 年 10 月 9 日
AWSGlueServiceRole — 對現有策略的次要更新。	收緊 CloudWatch 權限範圍，以更好地反映 AWS Glue 記錄。	2023 年 8 月 4 日
AWSGlueConsoleFullAccess — 對現有策略的次要更新。	將 `databrew` 配方清單和描述許可新增至政策。必須提供 AWS Glue 可存取配方的新功能的完整管理存取權。	2023 年 5 月 9 日
AWSGlueConsoleFullAccess — 對現有策略的次要更新。	將 `cloudformation:ListStacks` 新增至政策。在 AWS CloudFormation 授權需求變更後保留現有功能。	2023 年 3 月 28 日

變更	描述	日期
為互動式工作階段功能新增新的受管政策： • AwsGlueSessionUserRestrictedServiceRole • AwsGlueSessionUserRestrictedPolicy • AwsGlueSessionUserRestrictedNotebookServiceRole • AwsGlueSessionUserRestrictedNotebookPolicy	這些政策的設計目的是為 AWS Glue Studio 中的互動式工作階段和筆記本提供更多的安全性。這些政策限制對 `CreateSession` API 操作的存取，只有擁有者可以存取。	2021 年 11 月 30 日
AWSGlueConsoleSageMakerNotebookFullAccess — 更新至現有原則。	針對 AWS Glue 用於存放指令碼和臨時檔案的 Amazon S3 儲存貯體上授予讀取/寫入許可的動作，移除了冗餘資源 ARN (`arn:aws:s3:::aws-glue-*/*`)。 修正語法問題，方法是將 `"StringEquals"` 變更為 `"ForAnyValue:StringLike"` ，並將 `"Effect": "Allow"` 行移動到它們亂序的每個位置的 `"Action":` 行之前。	2021 年 7 月 15 日
AWSGlueConsoleFullAccess — 更新至現有原則。	針對 `arn:aws:s3:::aws-glue-*/*` 用於存放指令碼和臨時檔案的 Amazon S3 儲存貯體上授予讀取/寫入許可的動作，移除了冗餘資源 ARN (AWS Glue)。	2021 年 7 月 15 日

變更	描述	日期
AWS Glue 已開始追蹤變更。	AWS Glue開始追蹤其 AWS 受管理策略的變更。	2021 年 6 月 10 日

指定 AWS Glue 資源 ARN

在 AWS Glue 中，您可以使用 AWS Identity and Access Management (IAM) 政策控制對資源的存取。在政策中，您使用 Amazon Resource Name (ARN) 來識別要套用政策的資源。並不是所有 AWS Glue 中的資源都支援 ARN。

主題

Data Catalog ARN

Data Catalog 資源擁有階層結構，以 `catalog` 為根。

```
arn:aws:glue:region:account-id:catalog
```

每個 AWS 帳戶在 AWS 區域中都有單一的 Data Catalog ，並以 12 位數的帳戶 ID 做為目錄 ID。具有相關聯唯一 ARN 的資源，如下表所示。

Resource Type (資源類型)	ARN 格式
目錄	`arn:aws:glue:` *`region`*`:`*`account-id`* `:catalog` 例如：`arn:aws:glue:us-east-1:123456789012:catalog`
資料庫	`arn:aws:glue:` *`region`*`:`*`account-id`* `:database/` *`database name`*

Resource Type (資源類型)	ARN 格式
	例如：arn:aws:glue:us-east-1:123456789012:database/db1
資料表	arn:aws:glue: *region*:*account-id* :table/*database name*/*table name* 例如：arn:aws:glue:us-east-1:123456789012:table/db1/tbl1
使用者定義的函數	arn:aws:glue: *region*:*account-id* :userDefinedFunction/ *database name*/*user-defined function name* 例如：arn:aws:glue:us-east-1:123456789012:userDefinedFunction/db1/func1
Connection	arn:aws:glue: *region*:*account-id* :connection/ *connection name* 例如：arn:aws:glue:us-east-1:123456789012:connection/connection1
互動式工作階段	arn:aws:glue: *region*:*account-id* :session/ *interactive session id* 例如：arn:aws:glue:us-east-1:123456789012:session/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111

若要啟用精細定義的存取控制，您可以在您的 IAM 政策和資源政策中使用這些 ARN，以授予或拒絕特定資源的存取權。政策中允許萬用字元。例如，以下 ARN 會比對資料庫 default 中的所有資料表。

```
arn:aws:glue:us-east-1:123456789012:table/default/*
```

Important

Data Catalog 資源上執行的所有操作，都需要資源的許可與該資源所有上階的許可。例如，建立資料表分割區時，需要資料表、資料庫和資料表所在目錄的許可。以下範例說明要在 Data Catalog 的 `PrivateDatabase` 資料庫中建立 `PrivateTable` 資料表分割區的必要許可。

```
{
   "Sid": "GrantCreatePartitions",
   "Effect": "Allow",
   "Action": [
       "glue:BatchCreatePartitions"
   ],
   "Resource": [
       "arn:aws:glue:us-east-1:123456789012:table/PrivateDatabase/PrivateTable",
       "arn:aws:glue:us-east-1:123456789012:database/PrivateDatabase",
       "arn:aws:glue:us-east-1:123456789012:catalog"
   ]
}
```

除了資源的許可與其所有上階的許可，所有刪除操作也需要該資源所有子項的許可。例如，刪除資料庫，除需要資料庫及其所在目錄的許可之外，還需要資料庫中所有資料表和使用者定義函數的許可。以下範例說明要刪除 Data Catalog 中 `PrivateDatabase` 資料庫的必要許可。

```
{
   "Sid": "GrantDeleteDatabase",
   "Effect": "Allow",
   "Action": [
       "glue:DeleteDatabase"
   ],
   "Resource": [
       "arn:aws:glue:us-east-1:123456789012:table/PrivateDatabase/*",
       "arn:aws:glue:us-east-1:123456789012:userDefinedFunction/PrivateDatabase/
*",
       "arn:aws:glue:us-east-1:123456789012:database/PrivateDatabase",
       "arn:aws:glue:us-east-1:123456789012:catalog"
   ]
}
```

總而言之，對 Data Catalog 資源採取的動作需遵循以下許可規則：

- 針對目錄採取的動作只需要有目錄的許可即可。

- 針對資料庫採取的動作需要資料庫和目錄的許可。
- 針對資料庫的刪除動作需要資料庫和目錄的許可，再加上資料庫中所有資料表和使用者定義函數的許可。
- 針對資料表、分割區或資料表版本採取的動作需要資料表、資料庫和目錄的許可。
- 針對使用者定義函數採取的動作需要使用者定義函數、資料庫和目錄的許可。
- 針對連線採取的動作需要連線和目錄的許可。

AWS Glue 中非目錄物件的 ARN

有些 AWS Glue 資源允許資源層級許可使用 ARN 來控制存取。您可以在 IAM 政策中使用這些 ARN 來啟用精細定義的存取控制。下表列出的資源可以包含資源 ARN。

Resource Type (資源類型)	ARN 格式
爬蟲程式	arn:aws:glue: *region*:*account-id* :crawler/ *crawler-name* 例如：arn:aws:glue:us-east-1:123456789012:crawler/mycrawler
任務	arn:aws:glue: *region*:*account-id* :job/*job-name* 例如：arn:aws:glue:us-east-1:123456789012:job/testjob
觸發條件	arn:aws:glue: *region*:*account-id* :trigger/ *trigger-name* 例如：arn:aws:glue:us-east-1:123456789012:trigger/sampletrigger
開發端點	arn:aws:glue: *region*:*account-id* :devEndpoint/ *development-endpoint-name* 例如：arn:aws:glue:us-east-1:123456789012:devEndpoint/temporarydevendpoint

Resource Type (資源類型)	ARN 格式
機器學習轉換	arn:aws:glue: *region*:*account-id* :mlTransform/ *transform-id* 例如：arn:aws:glue:us-east-1:123456789012:mlTransform/tfm-1234567890

對 AWS Glue 非目錄單數 API 操作的存取控制

AWS Glue 非目錄單數 API 操作會在單一項目 (開發端點) 上執行動作。範例包括 GetDevEndpoint、CreateUpdateDevEndpoint 和 UpdateDevEndpoint。針對這些操作，政策的 "action" 區塊中必須放置 API 名稱，而 "resource" 區塊中必須放置資源 ARN。

假設您想要允許使用者呼叫 GetDevEndpoint 操作。下列政策會將最低必要許可授予名為 myDevEndpoint-1 的端點。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "MinimumPermissions",
            "Effect": "Allow",
            "Action": "glue:GetDevEndpoint",
            "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/
myDevEndpoint-1"
        }
    ]
}
```

以下政策允許 UpdateDevEndpoint 使用萬用字元 (*) 存取符合 myDevEndpoint- 的資源。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PermissionWithWildcard",
            "Effect": "Allow",
            "Action": "glue:UpdateDevEndpoint",
```

```
            "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/myDevEndpoint-
*"
        }
    ]
}
```

您可以結合兩個政策，如下範例所示。您可以查看 EntityNotFoundException 以取得名稱開頭為 A 的任何開發端點。不過，當您嘗試存取其他開發端點時會傳回拒絕存取錯誤。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CombinedPermissions",
            "Effect": "Allow",
            "Action": [
                "glue:UpdateDevEndpoint",
                "glue:GetDevEndpoint"
            ],
            "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/A*"
        }
    ]
}
```

可擷取多個項目的 AWS Glue 非目錄 API 操作存取控制

有些 AWS Glue API 操作可擷取多個項目 (例如多個開發端點)，像是 GetDevEndpoints。針對此操作，您只能指定萬用字元 (*) 資源，不能使用特定的 ARN。

例如，若要在政策中包含 GetDevEndpoints，資源範圍必須以萬用字元 (*) 放大。單數操作 (GetDevEndpoint、CreateDevEndpoint 和 DeleteDevendpoint) 也會將範圍限制為範例中的所有 (*) 資源。

```
{
            "Sid": "PluralAPIIncluded",
            "Effect": "Allow",
            "Action": [
                "glue:GetDevEndpoints",
                "glue:GetDevEndpoint",
                "glue:CreateDevEndpoint",
                "glue:UpdateDevEndpoint"
            ],
```

```
            "Resource": [
                "*"
            ]
}
```

AWS Glue 非目錄 BatchGet API 操作的存取控制

有些 AWS Glue API 操作可擷取多個項目 (例如多個開發端點)，像是 `BatchGetDevEndpoints`。處理此操作時，您可以指定一個 ARN 來限制可存取資源的範圍。

例如，若要允許存取特定的開發端點，可使用其資源 ARN 在政策中包含 `BatchGetDevEndpoints`。

```
{
            "Sid": "BatchGetAPIIncluded",
            "Effect": "Allow",
            "Action": [
                "glue:BatchGetDevEndpoints"
            ],
            "Resource": [
                "arn:aws:glue:us-east-1:123456789012:devEndpoint/de1"
            ]
}
```

使用此政策，您就可以成功存取名為 `de1` 的開發端點。不過，如果您嘗試存取名為 `de2` 的開發端點，此時就會傳回錯誤。

```
An error occurred (AccessDeniedException) when calling the BatchGetDevEndpoints
 operation: No access to any requested resource.
```

> **Important**
>
> 如需設定 IAM 政策的替代方法，例如使用 `List` 和 `BatchGet` API 操作，請參閱 AWS Glue 的身分型政策範例。

授予跨帳戶存取權

跨帳戶授予 Data Catalog 資源存取權可讓您的擷取、轉換和載入 (ETL) 任務查詢和聯結來自不同帳戶的資料。

主題

在 AWS Glue 中授予跨帳戶存取的方法

您可以使用AWS Glue方法或使用 AWS Lake Formation 跨 AWS 帳戶授權，將資料的存取權授予外部帳戶。這些AWS Glue方法使用 AWS Identity and Access Management (IAM) 政策實現精細的存取控制。Lake Formation 使用更簡單的 GRANT/REVOKE 許可模型，類似於關聯式資料庫系統中的 GRANT/REVOKE 命令。

本節說明如何使用 AWS Glue 方法。如需使用 Lake Formation 跨帳戶授予的詳細資訊，請參閱AWS Lake Formation 開發人員指南中的授予 Lake Formation 許可。

有兩個 AWS Glue 方法可授予資源的跨帳戶存取權：

- 使用 Data Catalog 資源政策
- 使用 IAM 角色

使用資源政策授予跨帳戶存取

以下是使用 Data Catalog 資源政策授予跨帳戶存取權的一般步驟：

1. 帳戶 A 中的管理員 (或其他授權身分) 會將資源政策連接到帳戶 A 中的 Data Catalog 。此政策會授予帳戶 B 特定跨帳戶許可，以對帳戶 A 目錄中的資源執行操作。
2. 帳戶 B 中的管理員會將 IAM 政策連接至帳戶 B 中委派接收自帳戶 A 之許可的 IAM 身分。

   帳戶 B 中的身分現在可存取帳戶 A 中所指定的資源。

   身分同時需要資源擁有者 (帳戶 A) 和其父帳戶 (帳戶 B) 的許可，才能存取資源。

使用 IAM 角色授予跨帳戶存取權

以下是使用 IAM 角色授予跨帳戶存取權的一般步驟：

1. 帳戶中擁有資源 (帳戶 A) 的管理員 (或其他授權身分) 會建立 IAM 角色。
2. 帳戶 A 中的管理員會將政策連接到角色，以授予跨帳戶許可以存取有問題的資源。
3. 帳戶 A 中的管理員會將信任政策連接到角色，以將不同帳戶 (帳戶 B) 中的 IAM 身分識別為可擔任該角色的委託人。

   如果您想要授與 AWS 服務權限來擔任角色，則信任原則中的主體也可以是 AWS 服務主體。
4. 帳戶 B 中的管理員現在會將許可委派給帳戶 B 中的一或多個 IAM 身分，讓他們可以擔任該角色。這樣做會將帳戶 A 中的資源存取權授予帳戶 B 中的這些身分。

如需有關使用 IAM 來委派許可的詳細資訊，請參閱《IAM 使用者指南》中的存取管理。如需使用者、群組、角色和許可的相關資訊，請參閱《IAM 使用者指南》中的身分 (使用者、群組和角色)。

有關這兩種方法的比較資料，請參閱《IAM 使用者指南》中的 IAM 角色與資源型政策有何差異。AWS Glue 支援這兩個選項，但限制是資源政策只能授予 Data Catalog 資源存取權。

例如，若要讓帳戶 B 的 Dev 角色存取帳戶 A 的 db1 資料庫，請將下列資源政策連接到帳戶 A 的目錄。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase"
      ],
      "Principal": {"AWS": [
        "arn:aws:iam::account-B-id:role/Dev"
      ]},
      "Resource": [
        "arn:aws:glue:us-east-1:account-A-id:catalog",
        "arn:aws:glue:us-east-1:account-A-id:database/db1"
      ]
    }
  ]
}
```

此外，帳戶 B 還必須先將下列 IAM 政策連接至 Dev 角色，他才能真正存取帳戶 A 的 db1。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase"
      ],
      "Resource": [
        "arn:aws:glue:us-east-1:account-A-id:catalog",
        "arn:aws:glue:us-east-1:account-A-id:database/db1"
      ]
    }
  ]
}
```

新增或更新 Data Catalog 資源政策

您可以使用主控台、API 或 AWS Command Line Interface (AWS CLI) 新增或更新資AWS Glue料目錄資源策略。

> **Important**
>
> 如果您已經使用 AWS Lake Formation從帳戶授予跨帳戶許可，則新增或更新 Data Catalog 資源政策需要額外的步驟。如需詳細資訊，請參閱《AWS Lake Formation 開發人員指南》中的同時使用 AWS Glue 和 Lake Formation 管理跨帳戶許可。
> 若要判斷 Lake Formation 跨帳戶授予是否存在，請使用 `glue:GetResourcePolicies` API 操作或 AWS CLI。如果 `glue:GetResourcePolicies` 傳回現有 Data Catalog 政策以外的任何政策，則存在 Lake Formation 授予。如需詳細資訊，請參閱AWS Lake Formation 開發人員指南中的使用 GetResourcePolicies API 作業檢視所有跨帳戶授權。

新增或更新 Data Catalog 資源政策 (主控台)

1. 開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。

   以具有`glue:PutResourcePolicy`權限的 AWS Identity and Access Management (IAM) 管理使用者身分登入。

2. 在導覽窗格中，選擇設定。

3. 在 Data catalog settings (Data Catalog 設定) 頁面，在 Permissions (許可) 下，將資源政策貼到文字區域。然後選擇 Save (儲存)。

   如果主控台顯示警示，指出政策中的許可將會新增至使用 Lake Formation 授予的任何許可，請選擇 Proceed (繼續)。

新增或更新 Data Catalog 資源政策 (AWS CLI)

- 提交 `aws glue put-resource-policy` 命令。如果 Lake Formation 授予已經存在，請確保您包含具有值 `'TRUE'` 的 `--enable-hybrid` 選項。

  如需使用此命令的範例，請參閱AWS Glue 以資源型為基礎的範例。

發起跨帳戶 API 呼叫

所有 AWS Glue Data Catalog 操作都會有 `CatalogId` 欄位。如已授予啟用跨帳戶存取的必要許可，發起人即可跨帳戶發起 Data Catalog API 呼叫。發起人可在 `CatalogId` 中傳遞目標 AWS 帳戶 ID 來執行此操作，以便存取該目標帳戶中的資源。

如果未提供任何 `CatalogId` 值，則根據預設，AWS Glue 會使用發起人的專屬帳戶 ID，而且呼叫未跨帳戶。

發起跨帳戶 ETL 呼叫

一些AWS Glue PySpark 和斯卡拉 API 有一個目錄 ID 字段。如果已授與所有必要權限來啟用跨帳戶存取，則 ETL 工作可以透過在目錄 ID 欄位中傳遞目標帳 AWS 戶 ID 以存取目標帳戶中的資料目錄資源，來跨帳戶對 API 作業進行 PySpark 和 Scala 呼叫。

如果未提供任何目錄 ID 值，則根據預設，AWS Glue 會使用發起人的專屬帳戶 ID，而且呼叫未跨帳戶。

如需支援的 PySpark APIcatalog_id，請參閱GlueContext 類。如需支援 `catalogId` 的 Scala API，請參閱 AWS Glue斯卡 GlueContext 拉。

下列範例顯示承授者執行 ETL 任務所需的許可。在此範例中，*`grantee-account-id`*是執行作業`catalog-id`的用戶端，並且*`grantor-account-id`*是資源的擁有者。此範例會授予授予者帳戶中所有目錄資源的許可。若要限制所授予資源的範圍，您可以提供目錄、資料庫、資料表和連線的特定 ARN。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetConnection",
                "glue:GetDatabase",
                "glue:GetTable",
                "glue:GetPartition"
            ],
            "Principal": {"AWS": ["arn:aws:iam::grantee-account-id:root"]},
            "Resource": [
                "arn:aws:glue:us-east-1:grantor-account-id:*"
            ]
        }
    ]
}
```

> **Note**
>
> 如果授予者帳戶中的資料表指向也在授予者帳戶中的 Amazon S3 位置，則承授者帳戶中用來執行 ETL 任務的 IAM 角色必須具備從授予者帳戶中列出和取得物件的許可。

如果帳戶 A 的用戶端已具備建立和執行 ETL 任務的許可，則設定跨帳戶存取權 ETL 任務的基本步驟如下：

1. 允許跨帳戶資料存取 (如果已設定 Amazon S3 跨帳戶存取權，則請略過此步驟)。
   a. 更新帳戶 B 中的 Amazon S3 儲存貯體政策，以允許從帳戶 A 的跨帳戶存取權。
   b. 更新帳戶 A 中的 IAM 政策，以允許存取帳戶 B 中的儲存貯體。
2. 允許跨帳戶 Data Catalog 存取。
   a. 建立或更新連接到帳戶 B 中 Data Catalog 的資源政策，以允許從帳戶 A 存取。
   b. 更新帳戶 A 中的 IAM 政策，以允許存取帳戶 B 中的 Data Catalog 。

跨帳戶 CloudTrail 記錄

當AWS Glue擷取、轉換和載入 (ETL) 工作存取透過 AWS Lake Formation 跨帳戶授權共用之「資料目錄」資料表的基礎資料時，會有其他 AWS CloudTrail 記錄行為。

為了進行此討論，共用資料表的 AWS 帳戶是擁有者帳戶，而共用資料表的帳戶是收件者帳戶。當收件者帳戶中的 ETL 工作存取擁有者帳戶中資料表中的資料時，新增至收件者帳戶記錄檔的資料存取 CloudTrail 事件會複製到擁有者帳戶的 CloudTrail 記錄檔。如此一來，擁有者帳戶可以追蹤各種收件者帳戶的資料存取。依預設， CloudTrail 事件不包含人類可讀的主參與者識別元 (主體 ARN)。收件者帳戶中的系統管理員可以選擇加入，在記錄中包含主體 ARN。

如需詳細資訊，請參閱AWS Lake Formation 開發人員指南中的跨帳戶 CloudTrail記錄。

> **另請參閱**
>
> - the section called “記錄和監控”

跨帳戶資源所有權和帳單

當一個 AWS 帳號中的使用者 (帳戶 A) 建立新資源 (例如在不同帳號 B) 中的資料庫時，該資源便會由建立該資源的帳號 B 所擁有。帳戶 B 中的管理員會自動取得存取新資源的完整許可 (包含讀取、寫入以及授予第三個帳戶的存取許可)。只有在帳戶 A 中的使用者具備帳戶 B 所授予的適當許可時，才能存取他們剛剛所建立的資源。

儲存成本以及與新資源直接建立關聯的其他成本都會計費到帳戶 B (資源擁有者)。來自建立資源之使用者的要求成本會計費到申請者帳戶 (帳戶 A)。

如需AWS Glue計費和定價的詳細資訊，請參閱定 AWS 價方式。

跨帳戶存取限制

AWS Glue 跨帳戶存取的限制如下：

- 如果您在區域支援 AWS Glue 之前使用 Amazon Athena 或 Amazon Redshift Spectrum 建立資料庫和資料表，並且資源擁有者帳戶尚未將 Amazon Athena Data Catalog 遷移至 AWS Glue，則不允許跨帳戶存取 AWS Glue。您可以使用 GetCatalogImportStatus (get_catalog_import_status) 找到目前移轉狀態。如需有關如何將 Athena 目錄移轉至的詳細資訊AWS Glue，請參閱 Amazon Athena 使用者指南AWS Glue Data Catalog step-by-step中的升級至。
- 只有 Data Catalog 資源 (包含資料庫、資料表、使用者定義的函數和連線) 支援跨帳戶存取。
- 從 Athena 跨帳戶存取 Data Catalog 需要您將目錄註冊為 Athena `DataCatalog` 資源。如需說明，請參閱《Amazon Athena 使用者指南》中的從另一個帳戶註冊 AWS Glue Data Catalog。

疑難排解 AWS Glue 身分識別

使用下列資訊可協助您診斷並修正使用 AWS Glue 和 IAM 時可能會遇到的常見問題。

主題

我沒有在 AWS Glue 中執行動作的授權

如果您收到錯誤，告知您未獲授權執行動作，您的政策必須更新，允許您執行動作。

下列範例錯誤會在mateojackson IAM 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 glue:*GetWidget* 許可時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 glue:GetWidget on resource: my-example-widget
```

在此情況下，必須更新 mateojackson 使用者的政策，允許使用 glue:*GetWidget* 動作存取 *my-example-widget* 資源。

如果您需要協助，請聯絡您的 AWS 系統管理員。您的管理員提供您的簽署憑證。

我沒有授權執行 iam：PassRole

如果您收到未獲授權執行iam:PassRole動作的錯誤訊息，您必須更新原則，才能讓您將角色傳遞給 AWS Glue。

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 的 IAM 使用者嘗試使用主控台在 AWS Glue 中執行動作時，發生下列範例錯誤。但是，動作請求服務具備服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 `iam:PassRole` 動作。

如果您需要協助，請聯絡您的 AWS 系統管理員。您的管理員提供您的簽署憑證。

我想允許我以外的人訪問我 AWS 帳戶 的 AWS Glue 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要瞭解 AWS Glue 是否支援這些功能，請參閱AWS Glue 如何與 IAM 搭配使用。
- 若要了解如何提供對您所擁有資源 AWS 帳戶 的存取權，請參閱《IAM 使用者指南》中您擁有的另一 AWS 帳戶 個 IAM 使用者提供存取權限。
- 若要了解如何將資源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 AWS 帳戶 擁有的存取權。
- 如需了解如何透過聯合身分提供存取權，請參閱 IAM 使用者指南中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 若要了解跨帳戶存取使用角色和以資源為基礎的政策之間的差異，請參閱 IAM 使用者指南中的 IAM 中的跨帳戶資源存取。

AWS Glue 中的記錄和監控

您可以自動執行您的 ETL (擷取、轉換和載入) 任務。AWS Glue 會提供您可監控的爬蟲程式和任務指標。為 AWS Glue Data Catalog 設定必要的中繼資料後，AWS Glue 便能提供關於環境運作狀態的統計資料。您可以根據 Cron 為基礎的時間排程自動化呼叫爬蟲程式和任務。也可以在事件型觸發因子引起時觸發任務。

AWS Glue 與 AWS CloudTrail 整合，該服務提供由使用者、角色或 AWS 服務在 AWS Glue 中所採取的動作記錄。如果您建立追蹤，就可以將 CloudTrail 事件持續交付到 Amazon Simple Storage Service (Amazon S3) 儲存貯體、Amazon CloudWatch Logs 和 Amazon CloudWatch Events。每一筆事件或日誌項目都會包含產生請求者的資訊。

您可以使用 Amazon CloudWatch Events 來自動化您的 AWS 服務，並自動回應應用程式可用性問題或資源變更等系統事件。AWS 服務的事件會以接近即時的方式傳送到 CloudWatch Events。您可編寫簡單的規則，來指示您在意的事件，以及當事件符合規則時所要自動執行的動作。

另請參閱

- 透過 CloudWatch Events 自動化 AWS Glue
- 跨帳戶 CloudTrail 記錄

雲端安全性的一個重要面向是日誌記錄。您必須以不會擷取祕密和機密材料的方式設定日誌記錄，同時要能擷取到用於除錯以及保障雲端基礎架構所需要的資訊。確認您熟悉正在記錄的日誌。

符合性驗證 AWS Glue

若要瞭解 AWS 服務 是否屬於特定規範遵循方案的範圍內，請參閱AWS 服務 遵循規範計劃方案中的，並選擇您感興趣的合規方案。如需一般資訊，請參閱AWS 規範計劃AWS。

您可以使用下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱下載中的報告中的 AWS Artifact。

您在使用時的合規責任取決 AWS 服務 於您資料的敏感性、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遵循法規：

- 安全性與合規性快速入門指南 — 這些部署指南討論架構考量，並提供部署以安全性和合規性 AWS 為重點的基準環境的步驟。
- 在 Amazon Web Services 上架構 HIPAA 安全性與合規性 — 本白皮書說明公司如何使用建立符合 HIPAA 資格的應 AWS 用程式。

  Note

  並非所有人 AWS 服務 都符合 HIPAA 資格。如需詳細資訊，請參閱 HIPAA 資格服務參照。

- AWS 合規資源AWS — 此工作簿和指南集合可能適用於您的產業和所在地。
- AWS 客戶合規指南 — 透過合規的角度瞭解共同的責任模式。這份指南總結了在多個架構 (包括美國國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準化組織 (ISO)) 中，保 AWS 服務 護指引並對應至安全控制的最佳實務。
- 使用AWS Config 開發人員指南中的規則評估資源 — 此 AWS Config 服務會評估您的資源組態符合內部實務、產業準則和法規的程度。

- AWS Security Hub— 這 AWS 服務 提供了內部安全狀態的全面視圖 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 — 透過監控環境中的 AWS 帳戶可疑和惡意活動，藉此 AWS 服務 偵測您的工作負載、容器和資料的潛在威脅。 GuardDuty 可協助您滿足特定合規性架構所要求的入侵偵測需求，例如 PCI DSS 等各種合規性需求。
- AWS Audit Manager— 這 AWS 服務 有助於您持續稽核您的 AWS 使用情況，以簡化您管理風險的方式，以及遵守法規和業界標準的方式。

韌性 AWS Glue

AWS 全球基礎架構是圍繞區 AWS 域和可用區域建立的。 AWS 區域提供多個實體分離和隔離的可用區域，這些區域透過低延遲、高輸送量和高度備援的網路連線。透過可用區域，您所設計與操作的應用程式和資料庫，就能夠在可用區域之間自動容錯移轉，而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如需區域和可用區域的相關 AWS 資訊，請參閱AWS 全域基礎結構。

如需有關 AWS Glue 工作恢復能力的詳細資訊，請參閱. AWS Glue

AWS Glue 中的基礎設施安全

AWS Glue 為受管服務，受到 Amazon Web Services：安全程序概觀白皮書所述的 AWS 全球網路安全程序所保護。

您可使用 AWS 發佈的 API 呼叫，透過網路存取 AWS Glue。用戶端必須支援 Transport Layer Security (TLS) 1.0 或更新版本。建議使用 TLS 1.2 或更新版本。用戶端也必須支援具備完美轉送私密 (PFS) 的密碼套件，例如臨時 Diffie-Hellman (DHE) 或橢圓曲線臨時 Diffie-Hellman (ECDHE)。現代系統 (如 Java 7 和更新版本) 大多會支援這些模式。

此外，請求必須使用存取索引鍵 ID 和與 IAM 主體相關聯的私密存取索引鍵來簽署。或者，您可以使用 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

主題

- AWS Glue 和介面 VPC 端點 (AWS PrivateLink)
- 共享 Amazon VPC

AWS Glue 和介面 VPC 端點 (AWS PrivateLink)

您可以建立介面 VPC 端點，以在您的 VPC 與 AWS Glue 間建立私有連線。介面端點是由 AWS PrivateLink 提供技術支援，這項技術可讓您在沒有網際網路閘道、NAT 裝置、VPN 連接或 AWS Direct Connect 連線的情況下私密地存取 AWS Glue API。VPC 中的執行個體不需要公有 IP 地址，即能與 AWS Glue API 通訊。 您的 VPC 與 AWS Glue 之間的網路流量都會在 Amazon 網路的範圍內。

每個介面端點都是由您子網路中的一或多個彈性網絡介面表示。

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)。

AWS Glue VPC 端點的考量事項

設定 AWS Glue 的介面 VPC 端點前，請務必檢閱《Amazon VPC 使用者指南》中的介面端點屬性和限制。

AWS Glue 支援從您的 VPC 呼叫其所有 API 動作。

為 AWS Glue 建立介面 VPC 端點

您可使用 Amazon VPC 主控台或 AWS Command Line Interface (AWS CLI) 來為 AWS Glue 服務建立 VPC 端點。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的建立介面端點。

使用下列服務名稱建立 AWS Glue 的 VPC 端點：

- com.amazonaws.*region*.glue

如果您為該端點啟用私有 DNS，您可以使用其區域的預設 DNS 名稱 (例如 `glue.us-east-1.amazonaws.com`)，向 AWS Glue 發出 API 請求。

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 AWS Glue 建立 VPC 端點政策

您可以將端點政策連接至控制 AWS Glue 存取權限的 VPC 端點。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：適用於 AWS Glue VPC 端點政策，允許建立和更新任務

以下是 AWS Glue 端點政策的範例。附加至端點後，此政策會針對所有資源上的所有主體，授予列出的 AWS Glue 動作的存取權限。

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
             "glue:CreateJob",
             "glue:UpdateJob",
             "iam:PassRole"
         ],
         "Resource":"*"
      }
   ]
}
```

範例：允許唯讀存取 Data Catalog 的 VPC 端點政策

以下是 AWS Glue 端點政策的範例。附加至端點後，此政策會針對所有資源上的所有主體，授予列出的 AWS Glue 動作的存取權限。

```
{
    "Statement": [
        {
            "Principal": "*",
            "Effect": "Allow",
            "Action": [
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:GetTable",
                "glue:GetTables",
                "glue:GetTableVersion",
                "glue:GetTableVersions",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:SearchTables"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

共享 Amazon VPC

AWS Glue 支援 Amazon Virtual Private Cloud 中的共用 Virtual Private Cloud (VPC)。Amazon VPC 支援多個 AWS 帳戶以將應用程式資源 (例如 Amazon EC2 執行個體和 Amazon Relational Database Service (Amazon RDS) 資料庫) 建立到共享、集中管理的 Amazon VPC。在此模型中，擁有 VPC (擁有者) 的帳戶會和屬於 中相同組織的其他帳戶 (參與者) 共用一個或多個子網路AWS Organizations 共用子網路後，參與者可以檢視、建立、修改及刪除與其共用之子網路中的應用程式資源。

在 AWS Glue 中，若要建立與共享子網路的連線，您必須在帳戶內建立安全群組，並將安全群組連接到共享子網路。

如需詳細資訊，請參閱以下主題：

- 如需詳細資訊，請參閱 Amazon VPC 使用者指南中的使用共用 VPC。
- AWS Organizations 使用者指南中的什麼是 AWS Organizations？

AWS Glue 疑難排解

主題

收集 AWS Glue 故障診斷資訊

如果您因 AWS Glue 發生錯誤或意外行為而需聯絡 AWS Support，您應該先收集與失敗動作相關的名稱、ID 和記錄等資訊。取得這些資訊可讓 AWS Support 協助您解決發生的問題。

除了您的帳戶 ID，請針對這些失敗類型收集以下資訊：

當爬蟲程式失敗，請收集以下資訊：

- 爬蟲程式名稱

  爬蟲程式執行日誌存放於 `/aws-glue/crawlers` 下的 CloudWatch Logs。

當測試連線失敗，請收集以下資訊：

- 連線名稱
- 連線 ID
- JDBC 連線字串，格式為 `jdbc:protocol://host:port/database-name`。

  測試連線日誌存放於 `/aws-glue/testconnection` 下的 CloudWatch Logs。

當任務失敗，請收集以下資訊：

- 任務名稱
- 任務執行 ID，格式為 `jr_xxxxx`。

  任務執行日誌存放於 `/aws-glue/jobs` 下的 CloudWatch Logs。

疑難排解 Spark 中 AWS Glue 的錯誤

如果在中遇到錯誤 AWS Glue，請使用下列解決方案來協助您尋找問題的來源並加以修正。

> **Note**
>
> AWS Glue GitHub 存放庫包含AWS Glue 常見問題集中的其他疑難排解指引。

主題

- 錯誤：爬蟲程式執行失敗
- 錯誤：未更新分割區
- 錯誤：由於版本不相符，任務書籤更新失敗
- 錯誤：當任務書籤啟用時，任務會重新處理資料
- 錯誤：中 VPC 之間的容錯移轉行為 AWS Glue
- 疑難排解爬蟲程式使用 Lake Formation 憑證時發生的錯誤

錯誤：資源無法使用

如果 AWS Glue 傳回無法使用資源的訊息，您可以檢視錯誤訊息或記錄檔，以協助您進一步瞭解問題。以下任務說明一般的故障診斷方法。

- 針對您所使用的任何連線和開發端點，請檢查您的叢集是否仍有彈性網路界面可用。

錯誤：無法在 VPC 中找到 subnetId 的 S3 端點或 NAT 閘道

檢查訊息中的子網路 ID 和 VPC ID，以協助診斷問題。

- 確認您已設定 Amazon S3 VPC 端點，此為 AWS Glue的要求。此外，如果組態內有 NAT 閘道，請檢查。如需詳細資訊，請參閱 Amazon S3 的 VPC 端點。

錯誤：安全群組需有傳入規則

必須至少有一個安全群組開啟所有傳入連接埠。若要限制流量，傳入規則中的來源安全群組可以限制為相同的安全群組。

- 針對您所使用的任何連線，請檢查自我參考的傳入規則的安全群組。如需詳細資訊，請參閱 設定對資料存放區的網路存取。
- 使用的是開發端點時，請檢查自我參考的傳入規則的安全群組。如需詳細資訊，請參閱 設定對資料存放區的網路存取。

錯誤：安全群組需有傳出規則

必須至少有一個安全群組開啟所有傳出連接埠。若要限制流量，傳出規則中的來源安全群組可以限制為相同的安全群組。

- 針對您所使用的任何連線，請檢查自我參考的傳出規則的安全群組。如需詳細資訊，請參閱 設定對資料存放區的網路存取。
- 使用的是開發端點時，請檢查自我參考的傳出規則的安全群組。如需詳細資訊，請參閱 設定對資料存放區的網路存取。

錯誤：Job 執行失敗，因為應將傳遞的角色指定為 AWS Glue 服務的角色權限

定義任務的使用者需擁有 AWS Glue的 `iam:PassRole` 許可。

- 當使用者建立 AWS Glue 工作時，請確認使用者的角色包含的策略包含`iam:PassRole` AWS Glue。如需詳細資訊，請參閱 步驟 3：連接政策到存取 AWS Glue 的使用者或群組。

錯誤： DescribeVpcEndpoints 動作未經授權。無法驗證虛擬私人雲端識別碼 vpc-id

- 檢查傳遞給的政策以 AWS Glue 獲取`ec2:DescribeVpcEndpoints`權限。

錯誤： DescribeRouteTables 動作未授權。無法驗證子網路識別碼：VPC ID 中的子網路識別碼：vpc-id

- 檢查傳遞給的政策以 AWS Glue 獲取`ec2:DescribeRouteTables`權限。

錯誤：無法調用 ec2：DescribeSubnets

- 檢查傳遞給的政策以 AWS Glue 獲取`ec2:DescribeSubnets`權限。

錯誤：無法調用 ec2：DescribeSecurityGroups

- 檢查傳遞給的政策以 AWS Glue 獲取 `ec2:DescribeSecurityGroups`權限。

錯誤：找不到可用區域的子網路

- 可用區域可能不適用於 AWS Glue。建立並使用可用區域中新的子網路，與訊息中所指定的可用區域不同。

錯誤：寫入到 JDBC 目標時發生任務執行例外

執行寫入到 JDBC 目標的任務時，任務在下列情況下可能發生錯誤：

- 如果您的任務寫入到 Microsoft SQL Server 資料表，且資料表有定義為 `Boolean` 類型的資料欄，則資料表必須預先在 SQL Server 資料庫中定義。當您在 AWS Glue 主控台上使用 SQL Server 目標與 [在資料目標中建立資料表] 選項來定義工作時，請勿將任何來源資料行對應至具有資料類型的目標資料行`Boolean`。執行任務時可能會發生錯誤。

  您可以執行以下動作避免錯誤：

  - 選擇含有 布林值 欄的現有資料表。
  - 編輯 `ApplyMapping` 轉換，並將來源的布林值欄映射至目標的數字或字串。
  - 編輯 `ApplyMapping` 轉換，移除來源中的布林值欄。
- 如果您的任務寫入到 Oracle 資料表，您可能需要調整 Oracle 物件名稱的長度。在某些版本的 Oracle，最大的 ID 長度限制為 30 位元組或 128 位元組。此限制影響 Oracle 目標資料存放區的資料表名稱和欄名稱。

  您可以執行以下動作避免錯誤：

  - 依照版本的限制為 Oracle 目標資料表命名。
  - 預設欄名稱從資料的欄位名稱產生。當欄名稱長度超過限制時，請使用 `ApplyMapping` 或 `RenameField` 轉換變更欄的名稱，以符合限制。

錯誤：Amazon S3：該操作對象的存儲類無效

如果 AWS Glue 傳回此錯誤，則您的 AWS Glue 任務可能已從跨 Amazon S3 儲存類別層級具有分割區的表格讀取資料。

- 透過使用儲存類別排除項，您可以確保您 AWS Glue 的工作可以在跨這些儲存類別層級具有分割區的表格上運作。如果沒有排除，則從這些層讀取資料的作業會失敗，並出現下列錯誤：`AmazonS3Exception: The operation is not valid for the object's storage class`。

如需詳細資訊，請參閱 排除 Amazon S3 儲存體方案。

錯誤：Amazon S3 逾時

如果 AWS Glue 傳回連線逾時錯誤，可能是因為它嘗試存取另一個 AWS 區域中的 Amazon S3 儲存貯體。

- Amazon S3 VPC 端點只能將流量路由到 AWS 區域內的儲存貯體。如果您需要連接到其他區域中的儲存貯體，可能的解決方法是使用 NAT 閘道。如需更多詳細資訊，請參閱 NAT 閘道。

錯誤：Amazon S3 存取遭拒

如果 AWS Glue 傳回 Amazon S3 儲存貯體或物件的存取遭拒錯誤，可能是因為提供的 IAM 角色沒有具有資料存放區許可的政策。

- ETL 任務必須要能存取 Amazon S3 資料存放區以做為來源或目標。爬蟲程式必須能夠存取編目的 Amazon S3 資料存放區。如需詳細資訊，請參閱 步驟 2：為 AWS Glue 建立 IAM 角色。

錯誤：Amazon S3 存取金鑰 ID 不存在

如果在運行作業時 AWS Glue 返回訪問密鑰 ID 不存在錯誤，則可能是由於以下原因之一：

- ETL 工作使用 IAM 角色存取任務資料存放區，在工作開始之前確認您工作的 IAM 角色尚未遭到刪除。
- IAM 角色包含存取您的資料存放區的許可，確認任何所附加包含 `s3:ListBucket` 的 Amazon S3 政策是正確的。

錯誤：使用 **`s3a://`** URI 存取 Amazon S3 時發生任務執行失敗

如果任務執行傳回錯誤 (像是無法使用處理常式類別剖析 XML 文件)，可能是因為使用 `s3a://` URI 嘗試列出數百個檔案時發生失敗。使用 `s3://` URI 存取資料存放區。以下例外狀況的追蹤會反白錯誤以尋找：

```
1. com.amazonaws.SdkClientException: Failed to parse XML document with handler class
 com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser$ListBucketHandler
```

```
2. at
 com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser.parseXmlInputStream(XmlRespons
3. at
 com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser.parseListBucketObjectsResponse
4. at com.amazonaws.services.s3.model.transform.Unmarshallers
$ListObjectsUnmarshaller.unmarshall(Unmarshallers.java:70)
5. at com.amazonaws.services.s3.model.transform.Unmarshallers
$ListObjectsUnmarshaller.unmarshall(Unmarshallers.java:59)
6. at
 com.amazonaws.services.s3.internal.S3XmlResponseHandler.handle(S3XmlResponseHandler.java:62)
7. at
 com.amazonaws.services.s3.internal.S3XmlResponseHandler.handle(S3XmlResponseHandler.java:31)
8. at
 com.amazonaws.http.response.AwsResponseHandlerAdapter.handle(AwsResponseHandlerAdapter.java:70
9. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.handleResponse(AmazonHttpClient.java:1554)
10. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.executeOneRequest(AmazonHttpClient.java:1272)
11. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.executeHelper(AmazonHttpClient.java:1056)
12. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.doExecute(AmazonHttpClient.java:743)
13. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.executeWithTimer(AmazonHttpClient.java:717)
14. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.execute(AmazonHttpClient.java:699)
15. at com.amazonaws.http.AmazonHttpClient$RequestExecutor.access
$500(AmazonHttpClient.java:667)
16. at com.amazonaws.http.AmazonHttpClient
$RequestExecutionBuilderImpl.execute(AmazonHttpClient.java:649)
17. at com.amazonaws.http.AmazonHttpClient.execute(AmazonHttpClient.java:513)
18. at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4325)
19. at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4272)
20. at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4266)
21. at com.amazonaws.services.s3.AmazonS3Client.listObjects(AmazonS3Client.java:834)
22. at org.apache.hadoop.fs.s3a.S3AFileSystem.getFileStatus(S3AFileSystem.java:971)
23. at
 org.apache.hadoop.fs.s3a.S3AFileSystem.deleteUnnecessaryFakeDirectories(S3AFileSystem.java:115
24. at org.apache.hadoop.fs.s3a.S3AFileSystem.finishedWrite(S3AFileSystem.java:1144)
25. at org.apache.hadoop.fs.s3a.S3AOutputStream.close(S3AOutputStream.java:142)
26. at org.apache.hadoop.fs.FSDataOutputStream
$PositionCache.close(FSDataOutputStream.java:74)
27. at org.apache.hadoop.fs.FSDataOutputStream.close(FSDataOutputStream.java:108)
28. at org.apache.parquet.hadoop.ParquetFileWriter.end(ParquetFileWriter.java:467)
```

```
29. at
 org.apache.parquet.hadoop.InternalParquetRecordWriter.close(InternalParquetRecordWriter.java:1
30. at
 org.apache.parquet.hadoop.ParquetRecordWriter.close(ParquetRecordWriter.java:112)
31. at
 org.apache.spark.sql.execution.datasources.parquet.ParquetOutputWriter.close(ParquetOutputWrit
32. at org.apache.spark.sql.execution.datasources.FileFormatWriter
$SingleDirectoryWriteTask.releaseResources(FileFormatWriter.scala:252)
33. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun
$org$apache$spark$sql$execution$datasources$FileFormatWriter$$executeTask
$3.apply(FileFormatWriter.scala:191)
34. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun
$org$apache$spark$sql$execution$datasources$FileFormatWriter$$executeTask
$3.apply(FileFormatWriter.scala:188)
35. at org.apache.spark.util.Utils
$.tryWithSafeFinallyAndFailureCallbacks(Utils.scala:1341)
36. at org.apache.spark.sql.execution.datasources.FileFormatWriter$.org$apache$spark
$sql$execution$datasources$FileFormatWriter$$executeTask(FileFormatWriter.scala:193)
37. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun$write$1$
$anonfun$3.apply(FileFormatWriter.scala:129)
38. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun$write$1$
$anonfun$3.apply(FileFormatWriter.scala:128)
39. at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:87)
40. at org.apache.spark.scheduler.Task.run(Task.scala:99)
41. at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:282)
42. at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
43. at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
44. at java.lang.Thread.run(Thread.java:748)
```

錯誤：Amazon S3 服務符記已過期

在移動 Amazon Redshift 的資料時，所使用的臨時 Amazon S3 憑證將於 1 小時之後過期。如果您有長時間執行的任務，則可能會失敗。如需如何設定長時間執行任務以便移動 Amazon Redshift 資料的詳細資訊，請參閱aws-glue-programming-etl-connect-redshift-home。

錯誤：找不到適用於網路介面的私有 DNS

如果任務失敗或開發端點無法佈建，可能是因為網路設定有問題。

- 如果您使用的是 Amazon 提供的 DNS，`enableDnsHostnames` 的值必須設為 true。如需更多詳細資訊，請參閱 DNS。

錯誤：開發端點佈建失敗

如果 AWS Glue 無法成功佈建開發端點，可能是因為網路設定中的問題。

- 定義開發端點時，VPC、子網路和安全群組會經過驗證，以確認其符合特定需求。
- 如果您提供可選的 SSH 公有金鑰，請確定其是有效的 SSH 公有金鑰。
- 在 VPC 主控台中，檢查您的 VPC 是否使用有效的 DHCP 選項集。如需詳細資訊，請參閱 DHCP 選項集。
- 如果叢集仍為 PROVISIONING 狀態，請聯絡 AWS Support。

錯誤：筆記本伺服器 CREATE_FAILED

如果 AWS Glue 無法為開發端點建立筆記本伺服器，可能是因為下列其中一個問題：

- AWS Glue 在設定筆記型電腦伺服器時，將 IAM 角色傳遞給 Amazon EC2。IAM 角色必須與 Amazon EC2 有信任關係。
- IAM 角色必須有相同名稱的執行個體描述檔。當您在 IAM 主控台建立 Amazon EC2 的角色時，便會自動建立相同名稱的執行個體描述檔。檢查日誌內是否有關於執行個體描述檔名稱 `iamInstanceProfile.name` 無效的錯誤。有關詳情，請參閱使用執行個體設定檔。
- 確認您的角色有許可存取傳送要建立筆記本伺服器之政策內的 `aws-glue*` 儲存貯體。

錯誤：本機筆記本無法啟動

如果您的本機筆記本無法啟動且回報找不到目錄或資料夾的錯誤，可能是因為下列其中一個問題：

- 如果您在 Microsoft Windows 上執行，請確定 `JAVA_HOME` 環境變數指向正確的 Java 目錄。無需更新此變數即可更新 Java，如果指向已不存在的資料夾，Jupyter 筆記本將無法啟動。

錯誤：爬蟲程式執行失敗

如果 AWS Glue 無法順利執行爬行者程式來編目您的資料，可能是因為下列其中一個原因。先檢查看看 AWS Glue 主控台爬蟲程式清單中是否出現錯誤。查看爬蟲程式名稱旁是否有驚嘆號 圖示，並將滑鼠移到圖示上方，查看任何相關的訊息。

- 檢查「記錄檔」`/aws-glue/crawlers` 下執行之爬行者程式的 CloudWatch 記錄檔。

錯誤：未更新分割區

如果您執行 ETL 工作時，資料目錄中的磁碟分割並未更新， CloudWatch 記錄檔中DataSink類別的這些記錄陳述式可能會有所幫助：

- "Attempting to fast-forward updates to the Catalog - nameSpace:" — 顯示此任務嘗試修改的資料庫、資料表和 catalogId。如果這裡沒有此陳述式，請檢查 enableUpdateCatalog 是否設為 true，並做為 getSink() 參數或在 additional_options 中正確傳遞。
- "Schema change policy behavior:" — 顯示您要傳入的結構描述 updateBehavior 值。
- "Schemas qualify (schema compare):" — 將會是 true 或 false。
- "Schemas qualify (case-insensitive compare):" — 將會是 true 或 false。
- 如果兩者皆為 false 且未設定為 UPDATE_IN_DATABASE，則您的 DynamicFrame 結構描述必須相同，或包含在「資料目錄」表格結構描述中看到的欄子集。updateBehavior

如需有關更新分割區的詳細資訊，請參閱 使用 AWS Glue ETL 工作更新結構描述，並在「資料目錄」中新增分割區。

錯誤：由於版本不相符，任務書籤更新失敗

您可能正在嘗試參數化任 AWS Glue 務，以便在 Amazon S3 的不同資料集上套用相同的轉換/邏輯。您希望在提供的位置上追蹤已處理的檔案。當您在同一個來源儲存貯體上執行同一個任務並同時寫入相同/不同的目的地 (並行數 > 1) 時，任務會失敗並出現以下錯誤：

```
py4j.protocol.Py4JJavaError: An error occurred while
 callingz:com.amazonaws.services.glue.util.Job.commit.:com.amazonaws.services.gluejobexecutor.m
 Continuation update failed due to version mismatch. Expected version 2 but found
 version 3
```

解決方案：將並行設定為 1 或不並行執行任務。

目前 AWS Glue 書籤不支援並行工作執行，且提交將會失敗。

錯誤：當任務書籤啟用時，任務會重新處理資料

在某些情況下，您可能已啟用 AWS Glue 工作書籤，但 ETL 工作正在重新處理先前執行中已處理的資料。請檢查此錯誤的常見原因：

最大並行數量

將工作的同時執行數目上限設定為大於預設值 1 可能會干擾工作書籤。當工作書籤檢查物件的上次修改時間以確認需要重新處理哪些物件時，就會發生這種情況。如需詳細資訊，請參閱設定 Spark 工作的工作屬性 AWS Glue中針對最大並行數量的說明。

找不到任務物件

確認您的任務執行指令碼結尾是否為下列遞交：

```
job.commit()
```

當您包含此物件時，會 AWS Glue 記錄工作執行的時間戳記和路徑。如果您使用相同的路徑再次執行工作，則只會 AWS Glue 處理新檔案。如果您未包含此物件並啟用任務書籤，任務就會重新處理已處理過的檔案與新的檔案，並在任務的目標資料存放區中建立冗餘項目。

缺少轉換內容參數

轉換細節是 `GlueContext` 類別中的選擇性參數，但如果您未包含此參數，任務書籤就無法運作。若要解決此錯誤，請在建立時新增轉換前後關聯參數 DynamicFrame，如下所示：

```
sample_dynF=create_dynamic_frame_from_catalog(database,
 table_name,transformation_ctx="sample_dynF")
```

輸入來源

如果您使用關聯式資料庫 (JDBC 連線) 做為輸入來源，則只有在資料表的主索引鍵為循序排列時任務書籤才能運作。任務書籤適用於新的資料列，但對更新的資料列無效。這是因為任務書籤會尋找主索引鍵，而主索引鍵已存在。如果您的輸入來源是 Amazon Simple Storage Service (Amazon S3)，則此選項不適用。

上次修改時間

若是 Amazon S3 輸入來源，任務書籤會檢查物件的上次修改時間而不是檔案名稱，以確認哪些物件需要重新處理。如果自上次任務執行之後已修改您的輸入來源資料，則在您重新執行任務時重新處理檔案。

錯誤：中 VPC 之間的容錯移轉行為 AWS Glue

以下過程用於 AWS Glue 4.0 和以前版本中的作業的故障轉移。

摘要：送出工作執行時會選取 AWS Glue 連線。如果任務執行時遇到一些問題 (缺少 IP 地址、連線至來源、路由問題)，任務執行將會失敗。如果已設定重試，則 AWS Glue 會以相同的連線重試。

1. 對於每次運行嘗試， AWS Glue 將按照作業配置中列出的順序檢查連接健康狀況，直到找到可以使用的一個為止。如果可用區域 (AZ) 失敗，來自該 AZ 的連線將會失敗檢查，並會略過。
2. AWS Glue 使用下列項目驗證連線：
   - 檢查是否具備有效的 Amazon VPC ID 和子網路。
   - 檢查 NAT 閘道或 Amazon VPC 端點是否存在。
   - 檢查子網路是否具備超過 0 個所分配的 IP 地址。
   - 檢查 AZ 是否狀態良好。

   AWS Glue 無法在工作執行提交時驗證連線。
3. 對於使用 Amazon VPC 的任務，所有驅動程式和執行器都會在同一個 AZ 中建立，並在任務執行提交時選取連線。
4. 如果已設定重試，則 AWS Glue 會以相同的連線重試。這是因為我們無法保證此連線在長期運作下不會發生問題。若 AZ 失敗，則該 AZ 中現有的任務執行 (視任務執行的階段而定) 可能會失敗。重試應會偵測 AZ 故障，並為新的執行選擇其他 AZ。

疑難排解爬蟲程式使用 Lake Formation 憑證時發生的錯誤

參考下列資訊診斷及修正各種問題，同時使用 Lake Formation 憑證設定爬蟲程式。

錯誤：S3 位置 (s3://examplepath) 尚未註冊

若要使用 Lake Formation 憑證執行爬蟲程式，您需要先設定 Lake Formation 許可。若要解決此錯誤，請向 Lake Formation 註冊目標 Amazon S3 位置。如需詳細資訊，請參閱 Registering an Amazon S3 location (註冊 Amazon S3 位置)。

錯誤：使用者/角色未經授權，無法在資源上執行 lakeformation:GetDataAccess

請使用 IAM 主控台或 AWS CLI，將 `lakeformation:GetDataAccess` 許可新增至爬蟲程式。有了此許可，Lake Formation 就會授與要求存取資料所需的臨時憑證。請參閱以下政策：

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

```
  }
}
```

錯誤：(資料庫名稱：exampleDatabase，資料表名稱：exampleTable) 上的 Lake Formation 許可不足

在 Lake Formation 主控台 (https://console.aws.amazon.com/lakeformation/) 授與爬蟲程式對資料庫 (指定作為輸出資料庫) 的角色存取許可 (`Create`、`Describe`、`Alter`)。您也可以授與資料表的使用許可。如需詳細資訊，請參閱 Granting database permissions using the named resource method (使用具名資源方法授與資料庫許可)。

錯誤：s3://examplepath 上的 Lake Formation 許可不足

1. 跨帳戶網路爬取
   a. 使用 Amazon S3 儲存貯體所註冊的帳戶資料 (帳戶 B)，登入 Lake Formation 主控台 (https://console.aws.amazon.com/lakeformation/)。將資料位置許可授與要執行爬蟲程式的帳戶。如此一來，爬蟲程式就能從目標 Amazon S3 位置讀取資料。
   b. 在建立爬蟲程式的帳戶 (帳戶 A) 中，將目標 Amazon S3 位置的資料位置許可授與爬蟲程式執行時所使用的 IAM 角色，以便爬蟲程式可以從 Lake Formation 的目的地讀取資料。如需詳細資訊，請參閱 Granting data location permissions (external account) (授與資料位置許可 (外部帳戶))。
2. 帳戶內 (爬蟲程式和所註冊的 Amazon S3 位置位於相同帳戶) 網路爬取 - 將資料位置許可授與爬蟲程式於 Amazon S3 位置執行時所使用的 IAM 角色，以便爬蟲程式可以從 Lake Formation 的目標讀取資料。如需詳細資訊，請參閱 Granting data location permissions (same account) (授與資料位置許可 (相同帳戶))。

使用 Lake Formation 憑證設定爬蟲程式的常見問題

1. 如何在 AWS 主控台使用 Lake Formation 憑證設定爬蟲程式，使其能順利執行？

   在 AWS Glue 主控台 (https://console.aws.amazon.com/glue/) 中，在設定爬蟲程式時，選取選項 Use Lake Formation credentials for crawling Amazon S3 data source (使用 Lake Formation 憑證對 Amazon S3 資料來源進行網路爬取)。若要進行跨帳戶網路爬取，請指定向 Lake Formation 註冊目標 Amazon S3 位置的 AWS 帳戶 ID。對於帳戶內網路爬取，accountId 為選填欄位。
2. 如何在 AWS CLI 使用 Lake Formation 憑證設定爬蟲程式，使其能順利執行？

   在 `CreateCrawler` API 呼叫期間，新增 `LakeFormationConfiguration`：

```
"LakeFormationConfiguration": {
    "UseLakeFormationCredentials": true,
    "AccountId": "111111111111" (AWS account ID where the target Amazon S3 location
 is registered with Lake Formation)
  }
```

3. 使用 Lake Formation 憑證的情況下，爬蟲程式支援哪些目標？

   使用 Lake Formation 憑證的爬蟲程式僅支援 Amazon S3 (帳戶內和跨帳戶網路爬取)、帳戶內資料型錄目標 (其中的基礎位置為 Amazon S3) 以及 Apache Iceberg 目標。

4. 可以使用 Lake Formation 憑證，以單一爬蟲程式對多個 Amazon S3 儲存貯體執行網路爬取作業嗎？

   不可以。如果使用 Lake Formation 憑證販售對目標執行網路爬取，基礎 Amazon S3 位置必須屬於同一個儲存貯體。舉例來說，客戶可以使用多個目標位置 (`s3://bucket1/folder1, s3://bucket1/folder2`)，但這些位置必須位於同一個儲存貯體 (儲存貯體 1)，不允許指定不同的儲存貯體 (s3://bucket1/folder1、s3://bucket2/folder2)。

對日誌中的 AWS Glue for Ray 錯誤進行疑難排解

AWS Glue 可讓您存取 Ray 程序在任務執行期間所發出的日誌。如果您在 Ray 任務中遇到錯誤或非預期的行為，請先從日誌中收集資訊，用以判斷失敗的原因。我們還為互動式工作階段提供類似的日誌。工作階段日誌帶有 `/aws-glue/ray/sessions` 字首。

當您的任務執行時，日誌行會即時傳送至 CloudWatch。列印陳述式會在執行完成後附加至 CloudWatch 日誌中。任務執行後，日誌會保留兩週。

檢查 Ray 任務日誌

如果任務失敗，請收集任務名稱和任務執行 ID。您可以在 AWS Glue 主控台中找到這些資訊。瀏覽至任務頁面，然後找到 Runs (執行) 索引標籤。Ray 任務日誌會存放在下列專用的 CloudWatch 日誌群組中。

- `/aws-glue/ray/jobs/script-log/` – 存放 Ray 主指令碼發出的日誌。
- `/aws-glue/ray/jobs/ray-monitor-log/` – 存放 Ray 自動縮放器程序發出的日誌。這些日誌是針對前端節點產生，而不是針對其他工作節點產生。
- `/aws-glue/ray/jobs/ray-gcs-logs/`：存放由 GCS (全域控制存放) 程序發出的日誌。這些日誌是針對前端節點產生，而不是針對其他工作節點產生。

- `/aws-glue/ray/jobs/ray-process-logs/`：存放在前端節點上執行的其他 Ray 程序 (主要是儀表板代理) 發出的日誌。這些日誌是針對前端節點產生，而不是針對其他工作節點產生。
- `/aws-glue/ray/jobs/ray-raylet-logs/`：存放每個 raylet 程序發出的日誌。系統會將這些日誌收集在每個工作節點的單一串流中，包括前端節點。
- `/aws-glue/ray/jobs/ray-worker-out-logs/`：將每個工作者的 `stdout` 日誌存放在叢集中。這些日誌會針對每個工作節點 (包括前端節點) 產生。
- `/aws-glue/ray/jobs/ray-worker-err-logs/`：將每個工作者的 `stderr` 日誌存放在叢集中。這些日誌會針對每個工作節點 (包括前端節點) 產生。
- `/aws-glue/ray/jobs/ray-runtime-env-log/`：存放有關 Ray 設定程序的日誌。這些日誌會針對每個工作節點 (包括前端節點) 產生。

對 Ray 任務錯誤進行故障診斷

若要了解 Ray 日誌群組的組織，並尋找可協助您針對錯誤進行故障診斷的日誌群組，取得 Ray 架構的背景資訊會很有幫助。

在 AWS Glue ETL 中，工作者會與執行個體相對應。在為 AWS Glue 任務設定工作者時，您可以設定任務專用的執行個體類型和數目。Ray 對 worker (工作、工作者) 一詞的使用方式有所不同。

Ray 使用前端節點和工作節點來區分 Ray 叢集中執行個體的責任。Ray 工作節點可以託管多個執行者程序，這些程序會執行計算以實現分佈式計算的結果。執行函數複本的執行者稱為複本。複本執行者也可以稱為工作者程序。複本也可以在前端節點 (稱為前端) 上執行，因為該節點會執行額外的程序來協調叢集。

參與計算的每個執行者會產生自己的日誌串流。這為我們提供了一些見解：

- 發出日誌的程序數目可能大於配置給任務的工作者數目。通常，每個執行個體上的每個核心都有一個執行者。
- Ray 前端節點會發出叢集管理和啟動日誌。相對地，Ray 工作節點只會針對在其上執行的工作發出日誌。

如需有關 Ray 架構的詳細資訊，請參閱 Ray 文件中的 Architecture Whitepapers (架構白皮書)。

問題領域：Amazon S3 存取

檢查任務執行的失敗訊息。如果提供的資訊不足，請查閱 `/aws-glue/ray/jobs/script-log/`。

問題領域：PIP 相依性管理

檢查 `/aws-glue/ray/jobs/ray-runtime-env-log/`。

問題領域：檢查主程序中的中繼值

從主指令碼中寫入 `stderr` 或 `stdout`，並從 `/aws-glue/ray/jobs/script-log/` 擷取日誌。

問題領域：檢查子程序中的中繼值

從 `remote` 函數寫入 `stderr` 或 `stdout`。然後，從 `/aws-glue/ray/jobs/ray-worker-out-logs/` 或 `/aws-glue/ray/jobs/ray-worker-err-logs/` 擷取日誌。函數可能已在任何複本上執行，因此您可能必須檢查多個日誌才能找到預期的輸出。

問題領域：解譯錯誤訊息中的 IP 地址

在某些錯誤情況下，您的任務可能會發出包含 IP 地址的錯誤訊息。這些 IP 地址是暫時的資訊，叢集會用於識別節點並在節點之間進行通訊。節點的日誌會發布至日誌串流，其具有以 IP 地址為基礎的唯一字尾。

在 CloudWatch 中，您可以透過識別此字尾來篩選日誌，以檢查特定於此 IP 地址的日誌。例如，假設為 *FAILED_IP* 和 *JOB_RUN_ID*，您可以透過以下方式識別字尾：

```
filter @logStream like /JOB_RUN_ID/
| filter @message like /IP-/
| parse @message "IP-[*]" as ip
| filter ip like /FAILED_IP/
| fields replace(ip, ":", "_") as uIP
| stats count_distinct by uIP as logStreamSuffix
| display logStreamSuffix
```

AWS Glue 機器學習例外狀況

本主題說明與機器學習相關之 AWS Glue 例外狀況的 HTTP 錯誤碼和字串。針對執行操作時可能發生的每個機器學習活動提供了錯誤碼和錯誤字串。此外，您也可以查看是否可能重試導致錯誤的操作。

CancelMLTaskRunActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)

- 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」
- 「找不到 [taskRunId] 的 ML 任務執行：在帳戶 [accountId] 中，針對轉換 [transformName]。」

可以重試：否。

CreateMLTaskRunActivity

此活動有下列例外狀況：

- InvalidInputException (400)
  - 「由於未預期的輸入導致內部服務故障。」
  - 「轉換中應該指定 AWS Glue 資料表輸入來源。」
  - 「在目錄中為輸入來源資料行 [columnName] 定義了無效的資料類型。」
  - 「只能提供一個輸入記錄資料表。」
  - 「應該指定資料庫名稱。」
  - 「應該指定資料表名稱。」
  - 「未在轉換上定義結構描述。」
  - 「結構描述應該包含指定的主索引鍵：[primaryKey]。」
  - 「擷取資料目錄結構描述時發生問題：[message]。」
  - 「無法同時設定「最大容量」和「工作者數目/類型」。」
  - 「WorkerType 和 NumberOfWorkers 兩者都應該設定。」
  - 「MaxCapacity 應該是 >= [maxCapacity]。」
  - 「NumberOfWorkers 應該是 >= [maxCapacity]。」
  - 「重試次數上限應該是非負數。」
  - 「尚未設定「尋找符合項目」參數。」
  - 「「尋找符合項目」參數中必須指定主索引鍵。」

  可以重試：否。
- AlreadyExistsException (400)
  - 「名稱為 [transformName] 的轉換已存在。」

  可以重試：否。
- IdempotentParameterMismatchException (400)

- 「轉換 [transformName] 的等冪建立請求具有不相符的參數。」

可以重試：否。

- InternalServiceException (500)
  - 「相依性失敗。」

可以重試：可。

- ResourceNumberLimitExceededException (400)
  - 「ML 轉換計數 ([count]) 已超過 [limit] 個轉換的限制。」

可以重試：可。刪除轉換，為這個新的轉換騰出空間即可。

DeleteMLTransformActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform」

可以重試：否。

GetMLTaskRunActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」
  - 「找不到 [taskRunId] 的 ML 任務執行：在帳戶 [accountId] 中，針對轉換 [transformName]。」

可以重試：否。

GetMLTaskRunsActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」

- 「找不到 [taskRunId] 的 ML 任務執行：在帳戶 [accountId] 中，針對轉換 [transformName]。」

可以重試：否。

GetMLTransformActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」

可以重試：否。

GetMLTransformsActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」

可以重試：否。

- InvalidInputException (400)
  - 「帳戶 ID 不可空白。」
  - 「資料行 [column] 不支援排序。」
  - 「[column] 不可空白。」
  - 「由於未預期的輸入導致內部服務故障。」

可以重試：否。

GetSaveLocationForTransformArtifactActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」

可以重試：否。

- InvalidInputException (400)
  - 「不支援的成品類型 [artifactType]。」
  - 「由於未預期的輸入導致內部服務故障。」

  可以重試：否。

GetTaskRunArtifactActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」
  - 「找不到 [taskRunId] 的 ML 任務執行：在帳戶 [accountId] 中，針對轉換 [transformName]。」

  可以重試：否。
- InvalidInputException (400)
  - 「用於發佈的檔案名稱 '[fileName]' 無效。」
  - 「無法擷取 [taskType] 任務類型的成品。」
  - 「無法擷取 [artifactType] 的成品。」
  - 「由於未預期的輸入導致內部服務故障。」

  可以重試：否。

PublishMLTransformModelActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」
  - 「找不到版本 [version] 的現有模型：帳戶 ID - [accountId] 和轉換 ID - [transformId]。」

  可以重試：否。
- InvalidInputException (400)
  - 「用於發佈的檔案名稱 '[fileName]' 無效。」
  - 「不帶正負號字串 [string] 有無效的前導負號。」

- 「[string] 結尾的數字錯誤。」
- 「字串值 [string] 超過不帶正負號長整數的範圍。」
- 「由於未預期的輸入導致內部服務故障。」

可以重試：否。

PullLatestMLTransformModelActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」

  可以重試：否。
- InvalidInputException (400)
  - 「由於未預期的輸入導致內部服務故障。」

  可以重試：否。
- ConcurrentModificationException (400)
  - 「由於使用不相符的參數競爭插入，導致無法建立要培訓的模型版本。」
  - 「轉換 ID [transformId] 的 ML 轉換模型已過時或正由另一個處理程序更新中；請重試。」

  可以重試：可。

PutJobMetadataForMLTransformActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」
  - 「找不到 [taskRunId] 的 ML 任務執行：在帳戶 [accountId] 中，針對轉換 [transformName]。」

  可以重試：否。
- InvalidInputException (400)
  - 「由於未預期的輸入導致內部服務故障。」
  - 「未知的任務中繼資料類型 [jobType]。」

- 「必須提供任務執行 ID 才能更新。」

可以重試：否。

StartExportLabelsTaskRunActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」
  - 「在帳戶 ID [accountId] 中沒有 transformId [transformId] 的 labelset。」

  可以重試：否。
- InvalidInputException (400)
  - 「[message]。」
  - 「所提供的 S3 路徑與轉換不在相同區域中。預期區域 - [region]，但得到 - [region]。」

  可以重試：否。

StartImportLabelsTaskRunActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」

  可以重試：否。
- InvalidInputException (400)
  - 「[message]。」
  - 「無效的標籤檔案路徑。」
  - 「無法存取位於 [labelPath] 的標籤檔案。[message]。」
  - 「無法使用轉換中提供的 IAM 角色。角色：[role]。」
  - 「標籤檔案大小 0 無效。」
  - 「所提供的 S3 路徑與轉換不在相同區域中。預期區域 - [region]，但得到 - [region]。」

  可以重試：否。

- ResourceNumberLimitExceededException (400)
  - 「標籤檔案已超過 [limit] MB 的限制。」

  可以重試：否。考慮將您的標籤檔案分成幾個較小的檔案。

StartMLEvaluationTaskRunActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」

  可以重試：否。
- InvalidInputException (400)
  - 「只能提供一個輸入記錄資料表。」
  - 「應該指定資料庫名稱。」
  - 「應該指定資料表名稱。」
  - 「尚未設定「尋找符合項目」參數。」
  - 「「尋找符合項目」參數中必須指定主索引鍵。」

  可以重試：否。
- MLTransformNotReadyException (400)
  - 「此操作只適用於處於「就緒」狀態的轉換。」

  可以重試：否。
- InternalServiceException (500)
  - 「相依性失敗。」

  可以重試：可。
- ConcurrentRunsExceededException (400)
  - 「ML 任務執行計數 [count] 已超過 [limit] 個任務執行的轉換限制。」
  - 「ML 任務執行計數 [count] 已超過 [limit] 個任務執行的限制。」

  可以重試：可。等待任務執行完成後。

StartMLLabelingSetGenerationTaskRunActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」

  可以重試：否。
- InvalidInputException (400)
  - 「只能提供一個輸入記錄資料表。」
  - 「應該指定資料庫名稱。」
  - 「應該指定資料表名稱。」
  - 「尚未設定「尋找符合項目」參數。」
  - 「「尋找符合項目」參數中必須指定主索引鍵。」

  可以重試：否。
- InternalServiceException (500)
  - 「相依性失敗。」

  可以重試：可。
- ConcurrentRunsExceededException (400)
  - 「ML 任務執行計數 [count] 已超過 [limit] 個任務執行的轉換限制。」

  可以重試：可。在任務執行完成後。

UpdateMLTransformActivity

此活動有下列例外狀況：

- EntityNotFoundException (400)
  - 「在帳戶 [accountId] 中找不到具有控制代碼 [transformName] 的 MLTransform。」

  可以重試：否。
- InvalidInputException (400)
  - 「另一個名稱為 [transformName] 的轉換已存在。」
  - 「[message]。」

- 「轉換名稱不可空白。」
- 「無法同時設定「最大容量」和「工作者數目/類型」。」
- 「WorkerType 和 NumberOfWorkers 兩者都應該設定。」
- 「MaxCapacity 應該是 >= [minMaxCapacity]。」
- 「NumberOfWorkers 應該是 >= [minNumWorkers]。」
- 「重試次數上限應該是非負數。」
- 「由於未預期的輸入導致內部服務故障。」
- 「尚未設定「尋找符合項目」參數。」
- 「「尋找符合項目」參數中必須指定主索引鍵。」

可以重試：否。

- AlreadyExistsException (400)
  - 「名稱為 [transformName] 的轉換已存在。」

  可以重試：否。
- IdempotentParameterMismatchException (400)
  - 「轉換 [transformName] 的等冪建立請求具有不相符的參數。」

  可以重試：否。

AWS Glue 配額

您可以針對 AWS 一般參考 中列出的服務配額，聯絡 AWS Support 以請求增加配額。除非另有說明，否則每個配額都是區域特定規定。如需詳細資訊，請參閱 AWS Glue 端點和配額。

改善 AWS Glue 效能

效能調校的基線策略

為了提高 AWS Glue 性能，您可以考慮更新某些與性能相關的 AWS Glue 參數。準備調校參數時，請使用以下最佳實務：

- 在開始識別問題之前，先決定您的效能目標。
- 嘗試變更調校參數之前，先使用指標來識別問題。

為在調校任務時得到最一致的結果，應制定調校工作的基線策略。

一般而言，效能調校依照以下工作流程進行：

1. 決定效能目標。
2. 量測指標。
3. 識別瓶頸。
4. 降低瓶頸的影響。
5. 重複步驟 2-4，直到達到預期的目標。

調整工作類型的策略

Spark 工作 — 請遵循《 AWS 規範指引》中 Apache Spark 工作效能調整 AWS Glue 的最佳實務指引中的指引。

其他工作 — 您可以調整 AWS Glue 其他執行階段環境中可用的策略來調整 Ray 和 AWS Glue Python 殼層工作。

改善 AWS Glue for Apache Spark 任務的效能

為改善 AWS Glue for Spark 效能，您可以考慮更新與 AWS Glue 和 Spark 參數有關的某些效能。

如需有關透過指標識別瓶頸並降低其影響的具體策略的詳細資訊，請參閱《AWS 規定指引》中的 AWS Glue for Apache Spark 任務效能調校的最佳實務。本指南向您介紹在所有執行階段環境中適用於 Apache Spark 的關鍵主題，例如 Spark 架構和彈性分散式資料集。本指南透過這些主題，引導您實作特定的效能調整策略，例如最佳化隨機處理和平行化工作。

您可以通過配置AWS Glue來顯示 Spark UI 來識別瓶頸。如需詳細資訊，請參閱 the section called “使用 Spark UI 進行監控”。

此外，還AWS Glue提供效能功能，這些功能可能適用於工作所連接的特定資料倉庫類型。您可以在中找到有關資料倉庫效能參數的參考資訊the section called “連線參數”。

使用 AWS Glue ETL 中的下推來最佳化讀取

下推是一種最佳化技術，可將有關擷取資料的邏輯推送到更接近資料來源的位置。來源可以是 Amazon S3 之類的資料庫或檔案系統。直接在來源上執行某些操作時，您可以不透過網路，將所有資料傳送至 AWS Glue 管理的 Spark 引擎，以節省時間和提高處理能力。

這個的另一種說法是下推減少了資料掃描。如需有關識別此技術何時適用的程序的詳細資訊，請參閱《AWS 規定指南》中 AWS Glue for Apache Spark 任務效能調校最佳實務指引中的減少資料掃描數量。

對 Amazon S3 上存放的檔案進行述詞下推

在 Amazon S3 上使用依字首整理的檔案時，您可以透過定義下推述詞，篩選目標 Amazon S3 路徑。您可以直接將篩選條件套用至 AWS Glue Data Catalog 中存放的分割區中繼資料，而不是在 `DynamicFrame` 中讀取完整的資料集並套用篩選條件。這種方法可讓您選擇性地僅列出和讀取必要的資料。如需有關此程序的詳細資訊，包括依分割區寫入儲存貯體，請參閱 the section called “管理分割區”。

您可以使用 `push_down_predicate` 參數在 Amazon S3 中達成述詞下推。考慮您在 Amazon S3 中依年份、月份和日期分割的儲存貯體。如果您想擷取 2022 年 6 月的客戶資料，可以指示 AWS Glue 僅讀取相關的 Amazon S3 路徑。在本案例中 `push_down_predicate` 為 `year='2022' and month='06'`。綜合來說，您可以依如下方式實現讀取操作：

Python

```
customer_records = glueContext.create_dynamic_frame.from_catalog(
    database = "customer_db",
    table_name = "customer_tbl",
    push_down_predicate = "year='2022' and month='06'"
)
```

Scala

```
val customer_records = glueContext.getCatalogSource(
```

```
database="customer_db",
tableName="customer_tbl",
pushDownPredicate="year='2022' and month='06'"
).getDynamicFrame()
```

在先前的案例中，push_down_predicate 會從 AWS Glue Data Catalog 擷取所有分割區的清單，並在讀取基礎 Amazon S3 檔案之前進行篩選。雖然這在大多數情況下很有用，但在處理具有數百萬個分割區的資料集時，分割區的列出程序可能很耗時。為了解決這個問題，可以使用伺服器端的分割區剔除來改善效能。此操作透過在 AWS Glue Data Catalog 中為資料建立分割區索引來完成。如需有關分割區索引的詳細資訊，請參閱 the section called “使用分割區索引”。然後，您可以使用 catalogPartitionPredicate 選項參考索引。如需使用 catalogPartitionPredicate 擷取分割區的範例，請參閱 the section called “目錄分割述詞”。

使用 JDBC 來源時下推

GlueContext 中使用的 AWS Glue JDBC 讀取器透過提供可直接在來源上執行的自訂 SQL 查詢，協助在支援的資料庫上進行下推。可以透過設定 sampleQuery 參數來實現這一點。範例查詢可以指定要選取的資料欄，以及提供下推述詞來限制向 Spark 引擎傳輸的資料。

範例查詢依預設會在單一節點上運作，但處理大量資料時可能會導致任務失敗。若要使用此功能大規模查詢資料，您應該將 enablePartitioningForSampleQuery 設定為 true 來設定查詢分割區，如此會將查詢分散到所選索引鍵之間的多個節點。查詢分割區還需要一些其他必要的組態參數。如需有關查詢分割區的詳細資訊，請參閱 the section called “從 JDBC 中平行讀取”。

設定 enablePartitioningForSampleQuery 時，AWS Glue 會在查詢資料庫時將下推述詞與分割述詞相結合。sampleQuery 的結尾必須是 AND，以便 AWS Glue 附加分割條件。(如果您沒有提供下推述詞，則 sampleQuery 的結尾必須是 WHERE)。請參閱下面的範例，其中我們向下推送一個述詞，以僅擷取 id 大於 1000 的資料列。此 sampleQuery 僅會傳回其中 id 大於指定值的資料列名稱和位置資料欄：

Python

```
sample_query = "select name, location from customer_tbl WHERE id>=1000 AND"
customer_records = glueContext.create_dynamic_frame.from_catalog(
    database="customer_db",
    table_name="customer_tbl",
    sample_query = "select name, location from customer_tbl WHERE id>=1000 AND",

    additional_options = {
                            "hashpartitions": 36 ,
```

```
                            "hashfield":"id",
                            "enablePartitioningForSampleQuery":True,
                            "sampleQuery":sample_query
                           }
)
```

Scala

```
val additionalOptions = Map(
        "hashpartitions" -> "36",
        "hashfield" -> "id",
        "enablePartitioningForSampleQuery" -> "true",
        "sampleQuery" -> "select name, location from customer_tbl WHERE id >= 1000
 AND"
        )

    val customer_records = glueContext.getCatalogSource(
        database="customer_db",
        tableName="customer_tbl").getDynamicFrame()
```

> **Note**
>
> 如果customer_tbl資料目錄和基礎資料存放區中的名稱不同，您必須在 sample_query 中提供基礎資料表名稱，因為查詢會傳遞至基礎資料存放區。

您也可以根據 JDBC 資料表進行查詢，而不需與 AWS Glue Data Catalog 整合。您可以透過提供 useConnectionProperties 和 connectionName 來重複使用來自預先存在之連線的憑證，而不是將使用者名稱和密碼作為參數提供給該方法。在本範例中，我們從名為 my_postgre_connection 的連線擷取憑證。

Python

```
connection_options_dict = {
    "useConnectionProperties": True,
    "connectionName": "my_postgre_connection",
    "dbtable":"customer_tbl",
    "sampleQuery":"select name, location from customer_tbl WHERE id>=1000 AND",
    "enablePartitioningForSampleQuery":True,
```

```
    "hashfield":"id",
    "hashpartitions":36
    }

customer_records = glueContext.create_dynamic_frame.from_options(
    connection_type="postgresql",
    connection_options=connection_options_dict
    )
```

Scala

```
val connectionOptionsJson = """
      {
        "useConnectionProperties": true,
        "connectionName": "my_postgre_connection",
        "dbtable": "customer_tbl",
        "sampleQuery": "select name, location from customer_tbl WHERE id>=1000 AND",
        "enablePartitioningForSampleQuery" : true,
        "hashfield" : "id",
        "hashpartitions" : 36
      }
    """

    val connectionOptions = new JsonOptions(connectionOptionsJson)

    val dyf = glueContext.getSource("postgresql",
 connectionOptions).getDynamicFrame()
```

AWS Glue 中下推的注意事項和限制

就概念來說，下推在從非串流來源讀取時適用。AWSGlue 支援多種來源，下推的能力取決於來源和連接器。

- 連線到 Snowflake 時，您可以使用 `query` 選項。在 AWS Glue 4.0 及更新版本的 Redshift 連接器中存在類似功能。如需有關使用 `query` 從 Snowflake 讀取的詳細資訊，請參閱 the section called “從 Snowflake 讀取”。
- DynamoDB ETL 讀取器不支援篩選條件或下推述詞。MongoDB 和 DocumentDB 也不支援這種功能。
- 從以開放資料表格式存放在 Amazon S3 中的資料進行讀取時，Amazon S3 中檔案的分割方法已不再足夠。若要使用開放資料表格式從分割區讀取和寫入，請參閱該格式的文件。

- DynamicFrame 方法不會執行 Amazon S3 投影下推。將從通過述詞篩選條件的檔案讀取所有資料欄。
- 使用 AWS Glue 中的 `custom.jdbc` 連接器時，下推的能力取決於來源和連接器。請檢閱適當的連接器文件，以確認連接器是否以及如何支援 AWS Glue 中的下推。

使用面向 AWS Glue 的自動擴展

在 AWS Glue 3.0 版或更新版本中，Auto Scaling 功能適用於 AWS Glue ETL 和串流任務。

啟用 Auto Scaling 功能後，您可以獲得下列好處：

- AWS Glue 根據每個階段的平行處理或任務執行的微批次，向叢集自動新增和從叢集自動移除工作者。
- 藉助此方法，不再需要進行實驗並決定為您的 AWS Glue ETL 任務指派的工作者數量。
- 如果選擇工作者數量上限，AWS Glue 將為工作負載選擇正確規模的資源。
- 您可以查看 AWS Glue Studio 中工作執行詳細資料頁面上的 CloudWatch 指標，查看工作執行期間叢集大小的變更情況。

適用於 AWS Glue ETL 和串流任務的 Auto Scaling 實現 AWS Glue 任務運算資源的隨需擴充規模和縮減規模。隨需縱向擴展可協助您在任務執行最初啟動時僅分配所需的運算資源，以及根據任務期間的需求佈建所需的資源。

Auto Scaling 還支援在任務執行過程中動態縮減 AWS Glue 任務資源的規模。在任務執行期間，當您的 Spark 應用程式請求更多執行器時，可將更多的工作者新增至叢集。當執行器在沒有已啟用運算任務的情況下閒置時，就會刪除該執行器和相應的工作者。

Auto Scaling 可協助您降低 Spark 應用程式成本與使用率的常見案例包括 Spark 驅動程式列出 Amazon S3 中的大量檔案，或在執行器處於非作用中狀態時執行負載，Spark 階段因為過度佈建而僅使用少數執行器執行，以及 Spark 階段之間的資料偏差或運算需求不均勻。

要求

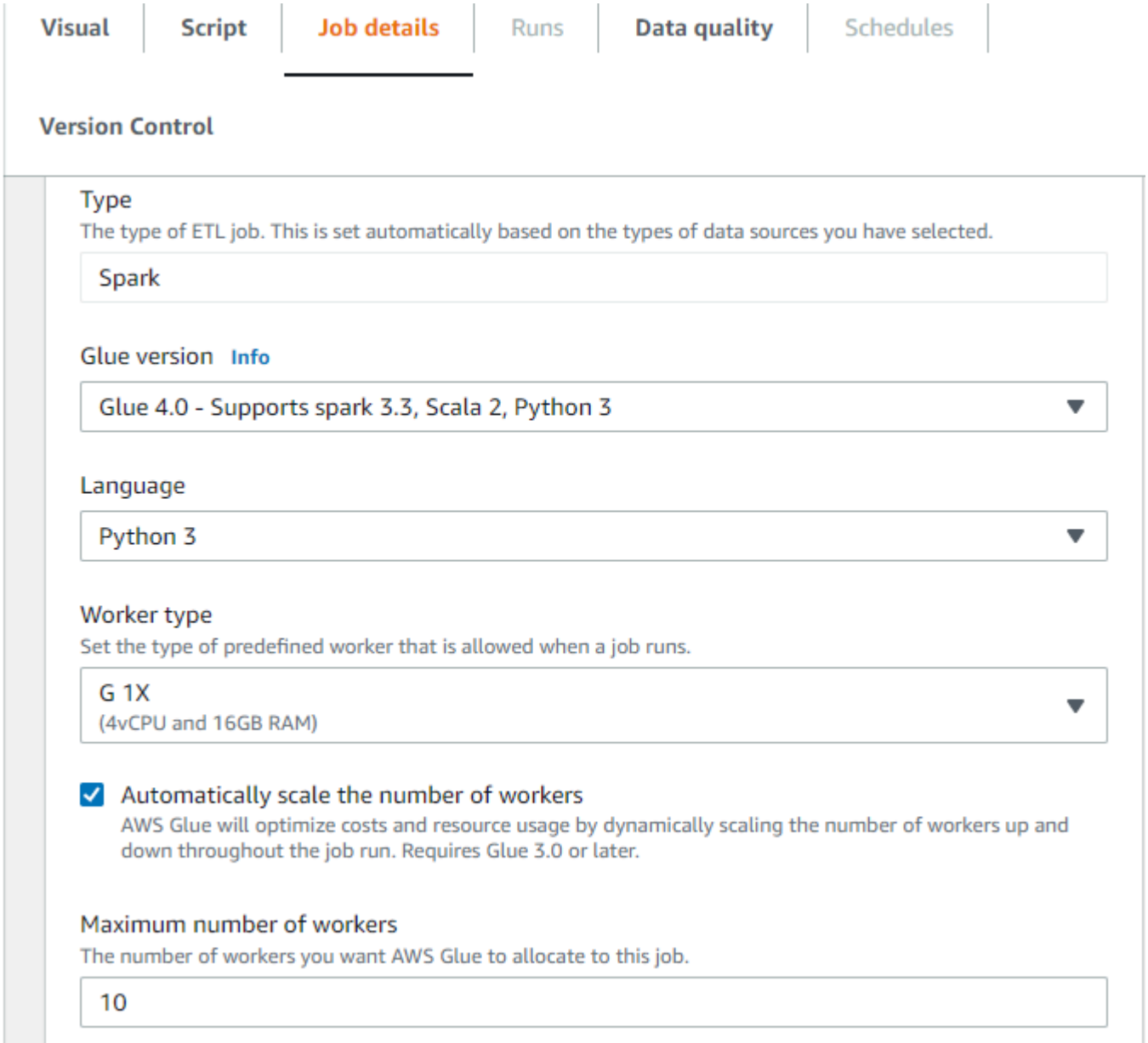
Auto Scaling 僅適用於 AWS Glue 3.0 版或更高版本。要使用 Auto Scaling，您可以遵循遷移指南，將現有的任務遷移至 AWS Glue 3.0 版或更高版本，或使用 AWS Glue 3.0 版或更高版本建立新任務。

自動擴展適用於具備 `G.1X`、`G.2X`、`G.4X`、`G.8X` 或 `G.025X` (僅限串流任務) 工作者類型的 AWS Glue 任務。不支援標準的 CPU。

在 AWS Glue Studio 中啟用 Auto Scaling

在 AWS Glue Studio 中的 Job details (任務詳細資訊) 索引標籤上，選擇 Spark 或 Spark Streaming 類型，並將 Glue version (Glue 版本) 選為 **`Glue 3.0`** 或 **`Glue 4.0`**。然後，一個核取方塊將顯示在 Worker type (工作者類型) 下方。

- 選取 Automatically scale the number of workers (自動擴展工作者數量) 選項。
- 設定 Maximum number of workers (工作者數上限) 以定義可提供給任務執行的工作者數上限。



透過 AWS CLI 或開發套件啟用 Auto Scaling

要從 AWS CLI 為任務執行啟用 Auto Scaling，請使用下列組態執行 `start-job-run`：

```
{
    "JobName": "<your job name>",
    "Arguments": {
        "--enable-auto-scaling": "true"
    },
    "WorkerType": "G.2X", // G.1X and G.2X are allowed for Auto Scaling Jobs
    "NumberOfWorkers": 20, // represents Maximum number of workers
    ...other job run configurations...
}
```

在 ETL 任務執行完成後，您也可以呼叫 `get-job-run` 以檢查任務執行的實際資源使用情況 (以 DPU 秒為單位)。注意：新欄位 DPUSeconds 只會在已啟用 Auto Scaling 功能的 AWS Glue 3.0 或更高版本上針對批任務顯示。此欄位不支援串流任務。

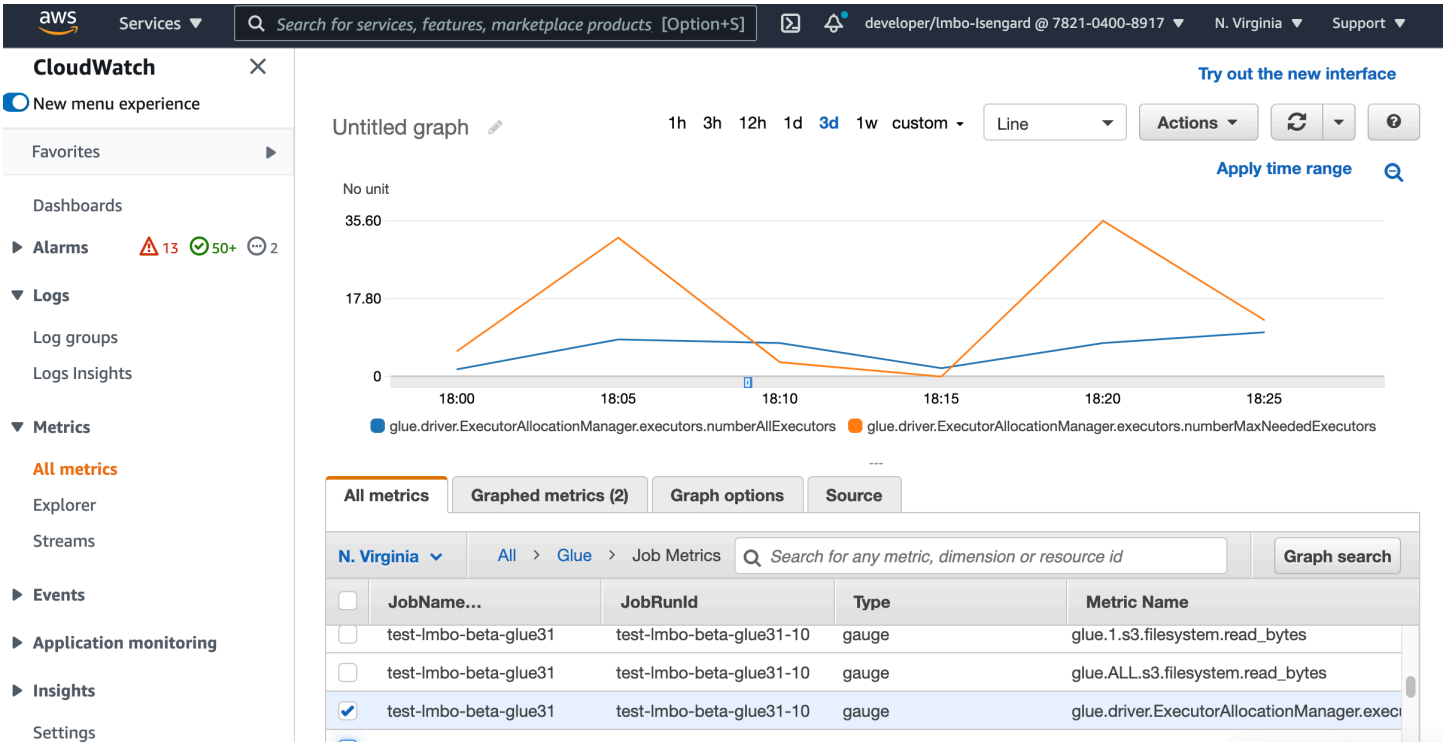
```
$ aws glue get-job-run --job-name your-job-name --run-id jr_xx --endpoint https://
glue.us-east-1.amazonaws.com --region us-east-1
{
    "JobRun": {
        ...
        "GlueVersion": "3.0",
        "DPUSeconds": 386.0
    }
}
```

您還可以使用具有相同組態的 AWS Glue SDK 為任務執行設定 Auto Scaling。

使用 Amazon CloudWatch 指標監控 Auto Scaling

如果您啟用「Auto Scaling 整」功能，則 CloudWatch 執行程式度量可用於 AWS Glue 3.0 或更新版本的工作。這些指標可用於監控在透過 Auto Scaling 啟用之 Spark 應用程式中的執行器需求和最佳化使用情況。如需詳細資訊，請參閱 使用 Amazon CloudWatch 指標監控 AWS Glue。

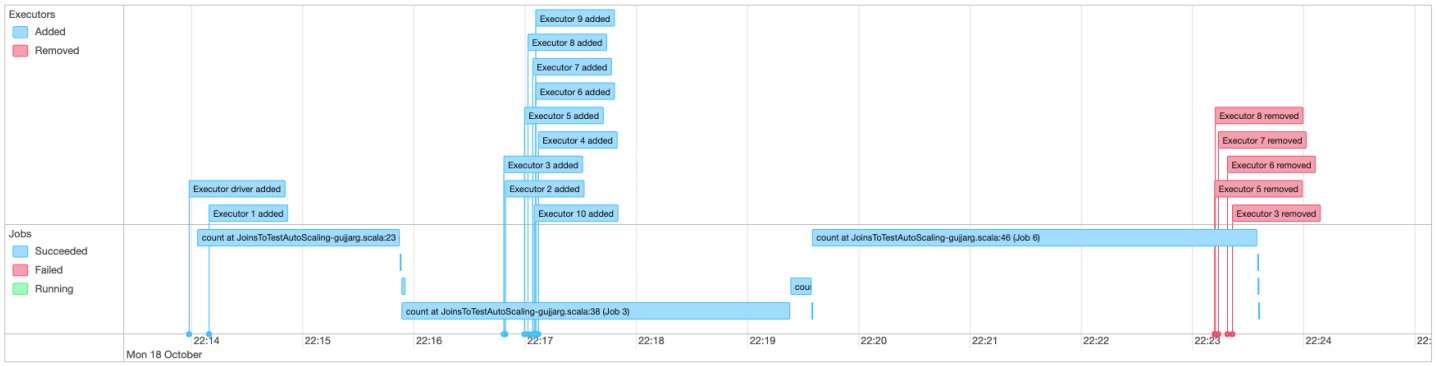
- 膠。司機。 ExecutorAllocationManager. 執行者。 numberAllExecutors
- 膠。司機。 ExecutorAllocationManager. 執行者。 numberMaxNeeded遺囑執行人



如需關於這些指標的詳細資訊，請參閱 DPU 容量規劃監控。

使用 Spark UI 監控 Auto Scaling

啟用 Auto Scaling 後，您還可以使用 Glue Spark UI，藉助動態擴充規模和縮減規模，根據 AWS Glue 任務中的需求監控正在新增和移除的執行器。如需詳細資訊，請參閱 為 AWS Glue 任務啟用 Apache Spark web UI。



監控 Auto Scaling 任務執行 DPU 使用情況

您可以使用 AWS Glue Studio 任務執行檢視以檢查 Auto Scaling 任務的 DPU 使用情況。

1. 從 AWS Glue Studio 導覽窗格選擇監控g。Monitoring (監控) 頁面隨即出現。

2. 向下捲動到 Job runs (任務執行) 圖表。
3. 導覽到您所需的任務執行，然後捲動到 DPU 時數欄以檢查特定任務執行的使用情況。

限制

AWS Glue串流 Auto Scaling 目前不支援在以外 DataFrame 建立靜態的串流 DataFrame 聯結`ForEachBatch`。在內部 DataFrame 創建的靜態`ForEachBatch`將按預期工作。

具有限制執行的工作負載分割

Spark 應用程式中的錯誤通常來自於效率低下的 Spark 指令碼，分散式記憶體執行大規模轉換以及資料集異常。有很多原因可能會導致驅動程式或執行程序的記憶體不足問題，例如資料偏斜，列出太多物件或大資料混洗。當您使用 Spark 處理大量積壓資料時，通常會出現這些問題。

AWS Glue 可讓您解決 OOM 問題，並透過工作負載分割讓 ETL 處理更輕鬆。啟用任務負載分割後，每個 ETL 任務執行只會挑選未處理的資料，而資料集大小或此任務執行時要處理的檔案數目上限。未來的任務執行將處理剩餘的資料。例如，如果需要處理 1000 個檔案，您可以將檔案數目設定為 500 個，並將它們分成兩個任務執行。

只有 Amazon S3 資料來源支援工作負載分割。

啟用工作負載分割

您可以透過手動設定指令碼中的選項或新增目錄資料表屬性來啟用限制執行。

若要在指令碼中啟用具有限制執行的工作負載分割：

1. 若要避免重新處理資料，請在新任務或現有任務中啟用任務書籤。如需詳細資訊，請參閱使用任務書籤追蹤已處理的資料。
2. 修改您的指令碼，並在AWS Glue `getSource` API 的附加選項中設定有界限制。您還應該設定工作書籤的轉換內容，以儲存 `state` 元素。例如：

   Python

   ```
   glueContext.create_dynamic_frame.from_catalog(
       database = "database",
       table_name = "table_name",
       redshift_tmp_dir = "",
   ```

```
    transformation_ctx = "datasource0",
    additional_options = {
        "boundedFiles" : "500", # need to be string
      # "boundedSize" : "1000000000" unit is byte
    }
)
```

Scala

```
val datasource0 = glueContext.getCatalogSource(
    database = "database", tableName = "table_name", redshiftTmpDir = "",
    transformationContext = "datasource0",
    additionalOptions = JsonOptions(
        Map("boundedFiles" -> "500") // need to be string
      //"boundedSize" -> "1000000000" unit is byte
    )
).getDynamicFrame()
```

```
val connectionOptions = JsonOptions(
    Map("paths" -> List(baseLocation), "boundedFiles" -> "30")
)
val source = glueContext.getSource("s3", connectionOptions, "datasource0", "")
```

若要在資料目錄資料表中啟用具有限制執行的工作負載分割：

1. 在資料目錄的資料表結構 `parameters` 欄位中設定鍵值對。如需詳細資訊，請參閱檢視與編輯資料表的詳細資訊。
2. 設定資料集大小或處理的檔案數目的上限：
   - 將 `boundedSize` 設定為資料集目標大小 (以位元組為單位)。從資料表達到目標大小後，任務執行將會停止。
   - 設定 `boundedFiles` 為目標檔案數量。處理檔案的目標數目後，任務執行將會停止。

> **Note**
>
> 您應該只設定一個 `boundedSize` 或 `boundedFiles`，因為只支援單一邊界。

設定 AWS Glue 觸發以自動執行任務

啟用限定執行之後，您可以設定 AWS Glue 觸發自動執行任務，並以循序執行遞增方式載入資料。前往 AWS Glue 主控台並建立觸發、設定排程時間，並連接至您的任務。然後它會自動觸發下一個任務執行並處理新批次的資料。

您也可以使用 AWS Glue 工作流程來協調多個任務，以平行處理來自不同分割區的資料。如需詳細資訊，請參閱 AWS Glue 觸發和 AWS Glue 工作流程。

如需更多關於使用案例和選項的詳細資訊，請參閱部落格在 AWS Glue 中使用工作負載分割來最佳化 Spark 應用程式。

AWS Glue 的已知問題

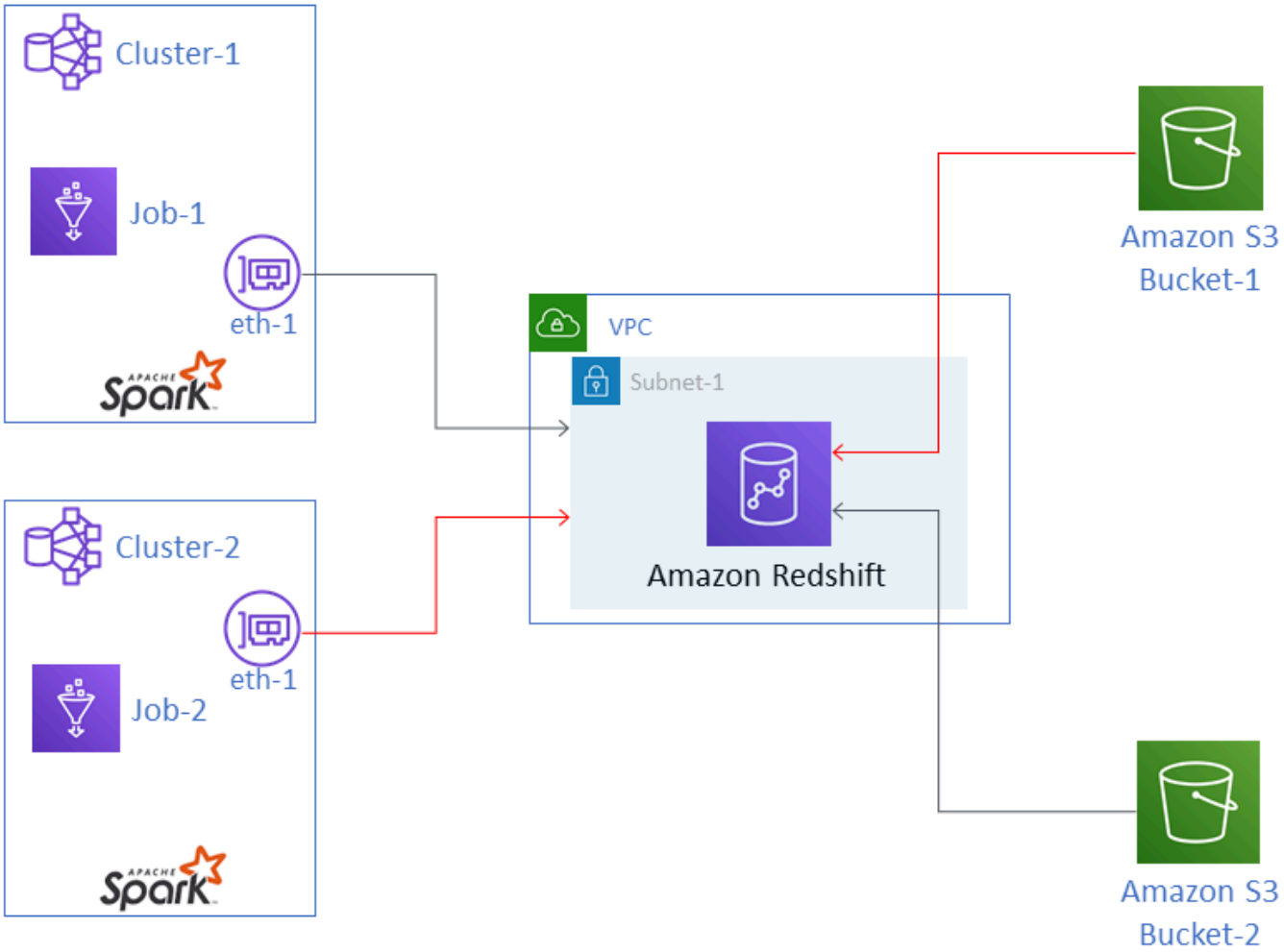
請注意，AWS Glue 具有下列已知問題。

主題

防止跨任務資料存取

假設單一 AWS Glue 帳戶中有兩個 AWS Spark 任務，且每個任務會在不同的 AWS Glue Spark 叢集中執行。這些任務會使用 AWS Glue 連線功能來存取相同虛擬私有雲端 (VPC) 中的資源。在這種情況下，其中一個叢集所執行的任務或許能讀取來自另一個叢集所執行任務的資料。

這種情況可用下圖表示。



在該圖中，AWS Glue Job-1 會在 Cluster-1 中執行，而 Job-2 則會在 Cluster-2 中執行。兩個任務所使用的是同一個 Amazon Redshift 執行個體，其位於 VPC 的 Subnet-1 中。Subnet-1 可能是公有或私有子網路。

Job-1 會轉換來自 Amazon Simple Storage Service (Amazon S3) Bucket-1 的資料，並將該資料寫入 Amazon Redshift。Job-2 則會以同樣方式來處理 Bucket-2 中的資料。Job-1 使用的是 AWS Identity and Access Management (IAM) 角色 Role-1 (圖中未顯示)，其可提供 Bucket-1 的存取權限。Job-2 使用的是 Role-2 (圖中未顯示)，可提供 Bucket-2 的存取權限。

這些任務可經由網路路徑與彼此的叢集通訊，進而存取對方的資料。舉例而言，Job-2 可以存取 Bucket-1 中的資料。這個路徑在圖中會顯示為紅色。

若要避免這種情況，建議您為 Job-1 與 Job-2 連接不同的安全組態。一旦連接安全組態，系統就可藉由 AWS Glue 建立的憑證來封鎖跨任務資料存取操作。安全組態可以是「虛擬」組態。也就是說，

您可以在不啟用 Amazon S3 資料、Amazon CloudWatch 資料或任務書籤加密的情況下，建立安全組態。這三種加密選項皆可停用。

如需安全組態的相關資訊，請參閱the section called “對 AWS Glue 寫入的資料加密”。

連接安全組態至任務

1. 開啟位於 https://console.aws.amazon.com/glue/ 的 AWS Glue 主控台。
2. 在任務的 Configure the job properties (設定任務屬性) 頁面上，展開 Security configuration, script libraries, and job parameters (安全組態、指令碼程式庫和工作屬性) 區段。
3. 在清單中選取安全組態。

AWS Glue 的文件歷史記錄

變更	描述	日期
Support AWS Glue 使用情況設定檔	管理員可以為帳戶內不同類別的 AWS Glue 使用者建立使用情況設定檔，例如開發人員、測試人員和產品團隊。這種靈活性使管理員可以為每個類別的用戶應用不同的用量和成本控制。如需詳細資訊，請參閱設定 AWS Glue 使用情況設定檔。	2024年6月18日
Support 適用於星火的 AWS Glue Salesforce 連接器	已新增 Salesforce 新 AWS Glue 連接器的相關資訊。這項功能可讓您在 AWS Glue 4.0 及更新版本中使用 AWS Glue Spark 來讀取和寫入 Salesforce。如需詳細資訊，請參閱連線至 Salesforce。	2024年5月22 日
Amazon Q 資料整合於 AWS Glue (GA)	中的 Amazon Q 資料整合 AWS Glue 是一項新的生成 AI 功能，可 AWS Glue 讓資料工程師和 ETL 開發人員使用自然語言建立資料整合任務。工程師和開發人員可以要求 Q 編寫任務、疑難排解問題，並回答有關 AWS Glue 資料整合的問題。如需詳細資訊，請參閱 AWS Glue中的 Amazon Q 資料整合。此功能包括AwsGlueSessionUserRestrictedPolicy 、AwsGlueSe	2024 年 4 月 30 日

	ssionUserRestricte dNotebookServiceRo le 和AwsGlueSe ssionUserRestricte dServiceRole AWS 受管理策略的更新。如需詳細資訊，請參閱AWS 受管理策略的更AWS Glue 新。	
Amazon Q 資料整合 AWS Glue (預覽)	中的 Amazon Q 資料整合 AWS Glue 是一項新的生成 AI 功能，可 AWS Glue 讓資料工程師和 ETL 開發人員使用自然語言建立資料整合任務。工程師和開發人員可以要求 Q 編寫任務、疑難排解問題，並回答有關 AWS Glue 資料整合的問題。如需詳細資訊，請參閱 AWS Glue中的 Amazon Q 資料整合。此功能包含受AwsGlueSessionUser RestrictedNotebook Policy AWS 管理策略的更新。如需詳細資訊，請參閱AWS 受管理策略的更AWS Glue 新。	2024年1月30日
更新至 AWS Glue 串流的文件	為「 AWS Glue 串流」新增了新章節，其中包含重新整理的內容。本內容說明串流的運作方式 AWS Glue、即時資料處理的特性，以及如何監視串流工作。如需詳細資訊，請參閱 AWS Glue 串流。	2023 年 12 月 27 日

支援使用微調敏感資料偵測	Detect Sensitive Data 轉換可偵測、遮罩或移除您定義或 AWS Glue 預先定義的實體。微調動作可讓您進一步針對每個實體套用特定動作。如需詳細資訊，請參閱使用微調敏感資料偵測。	2023 年 11 月 26 日
Support 使用 AWS Glue 可觀察性指標監視工作	使用 AWS Glue 可觀測性指標針對 Apache Spark 任務產生 AWS Glue 內部所發生之事件的深入解析，以改善問題的分類和分析。如需詳細資訊，請參閱使用 AWS Glue 可觀測性指標進行監控。	2023 年 11 月 26 日
Support 資 AWS Glue 料品質中的異常偵測	AWS Glue Data Quality 中的異常偵測功能會將機器學習 (ML) 演算法套用於一段時間內的資料統計資料，以偵測難以透過規則偵測的異常模式和隱藏的資料品質問題。如需詳細資訊，請參閱 AWS Glue Data Quality 中的異常偵測功能。	2023 年 11 月 26 日
更新為預設的 Spark UI 記錄行為	產生 Spark UI 記錄的 Spark 工作現在會以不同的檔案名稱模式寫入，以支援 AWS Glue 主控台中的 Spark UI。這不會變更 CloudWatch 記錄檔行為。您可以透過更新作業組態還原為舊版行為。如需詳細資訊，請參閱使用 Apache Spark web UI 監控作業。	2023 年 11 月 17 日

Support 於 Spark 的新資料來源 AWS Glue 的支援	與 Amazon OpenSearch 服務，Azure SQL，適用於 NoSQL 的 Azure 宇宙，SAP HANA 泰瑞數據和垂直的連接現在支持。 AWS Glue此外，連接到這些數據源，以及 MongoDB，現在可以在 AWS Glue 工作室可視化編輯器中使用。如需詳細資訊，請參閱 Spark 中 ETL 的連線類型和選項，以 AWS Glue 取得 Spark 支援的相關 AWS Glue 資訊和新增 AWS Glue 連線，以取得有關在 AWS Glue Studio 視覺化編輯器中使用的資訊。	2023 年 11 月 17 日
支援產生資料欄統計資料	您可以計算資料格式 (例如實木複合地板、ORC、JSON、ION、CSV 和 XML) 資料 AWS Glue Data Catalog 表的資料行層級統計資料，而無需設定其他資料管線。如需詳細資訊，請參閱使用資料欄統計資料。	2023 年 11 月 16 日
支援 Iceberg 資料表的資料壓縮	為了讓 Amazon Athena 和 Amazon EMR 和 AWS Glue ETL 任務等 AWS 分析服務獲得更好的讀取效能，資料型錄為資料目錄中的冰山表提供受管壓縮 (將小型 Amazon S3 物件壓縮為較大物件的程序)。如需詳細資訊，請參閱最佳化 Iceberg 資料表。	2023 年 11 月 13 日

任務執行等待行為更新	標準 Spark 和 Python Shell 任務執行目前會在特定情況下轉移至 WAITING，而非立即轉移至 FAILED。如需詳細資訊，請參閱 AWS Glue 任務執行狀態。	2023 年 11 月 8 日
整合至 AWS Glue 開發人員指南的 AWS Glue Studio 使用者指南	AWS Glue Studio 使用者指南已移至開發人員指南，以建立單一且統一的 AWS Glue Studio、AWS Glue 主控台和 AWS Glue Studio 程式設計存取相關使用者指南。	2023 年 10 月 25 日
更新到 AWSGlueServiceNotebookRole AWS 受管理的策略	已新增 AWSGlueServiceNotebookRole AWS 受管理原則的次要更新相關資訊。如需詳細資訊，請參閱AWS 受管理策略的更AWS Glue新。	2023 年 10 月 9 日
AWS Glue Studio 支援五種新的內建轉換	AWS Glue Studio 支援以下五種新的內置轉換：記錄比對、移除 Null 資料列、解析 JSON 資料欄、提取 JSON 路徑和 Regex 提取器。如需詳細資訊，請參閱編輯 AWS Glue 受管資料轉換節點。	2023 年 8 月 11 日
更新到 AWSGlueServiceRole AWS 受管理的策略	已新增 AWSGlueServiceRole AWS 受管理原則的次要更新相關資訊。如需詳細資訊，請參閱AWS 受管理策略的更AWS Glue新。	2023 年 8 月 4 日

支援網路爬取 Apache Hudi 資料表	已新增有關使 AWS Glue 用在 Amazon S3 儲存貯體中編目 Hudi 資料表，以及將 Hudi 資料表註冊到. AWS Glue Data Catalog如需詳細資訊，請參閱 Which data stores can I crawl? 和 Crawler properties。	2023 年 7 月 21 日
更新到 AWSGlueConsoleFull Access AWS 受管理的策略	已新增 AWSGlueConsoleFull Access AWS 受管理原則的次要更新相關資訊。如需詳細資訊，請參閱AWS 受管理策略的更AWS Glue新。	2023 年 7 月 14 日
支援網路爬取 Apache Iceberg 資料表	已新增有關使 AWS Glue 用在 Amazon S3 儲存貯體中抓取冰山資料表，以及將冰山資料表註冊到. AWS Glue Data Catalog如需詳細資訊，請參閱 Which data stores can I crawl? 和 Crawler properties。	2023 年 7 月 7 日
Support AWS Glue 使用雷	已新增 AWS Glue 有關 Ray (可支援 AWS Glue 工作的新引擎) 的資訊。重組現有 AWS Glue 與 Spark 內容消除歧義。	2023 年 5 月 30 日
Support 資 AWS Glue 料品質 (GA)	AWS Glue 資料品質現已正式推出。 AWS Glue 資料品質可協助您評估和監控資料品質。如需如何將資 AWS Glue 料品質與資料目錄搭配使用的相關資訊，請參閱資AWS Glue 料品質。若要瞭解的 AWS Glue 資料品質 AWS Glue Studio，請參閱使用評估資料品質 AWS Glue Studio。	2023 年 5 月 24 日

支援適用於 Apache Spark 任務的較大工作者類型	目前支援使用針對 Apache Spark 任務的 G.4X 與 G.8X 工作者類型。這些工作者類型適合工作負載包含最嚴苛轉換、彙總、聯結和查詢的任務。如需詳細資訊，請參閱在 AWS Glue 新增任務。	2023 年 5 月 8 日
支援在網路爬取資料表時建立分割區索引	新增了有關爬蟲程式如何支援為所偵測之資料表建立分割區索引的資訊。如需詳細資訊，請參閱 Setting the partition index crawler configuration option。	2023 年 4 月 24 日
支援資源用量指標	已新增有關在 Amazon 中檢視服務資源使用情況和設定警示的資訊 CloudWatch。如需詳細資訊，請參閱 AWS Glue resource monitoring。	2023 年 4 月 7 日
更新到 AWSGlueConsoleFullAccess AWS 受管理的策略	已新增 AWSGlueConsoleFullAccess AWS 受管理原則的次要更新相關資訊。如需詳細資訊，請參閱AWS 受管理策略的更AWS Glue新。	2023 年 3 月 28 日
已新增與 AWS SDK AWS Glue 搭配使用範例的指引	開AWS Glue發人員指南包含兩個新章節，提供協助您AWS Glue搭配 AWS SDK 使用的資訊。如需詳細資訊，請參閱AWS Glue 搭配 AWS SDK 使用和AWS Glue 使用 AWS SDK 的程式碼範例。	2023 年 2 月 23 日

使用以下方式更新 IAM 的文件 AWS Glue	重新組織並新增使用 IAM 的 AWS Glue相關資訊。如需詳細資訊，請參閱適用於 AWS Glue的 Identity and Access Management。	2023 年 2 月 15 日
支援在 AWS Glue 4.0 版中執行串流 ETL 任務	已新增有關在 Glue 4.0 版中執行串流 ETL 任務的支援、連線至 Kafka 叢集的新選項、適用於 Apache Kafka 叢集的 Amazon 受管串流，以及 Amazon Kinesis Data Streams 的相關資訊。如需詳細資訊，請參閱在 AWS Glue 中新增串流 ETL 任務和 AWS Glue 中的 ETL 連線類型和選項。	2023 年 2 月 8 日
支援網路爬取 MongoDB Atlas 資料來源	已新增有關使用 AWS Glue 編目 MongoDB 地圖集資料來源的資訊。如需詳細資訊，請參閱我可以編目哪些資料存放區？ ，MongoDB 和 MongoDB 的地圖集連接屬性，以及使用 MongoDB 或蒙古數據 MongoDB 的地圖集連接。	2023 年 2 月 6 日

支援使用原生 Delta Lake 連接器網路爬取 Delta Lake 資料表	已新增使用使用 AWS Glue 原生 Delta Lake 連接器編目 Delta Lake 資料表的相關資訊。此功能可讓您使用 AWS 查詢引擎直接查詢 Delta 交易日誌，並使用時間旅行和 ACID 保證等功能，並將您的 Delta Lake 中繼資料從 Amazon S3 交易檔案同步到資料目錄，以便在 Lake Formation 中的查詢啟用欄許可。如需詳細資訊，請參閱如何為 Delta Lake 的資料儲存指定配置選項和查詢 Delta Lake 資料表。	2022 年 12 月 15 日
AWS Glue 資料品質 Support (預覽)	AWS Glue 資料品質現已提供 Support (預覽)。 AWS Glue 資料品質可協助您在使用 AWS Glue 3.0 時評估和監控資料品質。如需如何搭配資料目錄使用「資 AWS Glue 料品質」的詳細資訊，請參閱資AWS Glue 料品質 (預覽)。若要瞭解的 AWS Glue 資料品質 AWS Glue Studio，請參閱使用評估資料品質 AWS Glue Studio。	2022 年 11 月 30 日

支援具有新功能和改進效能的全新 Amazon Redshift Spark 連接器	現支援具備新 JDBC 驅動程式的全新 Amazon Redshift Spark 連接器，可在資料擷取與轉換管道過程中搭配 AWS Glue ETL 任務使用，用於建置在 Amazon Redshift 中讀取和寫入資料的 Apache Spark 應用程式。如需詳細資訊，請參閱將資料移入及移出 Amazon Redshift。	2022 年 11 月 29 日
支援 AWS Glue 4.0 版。	新增了 AWS Glue 4.0 版支援的相關資訊。功能包括對 Apache Hudi、Delta Lake 和 Apache Iceberg 開源資料湖架構的原生支援，以及對以 Amazon S3 為基礎的雲端隨機排序儲存外掛程式 (一種 Apache Spark 外掛程式) 的原生支援，以針對隨機排序和彈性儲存容量使用 Amazon S3。如需詳細資訊，請參閱 AWS Glue 版本備註和將 AWS Glue 任務遷移至 AWS Glue 4.0 版。	2022 年 11 月 28 日
AWS Glue Studio 現在提供自訂視覺化轉換功能	自訂視覺化轉換可讓客戶在團隊之間定義、重複使用和共用業務專屬的 ETL 邏輯。如需詳細資訊，請參閱自訂視覺化轉換。	2022 年 11 月 28 日

支援使用 AWS Glue 爬蟲程式發佈 JDBC 資料存放區的中繼資料	現支援使用 AWS Glue 爬蟲程式，將諸如註解和原始類型等中繼資料發佈至適用於 JDBC 資料存放區的 Data Catalog。如需詳細資訊，請參閱依爬行者程式、爬行者程式特性和JdbcTarget 結構在資料目錄表格上設定的參數。	2022 年 11 月 18 日
支援網路爬取 Snowflake 資料存放區	現支援使用 AWS Glue 網路爬取 Snowflake 資料表和檢視，以及將中繼資料作為表格項目發佈至 Data Catalog。對於 Amazon S3 中的 Snowflake 外部資料表，爬蟲程式也會網路爬取 Amazon S3 位置和外部資料表的檔案格式類型，並填入為表格參數。如需詳細資訊，請參閱我可以爬取哪個資料存放區？、AWS Glue 連線屬性，以及爬蟲程式在 Data Catalog 資料表上設定的參數。	2022 年 11 月 18 日
支援改進 Spark 應用程式的隨機排序管理	現支援新的 Apache Spark 雲端隨機排序儲存外掛程式。如需詳細資訊，請參閱 AWS Glue Spark 隨機排序管理器與 Amazon S3 和 Cloud Shuffle Storage Plugin for Apache Spark (Apache Spark 雲端隨機排序儲存外掛程式)。	2022 年 11 月 15 日

新增了加速網路爬取 Amazon S3 事件通知時對 Data Catalog 目標的支援	除了對 Amazon S3 目標的現有支援外，現在還支援使用 Amazon S3 事件通知加速對 Data Catalog 目標的網路爬取。如需詳細資訊，請參閱使用 Amazon S3 事件通知加速網路爬取。	2022 年 10 月 13 日
支援指定爬蟲程式可建立的資料表數目上限	現在可支援指定爬蟲程式可建立的資料表數目上限。如需詳細資訊，請參閱如何指定爬蟲程式可建立的資料表數目上限。	2022 年 9 月 6 日
支援 AWS Glue 中 Python Shell 任務中的 Python 3.9	現在可支援執行 AWS Glue 中 Python Shell 作業中與 Python 3.9 相容的執行指令碼，並支援選擇使用預先封裝的程式庫集。如需詳細資訊，請參閱 AWS Glue 中的 Python Shell 任務。	2022 年 8 月 11 日
Support 在備用容量上執行非緊急或非時間敏感 AWS Glue 工作	現在可支援設定非緊急任務 (如生產前任務、測試和一次性資料載入) 的彈性任務執行。如需詳細資訊，請參閱在 AWS Glue 新增任務。	2022 年 8 月 9 日
支援串流任務的新工作者類型	目前支持使用適用於低容量串流任務的 `G.025X` 工作者類型。如需詳細資訊，請參閱在 AWS Glue 新增任務。	2022 年 7 月 14 日

支援在 AWS Glue 連線中使用 Kafka SASL	目前支援在 AWS Glue 連線中使用 Kafka SASL。如需詳細資訊，請參閱適用於用戶端身分驗證的 AWS Glue Kafka 連線屬性。	2022 年 7 月 5 日
支援適用於 protobuf 結構描述的 Apache kafka 連接器	目前支援適用於 Protobuf 結構描述的 Apache Kafka 連接器。如需詳細資訊，請參閱 AWS Glue 結構描述登錄檔。	2022 年 6 月 9 日
支援 AWS Glue 任務的 Auto Scaling (GA)	增加了有關在 AWS Glue 3.0 版本中對任務使用 Auto Scaling 以動態擴展運算資源的資訊。如需詳細資訊，請參閱為 AWS Glue 使用 Auto Scaling。	2022 年 4 月 14 日
與 AWS Glue 開發和測試 AWS Glue 任務指令碼相關的文件集更新	重新組織並補充了 AWS Glue 的可用開發和測試方法的相關資訊，包括使用 Docker 進行開發的說明。如需詳細資訊，請參閱開發和測試 AWS Glue 任務指令碼。	2022 年 3 月 14 日
新增協定緩衝區 (protobuf) 做為 AWS Glue 結構描述登錄檔的支援資料格式	新增有關 Protobuf 做為支援的資料格式 (除了 AVRO 和 JSON 之外) 的相關資訊。如需詳細資訊，請參閱 AWS Glue 結構描述登錄檔。	2022 年 2 月 25 日
支援爬取 Delta Lake 資料表	已新增有關使用 AWS Glue 編目 Delta 湖資料表的資訊。如需詳細資訊，請參閱如何為 Delta Lake 的資料儲存指定配置選項。	2022 年 2 月 24 日

AWS Glue 工作見解 Support	已新增有關使用 AWS Glue 工作見解來簡化工作偵錯和最佳化 AWS Glue 工作的資訊。如需詳細資訊，請參閱使用 AWS Glue 任務洞見進行監控。	2022 年 2 月 8 日
支援使用 VPC 端點爬取 Amazon S3 支援的 Data Catalog 資料表	除了 Amazon S3 資料存放區之外，您還可以設定 Amazon S3 支援的 Data Catalog 資料表以僅供 Amazon Virtual Private Cloud 環境 (Amazon VPC) 存取，以用於安全、稽核或控制目的。如需詳細資訊，請參閱使用 VPC 端點爬取 Amazon S3 資料存放區或 Amazon S3 支援的 Data Catalog 資料表。	2022 年 2 月 3 日
支援受 Lake Formation 管控的資料表	新增關於 AWS Glue 支援受 Lake Formation 管控之資料表的資訊，這些資料表支援 ACID 事務、自動資料壓縮和時間旅行查詢。如需詳細資訊，請參閱 AWS Glue API 和 AWS Lake Formation 開發人員指南。	2021 年 11 月 30 日
為互動式工作階段和筆記本新增了新的 AWS 受管	IAM 的新受管政策提供增強的安全性，可 AWS Glue 搭配互動式工作階段和筆記本使用。如需詳細資訊，請參閱 AWS Glue的AWS 受管政策。	2021 年 11 月 30 日

串流任務現在支援 Glue 結構描述登錄檔	您可以建立串流任務來存取屬於 Glue 結構描述登錄檔的資料表。如需詳細資訊，請參閱 AWS Glue 結構描述登錄檔和在 AWS Glue中新增串流 ETL 任務。	2021 年 11 月 15 日
支援全新的機器學習功能	已新增有關「尋找相符項目」機器學習轉換之新功能的資訊，包括增量改進比對和相符項目得分。如需詳細資訊，請參閱尋找增量改進相符項目和使用相符項目可信度分數估計項目相符品質。	2021 年 10 月 31 日
(私有預覽版) 支援 AWS Glue 彈性任務	新增關於設定具有彈性執行類別的 AWS Glue Spark 任務的資訊，適用於開始和完成時間可能會有所變化的時間敏感型任務。如需詳細資訊，請參閱在 AWS Glue 新增任務。	2021 年 10 月 29 日
支援使用 Amazon S3 事件通知加速網路爬取	新增使用 Amazon S3 事件通知加速網路爬取的相關資訊。如需詳細資訊，請參閱使用 Amazon S3 事件通知加速網路爬取。	2021 年 10 月 15 日
與存取控制和 VPC 相關的其他安全性組態選項	新增關於如何在 AWS Glue 以及 VPC 組態上設定全新存取控制許可的資訊。如需詳細資訊，請參閱中的AWS標記AWS Glue、使用條件金鑰或內容金鑰控制設定的身分型政策 (IAM 政策) 和設定透過 VPC 進行的所有 AWS 呼叫。	2021 年 10 月 13 日

支援 VPC 端點政策	新增在 AWS Glue 中支援虛擬私有雲端 (VPC) 端點政策的相關資訊。如需詳細資訊，請參閱 AWS Glue 和介面 VPC 端點 (AWS PrivateLink)。	2021 年 10 月 11 日
Glue Studio 現在可在中國區域中使用	中國北京和寧夏區域現在可以使用 AWS Glue Studio。	2021 年 10 月 11 日
AWS Glue Studio 提供筆記本編寫功能，用於互動式任務編輯	筆記本可協助您撰寫和執行程式碼、視覺化結果，以及分享深入解析。通常，資料科學家使用筆記本進行實驗和資料探索任務。如需詳細資訊，請參閱使用筆記本。	2021 年 10 月 1 日
現已推出直接存取串流來源功能	在視覺化編輯器中將資料來源新增至 ETL 任務時，您可以提供資訊來存取資料串流，而不必使用 Data Catalog 資料庫和資料表。	2021 年 9 月 30 日
記錄 AWS Glue 版本支援政策	新增關於 AWS Glue 版本支援政策以及某些 AWS Glue 版本生命週期結束階段的資訊。如需詳細資訊，請參閱 AWS Glue 版本支援政策。	2021 年 9 月 24 日
自訂連接器現在可以搭配資料預覽使用	使用自訂連接器編輯資料來源節點時，您可以選擇「資料預覽」索引標籤來預覽資料集。如需詳細資訊，請參閱自訂連接器。	2021 年 9 月 24 日

Support AWS Glue 互動式工作階段 (私人預覽)	(私人預覽) 已新增有關使用 AWS Glue 互動式工作階段從任何 Jupyter 筆記本在雲端執行 Spark 工作負載的相關資訊。當您使用 AWS Glue 2.0 或更新版本時，互動式工作階段是開發 AWS Glue 擷取、轉換和載入 (ETL) 程式碼的偏好方法。如需詳細資訊，請參閱為 Jupyter 記事本設定和執行 AWS Glue 互動式工作階段。	2021 年 8 月 24 日
支援從藍圖建立工作流程 (GA)	新增在藍圖中撰寫常用擷取、轉換和載入 (ETL) 使用案例的程式碼，然後從藍圖建立工作流程的相關資訊。可讓資料分析師輕鬆建立及執行複雜的 ETL 程序。如需詳細資訊，請參閱在 AWS Glue 中使用藍圖和工作流程執行複雜的 ETL 活動。	2021 年 8 月 23 日

支援 AWS Glue 3.0 版。	新增支援 AWS Glue 3.0 版的相關資訊，它支援執行 Apache Spark ETL 任務以及其他最佳化和升級的 Apache Spark 3.0 引擎升級。如需詳細資訊，請參閱 AWS Glue 版本備註和將 AWS Glue 任務遷移至 AWS Glue 3.0 版。此版本中的其他功能包括 AWS Glue 隨機排序管理器、SIMD 向量化 CSV 讀取器和目錄分割區述詞。如需更多詳細資訊，請參閱 AWS Glue Spark 隨機排序管理器與 Amazon S3、在 AWS Glue 中的 ETL 輸入與輸出格式選項，以及使用目錄分割述詞的伺服器端篩選。	2021 年 8 月 18 日
AWS GovCloud (US) Region	AWS Glue Studio現在可在 AWS GovCloud (US) Region	2021 年 8 月 18 日
Python Shell 編寫可用於 AWS Glue Studio	建立新任務時，您現在可以選擇建立 Python Shell 任務。如需詳細資訊，請參閱啟動任務建立程序和在 AWS Glue Studio 中編輯 Python Shell 任務。	2021 年 8 月 13 日
Support 使用 Amazon EventBridge 事件啟動工作流程	新增 AWS Glue 如何成為事件驅動架構中的事件取用者的相關資訊。有關詳情，請參閱使用 Amazon EventBridge 事件啟動AWS Glue工作流程和檢視啟 EventBridge 動工作流程的事件。	2021 年 7 月 14 日

新增 JSON 做為 AWS Glue 結構描述登錄檔的支援資料格式	新增有關 JSON 的資訊做為支援的資料格式 (除了 AVRO 之外) 的相關資訊。如需詳細資訊，請參閱 AWS Glue 結構描述登錄檔。	2021 年 6 月 30 日
建立不含 Data Catalog 資料表的 AWS Glue 串流任務	`create_data_frame_from_options` Python 函數或 `getSource` for Scala 指令碼支援建立可直接引用資料串流的 ETL 任務，而不是需要 Data Catalog 資料表。	2021 年 6 月 15 日
AWS Glue機器學習轉換現在支援 AWS Key Management Service 按鍵	使用主控台、CLI 或 AWS Glue API 設定 Machine L AWS Glue earning 轉換時，您可以指定安全性組態或 AWS KMS 金鑰。如需詳細資訊，請參閱在 Machine Learning 轉換中使用資料加密和 AWS Glue Machine Learning API。	2021 年 6 月 15 日
更新到 AWSGlueConsoleFullAccess AWS 受管理的策略	已新增 AWSGlueConsoleFullAccess AWS 受管理原則的次要更新相關資訊。如需詳細資訊，請參閱AWS 受管理策略的更AWS Glue新。	2021 年 6 月 10 日
在建立和編輯任務時檢視任務的資料集	您可以使用任務圖表中節點的新資料預覽索引標籤，以查看該節點處理的資料範例。如需詳細資訊，請參閱在視覺化任務編輯器中使用資料預覽。	2021 年 6 月 7 日

支援指定值以指出爬蟲程式輸出的資料表位置。	新增在設定爬蟲程式輸出時指出資料表位置值的相關資訊。如需詳細資訊，請參閱如何指定資料表位置。	2021 年 6 月 4 日
在爬取 Amazon S3 資料存放區時，支援爬取資料集中的檔案樣本	新增有關在爬取 Amazon S3 時如何爬取檔案範例的相關資訊。如需詳細資訊，請參閱爬蟲程式屬性。	2021 年 5 月 10 日
支援 AWS Glue 最佳化 Parquet 寫入器	已新增有關使用AWS Glue最佳化的實木複合地板寫入器 DynamicFrames 來建立或更新具有`parquet`分類的表格的資訊。如需詳細資訊，請參閱透過 AWS Glue ETL 任務在 Data Catalog 中建立資料表、更新結構描述，以及新增分割區以及在 AWS Glue 中的 ETL 輸入與輸出格式選項。	2021 年 5 月 4 日
支援 kafka 用戶端身分驗證密碼	新增 AWS Glue 中的串流 ETL 任務如何支援搭配 Apache Kafka 串流生產者的 SSL 用戶端憑證身分驗證的相關資訊。您現在在定義連至 Apache Kafka 叢集的 AWS Glue 連線時可以提供自訂憑證，AWS Glue 將用於進行身分驗證。如需詳細資訊，請參閱 AWS Glue 連線屬性和連線 API。	2021 年 4 月 28 日

支援在串流 ETL 任務中使用另一個帳戶中來自 Amazon Kinesis Data Streams 的資料	新增有關建立串流 ETL 任務以使用其他帳戶中 Amazon Kinesis Data Streams 資料的相關資訊。如需詳細資訊，請參閱在 AWS Glue 中新增串流 ETL 任務。	2021 年 3 月 30 日
可用的 SQL 轉換	您可以使用 SQL 轉換節點以 SQL 查詢的形式編寫自己的轉換。如需詳細資訊，請參閱使用 SQL 查詢轉換資料。	2021 年 3 月 23 日
支援從藍圖建立工作流程 (公開預覽)	(公開預覽) 新增在藍圖中撰寫常用擷取、轉換和載入 (ETL) 使用案例的程式碼，然後從藍圖建立工作流程的相關資訊。可讓資料分析師輕鬆建立及執行複雜的 ETL 程序。如需詳細資訊，請參閱在 AWS Glue 中使用藍圖和工作流程執行複雜的 ETL 活動。	2021 年 3 月 22 日
連接器可用於資料目標	現在支援為資料目標使用自訂或 AWS Marketplace 連接器。如需詳細資訊，請參閱使用自訂連接器編寫任務。	2021 年 3 月 15 日
支援 AWS Glue 機器學習轉換的資料欄重要性指標	新增有關使用 AWS Glue 機器學習轉換時檢視欄重要性指標的相關資訊。如需詳細資訊，請參閱在 AWS Glue 主控台上使用機器學習轉換。	2021 年 2 月 5 日

任務排程現在可用於 AWS Glue Studio	您可以在 AWS Glue Studio 中為任務執行定義以時間為基礎的排程。您可以使用主控台建立基本排程，或使用類似 UNIX 的 cron 語法定義更複雜的排程。如需詳細資訊，請參閱排程任務執行。	2020 年 12 月 21 日
已發行 AWS Glue 自訂連接器	AWS Glue 自訂連接器可讓您探索並訂閱 AWS Marketplace中的連接器。我們也發行了 AWS Glue Spark 執行時間介面，可插入為 Apache Spark 資料來源、Athena 聯合查詢和 JDBC API 建置的連接器。如需詳細資訊，請參閱搭配 AWS Glue Studio 使用連接器和連線。	2020 年 12 月 21 日
支援在 AWS Glue 2.0 版中執行串流 ETL 任務	新增有關支援在 Glue 2.0 版中執行串流 ETL 任務的相關資訊。如需詳細資訊，請參閱在 AWS Glue 中新增串流 ETL 任務。	2020 年 12 月 18 日
支援使用限制執行的工作負載分割	新增有關啟用工作負載分割以設定資料集大小上限，或 ETL 任務執行時處理的檔案數目的相關資訊。如需詳細資訊，請參閱具有限制執行的工作負載分割。	2020 年 11 月 23 日
支援增強的分割區管理	新增有關如何使用新 API 在現有資料表中新增或刪除分割區索引的相關資訊。如需詳細資訊，請參閱使用分割區索引。	2020 年 11 月 23 日

支援 AWS Glue 結構描述登錄檔	新增使用 AWS Glue 結構描述登錄檔集中探索、控制和演進結構描述的相關資訊。如需詳細資訊，請參閱 AWS Glue 結構描述登錄檔。	2020 年 11 月 19 日
支援串流 ETL 任務中的 grok 輸入格式	新增有關將 Grok 模式套用至串流來源 (例如日誌檔) 的相關資訊。如需詳細資訊，請參閱將 Grok 模式應用於串流來源。	2020 年 11 月 17 日
支援在 AWS Glue 主控台將標籤新增至工作流程	新增有關在使用 AWS Glue 主控台建立工作流程時新增標籤的相關資訊。如需詳細資訊，請參閱使用 AWS Glue 主控台建立和建構工作流程。	2020 年 10 月 27 日
支援增量爬蟲程式執行	新增有關支援增量爬蟲程式執行的相關資訊，這只會抓取自上次執行以來新增的 Amazon S3 資料夾。如需詳細資訊，請參閱增量網路爬取。	2020 年 10 月 21 日
支援串流 ETL 資料來源的結構描述偵測。支援 Avro 串流 ETL 資料來源和自我管理的 kafka	AWS Glue 中的串流擷取、轉換和載入 (ETL) 任務現在可以自動偵測傳入記錄的結構描述，並以每個記錄為基礎處理結構描述變更。現在支援自我管理的 Kafka 資料來源。串流 ETL 任務現在支援資料來源中的 Avro 格式。如需詳細資訊，請參閱 AWS Glue 的串流 ETL、定義串流 ETL 任務的任務屬性，以及 Avro 串流來源的注意事項與限制。	2020 年 10 月 7 日

支援網路爬取 MongoDB 和 DocumentDB 資料來源	新增有關支援網路爬取 MongoDB 和 Amazon DocumentDB (with MongoDB Compatibility) 資料來源的相關資訊。如需詳細資訊，請參閱定義爬蟲程式。	2020 年 10 月 5 日
支援 FIPS 合規	新增適用於使用 AWS Glue 存取資料時，需要 FIPS 140-2 驗證密碼編譯模組之客戶的 FIPS 端點的相關資訊。如需詳細資訊，請參閱 FIPS 合規。	2020 年 9 月 23 日
AWS Glue Studio 提供了一個易於使用的視覺化介面，用於建立和監控任務	您現在可以使用簡單的圖形介面來撰寫移動和轉換資料的任務，並在 AWS Glue 中執行它們。然後，您可以使用 AWS Glue Studio 中的任務執行儀表板來監控 ETL 執行，並確保您的任務按預期執行。如需詳細資訊，請參閱 AWS Glue Studio 使用者指南。	2020 年 9 月 23 日
支援建立資料表索引以改善查詢效能	新增有關建立資料表索引以讓您從資料表擷取分割區子集的相關資訊。如需詳細資訊，請參閱使用分割區索引。	2020 年 9 月 9 日
支援在 AWS Glue 2.0 版中執行 Apache Spark ETL 任務時減少啟動次數。	新增支援 AWS Glue 2.0 版的相關資訊，它為執行 Apache Spark ETL 任務提供升級的基礎結構、縮短啟動時間、變更記錄，並支援在任務層級指定其他 Python 模組。如需詳細資訊，請參閱 AWS Glue 版本備註和以縮短的啟動時間執行 Spark ETL 任務。	2020 年 8 月 10 日

支援限制並行工作流程執行的數目。	新增如何限制特定工作流程之並行工作流程執行次數的相關資訊。如需詳細資訊，請參閱使用 AWS Glue 主控台建立和建構工作流程。	2020 年 8 月 10 日
支援使用 VPC 端點爬取 Amazon S3 資料存放區	新增設定 Amazon S3 資料存放區以僅供 Amazon Virtual Private Cloud 環境 (Amazon VPC) 存取的相關資訊，以用於安全、稽核或控制目的。如需詳細資訊，請參閱使用 VPC 端點爬取 Amazon S3 資料存放區。	2020 年 8 月 7 日
支援繼續工作流程執行	新增有關如何繼續工作流程執行的相關資訊，這些工作流程執行僅部分完成，因為一或多個節點 (任務或爬蟲程式) 未順利完成。如需詳細資訊，請參閱修復和繼續工作流程執行。	2020 年 7 月 27 日
支援在 AWS Glue 中啟用 kafka 連線的私有 CA 憑證。	新增支援在 AWS Glue 中為 Kafka 連線啟用私有 CA 憑證的新連線選項的相關資訊。如需詳細資訊，請參閱 AWS Glue 中 ETL 的連線類型和選項和 AWS Glue 使用的特殊參數。	2020 年 7 月 20 日
支援讀取其他帳戶中的 DynamoDB 資料	新增 AWS Glue 支援從另一個 AWS 帳戶的 DynamoDB 資料表讀取資料的相關資訊。如需詳細資訊，請參閱從其他帳戶中的 DynamoDB 資料讀取。	2020 年 7 月 17 日

在 AWS Glue 1.0 版本或更新版本中支援 DynamoDB 寫入器連線	新增支援 DynamoDB 寫入器，以及 DynamoDB 讀取或寫入的新或更新連線選項的相關資訊。如需詳細資訊，請參閱 AWS Glue 中的 ETL 連線類型和選項。	2020 年 7 月 17 日
支援同時使用 AWS Glue 和 Lake Formation 的資源連結和跨帳戶存取控制	新增有關稱為資源連結的新 Data Catalog 物件，以及如何使用 AWS Glue 和 AWS Lake Formation在帳戶之間管理共用 Data Catalog 資源的內容。如需詳細資訊，請參閱授予跨帳戶存取權和資料表資源連結。	2020 年 7 月 7 日
支援爬取 DynamoDB 資料存放區時取樣記錄	已新增有關爬取 DynamoDB 資料存放區時可以設定新屬性的資訊。如需詳細資訊，請參閱爬蟲程式屬性。	2020 年 6 月 12 日
支援停用工作流程執行。	新增關於如何停止特定工作流程之工作流程執行的資訊。如需詳細資訊，請參閱停止工作流程執行。	2020 年 5 月 14 日
支援 Spark Streaming ETL 任務	新增關於使用串流資料來源建立擷取、轉換和載入 (ETL) 任務的資訊。如需詳細資訊，請參閱在 AWS Glue 中新增串流 ETL 任務。	2020 年 4 月 27 日

支援在執行 ETL 任務後在 Data Catalog 中建立資料表、更新結構描述，以及新增分割區	已新增有關如何啟用建立資料表、更新結構描述，以及新增分割區，以在 Data Catalog 中查看 ETL 任務結果的資訊。如需詳細資訊，請參閱透過 AWS Glue ETL 任務在 Data Catalog 中建立資料表、更新結構描述，以及新增分割區。	2020 年 4 月 2 日
支援指定 Apache Avro 資料格式的版本作為 AWS Glue 中的 ETL 輸入和輸出	已新增指定 Apache Avro 資料格式的版本作為 AWS Glue 中 ETL 輸入和輸出的相關資訊。預設版本 1.7。您可以使用 `version` 格式選項來指定 Avro 版本 1.8，以啟用邏輯讀取/寫入。如需詳細資訊，請參閱 AWS Glue 中 ETL 輸入和輸出的格式選項。	2020 年 3 月 31 日
支援 EMRFS S3 最佳化遞交者，以將 Parquet 資料寫入 Amazon S3	已新增如何設定新標記來啟用 EMRFR S3 最佳化遞交者，以在建立或更新 AWS Glue 任務時，將 Parquet 資料寫入 Amazon S3 的相關資訊。如需詳細資訊，請參閱 AWS Glue 使用的特殊參數。	2020 年 3 月 30 日
Support 機器學習轉換為由資源標籤管理的 AWS 資源	已新增有關使用 AWS 資源標籤來管理和控制機器學習轉換的存取權限的相關資訊AWS Glue。您可以將 AWS 資源標籤指派給中AWS Glue的工作、觸發器、端點、編目器和機器學習轉換。如需詳細資訊，請參閱 AWS Glue 中的AWS 標籤。	2020 年 3 月 2 日

支援不可覆寫的任務引數	已新增無法在觸發條件中覆寫，或在您執行任務時的特殊任務參數支援相關資訊。如需詳細資訊，請參閱在 AWS Glue 新增任務。	2020 年 2 月 12 日
支援用於 Amazon S3 中資料集的新轉換	新增有關 Apache Spark 應用程式搭配 Amazon S3 中資料集之新轉換 (合併、清除和轉移) 以及 Amazon S3 儲存體方案排除項目的相關資訊。如需有關對 Python 這些轉換的支援的詳細資訊，請參閱mergeDynamicFrame並在 Amazon S3 中使用資料集。對於斯卡拉，請參閱mergeDynamicFrames和AWS Glue斯卡拉 GlueContext API。	2020 年 1 月 16 日
支援使用來自 ETL 任務的新分割區資訊進行 Data Catalog 更新	已新增 AWS Glue Data Catalog 有關如何編寫擷取、轉換和載入 (ETL) 指令碼的資訊，以使用新的分割區資訊更新。利用這項功能，您不再需要在任務完成後重新執行爬蟲程式，即可檢視新的分割區。如需詳細資訊，請參閱使用新分割區更新 Data Catalog 。	2020 年 1 月 15 日
新增教學課程：使用 SageMaker 筆記本	已新增教學課程，示範如何使用 Amazon SageMaker 筆記本協助開發 ETL 和機器學習指令碼。請參閱教學課程：搭配開發端點使用 Amazon SageMaker 筆記本。	2020 年 1 月 3 日

支援從 MongoDB 和 Amazon DocumentDB (with MongoDB compatibility) 讀取	已新增有關讀取和寫入 MongoDB 和 Amazon DocumentDB (with MongoDB Compatibility) 的新連線類型和連線選項資訊。如需詳細資訊，請參閱 AWS Glue 中的 ETL 連線類型和選項。	2019 年 12 月 17 日
多個修正與說明	加入完整的修正與說明。已從「已知問題」章節中移除項目內容。新增在指定 Data Catalog 加密設定和建立安全性組態時，僅 AWS Glue 支援對稱客戶主金鑰 (CMK) 的警告。新增 AWS Glue 不支援寫入 Amazon DynamoDB 的附註。	2019 年 12 月 9 日
支援自訂 JDBC 驅動程式	新增有關使用 AWS Glue 未原生支援的 JDBC 驅動程式連線至資料來源和目標的資訊，例如 MySQL 版本 8 和 Oracle Database 版本 18。如需詳細資訊，請參閱 JDBC connectionType 值。	2019 年 11 月 25 日
Support 將 SageMaker 筆記型電腦連接至不同開發端點	已新增有關如何將 SageMaker 筆記本連接至不同開發端點的資訊。用於說明切換到新開發端點的新主控台動作以及新的 SageMaker IAM 政策的更新。如需詳細資訊，請參閱在AWS Glue主控台上使用筆記本和為 Amazon SageMaker 筆記本建立 IAM 政策。	2019 年 11 月 21 日

支援機器學習轉換中的 AWS Glue 版本	新增關於定義機器學習轉換中 AWS Glue 版本的資訊，以指出機器學習轉換相容哪個 AWS Glue 版本。如需詳細資訊，請參閱在 AWS Glue 主控台上使用機器學習轉換。	2019 年 11 月 21 日
支援倒轉您的任務書籤	已新增有關將您的工作書籤倒轉至任何先前的任務執行，導致後續任務只會從已加入書籤的任務執行重新處理資料的資訊。說明 `job-bookmark-pause` 選項兩個新的子選項，可讓您在兩個書籤之間執行任務。如需詳細資訊，請參閱使用任務書籤追蹤已處理的資料和 AWS Glue 使用的特殊參數。	2019 年 10 月 22 日
支援自訂 JDBC 憑證以連接到資料存放區	新增有關 AWS Glue 支援自訂 JDBC 憑證以 SSL 連線到 AWS Glue 資料來源或目標的資訊。如需詳細資訊，請參閱在 AWS Glue 主控台上使用連線。	2019 年 10 月 10 日
支援 Python Wheel	新增有關 AWS Glue 支援 wheel 檔案 (以及 egg 檔案) 做為 Python shell 任務相依性的資訊。如需詳細資訊，請參閱提供自己的 Python 程式庫。	2019 年 9 月 26 日

在 AWS Glue 中支援開發端點的版本控制	新增定義開發端點中的 `Glue version` 資訊。`Glue version` 決定 AWS Glue 支援的 Apache Spark 與 Python 版本。如需詳細資訊，請參閱新增開發端點。	2019 年 9 月 19 日
支援使用 Spark UI 監控 AWS Glue	新增有關使用 Apache Spark UI 監控和偵錯在 AWS Glue 任務系統上執行的 AWS Glue ETL 任務，以及在 AWS Glue 開發端點上的 Spark 應用程式。如需詳細資訊，請參閱使用 Spark UI 監控 AWS Glue。	2019 年 9 月 19 日
增強功能以支援使用公開 AWS Glue ETL 程式庫在本機開發 ETL 指令碼	更新 AWS Glue ETL 程式庫內容，以反映目前支援 AWS Glue 1.0 版。如需詳細資訊，請參閱使用 AWS Glue ETL 程式庫在本機開發及測試 ETL 指令碼。	2019 年 9 月 18 日
支援在執行任務時排除 Amazon S3 儲存體方案	新增相關資訊以說明在執行會從 Amazon S3 讀取檔案或分割區的 AWS Glue ETL 任務時，排除 Amazon S3 儲存體方案的方式。如需詳細資訊，請參閱排除 Amazon S3 儲存體方案。	2019 年 8 月 29 日
支援使用公開 AWS Glue ETL 程式庫在本機開發 ETL 指令碼	新增相關資訊以說明如何在本機開發及測試 Python 和 Scala ETL 指令碼，而無需連線至網路。如需詳細資訊，請參閱使用 AWS Glue ETL 程式庫在本機開發及測試 ETL 指令碼。	2019 年 8 月 28 日

已知問題	新增相關資訊以說明 AWS Glue 中的已知問題。如需詳細資訊，請參閱 AWS Glue 的已知問題。	2019 年 8 月 28 日
支援在 AWS Glue 中進行機器學習轉換	新增有關 AWS Glue 所提供機器學習功能以建立自訂轉換的資訊。當您建立任務時，您可以建立這些轉換。如需詳細資訊，請參閱 AWS Glue 中的機器學習轉換。	2019 年 8 月 8 日
支援共用 Amazon Virtual Private Cloud	新增 AWS Glue 支援共用 Amazon Virtual Private Cloud 的相關資訊。如需詳細資訊，請參閱共享 Amazon VPC。	2019 年 8 月 6 日
支援 AWS Glue 中的版本控制	新增在任務屬性中定義 `Glue version` 的資訊。AWS Glue 版本決定 AWS Glue 支援的 Apache Spark 與 Python 版本。如需詳細資訊，請參閱在 AWS Glue 新增任務。	2019 年 7 月 24 日
支援開發端點的其他組態選項	針對具有記憶體密集型工作負載的開發端點，新增組態選項的相關資訊。有兩個新的組態供您選擇，以提供每個執行程式更多的記憶體。如需詳細資訊，請參閱在 AWS Glue 主控台上使用開發端點。	2019 年 7 月 24 日

支援使用工作流程執行擷取、傳輸和載入 (ETL) 活動	新增使用稱為工作流程的新結構來設計複雜多任務擷取、轉換和載入 (ETL) 活動的資訊。AWS Glue 可將其作為單一實體執行和追蹤。如需詳細資訊，請參閱在 AWS Glue 中使用工作流程執行複雜的 ETL 活動。	2019 年 6 月 20 日
支援 Python Shell 任務中的 Python 3.6	新增有關支援在 Python shell 任務中支援 Python 3.6 的資訊。您可以指定 Python 2.7 或 Python 3.6 作為任務屬性。如需詳細資訊，請參閱在 AWS Glue 新增 Python Shell 任務。	2019 年 6 月 5 日
支援虛擬私有雲端 (VPC) 終端	新增有關透過您的 VPC 中的界面端點直接連接到 AWS Glue 的資訊。使用 VPC 界面端點時，VPC 和 AWS Glue 之間的通訊會完全在 AWS 網路內安全地進行。如需詳細資訊，請參閱使用 AWS Glue 搭配 VPC 端點。	2019 年 6 月 4 日
支援 AWS Glue 任務的即時、持續日誌記錄。	已新增啟用及檢視即時 Apache Spark 工作記錄的相關資訊，CloudWatch 包括驅動程式記錄檔、每個執行程式記錄檔和 Spark 工作進度列。如需詳細資訊，請參閱持續記錄 AWS Glue 任務。	2019 年 5 月 28 日

支援將現有的 Data Catalog 資料表做為爬蟲程式來源	新增將現有 Data Catalog 資料表清單指定為爬蟲程式來源的相關資訊。爬蟲程式即可在新資料可用時，偵測資料表結構描述的變更、更新資料表定義，並註冊新的分割區。如需詳細資訊，請參閱爬蟲程式屬性。	2019 年 5 月 10 日
支援記憶體密集型任務的額外組態選項	新增含記憶體密集型工作負載之 Apache Spark 任務的組態選項相關資訊。有兩個新的組態供您選擇，以提供每個執行程式更多的記憶體。如需詳細資訊，請參閱在 AWS Glue 新增任務。	2019 年 4 月 5 日
支援 CSV 自訂分類器	新增使用自訂 CSV 分類器以推斷各種 CSV 資料之結構描述的相關資訊。如需詳細資訊，請參閱撰寫自訂分類器。	2019 年 3 月 26 日
Support 資 AWS 源標籤	已新增有關使用 AWS 資源標籤的資訊，以協助您管理和控制資AWS Glue源存取權。您可以將 AWS 資源標籤指派給中AWS Glue的工作、觸發器、端點和編目器。如需詳細資訊，請參閱 AWS Glue 中的AWS 標籤。	2019 年 3 月 20 日

Spark SQL 任務適用的 Data Catalog 支援	已新增有關將AWS Glue工作和開發端點設定為作 AWS Glue Data Catalog 為外部 Apache Hive 中繼存放區使用的資訊。這可讓任務和開發端點直接對存放於 AWS Glue Data Catalog資料目錄之資料表直接執行 Apache Spark SQL 查詢。如需詳細資訊，請參閱 Spark SQL 任務的AWS Glue Data Catalog 支援。	2019 年 3 月 14 日
支援 Python shell 任務	Python shell 任務的新增資訊和新增欄位 Maximum capacity (容量上限)。如需詳細資訊，請參閱在 AWS Glue 新增 Python Shell 任務。	2019 年 1 月 18 日
支援在對資料庫和資料表進行變更時的通知	針對資料庫、資料表和分割區 API 呼叫進行變更時產生之事件的新增資訊 您可以在 CloudWatch 事件中配置動作來回應這些事件。如需詳細資訊，請參閱使用事件自AWS Glue CloudWatch 動化。	2019 年 1 月 16 日
支援加密連線密碼	針對用於連線物件的加密密碼新增資訊。如需詳細資訊，請參閱加密連線密碼。	2018 年 12 月 11 日
支援資源層級的許可和以資源為基礎的政策	新增搭配 AWS Glue 使用資源層級的許可和以資源為基礎的政策的資訊。如需詳細資訊，請參閱 AWS Glue 中的安全性內的主題。	2018 年 10 月 15 日

Support SageMaker 筆記型電腦	已新增有關搭配AWS Glue開發端點使用 SageMaker 筆記本的資訊 如需詳細資訊，請參閱管理筆記本。	2018 年 10 月 5 日
加密支援	新增相關資訊以說明如何透過 AWS Glue 使用加密功能。如需詳細資訊，請參閱靜態加密、傳輸中加密，以及設定 AWS Glue 中的加密。	2018 年 8 月 24 日
支援 Apache Spark 任務指標	新增有關使用 Apache Spark 指標的訊息，以便更佳的偵錯並分析 ETL 任務。您可以輕鬆追蹤執行時間指標，例如讀取和寫入的位元組、驅動程式和執行器的記憶體使用量和 CPU 負載，以及來自 AWS Glue 主控台的執行器之間的資料隨機排序。如需詳細資訊，請參閱 AWS Glue使用測 CloudWatch 量結果監視、Job 監視和偵錯，以及在AWS Glue主控台上使用工作。	2018 年 7 月 13 日
支援 DynamoDB 做為資料來源	新增有關爬取 DynamoDB 和使用它做為 ETL 任務的資料來源。如需詳細資訊，請參閱以爬蟲程式編目資料表和連線參數。	2018 年 7 月 10 日
對建立筆記本伺服器流程的更新	有關如何在與開發端點關聯的 Amazon EC2 執行個體建立筆記本伺服器的資訊。如需詳細資訊，請參閱建立和開發端點關聯的筆記本伺服器。	2018 年 7 月 9 日

現在可以透過 RSS 獲得更新	您現在可以訂閱更新 RSS 訊息，以接收 AWS Glue 開發人員指南的更新通知。	2018 年 6 月 25 日
支援任務的延遲通知	新增任務執行時有關設定延遲閾值的相關資訊。如需詳細資訊，請參閱在 AWS Glue 新增任務。	2018 年 5 月 25 日
設定爬蟲程式以附加新欄	已新增搜尋器新組態選項的相關資訊。 MergeNewColumns 如需詳細資訊，請參閱設定爬蟲程式。	2018 年 5 月 7 日
支援任務逾時	新增當任務執行時有關設定逾時閾值的相關資訊。如需詳細資訊，請參閱在 AWS Glue 新增任務。	2018 年 4 月 10 日
支援 Scala ETL 指令碼和根據額外的執行狀態來觸發任務	加入了使用 Scala 作為 ETL 程式設計語言的相關資訊。此外，觸發 API 現在支援在符合任何條件時觸發 (除了在符合所有條件時觸發之外)。另外，也可以根據「失敗的」或「停止的」任務執行來觸發任務 (除了根據「成功的」任務執行來觸發之外)。	2018 年 1 月 12 日

舊版更新

下表說明 2018 年一月前每個 AWS Glue 開發人員指南版本的重要變更。

變更	描述	日期
支援 XML 資料來源與新的爬蟲程式組態選項	針對 XML 資料來源的分類和變更分割區用的新爬蟲程式選項，新增了相關的資訊。	2017 年 11 月 16 日
新的轉換功能、支援其他 Amazon RDS 資料庫引擎，以及開發端點的增強功能	加入關於映射與篩選轉換的資訊、Amazon RDS Microsoft SQL Server 和 Amazon RDS Oracle 的支援，以及開發端點的新功能。	2017 年 9 月 29 日
AWS Glue 初始版本	這是初版的 AWS Glue 開發人員指南。	2017 年 8 月 14 日

AWS 詞彙表

有關最新 AWS 術語，請參閱AWS 詞彙表 參考文獻中的AWS 詞彙表。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。