aws

用户指南

AWS Glue

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Glue: 用户指南

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

什么是 AWS Glue？

AWS Glue 是一项无服务器数据集成服务，可让使用分析功能的用户轻松发现、准备、移动和集成来自多个来源的数据。您可以将其用于分析、机器学习和应用程序开发。它还包括用于编写、运行任务和实施业务工作流程的额外生产力和数据操作工具。

通过使用 AWS Glue，您可以发现并连接到 70 多个不同的数据来源，并在集中式数据目录中管理您的数据。您可以直观地创建、运行和监控“提取、转换、加载（ETL）”管道，以将数据加载到数据湖中。此外，您可以使用 Amazon Athena、Amazon EMR 和 Amazon Redshift Spectrum 立即搜索和查询已编目数据。

AWS Glue 将主要数据集成功能整合到一项服务中。其中包括数据发现、现代 ETL、清理、转换和集中式编目。这也是一项无服务器服务，即无需管理基础设施。通过在一项服务中灵活支持 ETL、ELT 和流式传输之类的所有工作负载，AWS Glue 可为不同工作负载和类型的用户提供支持。

此外，AWS Glue 可以轻松地在您的架构中集成数据。它可与 AWS 分析服务和 Amazon S3 数据湖集成。AWS Glue 具有集成式界面和任务编写工具，对于从开发人员到业务用户在内的所有用户来说，使用十分方便，还可针对不同的技术技能组合提供定制解决方案。

AWS Glue 可按需扩展，因此可帮助您专注于能最大限度地提高数据价值的高价值活动。可针对任何数据大小进行扩展，并支持所有数据类型和架构变化。为了提高灵活性并优化成本，AWS Glue 提供内置的高可用性和即付即用计费模式。

有关定价信息，请参阅 AWS Glue 定价。

AWS Glue Studio

AWS Glue Studio 采用图形界面，能让您轻松创建、运行和监控 AWS Glue 中的数据集成任务。您可以直观地编写数据转换工作流，并在 AWS Glue 中的基于 Apache Spark 的无服务器 ETL 引擎上无缝运行。

使用 AWS Glue Studio，您能够创建并管理收集、转换和清理数据的任务。您还可以使用 AWS Glue Studio 进行问题排查并编辑任务脚本。

主题

- AWS Glue 功能
- 了解 AWS Glue 中的创新
- 开始使用 AWS Glue

AWS Glue 功能

AWS Glue 功能分为三大类：

- 发现和整理数据
- 转换、准备和清理数据以进行分析
- 构建和监控数据管道

发现和整理数据

- 跨多个数据存储的统一和搜索 – 通过对 AWS 中的所有数据进行编目，跨多个数据来源和接收器进行存储、索引和搜索。
- 自动发现数据 – 使用 AWS Glue 爬网程序自动推断架构信息并将其集成到 AWS Glue Data Catalog。
- 管理架构和权限 – 验证和控制对数据库和表的访问。
- 连接到各种数据来源 – 利用本地和 AWS 的多个数据来源，使用 AWS Glue 连接构建您的数据湖，从而了解多个数据源。

转换、准备和清理数据以进行分析

- 使用作业画布界面直观地转换数据 – 在可视任务编辑器中定义 ETL 流程，并自动生成用于提取、转换和加载数据的代码。
- 通过简单的任务计划构建复杂的 ETL 管道 – 按计划、按需或按事件调用 AWS Glue 任务。
- 清理和转换传输中流数据 – 支持持续性的数据使用，并在传输过程中对其进行清理和转换。这样便可在数秒内在目标数据存储中完成分析。
- 通过内置的机器学习去除重复数据和清理数据 – 使用 `FindMatches` 功能，您无需成为机器学习专家也能轻松清理和准备数据以进行分析。此功能可去除重复项并查找彼此不完全匹配的记录。
- 内置任务笔记本 – 仅需在 AWS Glue 中进行最少设置，AWS Glue 任务笔记本即可提供无服务器笔记本，以便于您快速开始使用。
- 编辑、调试和测试 ETL 代码 – 通过 AWS Glue 交互式会话，您能够以交互方式探索和准备数据。您可以使用 IDE 或自己选择的笔记本以交互方式探索数据、对数据进行试验以及处理数据。

- 定义、检测和修复敏感数据 – AWS Glue 的敏感数据检测功能可让您定义、识别和处理数据管道和数据湖中的敏感数据。

构建和监控数据管道

- 根据工作负载自动扩展 – 根据工作负载动态扩展和缩减资源。仅在需要时才为工作人员分配任务。
- 使用基于事件的触发器自动处理任务 – 使用基于事件的触发器启动爬网程序或 AWS Glue 任务，并设计相互依赖的任务与爬网程序链。
- 运行和监控作业 - 使用您选择的引擎（Spark 或 Ray）运行 AWS Glue 作业。使用自动监控工具 AWS Glue 作业运行见解和 AWS CloudTrail 对其进行监控。使用 Apache Spark 用户界面改善对 Spark 支持的作业的监控。
- 定义 ETL 和集成活动的工作流程 – 为多个爬网程序、任务和触发器定义 ETL 和集成活动的工作流程。

了解 AWS Glue 中的创新

了解 AWS Glue 中最新的创新，听听客户如何使用 AWS Glue 在整个组织中实现自助式数据准备。

了解客户如何在传统设置之外扩展 AWS Glue，以及他们如何针对作业监控和性能配置 AWS Glue。

开始使用 AWS Glue

我们建议您首先阅读以下部分：

- AWS Glue 使用概述
- AWS Glue 概念
- 为 AWS Glue 设置 IAM 权限
- AWS Glue Data Catalog 入门
- 在 AWS Glue 中编写任务
- 开始使用 AWS Glue 交互式会话
- 在 AWS Glue 中编排

访问 AWS Glue

可以使用以下界面创建、查看和管理您的 AWS Glue 任务：

- AWS Glue 控制台 – 提供 Web 界面供您创建、查看和管理 AWS Glue 任务。要访问此控制台，请参阅 AWS Glue。
- AWS Glue Studio – 提供图形界面供您直观地创建和编辑 AWS Glue 任务。有关更多信息，请参阅什么是 AWS Glue Studio。
- AWS CLI 参考的 AWS Glue 部分 – 提供可与 AWS Glue 配合使用的 AWS CLI 命令。有关更多信息，请参阅适用于 AWS Glue 的 AWS CLI 参考。
- AWS Glue API – 为开发人员提供完整的 API 参考。有关更多信息，请参阅 AWS Glue API。

相关服务

AWS Glue 的用户也使用：

- AWS Lake Formation – 此服务是授权层，提供对 AWS Glue Data Catalog 中的资源访问权的精细控制。
- AWS Glue DataBrew – 是一种可视化数据准备工具，让您无需编写任何代码即可清理数据并实现标准化。

AWS Glue：工作原理

AWS Glue 使用其他 AWS 服务以协调您的 ETL（提取、转换和加载）任务来构建数据仓库或数据湖，并生成输出流。AWS Glue 调用 API 操作来转换您的数据、创建运行时日志、存储您的任务逻辑，以及创建通知来帮助您监控任务运行。AWS Glue 控制台将这些服务连接到托管应用程序，因此您可以专注于创建和监控您的 ETL 工作。控制台代表您执行管理和作业开发操作。您可以向 AWS Glue 提供凭证和其他属性，以访问您的数据源并将内容写入数据目标。

AWS Glue 将负责预置和管理运行您的工作负载所需的资源。您不需要为 ETL 工具创建基础设施，因为 AWS Glue 为您提供了它。当需要资源时，为了减少启动时间，AWS Glue 会使用其热实例池中的实例来运行您的工作负载。

借助 AWS Glue，您可以使用数据目录中的表定义创建任务。作业由包含执行转换的编程逻辑的脚本组成。您可以使用触发器按计划或作为指定事件的结果启动作业。您可以确定目标数据驻留的位置以及哪些源数据填充目标。借助您的输入，AWS Glue 生成将您的数据从源转换到目标必需的代码。您还可以在 AWS Glue 控制台或 API 中提供脚本来处理您的数据。

数据来源和目标

AWS Glue for Spark 允许您从多个系统和数据库读取和写入数据，包括：

- Amazon S3
- Amazon DynamoDB
- Amazon Redshift
- Amazon Relational Database Service (Amazon RDS)
- 第三方 JDBC 可访问的数据库
- MongoDB 和 Amazon DocumentDB (with MongoDB compatibility)
- 其他市场连接器和 Apache Spark 插件

数据流

AWS Glue for Spark 可以流式传输来自以下系统的数据：

- Amazon Kinesis Data Streams
- Apache Kafka

AWS Glue 可在多个 AWS 区域中使用。有关更多信息，请参阅 AWS 中的 Amazon Web Services 一般参考 区域和终端节点。

主题

无服务器 ETL 作业在隔离环境中运行

AWS Glue 在您选择的引擎 Spark 或 Ray 中在无服务器环境中运行 ETL 作业。AWS Glue 在用其自己的服务账户预置和管理的虚拟资源上运行这些作业。

AWS Glue 专用于执行以下操作：

- 隔离客户数据。
- 保护传输中的和静态的客户数据。
- 仅在响应客户请求过程中需要时，才使用临时的、限定范围的凭证或客户对其账户中的 IAM 角色的同意来访问客户数据。

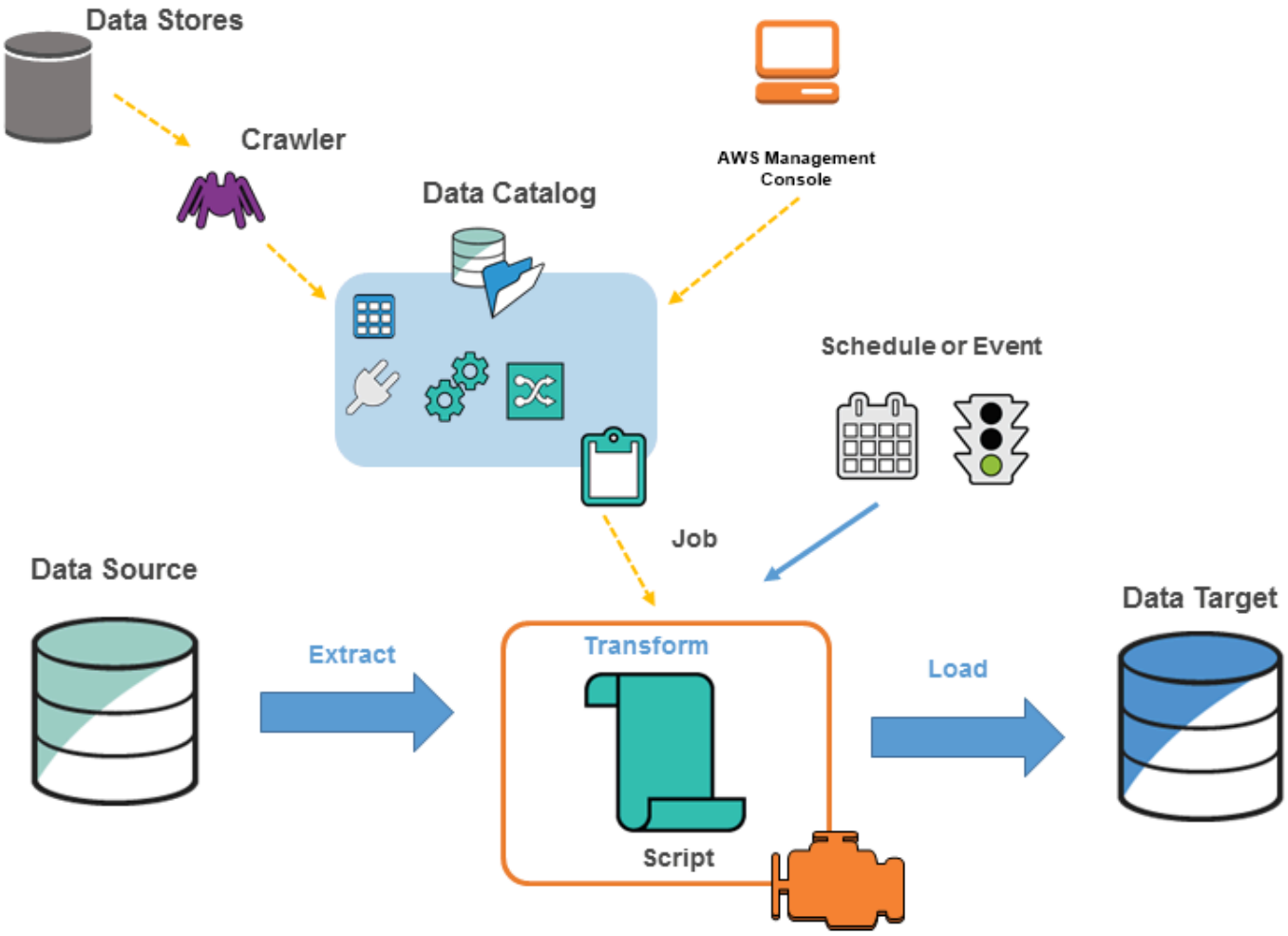
在 ETL 作业的预置过程中，您可以在 Virtual Private Cloud（VPC）中提供输入数据源和输出数据目标。此外，您还提供访问数据源和目标所需的 IAM 角色、VPC ID、子网 ID 和安全组。对于每个元组（客户账户 ID、IAM 角色、子网 ID 和安全组），AWS Glue 会创建一个新环境，此环境在网络和管理级别上与 AWS Glue 服务账户内的所有其他环境隔离。

AWS Glue 使用私有 IP 地址在子网中创建弹性网络接口。作业使用这些弹性网络接口来访问数据来源和数据目标。进出和位于作业运行环境中的流量是由您的 VPC 和网络策略控制的，但有一个例外：对 AWS Glue 库的调用可以通过 AWS Glue VPC 将流量委托给 AWS Glue API 操作。所有 AWS Glue API 调用都被记录下来，因此，数据所有者可以通过启用将审核日志提供给您账户的 AWS CloudTrail 来审核 API 访问权限。

运行 ETL 任务的 AWS Glue 托管环境受到与其他 AWS 服务遵循的相同安全做法的保护。有关实践和安全责任共担的概述，请参阅 AWS 安全流程简介白皮书。

AWS Glue 概念

下图表明了 AWS Glue 环境的架构。



您可以在 中定义作业AWS Glue来完成从数据源提取、转换和加载 (ETL) 数据到数据目标所需的工作。您通常会执行以下操作：

- 对于数据存储源，您定义爬网程序 以使用元数据表定义填充 AWS Glue Data Catalog。您将爬网程序指向数据存储，并且爬网程序在数据目录中创建表定义。对于流式处理源，您可以手动定义数据目录表并指定数据流属性。

  除了表定义，AWS Glue Data Catalog 还包含定义其他 ETL 作业所需的其他元数据。在定义作业以转换数据时，您可以使用此元数据。

- AWS Glue 可以生成用于转换数据的脚本。或者，您可以在 AWS Glue 控制台或 API 中提供脚本。

- 您可以按需运行任务，也可以将其设置为在指定的触发器发生时启动。触发器可以是基于时间的计划或事件。

  当您的任务运行时，脚本从数据源中提取数据，转换数据，并将其加载到数据目标。该脚本在 AWS Glue 中的 Apache Spark 环境内运行。

> **Important**
>
> AWS Glue 中的表和数据库是 AWS Glue Data Catalog 中的对象。它们包含元数据；它们不包含数据存储中的数据。

基于文本的数据（如 CSV）必须采用 **UTF-8** 进行编码，以便 AWS Glue 成功处理该数据。有关更多信息，请参阅 Wikipedia 中的 UTF-8。

AWS Glue 术语

AWS Glue 依赖于多个组件之间的交互来创建和管理您的提取、转换、加载（ETL）工作流程。

AWS Glue Data Catalog

AWS Glue 中的持久元数据存储。它包含表定义、作业定义以及其他用于管理您的 AWS Glue 环境的控制信息。每个 AWS 账户在每个区域有一个 AWS Glue Data Catalog。

分类器

确定您的数据的架构。AWS Glue 提供常见文件类型的分类器，如 CSV、JSON、Avro、XML 等。此外，它还使用 JDBC 连接为常用关系数据库管理系统提供分类器。您可以使用 grok 模式或在 XML 文档中指定行标记来编写自己的分类器。

Connection

数据目录对象，其中包含了连接到特定数据存储所需的属性。

爬网程序

该程序连接到数据存储（源或目标），通过分类器的优先级列表不断更新以确定您的数据架构，然后在 AWS Glue Data Catalog 中创建元数据表。

数据库

一组关联的数据目录表定义，这些定义组织到一个逻辑组内。

数据存储、数据源、数据目标

数据存储是持久存储数据的存储库。示例包括 Amazon S3 存储桶和关系数据库。数据源是用作进程或转换输入的数据存储。数据目标是进程或转换写入的数据存储。

开发终端节点

一种可以用来开发和测试 AWS Glue ETL 脚本的环境。

动态帧

支持嵌套数据（如结构和数组）的分布式表。每条记录都是自我描述，旨在使用半结构化数据实现架构灵活性。每条记录都包含数据和描述该数据的架构。您可以在 ETL 脚本中同时使用动态帧和 Apache Spark DataFrames，并在它们之间进行转换。动态帧为数据清理和 ETL 提供了一系列转换。

作业

执行 ETL 工作所需的业务逻辑。它由转换脚本、数据源和数据目标组成。作业运行通过可由事件计划或触发的触发器启动。

任务性能控制面板

AWS Glue 为您的 ETL 任务提供一个全面的运行控制面板。控制面板显示特定时间范围内运行的任务的相关信息。

笔记本界面

通过一键式设置增强了笔记本体验，便于作业创作和数据探索。笔记本和连接都将自动为您配置。您可以使用基于 Jupyter Notebook 的笔记本界面，使用 AWS Glue 无服务器 Apache Spark ETL 基础设施以交互方式开发、调试和部署脚本和工作流。您还可以在笔记本环境中执行临时查询、数据分析和可视化（例如，表格和图表）。

Script

从源中提取数据、转换并将其加载到目标中的代码。AWS Glue 会生成 PySpark 或 Scala 脚本。

表

表示您的数据的元数据定义。无论您的数据是位于 Amazon Simple Storage Service（Amazon S3）文件、Amazon Relational Database Service（Amazon RDS）表还是其他数据集中，表都定义了数据的架构。AWS Glue Data Catalog 中的表包含列的名称、数据类型定义、分区信息以及有关基本数据集的其他元数据。数据的架构在您的 AWS Glue 表定义中表示。实际数据留存在原始数据存储中，无论是在文件中还是关系数据库表中。AWS Glue 在 AWS Glue Data Catalog 中编录您的文件和关系数据库表。在您创建 ETL 任务时，它们用作源和目标。

转换

用于将数据处理为其他格式的代码逻辑。

触发器

启动 ETL 任务。可以根据计划时间或事件来定义触发器。

可视化任务编辑器

可视化作业编辑器是一个图形界面，可以方便地在 AWS Glue 中创建、运行和监控提取、转换和加载（ETL）任务。您可以直观地编写数据转换工作流，并在 AWS Glue 的基于 Apache Spark 的无服务器 ETL 引擎上顺畅运行，检查作业的每个步骤中的模式和数据结果。

工作线程

借助 AWS Glue，您只需按 ETL 任务运行所需时间付费。没有要管理的资源，无预付费用，您无需为启动或关闭时间付费。您将根据用于运行 ETL 任务的数据处理单元（缩写为 DPU）的数量按小时费率支付费用。单个数据处理单元 (DPU) 也称为一个工件。AWS Glue 附带三种工件类型，可帮助您选择符合任务延迟和成本要求的配置。Worker 的配置值包括 Standard、G.1X、G.2X 和 G.025X。

AWS Glue 组件

AWS Glue 提供控制台和 API 操作来设置和管理您的提取、转换和加载 (ETL) 工作负载。您可以通过多个特定于语言的开发工具包和 AWS Command Line Interface (AWS CLI) 来使用 API 操作。有关使用 AWS CLI 的信息，请参阅 AWS CLI 命令参考。

AWS Glue 使用 AWS Glue Data Catalog来存储有关数据源、转换和目标的元数据。数据目录是 Apache Hive 元存储的简易替代。AWS Glue Jobs system提供用于为您的数据定义、安排和运行 ETL 操作的托管基础设施。有关 AWS Glue API 的更多信息，请参阅AWS Glue API。

AWS Glue 控制台

您可以使用 AWS Glue 控制台来定义和协调您的 ETL 工作流程。该控制台在 AWS Glue Data Catalog 和 AWS Glue Jobs system中调用多个 API 操作以执行以下任务：

- 定义 AWS Glue 对象，如作业、表、爬网程序和连接。
- 安排爬网程序的运行时间。
- 为作业触发器定义事件或计划。
- 搜索和筛选 AWS Glue 对象的列表。
- Edit 转换脚本。

AWS Glue Data Catalog

AWS Glue Data Catalog 是您在 AWS 云中的持久性技术元数据存储。

每个 AWS 账户在每个 AWS 区域有一个 AWS Glue Data Catalog。每个数据目录都是组织成数据库的高度可扩展的表集合。表是存储在 Amazon RDS、Apache Hadoop Distributed File System、Amazon OpenSearch Service 等源中的结构化或半结构化数据集合的元数据表示形式。AWS Glue Data Catalog 提供了一个统一的存储库，不同的系统可以在其中存储和查找元数据来跟踪数据孤岛中的数据。然后，您可以使用元数据在各种应用程序中以统一的方式查询和转换该数据。

您将数据目录与 AWS Identity and Access Management 策略和 Lake Formation 一同使用，从而控制对表和数据库的访问。通过这样做，您可以允许企业中的不同组将数据安全地发布到更广泛的组织，同时以高度精细的方式保护敏感信息。

数据目录以及 CloudTrail 和 Lake Formation 还提供全面的审计和监管功能，其中有架构更改跟踪和数据访问控制。这有助于确保数据不会被不当修改或无意中共享。

有关保护及审计 AWS Glue Data Catalog 的信息，请参阅：

- AWS Lake Formation – 有关更多信息，请参阅《AWS Lake Formation 开发人员指南》中的 什么是 AWS Lake Formation？
- CloudTrail – 有关更多信息，请参阅 AWS CloudTrail 用户指南中的什么是 CloudTrail？

以下是其他 AWS 服务和使用 AWS Glue Data Catalog 的开源项目：

- Amazon Athena – 有关更多信息，请参阅《Amazon Athena 用户指南》中的 了解表、数据库和数据目录。

- Amazon Redshift Spectrum – 有关更多信息，请参阅《Amazon Redshift 数据库开发人员指南》中的 使用 Amazon Redshift Spectrum 查询外部数据。
- Amazon EMR – 有关更多信息，请参阅《Amazon EMR 管理指南》中的 使用基于资源的策略实现 Amazon EMR 对 AWS Glue Data Catalog 的访问。
- Apache Hive 元存储的 AWS Glue Data Catalog 客户端 – 有关此 GitHub 项目的更多信息，请参阅 Apache Hive 元存储的 AWS Glue Data Catalog 客户端。

AWS Glue 爬网程序和分类器

AWS Glue 还能让您设置爬网程序，它可以扫描所有类型的存储库中的数据，对其进行分类，从中提取架构信息，并自动在 AWS Glue Data Catalog中存储元数据。然后 AWS Glue Data Catalog 可用于指导 ETL 操作。

有关如何设置爬网程序和分类器的信息，请参阅使用爬网程序填充 Data Catalog 。有关如何使用 AWS Glue API 编程爬网程序和分类器的信息，请参阅爬网程序和分类器 API。

AWS Glue ETL 操作

通过使用数据目录中的元数据，AWS Glue 可以自动生成具有 AWS Glue 扩展的 Scala 或 PySpark（用于 Apache Spark 的 Python API）脚本，您可以使用和修改它来执行各种 ETL 操作。例如，您可以提取、清除和转换原始数据，然后将结果存储在不同的存储库中，以便可以对其进行查询和分析。此类脚本可能会将 CSV 文件转换为关系形式并将其保存到 Amazon Redshift 中。

有关如何使用 AWS Glue ETL 功能的更多信息，请参阅Spark 脚本编程。

AWS Glue 中的流式处理 ETL

通过 AWS Glue，您能够使用连续运行的任务对流数据执行 ETL 操作。AWS Glue 流式处理 ETL 基于 Apache Spark Structured Streaming 引擎而构建，可以从 Amazon Kinesis Data Streams、Apache Kafka 和 Amazon Managed Streaming for Apache Kafka（Amazon MSK）提取流。流式处理 ETL 可以清理和转换流数据，并将其加载到 Amazon S3 或 JDBC 数据存储中。在 AWS Glue 中使用流式处理 ETL 可以处理 IoT 流、点击流和网络日志等事件数据。

如果您知道流数据源的架构，则可以在数据目录表中指定该架构。如果没有，则可以在流式 ETL 任务中启用架构检测。然后，任务会根据传入的数据自动确定架构。

流式处理 ETL 任务可以同时使用 AWS Glue 内置转换和 Apache Spark Structured Streaming 的原生转换。有关更多信息，请参阅 Apache Spark 网站上的流式处理 DataFrame/数据集的操作。

有关更多信息，请参阅 the section called “流式处理 ETL 作业”。

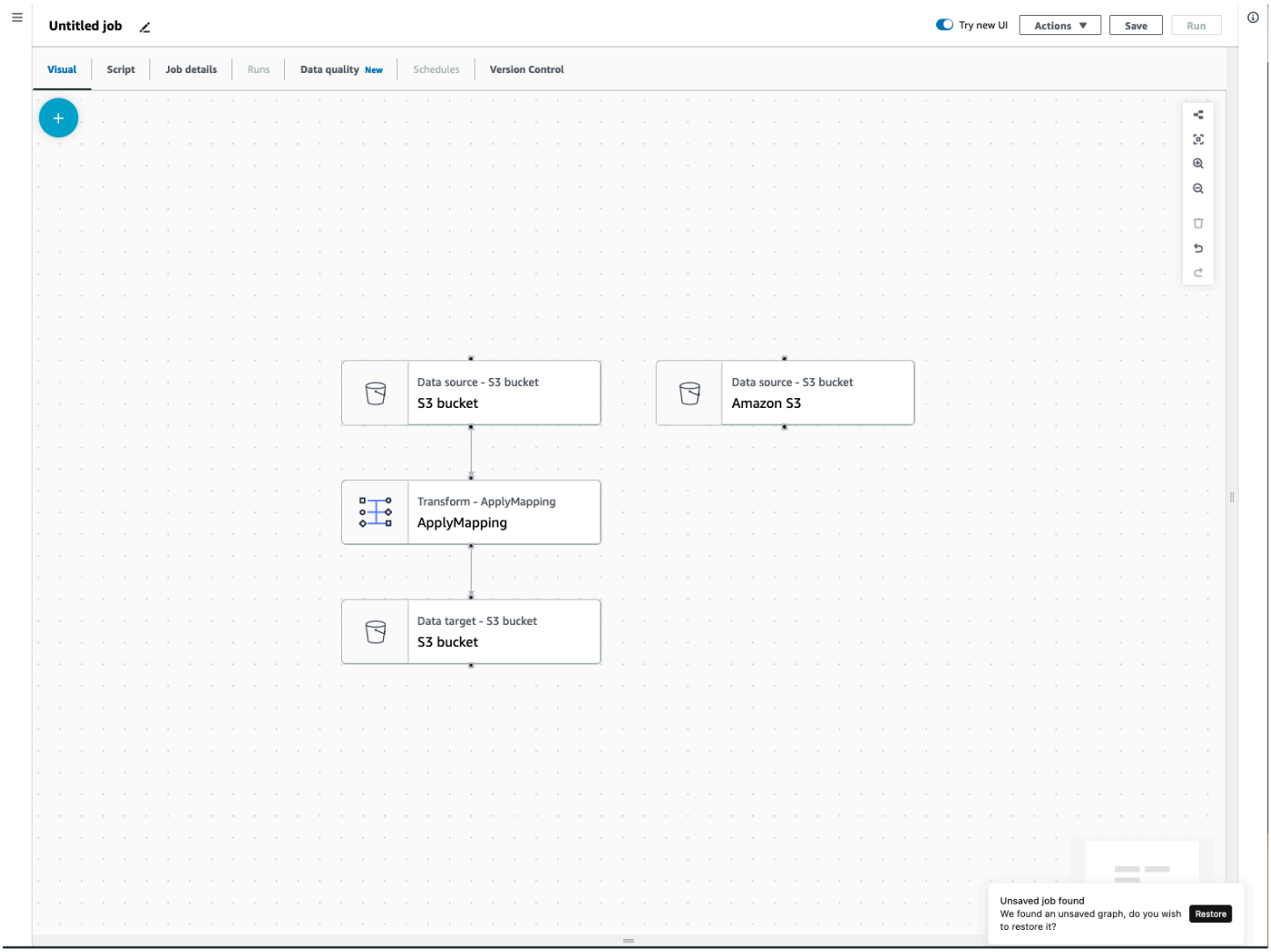
AWS Glue 作业系统

AWS Glue Jobs system提供托管基础设施以协调 ETL 工作流程。您可以在 AWS Glue 中创建作业，用于自动处理您用于提取、转换数据并将数据传输到不同位置的脚本。作业可以安排和串联，也可以由诸如新数据到达之类的事件触发。

有关如何使用 AWS Glue Jobs system的更多信息，请参阅监控 AWS Glue。有关使用 AWS Glue Jobs system API 编程的信息，请参阅作业 API。

Visual ETL 组件

使用 AWS Glue 可以通过可操作的可视画布创建 ETL 作业。



ETL 作业菜单

使用画布顶部的菜单选项可以访问有关作业的各种视图和配置详细信息。

- 可视化 — 可视化任务编辑器画布。您可以在此处添加节点以创建任务。
- 脚本 — 您的 ETL 作业的脚本表示形式。AWS Glue 根据作业的可视化表示生成脚本。您也可以编辑脚本或下载脚本。

> **Note**
>
> 如果您选择编辑脚本，则作业创作体验将永久转换为纯脚本模式。之后，您将无法再使用可视化编辑器来编辑作业。在选择编辑脚本之前，您应该添加所有作业源、转换和目标，并使用可视化编辑器进行所需的所有更改。

- 作业详细信息 —“作业详细信息”选项卡允许您通过设置作业属性来配置作业。有基本属性，例如您的作业名称和描述、IAM 角色、作业类型、AWS Glue 版本、语言、工作线程类型、工作线程数量、作业书签、弹性执行、重试次数和作业超时，还有一些高级属性，例如连接、库、作业参数和标签。
- 运行 — 作业运行后，可以访问此选项卡以查看您过去的作业运行情况。
- 数据质量 — 数据质量评估和监控数据资产的质量。您可以在此选项卡上详细了解如何使用数据质量，并在作业中添加数据质量转换。
- 计划 — 您已计划的作业显示在此选项卡中。如果此作业没有附加计划，则无法访问此选项卡。
- 版本控制 — 您可以将作业配置到 Git 存储库，从而将 Git 用于您的作业。

Visual ETL 面板

当您在画布中工作时，有几个面板可以帮助您配置节点，或者帮助您预览数据和查看输出架构。

- 属性 — 当您在画布上选择节点时，将出现“属性”面板。
- 数据预览 —“数据预览”面板提供数据输出的预览，这样您就可以在运行作业和检查输出之前做出决策。
- 输出架构 —“输出架构”选项卡允许您查看和编辑转换节点的架构。

调整面板大小

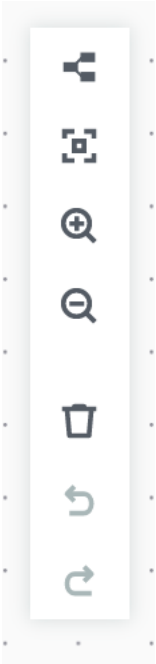
您可以调整屏幕右侧的“属性”面板以及包含“数据预览”和“输出架构”选项卡的底部面板的大小，方法是单击面板边缘并向左和向右或向上和向下拖动。

- 属性面板 — 通过单击并拖动屏幕右侧画布边缘来调整属性面板的大小，然后向左拖动以扩大其宽度。默认情况下，面板处于折叠状态，当选择节点时，属性面板会以其默认大小打开。
- 数据预览和输出架构面板 — 通过单击并拖动屏幕底部画布的底部边缘来调整底部面板的大小，然后向上拖动以扩大其高度。默认情况下，面板处于折叠状态，当选择节点时，底部面板会以其默认大小打开。

作业画布

您可以直接在 Visual ETL 画布上添加、移除和移动/重新排序节点。可以把它想象成您的工作空间，用来创建一个功能齐全的 ETL 作业，该作业从数据来源开始，可以以数据目标结尾。

当您在画布上处理节点时，您可以使用一个工具栏帮助您放大和缩小、移除节点、建立或编辑节点之间的连接、更改作业流方向以及撤销或重做操作。



浮动工具栏固定在画布的右上角大小，其中包含几张执行操作的图像：

- 布局图标 — 工具栏中的第一个图标是布局图标。默认情况下，视觉作业的方向是从上到下。它通过从左到右水平排列节点来重新排列可视化作业的方向。再次单击布局图标将方向更改为从上到下。
- 重新居中图标 — 重新居中图标通过居中来更改画布视图。您可以用它来处理大型作业，回到中心位置。
- 放大图标 — 放大图标可放大画布上节点的大小。
- 缩小图标 — 缩小图标可缩小画布上节点的大小。

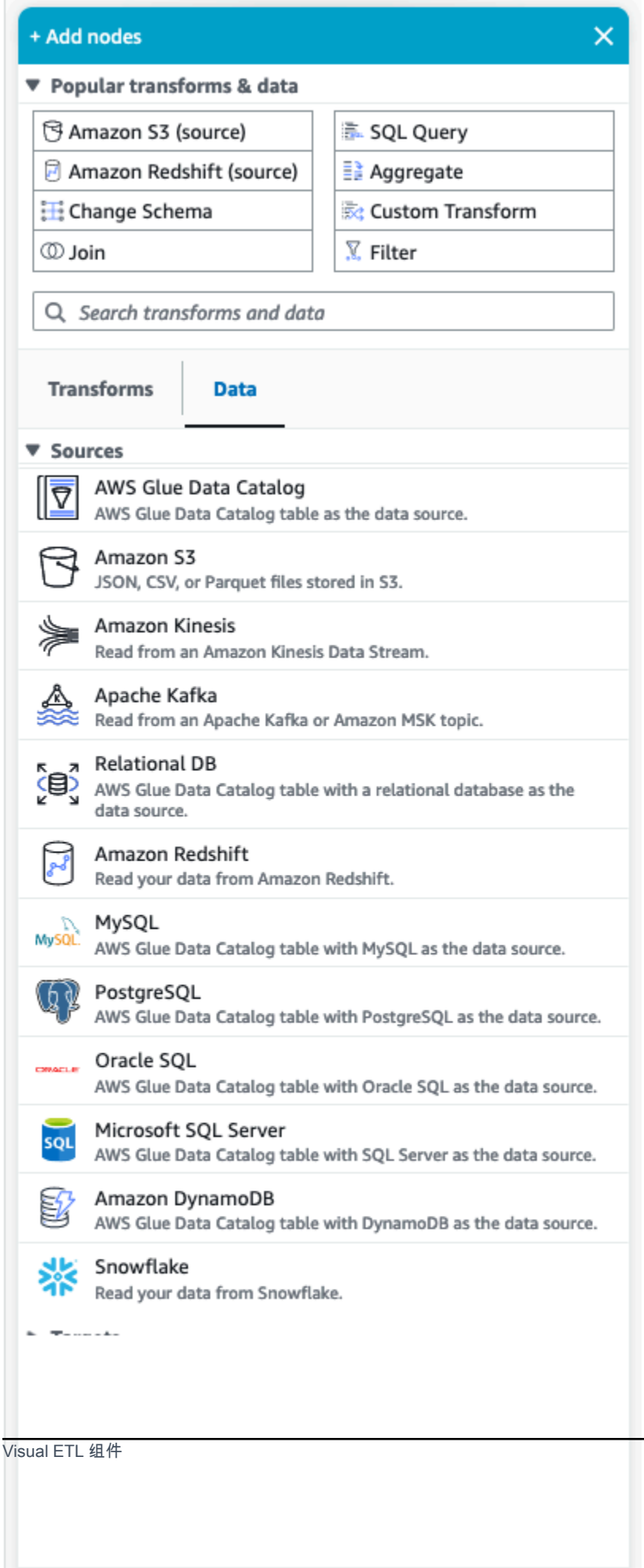
- 垃圾桶图标 — 垃圾桶图标将节点从可视化作业中移除。必须先选择一个节点。
- 撤销图标 — 撤销图标会撤销上次对可视化作业执行的操作。
- 重做图标 — 重做图标会重复对可视化作业执行的上一个操作。

使用迷你地图

资源面板

资源面板包含所有可用的数据来源、转换操作和连接。单击“+”图标在画布上打开资源面板。这将打开资源面板。

要关闭资源面板，请单击资源面板右上角的 X。这将隐藏面板，直到您准备好再次打开它为止。


+ 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.

常用转换和数据

面板顶部是常用转换和数据的集合。这些节点通常用于 AWS Glue。选择一个将其添加到画布中。也可以通过单击常用转换和数据标题旁边的三角形来隐藏常用转换和数据。

在常用转换和数据部分下方，您可以搜索转换和数据来源节点。在您键入时会显示结果。您在搜索查询中添加的字母越多，结果列表就会越小。搜索结果是根据节点名称和/或描述填充的。选择一个节点将其添加到画布中。

转换和数据

有两个选项卡可将节点组织为转换和数据。

转换 — 选择转换选项卡时，可以选择所有可用的转换。选择一个转换将其添加到画布中。您也可以选择转换列表底部的添加转换，这将打开一个用于创建自定义视觉转换的文档的新页面。按照这些步骤操作可以创建您自己的转换。然后，您的转换将出现在可用转换列表中。

数据 — 数据选项卡包含来源和目标的所有节点。您可以通过单击“来源”或“目标”标题旁边的三角形来隐藏“来源”和“目标”。您可以通过再次单击三角形来取消隐藏来源和目标。选择来源节点或目标节点将其添加到画布中。您也可以选择管理连接来添加新连接。这将在控制台中打开“连接”页面。

AWS Glue for Spark 和 AWS Glue for Ray

在 AWS Glue on Apache Spark（AWS Glue ETL），您可以使用 PySpark 编写 Python 代码来大规模处理数据。Spark 是解决这个问题的熟悉解决方案，但是具有以 Python 为中心背景的数据工程师可能会发现这种过渡并不直观。Spark DataFrame 模型不是无缝的“Pythonic”，它反映了它所构建的 Scala 语言和 Java 运行时。

在 AWS Glue 中，您可以使用 Python shell 作业来运行原生 Python 数据集成。这些作业在单个 Amazon EC2 实例上运行，并且受到该实例容量的限制。这限制了您可以处理的数据的吞吐量，并且在处理大数据时维护成本会很高。

AWS Glue for Ray 允许您纵向扩展 Python 工作负载，而无需在学习 Spark 上投入大量资金。您可以利用 Ray 表现某些更好的场景。通过为您提供选择，您可以利用 Spark 和 Ray 的优势。

AWS Glue ETL 和 AWS Glue for Ray 在底层是不同的，因此它们支持不同的功能。请查看文档以了解支持的功能。

AWS Glue for Ray 有什么用？

Ray 是一个开源分布式计算框架，您可以使用它来扩展工作负载，重点是 Python。有关 Ray 的更多信息，请参阅 Ray 网站。AWS GlueRay 作业和交互式会话允许您在 AWS Glue 中使用 Ray。

您可以使用 AWS Glue for Ray 为计算编写 Python 脚本，这些脚本将在多台机器上并行运行。在 Ray 作业和交互式会话中，您可以使用熟悉的 Python 库（例如，pandas），使您的工作流程易于编写和运行。有关 Ray 数据集的更多信息，请参阅 Ray 文档中的 Ray 数据集。有关 pandas 的更多信息，请参阅 Pandas 网站。

使用 AWS Glue for Ray 时，只需几行代码，即可针对企业级的大数据运行 pandas 工作流程。可从 AWS Glue 控制台或 AWS 开发工具包创建 Ray 作业。您也可以打开 AWS Glue 交互式会话，在无服务器 Ray 环境中运行您的代码。AWS Glue Studio 中尚不支持可视化作业。

AWS Glue for Ray 作业允许您按计划运行脚本或响应来自 Amazon EventBridge 的事件。作业将日志信息和监控统计信息存储在 CloudWatch 中，使您能够了解脚本的运行状况和可靠性。有关 AWS Glue 作业系统的更多信息，请参阅 the section called “处理 Ray 作业”。

AWS Glue for Ray 交互式会话（预览版）允许您针对相同的预配置资源一个接一个地运行代码片段。您可以使用它来高效地制作原型和开发脚本，或者构建自己的交互式应用程序。您可以在 AWS Management Console 中使用 AWS Glue Studio Notebooks 中的 AWS Glue 交互式会话。有关更多信息，请参阅将笔记本与 AWS Glue Studio 和 AWS Glue 结合使用。也可以通过 Jupyter 内核使用它们，它允许您从支持 Jupyter Notebooks 的现有代码编辑工具（例如，VSCode）运行交互式会话。有关更多信息，请参阅 the section called “AWS Glue 适用于 Ray 交互式会话（预览版）”。

Ray 可自动扩展 Python 代码的工作，方法是将处理分布在一组机器上，然后根据负载对其进行实时重新配置。这可以提高某些工作负载的每美元性能。借助 Ray 作业，我们在 AWS Glue 作业模型中原生内置了自动扩缩功能，因此您可以充分利用此功能。Ray 任务在 AWS Graviton 上运行，从而提高了整体性价比。

除了节省成本外，您还可以使用原生自动扩缩来运行 Ray 工作负载，而无需花时间进行集群维护、调整和管理。您可以开箱即用熟悉的开源库，比如 pandas 和适用于 Panda 的 AWS 开发工具包。这些可以提高您在 AWS Glue for Ray 上开发时的迭代速度。使用 AWS Glue for Ray 时，您将能够快速开发和运行具有成本效益的数据集成工作负载。

借助 AWS Glue 将半结构化架构转换为关系架构

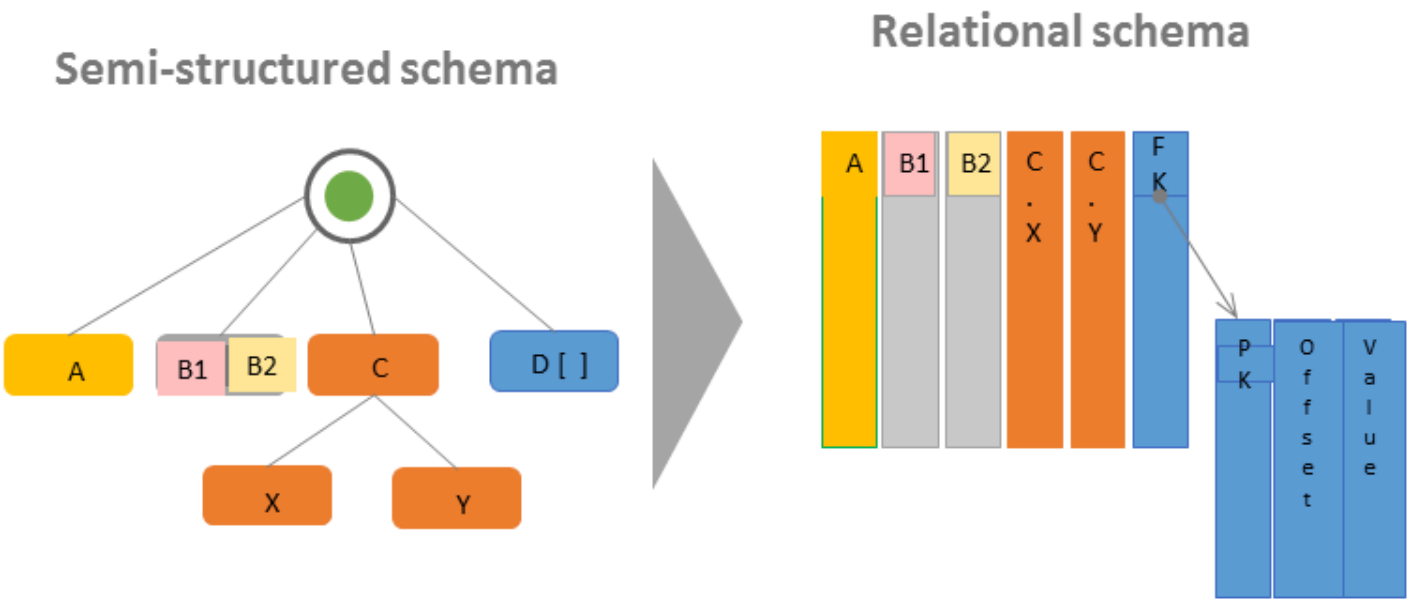
通常会想将半结构化数据转换为关系表。从概念上说，您是将分层架构展平成关系架构。AWS Glue 可以即时为您执行此转换。

半结构化数据通常包含标记以标识数据中的实体。它可以具有不带固定架构的嵌套数据结构。有关半结构化数据的更多信息，请参阅维基百科中的半结构化数据。

关系数据由包含行和列的表表示。表之间的关系可以由主键 (PK) 与外键 (FK) 关系表示。有关更多信息，请参阅维基百科中的关系数据库。

AWS Glue 使用爬网程序来推断半结构化数据的架构。然后，它使用 ETL (提取、转换和加载) 作业将数据转换为关系架构。例如，您可能想要将 Amazon Simple Storage Service（Amazon S3）源文件中的 JSON 数据解析为 Amazon Relational Database Service（Amazon RDS）表。了解 AWS Glue 如何处理架构之间的差异可以帮助您了解转换过程。

此图显示了 AWS Glue 如何将半结构化架构转换为关系架构。



该图阐释了以下内容：

- 单值 A 直接转换为关系列。

- 值对 B1 和 B2 转换为两个关系列。
- 结构 C 以及子结构 X 和 Y 转换为两个关系列。
- 数组 D[] 转换为具有指向另一个关系表的外键 (FK) 的关系列。除了主键 (PK)，第二个关系表还具有包含数组中项的偏移和值的列。

AWS Glue 类型系统

AWS Glue 使用多种类型的系统为以截然不同的方式存储数据的数据系统提供多功能接口。本文档消除了 Glue 类型 AWS 系统和数据标准的歧义。

AWS Glue 数据目录类型

Data Catalog 是存储在各种数据系统中的表和字段的注册表，即元存储。当 AWS Glue 组件（例如 AWS Glue crawlers 和 AWS Glue with Spark 作业）写入数据目录时，它们使用内部类型系统来跟踪字段的类型。这些值显示在 AWS Glue 控制台中表架构的数据类型列中。这种类型系统基于 Apache Hive 的类型系统。有关 Apache Hive 类型系统的更多信息，请参阅 Apache Hive 维基中的类型。有关特定类型和支持的更多信息， AWS Glue 控制台中提供了示例，该示例是架构生成器的一部分。

验证、兼容性和其他使用

Data Catalog 不验证写入类型字段的类型。当 AWS Glue 组件读取和写入数据目录时，它们将相互兼容。 AWS Glue 组件还旨在保持与 Hive 类型的高度兼容性。但是， AWS Glue 组件并不能保证与所有 Hive 类型兼容。这允许在处理 Data Catalog 中的表时与 Athena DDL 等工具进行互操作。

由于 Data Catalog 不验证类型，其他服务可能会使用数据目录来跟踪使用严格符合 Hive 类型系统的系统或任何其他系统的类型。

使用 Spark 脚本 AWS 在 Glue 中键入内容

当 AWS Glue with Spark 脚本解释或转换数据集时DynamicFrame，我们会提供脚本中使用的数据集的内存中表示形式。DynamicFrame 的目标与 Spark DataFrame 的目标类似：它会为数据集建模，这样 Spark 就可以对数据进行调度和执行转换。我们通过提供 toDF 和 fromDF 方法来保证 DynamicFrame 的类型表示形式与 DataFrame 相互兼容。

如果可以推断出类型信息或将其提供给 DataFrame，则可以推断出类型信息或将其提供给 DynamicFrame，除非另有记录。当我们为特定数据格式提供优化的读取器或写入器时，如果 Spark 可以读取或写入您的数据，我们提供的读取器和写入器也可以，会受记录限制。有关读取器和写入器的更多信息，请参阅 the section called “数据格式选项”。

选择类型

DynamicFrames 提供在数据集中为字段建模的机制，这些字段的值在磁盘上的不同行中可能具有不一致的类型。例如，一个字段可以在某些行中将数字存储为字符串，在其他行中则存储为整数。此机制是一种名为 Choice 的内存中类型。我们提供诸如ResolveChoice方法之类的变换，将 Choice 列解析为具体类型。 AWS 在正常操作过程中，Glue ETL 不会将选择类型写入数据目录；选择类型仅存在于数据集的 DynamicFrame 内存模型环境中。有关选择类型的使用示例，请参阅 the section called “数据准备示例”。

AWS Glue Crawler 类型

抓取工具的目标是为您的数据集生成一致且可用的架构，然后将其存储在数据目录中以用于其他 AWS Glue 组件和 Athena。爬网程序处理 Data Catalog 上一节中描述的类型，the section called “AWS Glue 数据目录类型”。为了在“选择”类型场景（其中一列包含两种或更多种类型的值）中生成可用类型，爬网程序将创建一种为潜在类型建模的 struct 类型。

开始使用 AWS Glue

以下部分提供有关设置 AWS Glue 的信息。要开始使用 AWS Glue，并非所有的设置部分都是必需的。如果使用 VPC 环境访问数据存储或使用交互式会话，您可以根据需要使用说明来设置 IAM 权限、加密和 DNS。

主题

AWS Glue 使用概述

利用 AWS Glue，您可以在 AWS Glue Data Catalog中存储元数据。您可以使用此元数据来协调转换数据源和加载数据仓库或数据湖的 ETL 作业。以下步骤介绍了一般工作流程以及您在使用 AWS Glue 时所做的一些选择。

> **Note**
>
> 您可以使用以下步骤，也可以创建一个工作流来自动执行步骤 1 到步骤 3。有关更多信息，请参阅 the section called “使用蓝图和工作流执行复杂的 ETL 活动”。

1. 使用表定义填充 AWS Glue Data Catalog。

   在控制台中，对于持久数据存储，您可以添加一个爬网程序来填充 AWS Glue Data Catalog。您可以从表列表或爬网程序列表中启动 Add crawler (添加爬网程序) 向导。您可以选择一个或多个数据存储供爬网程序访问。您也可以创建一个计划来确定运行爬网程序的频率。对于数据流，您可以手动创建表定义并定义流属性。

   (可选) 您可以提供一个推断数据架构的自定义分类器。您可以使用 grok 模式创建自定义分类器。但是，AWS Glue 提供了内置分类器，如果自定义分类器无法识别您的数据，则爬网程序将自动使用

内置分类器。在定义爬网程序时，您不必选择分类器。有关 AWS Glue 中的分类器的更多信息，请参阅在 AWS Glue 中向爬网程序添加分类器。

对某些类型的数据存储进行网络爬取需要一个提供身份验证和位置信息的连接。如果需要，您可以在 AWS Glue 控制台中创建提供此所需信息的连接。

爬网程序将读取您的数据存储并在 AWS Glue Data Catalog中创建数据定义和已命名的表。这些表将组织到您选择的数据库中。您也可以使用手动创建的表填充数据目录。通过这种方法，您可以提供架构和其他元数据，从而在数据目录中创建表定义。由于此方法可能有点繁琐又容易出错，通常最好是让爬网程序创建表定义。

有关使用表定义填充 AWS Glue Data Catalog的更多信息，请参阅创建表。

2. 定义一个作业，该作业描述数据从源到目标的转换。

   通常，要创建作业，您必须进行以下选择：

   - 从 AWS Glue Data Catalog中选择一个表作为作业的源。您的作业使用此表定义访问数据源和解释数据的格式。
   - 从 AWS Glue Data Catalog中选择一个表或位置作为作业的目标。您的作业使用此信息访问数据存储。
   - 告知 AWS Glue 生成一个脚本，以将源转换为目标。AWS Glue 将生成用于调用内置转换的代码，以将数据从其源架构转换到目标架构格式。这些转换将执行复制数据、重命名列和筛选数据等操作，以便根据需要转换数据。您可以在 AWS Glue 控制台中修改此脚本。

   有关在 AWS Glue 中定义作业的更多信息，请参阅使用 AWS Glue Studio 构建可视化 ETL 作业。

3. 运行作业以转换数据。

   您可以按需运行作业，或根据这些触发器类型之一来启动它：

   - 基于 cron 计划的触发器。
   - 基于事件的触发器；例如，另一个作业的成功完成可能启动一个 AWS Glue 作业。
   - 按需启动作业的触发器。

   有关 AWS Glue 中的触发器的更多信息，请参阅使用触发器启动作业和爬网程序。

4. 监控您的已计划的爬网程序和已触发的作业。

   使用 AWS Glue 控制台查看以下内容：

   - 作业运行详细信息和错误。
   - 爬网程序运行详细信息和错误。

- 有关 AWS Glue 活动的任何通知

有关在 AWS Glue 中监控爬网程序和作业的更多信息，请参阅监控 AWS Glue。

为 AWS Glue 设置 IAM 权限

本主题中的说明可帮助您为 AWS Glue 快速设置 AWS Identity and Access Management（IAM）权限。您需要完成以下任务：

- 授予您的 IAM 身份访问 AWS Glue 资源的权限。
- 创建用于运行作业、访问数据和运行 AWS Glue Data Quality 任务的服务角色。

有关可用于为 为 AWS Glue 配置 IAM 权限 自定义 IAM 权限的详细说明，请参阅 AWS Glue。

在 AWS Management Console 中为 AWS Glue 设置 IAM 权限

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 选择开始使用。
3. 在为 AWS Glue 准备好账号下，选择设置 IAM 权限。
4. 选择您要向其授予 AWS Glue 权限的 IAM 身份（角色或用户）。AWS Glue 将 `AWSGlueConsoleFullAccess` 托管策略附加到这些身份。如果您想手动设置这些权限或只想设置默认服务角色，则可以跳过此步骤。
5. 选择下一步。
6. 选择您的角色和用户需要的 Amazon S3 访问权限级别。您在此步骤中选择的选项将应用于您选择的所有身份。

   a. 在选择 S3 地点下，选择您想要授予访问权限的 Amazon S3 地点。

   b. 接下来，选择您的身份应该对您之前选择的位置具有只读（推荐）还是读写权限。AWS Glue 根据您选择的位置和读取或写入权限的组合为您的身份添加权限策略。

      下表显示了 AWS Glue 为访问 Amazon S3 而附加的权限。

如果选择.....	AWS Glue 附加.....
无更改	没有权限。AWS Glue 不会对您的身份权限执行任何更改。
授予对特定 Amazon S3 地点的访问权限（只读）	在您选择的 IAM 身份中嵌入内联策略。有关更多信息，请参阅《IAM 用户指南》中的 Inline policies。 AWS Glue 使用以下惯例命名策略：`AWSGlueConsole` *`<Role/User>`* `InlinePolicy-read-specific-access-` *`<UUID>`*。例如：`AWSGlueConsoleRole InlinePolicy-read-specific-access-123456780123` 。 以下是 AWS Glue 附加的内联策略示例，该策略用于授予对指定 Amazon S3 位置的只读访问权限。 `{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Action": [ "s3:Get*", "s3:List*" ], "Resource": [ "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*" ] } ] }`

如果选择……	AWS Glue 附加……
授予对特定 Amazon S3 地点的访问权限（读写）	在您选择的 IAM 身份中嵌入内联策略。有关更多信息，请参阅《IAM 用户指南》中的 Inline policies。 AWS Glue 使用以下惯例命名策略：`AWSGlueConsole` *`<Role/User>`* `InlinePolicy-read -and-write-specific-access-` *`<UUID>`*。例如：`AWSGlueConsoleRole InlinePolicy-read-and-write-specific-access-123456780123` 。 以下是 AWS Glue 附加的内联策略示例，该策略用于授予对指定 Amazon S3 位置的读写访问权限。 `{ "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/*" ] } ] }`

如果选择……	AWS Glue 附加……
授予对 Amazon S3 的完全访问权限（只读）	`AmazonS3ReadOnlyAccess` 托管 IAM policy。要了解更多信息，请参阅 AWS managed policy: AmazonS3ReadOnlyAccess。
授予对 Amazon S3 的完全访问权限（读写）	`AmazonS3FullAccess` 托管 IAM policy。要了解更多信息，请参阅 AWS managed policy: AmazonS3FullAccess。

7. 选择下一步。
8. 为您的账户选择默认 AWS Glue 服务角色。服务角色是一个 IAM 角色，AWS Glue 用于代表您访问其他 AWS 服务中的资源。有关更多信息，请参阅 AWS Glue 的服务角色。

   - 当您选择标准 AWS Glue 服务角色时，AWS Glue 会在您的名为 `AWSGlueServiceRole` 的 AWS 账户 中创建一个新的 IAM 角色，并附加以下托管策略。如果您的账户已经有一个名为 `AWSGlueServiceRole` 的 IAM 角色，AWS Glue 会将这些策略附加到现有角色。
     - AWSGlueServiceRole
     - AmazonS3FullAccess
   - 当您选择现有 IAM 角色时，AWS Glue 会将该角色设置为默认角色，但不会向其添加任何权限。确保您已将角色配置为用作 AWS Glue 的服务角色。有关更多信息，请参阅步骤 1：为 AWS Glue 服务创建 IAM policy 和步骤 2：为 AWS Glue 创建 IAM 角色。

9. 选择下一步。
10. 最后，检查您选择的权限，然后选择应用更改。当您应用更改时，AWS Glue 会向您选择的身份添加 IAM 权限。您可以在 IAM 控制台中查看或修改新权限，网址为 https://console.aws.amazon.com/iam/。

现在，您已经完成了为 AWS Glue 设置最低 IAM 权限。在生产环境中，我们建议您熟悉 AWS Glue 中的安全性 和 AWS Glue 的身份和访问管理，帮助您保护用例的 AWS 资源。

后续步骤

现在您已经设置了 IAM 权限，您可以探索以下主题以开始使用 AWS Glue：

- AWS Skill Builder 中的 AWS Glue 入门

- AWS Glue Data Catalog入门

对 AWS Glue Studio 进行设置

如果您首次将 AWS Glue 用于可视化 ETL，请完成本部分中的任务：

主题

- 审核 AWS Glue Studio 用户需要 IAM 权限
- 审核 ETL 任务所需的 IAM 权限
- 为 AWS Glue Studio 设置 IAM 权限
- 为 ETL 任务配置 VPC

审核 AWS Glue Studio 用户需要 IAM 权限

要使用 AWS Glue Studio，用户必须有权访问各种 AWS 资源。用户必须能够查看和选择 Amazon S3 存储桶、IAM policy 和角色，以及 AWS Glue Data Catalog 对象。

AWS Glue 服务权限

AWS Glue Studio 使用 AWS Glue 服务的操作和资源。您的用户需要这些操作和资源的权限，从而有效使用 AWS Glue Studio。您可以授予 AWS Glue Studio 用户 `AWSGlueConsoleFullAccess` 托管式策略，或创建具有较小权限集的自定义策略。

> **Important**
>
> 根据安全性最佳实践，建议通过收紧策略来限制访问，从而进一步限制对 Amazon S3 存储桶和 Amazon CloudWatch 日志组的访问。有关示例 Amazon S3 策略，请参阅编写 IAM policy：如何授予对 Amazon S3 存储桶的访问权限。

为 AWS Glue Studio 创建自定义 IAM policy

您可以为 AWS Glue Studio 创建一个包含较小权限集的自定义策略。该策略可以为对象或操作子集授予权限。创建自定义策略时，请使用以下信息。

要使用 AWS Glue Studio API，请在 IAM 权限的操作策略中包括 `glue:UseGlueStudio`。使用 `glue:UseGlueStudio` 将允许您访问所有 AWS Glue Studio，即使随着时间的推移向 API 中添加了更多操作。

有向非循环图 (DAG) 操作

- CreateDag
- UpdateDag
- GetDag
- DeleteDag

任务操作

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

任务运行操作

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

架构操作

- GetSchema
- *GetInferredSchema

数据库操作

- GetDatabases

计划操作

- GetPlan

表操作

- SearchTables
- GetTables
- GetTable

连接操作

- CreateConnection
- DeleteConnection
- UpdateConnection
- GetConnections
- GetConnection

映射操作

- GetMapping

S3 代理操作

- ListBuckets
- ListObjectsV2
- GetBucketLocation

安全配置操作

- GetSecurityConfigurations

脚本操作

- CreateScript（不同于 AWS Glue 中的同名 API）

访问 AWS Glue Studio API

要访问 AWS Glue Studio，请在 IAM 权限的操作策略列表中添加 `glue:UseGlueStudio`。

在以下示例中，`glue:UseGlueStudio` 已包括在操作策略中，但 AWS Glue Studio API 不是单独识别的。这是因为当您包括 `glue:UseGlueStudio` 时，系统将会向您自动授予访问内部 API 的权限，而无需在 IAM 权限中指定单个 AWS Glue Studio API。

在该示例中，列出的其他操作策略（例如，`glue:SearchTables`）并非 AWS Glue Studio API，因此，需要根据需要将它们包含在 IAM 权限中。此外，您可能还希望包括 Amazon S3 代理操作，以指定要授予的 Amazon S3 访问权限级别。以下示例策略提供对开放 AWS Glue Studio、创建可视化任务及保存/运行任务（如果选定 IAM 角色具有足够权限）的访问权限。

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

笔记本和数据预览权限

数据预览和笔记本允许您在任务的任何阶段（读取、转换、写入）查看数据样本，而无需运行任务。您可以为 AWS Glue Studio 指定访问数据时要使用的 AWS Identity and Access Management (IAM) 角色。IAM 角色可代入，没有关联的标准长期凭证（如密码或访问密钥）。相反，当 AWS Glue Studio 代入角色时，IAM 会为其提供临时安全凭证。

要确保数据预览和笔记本命令正常工作，请使用名称开头为字符串 `AWSGlueServiceRole` 的角色。如果选择为角色使用其他名称，则必须在 IAM 中为角色添加 `iam:passrole` 权限并配置策略。有关更多信息，请参阅 为未命名为“AWSGlueServiceRole*”的角色创建 IAM policy。

> **Warning**
>
> 如果角色为笔记本授予 `iam:passrole` 权限，并且您实施了角色链接，则用户可能会无意中获得访问笔记本的权限。目前没有实施审计，因此不允许您监控哪些用户已被授予访问笔记本的权限。

如果您想拒绝 IAM 身份创建数据预览会话，请参阅以下示例 the section called “拒绝某个身份创建数据预览会话”。

Amazon CloudWatch 权限

您可以使用 Amazon CloudWatch 来监控 AWS Glue Studio 任务，此工具可从 AWS Glue 收集原始数据，并将数据处理为可读的近实时指标。默认情况下，AWS Glue 指标数据自动发送到 CloudWatch。有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的什么是 Amazon CloudWatch？和《AWS Glue 开发人员指南》中的 AWS Glue 指标。

要访问 CloudWatch 控制面板，则访问 AWS Glue Studio 的用户需要以下内容之一：

- `AdministratorAccess` 策略
- `CloudWatchFullAccess` 策略
- 包含以下一个或多个特定权限的自定义策略：
  - `cloudwatch:GetDashboard` 和 `cloudwatch:ListDashboards`，以查看控制面板
  - `cloudwatch:PutDashboard`，以创建或修改控制面板
  - `cloudwatch:DeleteDashboards`，以删除控制面板

有关使用策略更改 IAM 用户权限的更多信息，请参阅《IAM 用户指南》中的更改 IAM 用户的权限。

审核 ETL 任务所需的 IAM 权限

当您使用 AWS Glue Studio 创建任务时，该任务代入您在创建它时指定的 IAM 角色的权限。此 IAM 角色必须有权限从您的数据源中提取数据，将其写入您的目标，并访问 AWS Glue 资源。

为任务创建的角色的名称必须以字符串 `AWSGlueServiceRole` 开头，以便其能够正确地被 AWS Glue Studio 使用。例如，您可以将角色命名为 `AWSGlueServiceRole-FlightDataJob`。

数据源和数据目标权限

AWS Glue Studio 任务必须具有您在任务中使用的任何源、目标、脚本和临时目录等 Amazon S3 的访问权限。您可以创建策略，提供对特定 Amazon S3 资源的精细访问权限。

- 数据源需要 `s3:ListBucket` 和 `s3:GetObject` 权限。
- 数据目标需要`s3:ListBucket`、`s3:PutObject` 和 `s3:DeleteObject` 权限。

如果选择 Amazon Redshift 作为数据源，那么您可以为集群权限提供角色。针对 Amazon Redshift 集群运行的任务会发出命令，使用临时凭证访问 Amazon S3 进行临时存储。如果您的任务运行超过一小时，这些凭证将会过期，导致任务失败。若要避免此问题，您可以将角色分配给 Amazon Redshift 集群本身，其授予使用临时凭证的任务所需的权限。有关更多信息，请参阅《AWS Glue 开发人员指南》中的将数据移入和移出 Amazon Redshift。

如果任务使用 Amazon S3 以外的数据源或目标，则您必须为任务使用的 IAM 角色附上所需的权限，从而访问这些数据源和目标。有关更多信息，请参阅《AWS Glue 开发人员指南》中的设置环境以访问数据存储。

如果要为数据存储使用连接器和连接，则您需要在the section called “使用连接器所需的权限”中描述的其他权限。

删除任务所需的权限

在 AWS Glue Studio 中，您可以在控制台选择多个任务删除。若要执行此操作，您必须具有 `glue:BatchDeleteJob` 权限。这不同于需要 `glue:DeleteJob` 权限来删除任务的 AWS Glue 控制台。

AWS Key Management Service 权限

如果您计划访问使用 AWS Key Management Service（AWS KMS）进行服务器端加密的 Amazon S3 源和目标，则需为任务使用的 AWS Glue Studio 角色附上策略，从而使任务能够解密数据。任务角色需要 `kms:ReEncrypt`、`kms:GenerateDataKey` 和 `kms:DescribeKey` 权限。此外，任务角色需要 `kms:Decrypt` 权限来上载或下载使用 AWS KMS 客户主密钥（CMK）加密的 Amazon S3 对象。

使用 AWS KMS CMK 需支付额外费用。有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 AWS Key Management Service 概念 – 客户主密钥（CMK）和 AWS Key Management Service 定价。

使用连接器所需的权限

如果您使用 AWS Glue 自定义连接器和连接来访问数据存储，则需要为用于运行 AWS Glue ETL 任务的角色附上其他权限：

- 亚马逊云科技托管式策略 `AmazonEC2ContainerRegistryReadOnly`，用于访问购买于 AWS Marketplace 的连接器。
- `glue:GetJob` 和 `glue:GetJobs` 权限。
- AWS Secrets Manager 权限，用于访问与连接一起使用的密钥。有关示例 IAM 策略，请参阅 Example: Permission to retrieve secret values（示例：检索密钥值的权限）。

如果您的 AWS Glue ETL 任务在运行 Amazon VPC 的 VPC 中运行，则 VPC 必须按照the section called “为 ETL 任务配置 VPC”中的描述进行配置。

为 AWS Glue Studio 设置 IAM 权限

您可以通过使用 AWS 管理员用户，创建角色并将策略分配给用户和任务角色。

您可以使用 AWSGlueConsoleFullAccess AWS 托管式策略提供使用 AWS Glue Studio 控制台所需的权限。

要创建您自己的策略，请按照《AWS Glue 开发人员指南》中的为 AWS Glue 服务创建 IAM policy 记录的步骤进行操作。包括前面在 审核 AWS Glue Studio 用户需要 IAM 权限 中描述的 IAM 权限。

主题

- 将策略附加至 AWS Glue Studio 用户
- 为未命名为“AWSGlueServiceRole*”的角色创建 IAM policy

将策略附加至 AWS Glue Studio 用户

登录 AWS Glue Studio 控制台的任何 AWS 用户都必须具有访问特定资源的权限。您可使用向用户分配 IAM 策略来提供这些权限。

向用户附加 AWSGlueConsoleFullAccess 托管策略

1. 登录到 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择策略。

3. 在策略列表中，选中 AWSGlueConsoleFullAccess 旁边的复选框。您可以使用 Filter 菜单和搜索框来筛选策略列表。
4. 选择 Policy actions（策略操作），然后选择 Attach（附加）。
5. 选择要将策略附加到的用户。您可以使用 Filter（筛选条件）菜单和搜索框来筛选委托人实体列表。在选择要将策略附加到的用户后，选择 Attach policy (附加策略)。
6. 根据需要，重复前面的步骤向用户附上其他策略。

为未命名为“AWSGlueServiceRole*”的角色创建 IAM policy

要为 AWS Glue Studio 所使用的角色配置 IAM policy

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 添加新的 IAM policy。您可以添加到现有策略或创建新的 IAM 内联策略。若要创建 IAM policy：
   1. 选择 Policies，然后选择 Create Policy。如果 Get Started (开始使用) 按钮出现，选择此按钮，然后选择 Create Policy (创建策略)。
   2. 在 Create Your Own Policy 旁，选择 Select。
   3. 对于 Policy Name (策略名称)，键入一个便于您稍后参考的值。（可选）在 Description (描述) 中键入说明性文本。
   4. 对于 Policy Document (策略文档)，请使用以下格式键入策略语句，然后选择 Create Policy (创建策略)：
3. 将以下块复制并粘贴到“Statement”数组下的策略中，将 *my-interactive-session-role-prefix* 替换为所有要与 AWS Glue 的权限关联的常用角色的前缀。

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

```
}
```

以下是策略中包含的版本和语句阵列的完整示例

```
{
  "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. 要对某个用户启用策略，请选择用户。
5. 选择您要向其挂载策略的 用户。

为 ETL 任务配置 VPC

您可以使用 Amazon Virtual Private Cloud (Amazon VPC) 在 AWS Cloud 内您自己的逻辑隔离区域中定义虚拟化网络，我们称之为 虚拟私有云 (VPC) 。可在 VPC 中启动实例等 AWS 资源。您的 VPC 与您可能在自己的数据中心中运行的传统网络极为相似，同时享有使用来自 AWS 的可扩展基础设施的优势。您可以配置您的 VPC；您可以选择它的 IP 地址范围、创建子网并配置路由表、网关和安全设置。您可以将您的 VPC 中的实例连接到网络。您可以将您的 VPC 连接到公司的数据中心，并将 AWS Cloud 作为数据中心的延伸。要保护各个子网中的资源，您可以利用多种安全层，包括安全组和网络访问控制列表。有关更多信息，请参阅 Amazon VPC 用户指南。

使用连接器时，您可以配置您的 AWS Glue ETL 任务，使其在 VPC 内运行。您必须根据需要为以下内容配置 VPC：

- 不在 AWS 中的数据存储的公有网络访问。必须能够从 VPC 子网使用任务访问的所有数据存储。

- 如果您的任务既要访问 VPC 资源又要访问公有 Internet，VPC 内部必须具有网络地址转换（NAT）网关。

  有关更多信息，请参阅《AWS Glue 开发人员指南》中的设置环境以访问数据存储。

在 AWS Glue Studio 中开启笔记本

当您通过 AWS Glue Studio 开启笔记本时，所有配置步骤都已为您完成，让您在几秒钟后即可浏览数据并开始开发任务脚本。

以下部分介绍如何创建角色并授予在 AWS Glue Studio for ETL 作业中使用笔记本的适当权限。

主题

为 IAM 角色授予权限

设置 AWS Glue Studio 是使用笔记本的先决条件。

要在 AWS Glue 中使用笔记本，您的角色需要满足以下条件：

- 与 AWS Glue 建立信任关系以进行 `sts:AssumeRole` 操作，如果您要进行标记操作，则与 `sts:TagSession` 建立信任关系。
- 包含笔记本、AWS Glue 和交互式会话的所有 API 操作的 IAM policy。
- 传递角色的 IAM policy，因为该角色需要能够将自己从笔记本传递到交互式会话。

例如，在创建新角色时，您可以向该角色添加标准 AWS 托管策略（如 `AWSGlueConsoleFullAccessRole`），然后为笔记本操作添加新策略，为 IAM PassRole 策略添加另一个策略。

与 AWS Glue 建立信任关系所需的操作

启动笔记本会话时，您必须将 `sts:AssumeRole` 添加至传递到笔记本的角色的信任关系中。如果您的会话包括标签，您也必须传递 `sts:TagSession` 操作。如果未进行这些操作，则笔记本会话无法开启。

例如：

```
{
```

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

包含笔记本 API 操作的策略

以下示例策略描述了笔记本所需的 AWS IAM 权限。如果您要创建新角色，请创建包含以下内容的策略：

```
{
    "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 策略允许访问特定资源：

- AwsGlueSessionUserRestrictedNotebookServiceRole：提供对除会话外所有 AWS Glue 资源的完全访问权限。允许用户仅创建和使用与用户关联的笔记本会话。此策略还包括由 AWS Glue 管理其他 AWS 服务中的 AWS Glue 资源所需的其他权限。

- AwsGlueSessionUserRestrictedNotebookPolicy：提供允许用户仅创建和使用与用户关联的笔记本会话的权限。此策略还包括明确允许用户传递受限 AWS Glue 会话角色的权限。

传递角色的 IAM policy

当您创建具有角色的笔记本时，该角色将传递至交互式会话，以便同一角色可以在两个位置使用。因此，iam:PassRole 权限需要成为角色策略的一部分。

使用以下示例为您的角色创建新策略。将账号替换为您的账号，并将角色名称替换为您的角色名称。

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

设置 AWS Glue 使用情况配置文件

使用云平台的主要优势之一是其灵活性。但是，由于这种创建计算资源的便捷性，如果不受管理且没有护栏，则存在云成本螺旋式上升的风险。因此，管理员需要在避免高昂的基础架构成本与允许用户在没有不必要摩擦的情况下工作之间取得平衡。

通过 AWS Glue 使用情况配置文件，管理员可以为账户中的不同类别的用户（例如开发人员、测试人员和产品团队）创建不同的配置文件。每个配置文件都是一组独特的参数，可以分配给不同类型的用户。例如，开发人员可能需要更多的工作人员，并且可以拥有更高的最大员工人数，而产品团队可能需要更少的工作人员，以及更低的超时或空闲超时值。

作业和作业运行行为示例

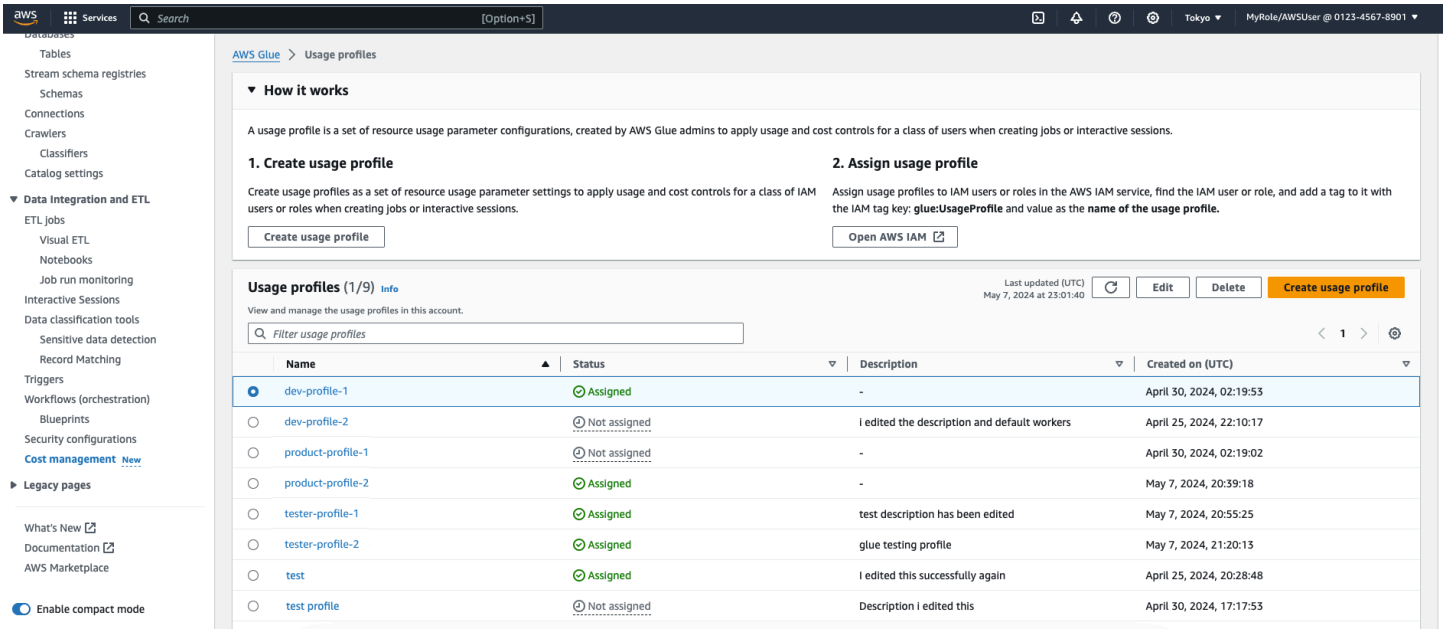
假设任务是由用户 A 创建的，配置文件 A 是使用特定的参数值保存的。配置文件 B 的用户 B 将尝试运行该作业。

当用户 A 创作作业时，如果他没有设置特定数量的工作人员，则会应用用户 A 个人资料中的默认设置，并与作业定义一起保存。

当用户 B 运行作业时，它会使用为其保存的任何值运行。如果用户 B 自己的配置文件更具限制性，并且不允许与那么多工作人员一起运行，则作业运行将失败。

作为资源的使用情况配置文件

AWS Glue 使用情况配置文件是一种由 Amazon 资源名称 (ARN) 标识的资源。所有默认 IAM（Identity and Access Management）控制均适用，包括基于操作的授权和基于资源的授权。管理员应更新创建 AWS Glue 资源的用户的 IAM 政策，授予他们使用配置文件的权限。



主题

- 创建和管理使用情况配置文件
- 使用情况资料和职位

创建和管理使用情况配置文件

创建 AWS Glue 使用情况配置文件

管理员应创建使用情况配置文件，然后将其分配给不同的用户。创建使用情况配置文件时，您可以为各种作业和会话参数指定默认值以及允许值的范围。必须为作业或交互式会话配置至少一个参数。您可以自定义未为作业提供参数值时使用的默认值，和/或如果用户在使用此配置文件时提供了参数值，则可以设置范围限制或一组允许的值进行验证。

默认值是管理员为帮助作业作者而设置的最佳实践。当用户创建新作业但未设置超时值时，使用情况配置文件的默认超时将适用。如果作者没有个人资料，则 AWS Glue 服务默认值将适用并保存在作

业定义中。在运行时， AWS Glue 强制执行配置文件中设置的限制（最小值、最大值、允许的工作人员）。

配置参数后，所有其他参数均为可选参数。可以为作业或交互式会话自定义的参数有：

- 工作人员数量 — 限制工作人员数量以避免过度使用计算资源。您可以设置默认值、最小值和最大值。最小值为 1。
- 工作人员类型-限制工作负载的相关工作人员类型。您可以为用户配置文件设置默认类型并允许工作人员类型。
- 超时-定义任务或交互式会话在终止之前可以运行和消耗资源的最长时间。设置超时值以避免长时间运行的作业。

  您可以以分钟为单位设置默认值、最小值和最大值。最小值为 1（分钟）。虽然 AWS Glue 默认超时时间为 2880 分钟，但您可以在使用情况配置文件中设置任何默认值。

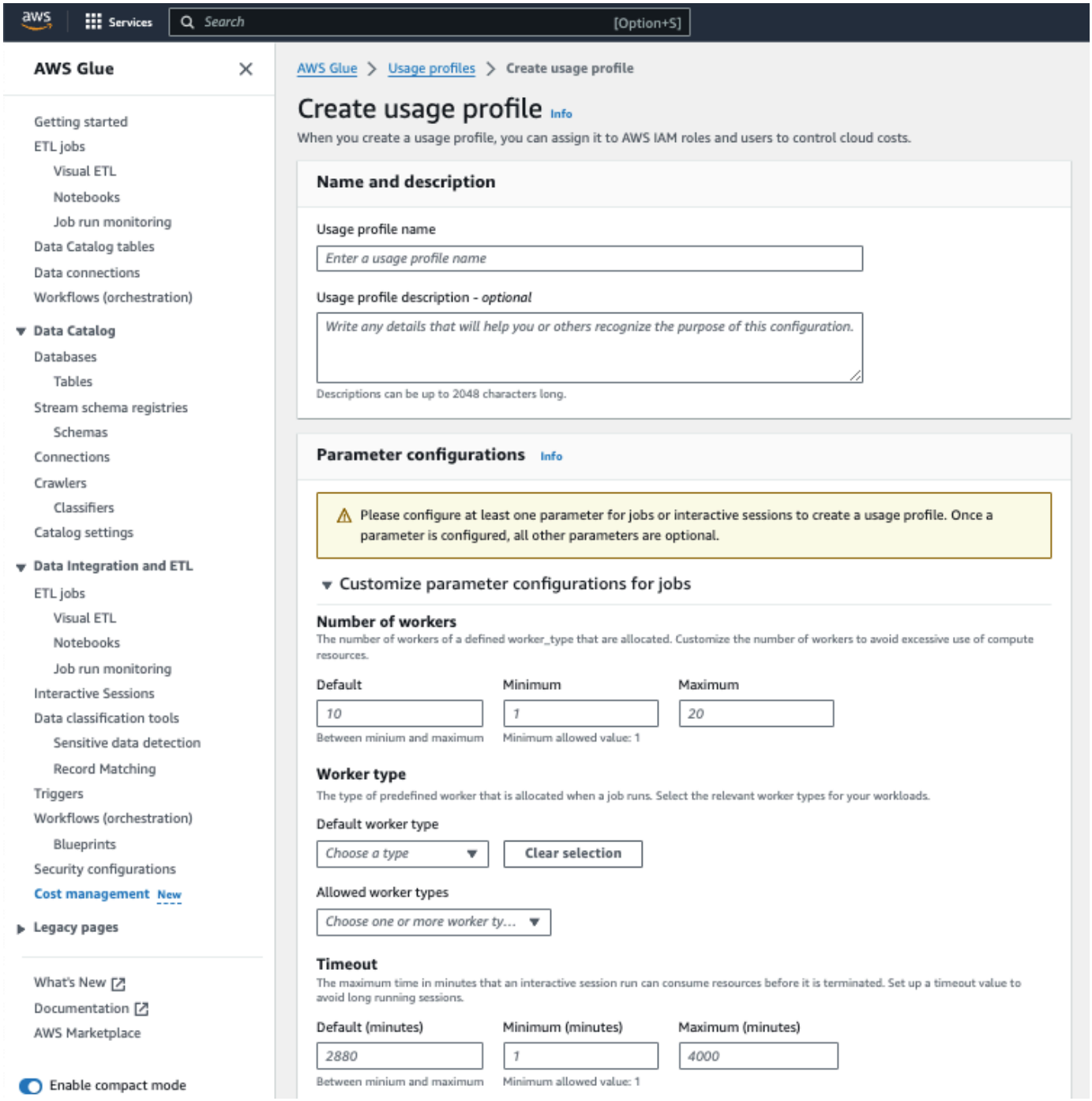
  最佳做法是为 “默认” 设置一个值。如果用户未设置任何值，则此值将用于创建任务或会话。
- 空闲超时 — 定义交互式会话在单元运行后超时之前处于非活动状态的分钟数。定义交互式会话在工作完成后终止的空闲超时。空闲超时范围应在超时限制范围内。

  您可以以分钟为单位设置默认值、最小值和最大值。最小值为 1（分钟）。虽然 AWS Glue 默认超时时间为 2880 分钟，但您可以在使用情况配置文件中设置任何默认值。

  最佳做法是为 “默认” 设置一个值。如果用户未设置任何值，则此值将用于创建会话。

以管理员身份创建 AWS Glue 使用情况配置文件（控制台）

1. 在左侧导航菜单中，选择成本管理。
2. 选择 “创建使用情况配置文件”。
3. 输入使用情况配置文件的使用情况配置文件名称。
4. 输入可选描述，以帮助其他人识别使用情况配置文件的用途。
5. 在配置文件中至少定义一个参数。表单中的任何字段都是参数。例如，会话空闲超时最小值。
6. 定义适用于使用情况配置文件的所有可选标签。
7. 选择保存。



创建使用情况配置文件 (AWS CLI)

1. 输入以下 命令。

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

其中 config.json 可以为交互式会话 (SessionConfiguration) 和作业 (JobConfiguration) 定义参数值：

```
//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. 输入以下命令以查看创建的使用情况配置文件：

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

回应：

```
{
    "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"
}
```

用于管理使用情况配置的其他 CLI 命令：

- aws 胶水 list-usage-profiles
- *aws glu update-usage-profile e--name p rofile-name--####://config.*
- *aws glue delete-usage-profile --name #####*

编辑使用情况配置文件

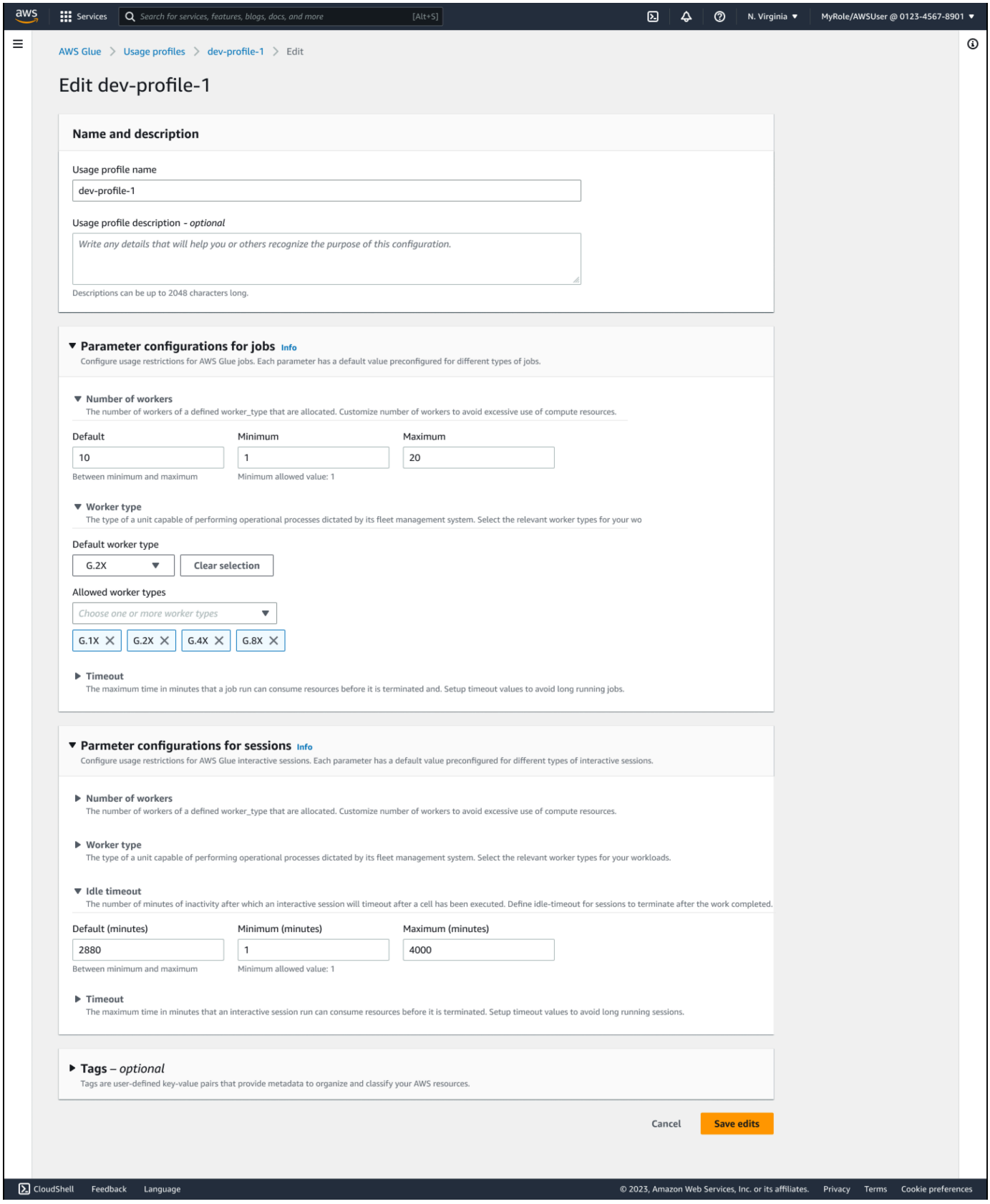
管理员可以编辑他们创建的使用情况配置文件，以更改作业和交互式会话的配置文件参数值。

要编辑使用情况配置文件，请执行以下操作：

以管理员身份编辑 AWS Glue 使用情况配置文件（控制台）

1. 在左侧导航菜单中，选择成本管理。

2. 选择您有权编辑的使用情况配置文件，然后选择编辑。
3. 根据需要对配置文件进行更改。默认情况下，已有值的参数会被展开。
4. 选择 “保存编辑内容”。


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

编辑使用情况配置文件 (AWS CLI)

- 输入以下 命令。与上面的 create 命令中使用的`--configuration`文件语法相同。

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

其中 config.json 定义了交互式会话 (`SessionConfiguration`) 和作业 (`JobConfiguration`) 的参数值：

分配使用情况配置文件

“使用情况配置文件” 页面中的 “利用率状态” 列显示是否向用户分配了使用情况配置文件。将鼠标悬停在状态上会显示分配的 IAM 实体。

管理员可以为创建 AWS Glue 资源的用户/角色分配 AWS Glue 使用情况配置文件。分配配置文件是以下两个操作的组合：

- 使用`glue:UsageProfile`密钥更新 IAM 用户/角色标签，然后
- 更新用户/角色的 IAM 策略。

对于使用 AWS Glue Studio 创建作业/交互式会话的用户，管理员会标记以下角色：

- 为了限制作业，管理员会标记已登录的控制台角色
- 对于交互式会话的限制，管理员会标记用户在创建笔记本时提供的角色

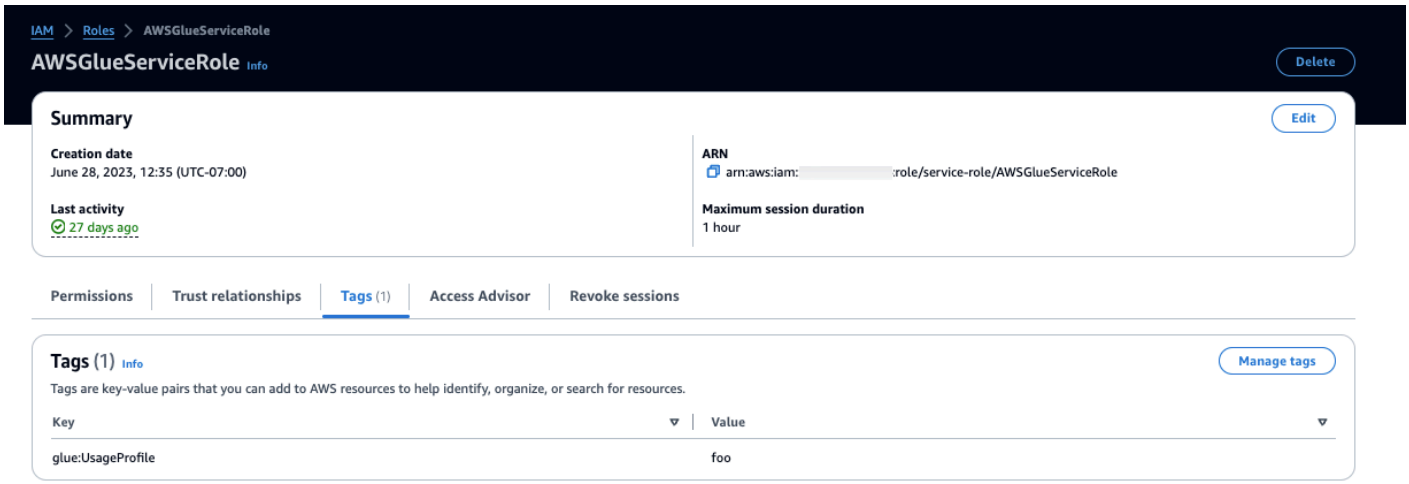
以下是管理员需要更新的有关创建 AWS Glue 资源的 IAM 用户/角色的策略示例：

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

AWS Glue 根据 AWS Glue 使用情况配置文件中指定的值验证作业、作业运行和会话请求，如果请求被禁止，则会引发异常。对于同步 API，将向用户抛出错误。对于异步路径，会创建失败的作业运行，并显示错误消息，即输入参数超出了为用户/角色分配的配置文件所允许的范围。

要为用户/角色分配使用情况配置文件，请执行以下操作：

1. 打开（Identity and Access Management）IAM 控制台。
2. 在左侧导航栏中，选择 “用户” 或 “角色”。
3. 选择用户或角色。
4. 选择标签选项卡。
5. 选择添加新标签。
6. 添加带有使用情况配置文件名称的密钥`glue:UsageProfile`和值的标签。
7. 选择 Save changes（保存更改）



查看您分配的使用情况配置文件

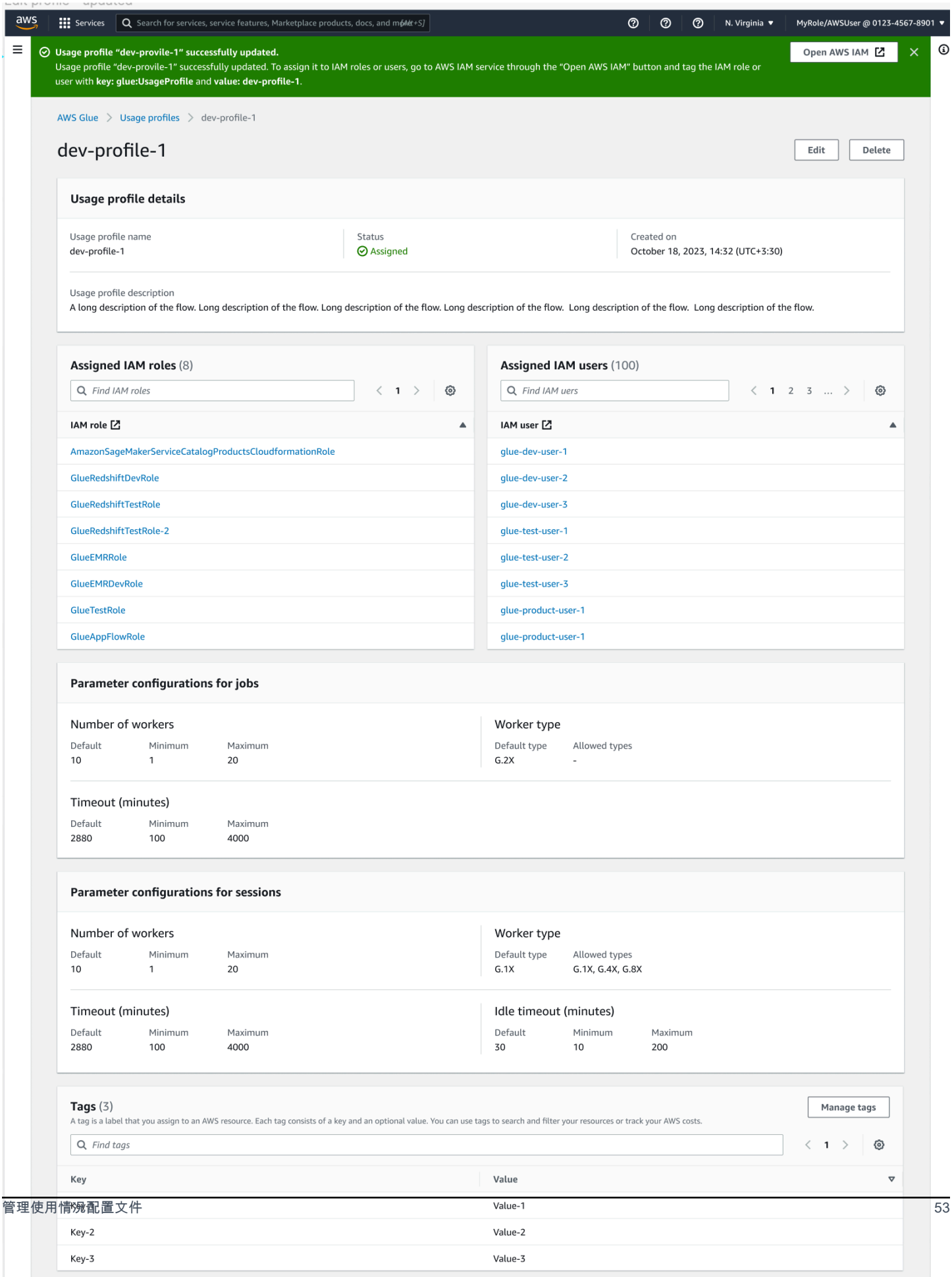
用户可以查看他们分配的使用情况配置文件，并在调用 API 以创建 AWS Glue 作业和会话资源或启动作业时使用它们。

IAM 策略中提供了个人资料权限。只要来电者策略具有`glue:UsageProfile`权限，用户就可以查看个人资料。否则，您将收到拒绝访问的错误。

要查看分配的使用情况配置文件，请执行以下操作：

1. 在左侧导航菜单中，选择成本管理。

2. 选择您有权查看的使用情况配置文件。


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

使用情况资料和职位

使用使用情况配置文件创作作业

在创作作业时，您的使用情况配置文件中设置的限制和默认值将适用。保存后，您的个人资料将分配给该职位。

使用使用情况配置文件运行作业

当你开始运行任务时，会 AWS Glue 强制执行来电者个人资料中设置的限制。如果没有直接来电者，Glue 将应用作者分配给该职位的个人资料中的限制。

> **Note**
>
> 当作业按计划运行（通过 AWS Glue 工作流程或 AWS Glue 触发器）时，作者将应用分配给该作业的配置文件。
>
> 当任务由外部服务（Step Functions、MWAA）或 `StartJobRun` API 运行时，将强制执行调用者的个人资料限制。

对于 AWS Glue 工作流程或 AWS Glue 触发器：需要更新先前存在的作业以保存新的配置文件名称，以便在计划运行时强制执行配置文件的限制（最小、最大和允许的工作人员）。

查看为任务分配的使用情况配置文件

要查看分配给您的作业的配置文件（将在运行时与计划 AWS Glue 的工作流程或 AWS Glue 触发器一起使用），您可以查看任务详细信息选项卡。您还可以在作业运行详细信息选项卡中查看过去运行中使用的配置文件。

更新或删除附加到作业的使用情况配置文件

分配给某项工作的配置文件在更新时会发生变化。如果未向作者分配使用情况配置文件，则之前附加到该职位的任何个人资料都将从中删除。

AWS Glue Data Catalog入门

AWS Glue Data Catalog 是您的持久性技术元数据存储。它是一项托管式服务，可用于存储、注释和共享 AWS 云中的元数据。有关更多信息，请参阅 AWS Glue Data Catalog。

AWS Glue 控制台和部分用户界面已于近期更新。

概述

您可以使用本教程创建您的第一个 AWS Glue 数据目录，其使用 Amazon S3 存储桶作为您的数据源。

在本教程中，您将使用 AWS Glue 控制台执行以下操作：

1. 创建一个数据库
2. 创建表
3. 使用 Amazon S3 桶作为数据来源

完成这些步骤后，您将成功使用 Amazon S3 存储桶作为数据源来填充 AWS Glue 数据目录。

步骤 1：创建数据库

要开始使用，请登录 AWS Management Console，然后打开 AWS Glue 控制台。

要使用 AWS Glue 控制台创建数据库：

1. 在 AWS Glue 控制台中，从左侧菜单中选择 Data catalog（数据目录）下的 Databases（数据库）。
2. 选择 Add database（添加数据库）。
3. 在“创建数据库”页面中，输入数据库的名称。在位置 - 可选部分中，设置 URI 位置以供 Data Catalog 的客户端使用。如果您不知道信息，则可以继续创建数据库。
4. （可选）。输入数据库的描述。
5. 选择创建数据库。

恭喜您，您刚刚使用 AWS Glue 控制台搭建了您的第一个数据库。您的新数据库将显示在可用数据库列表中。您可以在 Databases（数据库）控制面板中选择数据库名称，编辑数据库。

后续步骤

Other ways to create a database:（创建数据库的其他方法：）

您刚刚使用 AWS Glue 控制台创建了一个数据库，但还有其他方法也可以创建数据库：

- 您可以使用爬网程序自动为自己创建数据库和表。要使用爬网程序设置数据库，请参阅 在 AWS Glue 控制台中使用爬网程序。
- 您可以使用 AWS CloudFormation 模板。请参阅 使用 AWS Glue Data Catalog 模板创建 AWS Glue 资源。
- 您还可以使用 AWS Glue 数据库 API 操作来创建数据库。

  要使用 `create` 操作来创建数据库，可通过添加 `DatabaseInput`（必需）参数来构建请求。

  例如：

  以下示例说明了如何使用 CLI、Boto3 或 DDL 根据您在教程中使用的 S3 存储桶中的相同 flights_data.csv 文件来定义表。

  CLI

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

  Boto3

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

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

有关数据库 API 数据类型、结构和操作的更多信息，请参阅 数据库 API。

后续步骤

在下一个部分中，您将创建一个表并将该表添加到您的数据库中。

您还可以了解您数据目录的设置和权限。请参阅 在 AWS Glue 控制台中使用数据目录设置。

第 2 步。创建表

在本步骤中，使用 AWS Glue 控制台创建表。

1. 在 AWS Glue 控制台中，从左侧菜单中选择 Tables（表）。
2. 选择 Add table (添加表)。
3. 通过在 Table details（表详细信息）中输入表的名称，设置表的属性。
4. 在 Databases（数据库）部分中，从下拉菜单选择您在步骤 1 中创建的数据库。
5. 在 Add a data store（添加数据存储）部分中，默认将选择 S3 作为源类型。
6. 对于 Data is located in（数据位置），选择 Specified path in another account（另一个账户中的指定路径）。
7. 复制并粘贴 Include path（包含路径）输入字段的路径：

   ```
   s3://crawler-public-us-west-2/flight/2016/csv/
   ```

8. 在 Data format（数据格式）部分中，对于 Classification（分类），选择 CSV，对于 Delimiter（分隔符），选择 comma (,) [逗号（,）]。选择下一步。
9. 系统会要求您定义架构。架构定义了数据记录的结构和格式。选择 Add column（添加列）。（有关更多信息，请参阅 架构注册表）。
10. 指定列属性：

    a. 输入列名。

    b. 对于 Column Type（列类型），默认情况下已选中 string（字符串）。

    c. 对于 Column number（列编号），默认情况下已选中 1。

    d. 选择 添加。

11. 系统会要求您添加分区索引。该项为可选项。要跳过此步，选择 Next（下一步）。
12. 此时将显示表属性的摘要。如果一切都符合预期，请选择创建。否则，请选择 Back（返回）并根据需要进行编辑。

恭喜您，您已手动创建了一个表并成功将其关联到数据库。新创建的表将显示在 Tables（表）控制面板中。您可以在控制面板中修改和管理您的所有表。

有关更多信息，请参阅 在 AWS Glue 控制台中使用表。

后续步骤

后续步骤

现已填充数据目录，您可以开始在 AWS Glue 中编写任务了。请参阅使用 AWS Glue Studio 构建可视化 ETL 作业。

除了使用控制台之外，还有其他方法可以在数据目录中定义表，包括：

- 创建和运行爬网程序
- 在 AWS Glue 中向爬网程序添加分类器
- 使用 AWS Glue 表 API
- 使用 AWS Glue Data Catalog 模板
- 迁移 Apache Hive 元存储
- 使用 AWS CLI、Boto3 或数据定义语言（DDL）

以下示例说明了如何使用 CLI、Boto3 或 DDL 根据您在教程中使用的 S3 存储桶中的相同 flights_data.csv 文件来定义表。

请参阅有关如何构造 AWS CLI 命令的文档。CLI 示例包含“aws glue create-table --table-input”值的 JSON 语法。

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')
```

设置对数据存储的网络访问

要运行提取、转换和加载 (ETL) 作业，AWS Glue 必须能够访问您的数据存储。如果不需要在您的 Virtual Private Cloud（VPC）子网中运行任务（例如，将数据从 Amazon S3 转换到 Amazon S3），则无需进行额外的配置。

如果需要在您的 VPC 子网中运行任务（例如，转换私有子网中的 JDBC 数据存储中的数据），AWS Glue 将设置弹性网络接口，使您的任务能够安全可靠地连接您 VPC 中的其他资源。每个弹性网络接口都会从您指定的子网中的 IP 地址范围内获得一个私有 IP 地址。不会获得公有 IP 地址。在 AWS Glue 连接中指定的安全组应用于每个弹性网络接口上。有关更多信息，请参阅 设置 Amazon VPC 以通过建立从 AWS Glue 到 Amazon RDS 数据存储的 JDBC 连接。

必须能够从 VPC 子网使用作业访问的所有 JDBC 数据存储。要从您的 VPC 内访问 Amazon S3，需要一个 VPC 终端节点。如果您的任务既要访问 VPC 资源又要访问公有 Internet，VPC 内部必须具有网络地址转换（NAT）网关。

一个作业或开发终端节点一次只能访问一个 VPC (以及子网)。如果需要访问不同 VPC 中的数据存储，可以进行以下选择：

- 使用 VPC 对等访问数据存储。有关 VPC 对等的更多信息，请参阅 VPC 对等基本知识
- 使用 Amazon S3 存储桶作为中间存储位置。将工作拆分成两个任务，将任务 1 的 Amazon S3 输出作为任务 2 的输入。

有关如何使用 Amazon VPC 连接到 Amazon Redshift 数据存储的详细信息，请参阅 the section called “配置 Redshift”。

有关如何使用 Amazon VPC 连接到 Amazon RDS 数据存储的详细信息，请参阅 the section called “设置 Amazon VPC 以连接到 Amazon RDS 数据存储”。

在 Amazon VPC 中设置必要规则后，您就可以在 AWS Glue 中创建一个连接以连接您的数据存储所需的属性。有关连接的更多信息,请参阅 连接到数据。

> **Note**
>
> 确保针对 AWS Glue 设置您的 DNS 环境。有关更多信息，请参阅 在 VPC 中设置 DNS。

主题

设置 VPC 以连接到 PyPI for AWS Glue

Python Package Index（PyPI）是 Python 编程语言的软件库。本主题介绍支持使用 pip install 的软件包所需的详细信息（由会话创建者使用 `--additional-python-modules` 标志指定）。

使用与连接器的 AWS Glue 交互式会话会导致通过为连接器指定的子网使用 VPC 网络。因此，除非您设置特殊配置，否则 AWS 服务和其他网络目标不可用。

该问题的解决方法包括：

- 使用您的会话可以访问的互联网网关。
- 设置并使用带有 PyPI/simple 存储库的 S3 存储桶，其中包含软件包集依赖关系的传递闭包。
- 使用镜像 PyPI 并连接到您的 VPC 的 CodeArtifact 存储库。

设置互联网网关

技术方面在 NAT 网关用例中有详细介绍，但请注意使用 `--additional-python-modules` 的这些要求。具体而言，`--additional-python-modules` 需要访问 pypi.org，这取决于您的 VPC 的配置。请注意以下要求：

1. 要求通过 pip install 为用户会话安装其他 python 模块。如果会话使用连接器，则您的配置可能会受到影响。

2. 当连接器与 `--additional-python-modules` 一起使用时，启动会话时，与该连接器的 `PhysicalConnectionRequirements` 关联的子网必须提供到达 pypi.org 的网络路径。
3. 您必须确定配置是否正确。

设置 Amazon S3 存储桶以托管目标 PyPI/simple 存储库

此示例在 Amazon S3 中为一组软件包及其依赖关系设置了 PyPI 镜像。

要为一组软件包设置 PyPI 镜像，请执行以下操作：

```
# 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/*
```

如果您已经有现有构件存储库，它将有一个索引 URL 供 pip 使用，您可以提供该索引 URL 来代替上述 Amazon S3 存储桶的示例 URL。

要使用自定义 index-url，请使用一些示例包：

```
%%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"
}
```

设置连接到您的 VPC 的 pypi 的 CodeArtifact 镜像

要设置镜像，请执行以下操作：

1. 在与连接器使用的子网相同的区域中创建存储库。

   选择 `Public upstream repositories`，然后选择 `pypi-store`。
2. 提供从 VPC 访问子网存储库的权限。
3. 使用 `python-modules-installer-option` 指定正确的 `--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"
}
```

有关更多信息，请参阅 Use CodeArtifact from a VPC。

在 VPC 中设置 DNS

域名系统 (DNS) 是 Internet 中名称使用的标准，以将名称解析到各自相应的 IP 地址。DNS 主机名称可以唯一的命名计算机，它由主机名称和域名组成。DNS 服务器会将 DNS 主机名称解析到其相应的 IP 地址。

要在 VPC 中设置 DNS，请确保您的 VPC 中同时启用了 DNS 主机名和 DNS 解析。VPC 网络属性 `enableDnsHostnames` 和 `enableDnsSupport` 必须设置为 `true`。要查看和修改这些属性，请转到 VPC 控制台：https://console.aws.amazon.com/vpc/ 的 。

有关更多信息，请参阅将 DNS 与您的 VPC 一起使用。此外，您可以使用 AWS CLI 并调用 modify-vpc-attribute 命令来配置 VPC 网络属性。

> **Note**
>
> 如果您使用的是 Route 53，请确认您的配置不会覆盖 DNS 网络属性。

在 AWS Glue 中设置加密

以下示例工作流程重点介绍在对 AWS Glue 使用加密时要配置的选项。该示例演示如何使用特定的 AWS Key Management Service (AWS KMS) 密钥，但您可以根据您的特定需求选择其他设置。此工作流程只重点介绍在设置 AWS Glue 时与加密有关的选项。

1. 如果 AWS Glue 控制台的用户不使用允许所有 AWS Glue API 操作（例如，`"glue:*"`）的权限策略，请确认允许以下操作：
   - `"glue:GetDataCatalogEncryptionSettings"`
   - `"glue:PutDataCatalogEncryptionSettings"`
   - `"glue:CreateSecurityConfiguration"`
   - `"glue:GetSecurityConfiguration"`

- "glue:GetSecurityConfigurations"
- "glue:DeleteSecurityConfiguration"

2. 访问加密目录的任何客户端，即任何控制台用户、爬网程序、任务或开发终端节点都需要以下权限：

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

3. 访问加密连接密码的任何用户或角色都需要以下权限。

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

4. 将加密数据写入 Amazon S3 的任何提取、转换和加载（ETL）任务的角色都需要以下权限。

```
{
 "Version": "2012-10-17",
  "Statement": {
     "Effect": "Allow",
     "Action": [
           "kms:Decrypt",
           "kms:Encrypt",
           "kms:GenerateDataKey"
      ],
```

```
      "Resource": "<key-arns-used-for-s3>"
    }
}
```

5. 任何写入加密 Amazon CloudWatch Logs 的 ETL 作业或爬网程序在密钥政策和 IAM policy 中都需要以下权限。

   在密钥政策（而不是在 IAM policy）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>"
 }
```

   有关 密钥策略的更多信息，请参阅《AWS Key Management Service 开发人员指南》中的使用 AWS KMS 中的密钥策略。

   在 IAM policy 中，请附加 `logs:AssociateKmsKey` 权限：

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

6. 任何使用加密作业书签的 ETL 作业都需要以下权限。

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

7. 在 AWS Glue 控制台的导航窗格上选择 Settings (设置)。

   a. 在存储库的 Data catalog settings (数据目录设置) 页面上，选择 Metadata encryption (元数据加密)，加密您的数据目录。此选项将使用您选择的 AWS KMS 密钥加密数据目录中的所有对象。

   b. 对于 AWS KMS 密钥 (Amazon KMS 密钥)，请选择 aws/glue。您也可以选择您创建的 AWS KMS 键。

   > **Important**
   >
   > AWS Glue 只支持对称客户主密钥（CMK）。AWS KMS key (Amazon KMS 密钥) 列表仅显示对称密钥。但是，如果选择 Choose a AWS KMS key ARN (选择 Amazon KMS 密钥 ARN)，控制台允许您为任何密钥类型输入 ARN。确保仅为对称密钥输入 ARN。

   启用了加密时，正在访问数据目录的客户端必须具有 AWS KMS 权限。

8. 在导航窗格中，选择 Security configurations (安全配置)。安全配置是可用于配置 AWS Glue 过程的一组安全属性。然后选择 Add security configuration (添加安全配置)。在配置中，选择以下任何选项：

   a. 选择 S3 加密。对于 Encryption mode (加密模式)，选择 SSE-KMS。对于 AWS KMS key (Amazon KMS 密钥)，选择 aws/s3（确保用户有权限使用此密钥）。这支持任务写入 Amazon S3 的数据可以使用 AWS 托管的 AWS Glue AWS KMS 密钥。

   b. 选择 CloudWatch 日志加密，然后选择 CMK。（确保用户有权使用此密钥）。有关更多信息，请参阅《AWS Key Management Service 用户指南》中的使用 AWS KMS 加密 CloudWatch Logs 中的日志数据。

> **Important**
>
> AWS Glue 只支持对称客户主密钥（CMK）。AWS KMS key (Amazon KMS 密钥) 列表仅显示对称密钥。但是，如果选择 Choose a AWS KMS key ARN (选择 Amazon KMS 密钥 ARN)，控制台允许您为任何密钥类型输入 ARN。确保仅为对称密钥输入 ARN。

c. 选择高级属性，然后选择任务书签加密。对于 AWS KMS key (Amazon KMS 密钥)，选择 aws/glue（确保用户有权限使用此密钥）。这样，就可以使用 AWS Glue AWS KMS 密钥对写入 Amazon S3 的任务书签进行加密。

9. 在导航窗格中，选择 Connections (站点到站点 VPN 连接)。
   a. 选择 Add connection (添加连接) 以创建到作为 ETL 作业目标的 Java 数据库连接 (JDBC) 数据存储的连接。
   b. 要强制使用安全套接字层 (SSL) 加密，请选择 Require SSL connection (需要 SSL 连接)，并测试您的连接。
10. 在导航窗格中，选择作业。
    a. 选择 Add job (添加作业) 以创建转换数据的作业。
    b. 在作业定义中，选择您创建的安全配置。
11. 在 AWS Glue 控制台中，按需运行作业。验证任务写入的任何 Amazon S3 数据、任务写入的 CloudWatch Logs 以及任务书签都已全部加密。

为 AWS Glue 设置开发网络

要使用 AWS Glue 运行您的提取、转换和加载 (ETL) 脚本，您可以使用开发端点开发和测试脚本。不支持将开发终端节点与 AWS Glue 版本 2.0 任务一起使用。对于版本 2.0 和更高版本，首选的开发方法是使用带有 AWS Glue 内核之一的 Jupyter Notebook。有关更多信息，请参阅 the section called “开始使用 AWS Glue 交互式会话”。

针对开发终端节点设置您的网络

在设置开发终端节点时，您需要指定一个 Virtual Private Cloud (VPC)、子网和多个安全组。

> **Note**
>
> 确保针对 AWS Glue 设置您的 DNS 环境。有关更多信息，请参阅 在 VPC 中设置 DNS。

要支持 AWS Glue 访问所需资源，请在您的子网路由表中添加一行，以将 Amazon S3 的前缀列表关联到 VPC 终端节点。前缀列表 ID 对于创建允许来自 VPC 的流量通过 VPC 终端节点访问 AWS 服务的出站安全组规则来说是必需的。为了轻松连接到与此开发终端节点关联的笔记本电脑服务器，请从您的本地计算机向路由表添加一行，以添加互联网网关 ID。有关更多信息，请参阅 VPC 端点。更新子网路由表，使之类似于下表：

目标位置	目标		
10.0.0.0/16	本地		
适用于 Amazon S3 的 pl-id	vpce-id		
0.0.0.0/0	igw-xxxx		

要支持 AWS Glue 在其组件之间通信，请为所有 TCP 端口指定一个具有自引用入站规则的安全组。通过创建自引用规则，您可以将源限制为 VPC 中的同一安全组，而不将其对所有网络开放。VPC 的默认安全组可能已经为所有流量设置了自引用入站规则。

设置安全组

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 在左侧导航窗格中，选择 Security Groups (安全组)。
3. 从列表中选择一个现有安全组，或 Create Security Group (创建安全组)，以用于开发终端节点。
4. 在安全组窗格中，导航到 Inbound (入站) 选项卡。
5. 添加一个自引用规则，以允许 AWS Glue 组件进行通信。具体来讲，添加或确认有一条类型为 `All TCP` 的规则，协议为 `TCP`，端口范围包括所有端口，其源具有与组 ID相同的安全组名。

   入站规则类似于以下内容：

类型	协议	端口范围	来源
所有 TCP	TCP	0–65535	*security-group*

   下面显示了一个自引用入站规则示例：

6. 同时为出站流量添加一条规则。打开到所有端口的出站流量，或创建一条 Type (键入) `All TCP` 的自引用规则，Protocol (协议) 为 TCP，Port Range (端口范围) 包括所有端口以及其 Source (源) 具有与 Group ID (组 ID) 相同的安全组名称。

   该出站规则类似于以下规则之一：

类型	协议	端口范围	目标位置
所有 TCP	TCP	0–65535	*security-group*
所有流量	ALL	ALL	0.0.0.0/0

针对笔记本电脑服务器设置 Amazon EC2

借助开发端点，您可以创建一个笔记本服务器，用于使用 Jupyter Notebook 测试您的 ETL 脚本。要启用与笔记本的通信，请指定一个包含适用于 HTTPS (端口 443) 和 SSH (端口 22) 的入站规则的安全组。确保该规则的源是 0.0.0.0/0 或连接笔记本的计算机的 IP 地址。

设置安全组

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 在左侧导航窗格中，选择 Security Groups (安全组)。
3. 从列表中选择一个现有安全组，或 Create Security Group (创建安全组)，以用于笔记本服务器。与开发终端节点关联的安全组也可用于创建笔记本服务器。
4. 在安全组窗格中，导航到 Inbound (入站) 选项卡。
5. 添加类似于以下内容的入站规则：

类型	协议	端口范围	来源
SSH	TCP	22	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0

   下面显示了安全组的入站规则示例：

Security Group: sg-19e1b768

Description | **Inbound** | Outbound | Tags

Edit

Type	Protocol	Port Range	Source
SSH	TCP	22	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0

AWS Glue 中的数据发现和编目

AWS Glue Data Catalog 是一个集中式存储库，用于存储有关您组织数据集的元数据。它充当数据来源的位置、架构和运行时指标的索引。元数据存储在元数据表中，其中每个表代表一个单一数据存储。

您可以使用爬网程序填充 Data Catalog，它会自动扫描您的数据来源并提取元数据。爬网程序可以连接到 AWS 内部（基于 AWS）和外部的数据来源。

有关支持的数据来源的更多信息，请参阅我可以爬取哪些数据存储？。

您也可以根据自身的特定要求，通过定义表结构、架构和分区结构在 Data Catalog 中手动创建表。

有关手动创建元数据表的更多信息，请参阅手动定义元数据。

您可以使用 Data Catalog 中的信息创建和监控您的 ETL 任务。Data Catalog 与其他 AWS 分析服务集成，提供统一的数据来源视图，助您更轻松地管理和分析数据。

- Amazon Athena – 使用 SQL 在 Data Catalog 中存储和查询 Amazon S3 数据的表元数据。
- AWS Lake Formation – 集中定义和管理精细的数据访问策略并审核数据访问权限。
- Amazon EMR – 访问 Data Catalog 中定义的数据来源以进行大数据处理。
- Amazon SageMaker – 快速、自信地构建、训练和部署机器学习模型。

Data Catalog 的主要功能

以下是 Data Catalog 的主要方面。

元数据存储库

Data Catalog 充当中央元数据存储库，存储有关数据来源的位置、架构和属性的信息。该元数据被组织成数据库和表，类似于传统的关系数据库目录。

自动发现数据

AWS Glue 爬网程序 可以自动发现新的或更新的数据来源并对其进行编目，从而减少手动元数据管理的开销，并确保您的 Data Catalog 保持最新状态。通过对数据来源进行编目，Data Catalog 能让用户和应用程序更轻松地发现和了解组织内的可用数据资产，从而促进数据的重用和协作。

Data Catalog 支持各种数据来源，包括 Amazon S3、Amazon RDS、Amazon Redshift、Apache Hive 等。它可以使用 AWS Glue 爬网程序 自动推断和存储来自这些来源的元数据。

有关更多信息，请参阅使用爬网程序填充 Data Catalog 。

架构管理

Data Catalog 会自动捕获和管理数据来源的架构，包括架构推断、发展和版本控制。您可以使用 AWS Glue ETL 任务在 Data Catalog 中更新架构和分区。

表优化

为提高 AWS 分析服务（例如 Amazon Athena 和 Amazon EMR）和 AWS Glue ETL 任务的读取性能，Data Catalog 为 Data Catalog 中的 Iceberg 表提供了托管式压缩功能（一种将小的 Amazon S3 对象压缩成较大对象的进程）。您可以使用 AWS Glue 控制台、AWS Lake Formation 控制台、AWS CLI 或 AWS API 为 Data Catalog 中的单个 Iceberg 表启用或禁用压缩。

有关更多信息，请参阅 优化 Iceberg 表。

列统计数据

无需设置其他数据管道，即可为 Parquet、ORC、JSON、ION、CSV 和 XML 等数据格式的 Data Catalog 表计算列级别的统计数据。借助列统计数据，您可以深入洞察列中的值，从而了解数据特征。Data Catalog 支持生成列值统计数据，例如最小值、最大值、空值总计、非重复值总计、值的平均长度和真实值的总出现次数等。

有关更多信息，请参阅 使用列统计数据优化查询性能。

数据沿袭

Data Catalog 保留对您的数据执行的转换和操作的记录，并提供数据沿袭信息。这些沿袭信息对于审核、合规和了解数据的来源非常有价值。

与其他 AWS 服务集成

Data Catalog 与其他 AWS 服务无缝集成，例如 AWS Lake Formation、Amazon Athena、Amazon Redshift Spectrum 和 Amazon EMR。您可利用这一集成，使用单一、一致的元数据层查询和分析各种数据存储中的数据。

安全性和访问控制

AWS Glue 与 AWS Lake Formation 集成，以支持对 Data Catalog 资源的精细访问控制，从而允许您根据组织的策略和要求管理对数据资产的权限和安全访问。AWS Glue 与 AWS Key Management Service（AWS KMS）集成，以便加密存储在 Data Catalog 中的元数据。

主题

- 管理 Data Catalog
- 访问 Data Catalog
- AWS Glue Data Catalog 最佳实践
- AWS Glue 架构注册表

填充 AWS Glue Data Catalog

您可以使用以下方法填充 AWS Glue Data Catalog：

- AWS Glue 爬网程序 – AWS Glue 爬网程序 可以自动发现数据库、数据湖和流式传输数据等数据来源并对其进行分类。爬网程序可以自动发现和推断各种数据来源的元数据，因此是填充 Data Catalog 的最常用和最推荐的方法。
- 手动添加元数据 – 您可以使用 AWS Glue 控制台、Lake Formation 控制台、AWS CLI 或 AWS Glue API 手动定义数据库、表和连接详细信息，并将其添加到 Data Catalog 中。当您要对无法爬取的数据来源进行分类时，手动输入非常有用。
- 与其他 AWS 服务集成 – 您可以使用来自 AWS Lake Formation 和 Amazon Athena 等服务的元数据填充 Data Catalog。这些服务可以在 Data Catalog 中发现和注册数据来源。
- 从现有元数据存储库中填充 – 如果您有 Apache Hive Metastore 这样的现有元数据存储，则可以使用 AWS Glue 将该元数据导入到 Data Catalog 中。有关更多信息，请参阅 GitHub 上的在 Hive 元存储和 AWS Glue Data Catalog 之间迁移。

主题

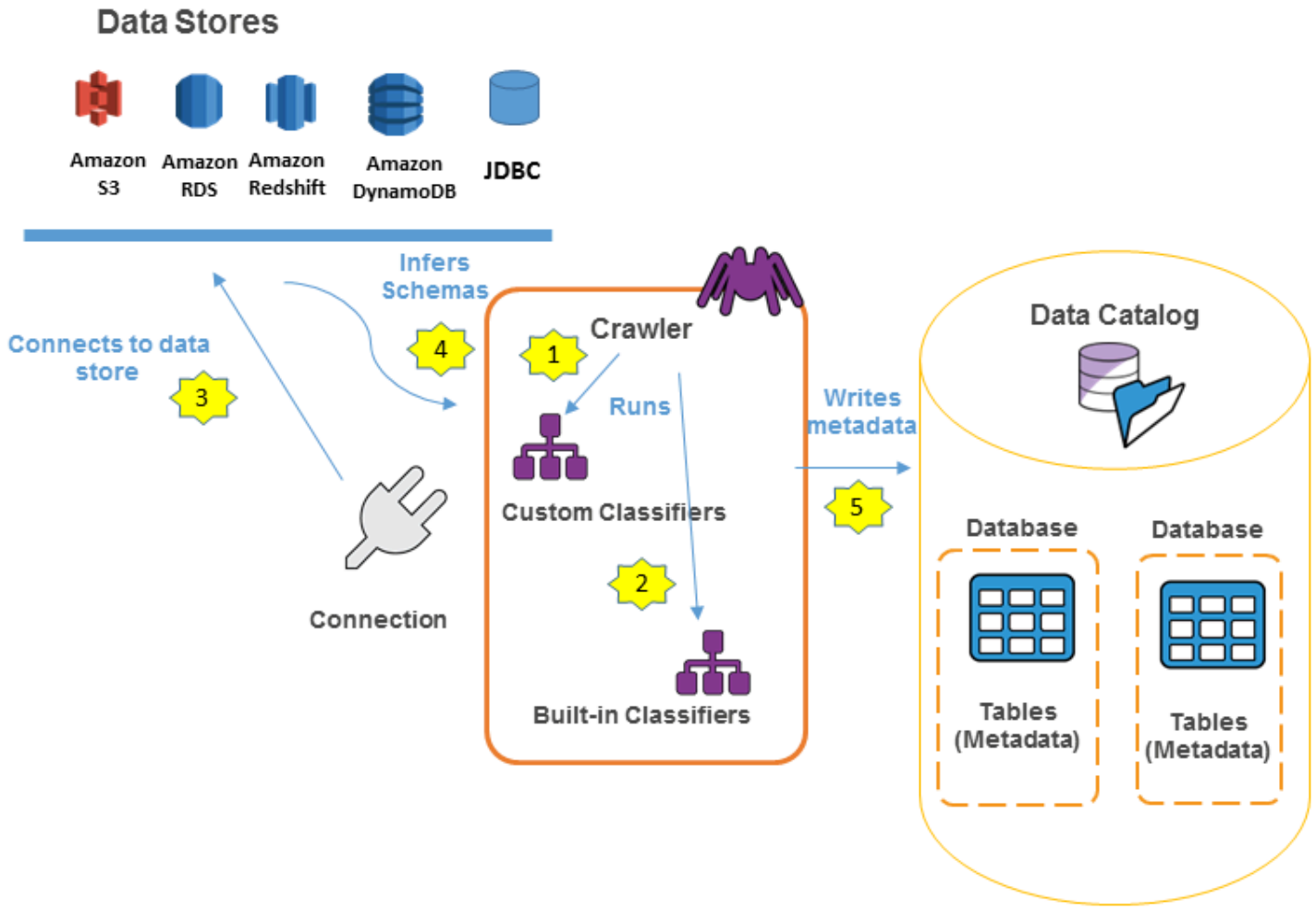
- 使用爬网程序填充 Data Catalog
- 手动定义元数据
- 与其他 AWS 服务集成
- Data Catalog 设置

使用爬网程序填充 Data Catalog

您可以使用 AWS Glue 爬网程序，将数据库和表填充到 AWS Glue Data Catalog。这是大多数 AWS Glue 用户使用的主要方法。爬网程序可以在单次运行中爬取多个数据存储。完成后，爬网程序会在数据目录中创建或更新一个或多个表。您在 AWS Glue 中定义的提取、转换和加载（ETL）任务使用这些数据目录表作为源和目标。ETL 任务从在源和目标数据目录表中指定的数据存储中读取内容并向其中写入内容。

工作流

以下流程图显示了 AWS Glue 爬网程序如何与数据存储和其他元素交互来填充数据目录。



以下是爬网程序如何填充 AWS Glue Data Catalog 的一般工作流程：

1. 爬网程序运行您为推断数据的格式和架构而选择的任何自定义分类器。您为自定义分类器提供代码，它们按您指定的顺序运行。

   第一个成功识别您的数据结构的自定义分类器用于创建架构。将会跳过列表中较低的自定义分类器。

2. 如果没有自定义分类器与您的数据的架构匹配，则内置分类符会尝试识别数据的架构。内置分类器的示例是一个可识别 JSON 的分类器。
3. 爬网程序连接到数据存储。某些数据存储需要使用连接属性才能访问爬网程序。

4. 将会为您的数据创建推断的架构。
5. 爬网程序向数据目录写入元数据。表定义包含有关您的数据存储中的数据的元数据。该表被写入一个充当数据目录中表的容器的数据库。表的属性包括分类，它是由推断表架构的分类器创建的标签。

主题

爬网程序的工作原理

爬网程序在运行时，会执行以下操作来询问数据存储：

- 对数据分类，以确定原始数据的格式、架构和关联属性 – 您可以通过创建自定义分类器来配置分类结果。
- 将数据分组为表或分区 – 根据爬网程序探试算法对数据分组。
- 将元数据写入数据目录 – 您可以配置爬网程序如何添加、更新和删除表和分区。

在定义爬网程序时，您可以选择一个或多个分类器来评估用于推断架构的数据的格式。当爬网程序运行时，列表中的第一个分类器可以成功识别您的数据存储，用于为表创建架构。您可以使用内置分类器或定义您自己的分类器。您可以在定义爬网程序之前，在单独的操作中定义您的自定义分类器。AWS Glue 提供内置分类器，用于使用包含 JSON、CSV 和 Apache Avro 的格式从公共文件中推断架构。有关 AWS Glue 中内置分类器的当前列表，请参阅AWS Glue 中的内置分类器。

爬网程序创建的元数据表包含在定义爬网程序时的数据库中。如果爬网程序未指定数据库，则您的表将放置在默认数据库中。此外，每个表都有一个分类列，由第一个成功识别数据存储的分类器填充。

如果已爬取的文件被压缩，则爬网程序必须下载它才能处理它。在爬网程序运行时，它会询问文件以确定其格式和压缩类型，并将这些属性写入数据目录。某些文件格式（例如 Apache Parquet）允许您在写入文件时压缩文件的一部分。对于这些文件，压缩的数据是文件的内部组件，在将表写入数据目录时，AWS Glue 不会填充 `compressionType` 属性。相反，如果整个文件通过压缩算法（例如 gzip）来压缩，则在将表写入数据目录时，会填充 `compressionType` 属性。

爬网程序为它创建的表生成名称。存储在 AWS Glue Data Catalog 中的表的名称遵循以下规则：

- 仅允许使用字母数字字符和下划线 (_)。
- 任何自定义前缀均不能超过 64 个字符。
- 名称的最大长度不能超过 128 个字符。爬网程序截断生成的名称以适应限制范围。
- 如果遇到重复的表名，则爬网程序会向名称中添加哈希字符串后缀。

如果您的爬网程序多次运行 (也许是按计划)，它会在您的数据存储中查找新的或已更改的文件或表。爬网程序的输出包括自上次运行以来找到的新表和分区。

我可以爬取哪些数据存储？

爬网程序可以同时爬取以下基于文件的数据存储和基于表的数据存储。

爬网程序使用的访问类型	数据存储
本机客户端	• Amazon Simple Storage Service (Amazon S3) • Amazon DynamoDB • Delta Lake 2.0.x • Apache Iceberg 1.5 • Apache Hudi 0.14
JDBC	Amazon Redshift Snowflake 在 Amazon Relational Database Service（Amazon RDS）内部或 Amazon RDS 外部： • Amazon Aurora • MariaDB

爬网程序使用的访问类型	数据存储
	• Microsoft SQL Server • MySQL • Oracle • PostgreSQL
MongoDB 客户端	• MongoDB • MongoDB Atlas • Amazon DocumentDB (with MongoDB compatibility)

> **Note**
>
> 目前，AWS Glue 不支持数据流的爬网程序。

对于 JDBC、MongoDB、MongoDB Atlas 和 Amazon DocumentDB（与 MongoDB 兼容）数据存储，您必须指定爬网程序可用于连接到数据存储的 AWS Glue 连接。对于 Amazon S3，您可以选择指定网络类型的连接。连接是存储连接信息（例如凭证、URL、Amazon Virtual Private Cloud 信息等）的数据目录对象。有关更多信息，请参阅 连接到数据。

以下是爬网程序支持的驱动程序版本：

产品	爬网程序支持的驱动程序
PostgreSQL	42.2.1
Amazon Aurora	与原生爬网程序爬虫驱动程序相同
MariaDB	8.0.13
Microsoft SQL Server	6.1.0
MySQL	8.0.13
Oracle	11.2.2
Amazon Redshift	4.1

产品	爬网程序支持的驱动程序
Snowflake	3.13.20
MongoDB	4.7.2
MongoDB Atlas	4.7.2

以下是有关各种数据存储的说明。

Amazon S3

您可以选择在您的账户中或在其他账户中爬取路径。如果文件夹中的所有 Amazon S3 文件具有相同的架构，爬网程序会创建一个表。此外，如果对 Amazon S3 对象进行了分区，则只会创建一个元数据表，并将分区信息添加到该表的数据目录中。

Amazon S3 和 Amazon DynamoDB

爬网程序使用 AWS Identity and Access Management（IAM）权限角色来访问您的数据存储。传递给爬网程序的角色必须有权访问所爬取的 Amazon S3 路径和 Amazon DynamoDB 表。

Amazon DynamoDB

在使用 AWS Glue 控制台定义爬网程序时，请指定一个 DynamoDB 表。如果您使用的是 AWS Glue API，则可以指定表的列表。您可以选择仅爬取一小部分数据样本以减少爬网程序的运行时间。

Delta Lake

对于每个 Delta Lake 数据存储，可以指定如何创建 Delta 表：

- 创建原生表：允许与支持直接查询 Delta 事务日志的查询引擎集成。有关更多信息，请参阅查询 Delta Lake 表。
- 创建符号链接表：根据指定的配置参数，使用由分区键分区的清单文件创建 `_symlink_manifest` 文件夹。

Iceberg

对于每个 Iceberg 数据存储，您可以指定一个 Amazon S3 路径，其中包含您的 Iceberg 表的元数据。爬网程序发现 Iceberg 表元数据后，会在 Data Catalog 中注册该元数据。您可以为爬网程序设置计划以保持表格最新。

您可以为数据存储定义以下参数：

- 排除项：允许您跳过某些文件夹。
- 最大遍历深度：设置爬网程序可以在您的 Amazon S3 存储桶中爬取的深度限制。默认的最大遍历深度为 10；您可以设置的最大深度为 20。

Hudi

对于每个 Hudi 数据存储，您可以指定一个 Amazon S3 路径，其中包含您的 Hudi 表的元数据。爬网程序发现 Hudi 表元数据后，会在 Data Catalog 中注册该元数据。您可以为爬网程序设置计划以保持表格最新。

您可以为数据存储定义以下参数：

- 排除项：允许您跳过某些文件夹。
- 最大遍历深度：设置爬网程序可以在您的 Amazon S3 存储桶中爬取的深度限制。默认的最大遍历深度为 10；您可以设置的最大深度为 20。

> **Note**
>
> 由于与 Hudi 0.13.1 和时间戳类型不兼容，逻辑类型为 `millis` 的时间戳列将被解释为 `bigint`。在即将发布的 Hudi 版本中可能会提供解决方案。

Hudi 表分为以下几类，每个表都有具体含义：

- 写入时复制（CoW）：数据以列式（Parquet）存储，并且每次更新都会在写入过程中创建一个新版本的文件。
- 读取时合并（MOR）：数据使用列式（Parquet）和基于行（Avro）的格式的组合进行存储。更新记录到基于行的增量文件中，并根据需要进行压缩以创建新版本的列式文件。

对于 CoW 数据集，每次更新记录时，包含该记录的文件都会使用更新后的值进行重写。对于 MoR 数据集，每次进行更新时，Hudi 仅写入已更改记录对应的行。MoR 更适合写入或更改繁重而读取量较少的工作负载。CoW 更适合更改频率较低但读取量繁重的工作负载。

Hudi 提供三个查询类型用于访问数据：

- 快照查询：该查询用于查看截至给定提交或压缩操作时表的最新快照。对于 MOR 表，快照查询通过在查询时合并最新文件切片的基本文件和增量文件来显示表的最新状态。
- 增量查询：查询只能看到自给定提交/压缩以来，写入表的新数据。这有效地提供了更改流以启用增量数据管道。
- 读取优化查询：对于 MoR 表，查询将看到最新压缩的数据。对于 CoW 表，查询将看到最新提交的数据。

对于写入时复制表，爬网程序使用 ReadOptimized serde 在 Data Catalog 中创建单个表。`org.apache.hudi.hadoop.HoodieParquetInputFormat`

对于读取时合并表，爬网程序在 Data Catalog 中为同一个表位置创建两个表：

- 带有后缀 `_ro` 的表，使用 ReadOptimized serde `org.apache.hudi.hadoop.HoodieParquetInputFormat`。
- 带有后缀 `_rt` 的表，使用允许快照查询的 RealTime Serde：`org.apache.hudi.hadoop.realtime.HoodieParquetRealtimeInputFormat`。

MongoDB 和 Amazon DocumentDB (with MongoDB compatibility)

支持 MongoDB 版本 3.2 及更高版本。您可以选择仅爬取一小部分数据样本以减少爬网程序的运行时间。

关系数据库

使用数据库用户名和密码进行身份验证。根据数据库引擎的类型，您可以选择要爬网哪些对象，如数据库、架构和表。

Snowflake

Snowflake JDBC 爬网程序支持爬取表、外部表、视图和实体化视图。不会填充实体化视图定义。

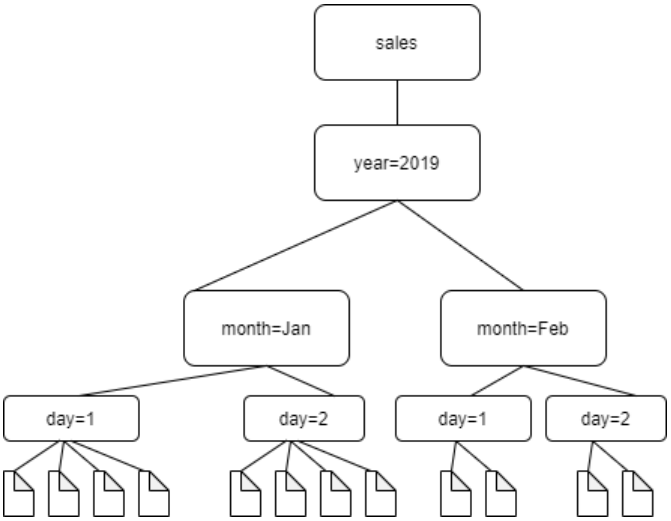
对于 Snowflake 外部表，仅爬网程序指向 Amazon S3 位置时才会爬取。除了表架构外，爬网程序还将爬取 Amazon S3的位置、文件格式和输出作为数据目录表中的表参数。请注意，未填充分区外部表的分区信息。

使用 Snowflake 爬网程序创建的数据目录表目前不支持 ETL。

爬网程序如何确定何时创建分区？

当 AWS Glue 爬网程序扫描 Amazon S3 并检测存储桶中的多个文件夹时，它会在文件夹结构中确定表的根，以及哪些文件夹是表的分区。表的名称基于 Amazon S3 前缀或文件夹名称。您提供一个包含路径，它指向要爬网的文件夹级别。当某个文件夹级别的大部分架构都相似时，爬网程序将创建一个表而不是独立表的分区。要影响爬网程序，以创建单独的表，可在定义爬网程序时将每个表的根文件夹添加为单独的数据存储。

例如，考虑以下 Amazon S3 文件夹结构。



四个最低级别文件夹的路径如下：

```
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
```

假设爬网程序目标设置为 Sales，并且 day=*n* 文件夹中的所有文件都具有相同的格式（例如 JSON，未加密），并且具有相同或非常相似的架构。爬网程序将创建一个包含四个分区的表，分区键为 year、month 和 day。

在下一个示例中，考虑以下 Amazon S3 结构：

```
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 下的文件架构相似，并且在爬网程序中通过包含路径 s3://bucket01/folder1/ 定义了单个数据存储，爬网程序将创建一个具有两个分区键列的表。第一个分区键列包含 table1 和 table2，第二个分区键列包含 table1 分区的 partition1 到 partition3 以及 table2 分区的 partition4 和 partition5。要创建两个单独的表，可用两个数据存储来定义爬网程序。在本示例中，将第一个包含路径定义为 s3://bucket01/folder1/table1/，将第二个定义为 s3://bucket01/folder1/table2。

> **Note**
>
> 在 Amazon Athena 中，每个表对应一个 Amazon S3 前缀，所有对象都在其中。如果对象具有不同架构，Athena 则不会将同一前缀内的不同对象识别为不同的表。如果爬网程序从同一个 Amazon S3 前缀中创建多个表，则可能出现这种情况。这可能会导致 Athena 中的查询返回零个结果。为使 Athena 正确识别和查询表，请在创建爬网程序时，使 Amazon S3 文件夹结构中的每个不同的表架构具有单独的包含路径。有关更多信息，请参阅将 Athena 与 AWS Glue 一起使用时的最佳实践以及这篇 AWS 知识中心文章。

爬网程序先决条件

爬网程序代入您在定义它时指定的 AWS Identity and Access Management（IAM）角色的权限。此 IAM 角色必须有权从您的数据存储中提取数据并将其写入数据目录。AWS Glue 控制台仅列出已为 AWS Glue 委托人服务附加信任策略的 IAM 角色。从控制台中，您还可以创建具有 IAM policy 的 IAM 角色，该策略允许访问爬网程序所访问的 Amazon S3 数据存储。有关为 AWS Glue 提供角色的更多信息，请参阅Glue 基于身份的策略 AWS。

> **Note**
>
> 在爬取 Delta Lake 数据存储时，您必须拥有该 Amazon S3 位置的读/写权限。

对于您的爬网程序，您可以创建一个角色并附加以下策略：

- `AWSGlueServiceRole` AWS 托管策略，授予对数据目录所需的权限
- 授予数据源权限的内联策略。
- 对角色授予 `iam:PassRole` 权限的内联策略。

一种更快的方法是让 AWS Glue 控制台爬网程序向导为您创建一个角色。它创建的角色专用于爬网程序，包括 `AWSGlueServiceRole` AWS 托管策略以及指定数据源所需的内联策略。

如果您为爬网程序指定现有角色，请确保它包含 `AWSGlueServiceRole` 策略或等效策略（或此策略的范围缩小版本），以及所需的内联策略。例如，对于 Amazon S3 数据存储，内联策略至少为以下内容：

```
{
```

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

对于 Amazon DynamoDB 数据存储，策略至少为以下内容：

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

此外，如果爬网程序读取 AWS Key Management Service（AWS KMS）加密的 Amazon S3 数据，则 IAM 角色必须具有 AWS KMS 密钥的解密权限。有关更多信息，请参阅 步骤 2：为 AWS Glue 创建 IAM 角色。

配置爬网程序

爬网程序访问您的数据存储，提取元数据并在 AWS Glue Data Catalog 中创建表定义。AWS Glue 控制台中的 Crawlers (爬网程序) 窗格列出了您创建的所有爬网程序。此列表显示上次运行的爬网程序的状态和指标。

> **Note**
>
> 如果您选择引入自己的 JDBC 驱动程序版本，则 AWS Glue 爬网程序将消耗 AWS Glue 作业和 Amazon S3 存储桶中的资源，以确保您提供的驱动程序在您的环境中运行。额外的资源使用量将反映在您的账户中。此外，提供自己的 JDBC 驱动程序并不意味着爬网程序能够利用该驱动程序的所有功能。驱动程序仅限于添加 AWS Glue 连接中描述的属性。

配置爬网程序

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。在导航窗格中选择 Crawlers (爬网程序)。
2. 选择添加爬网程序，然后按照添加爬网程序向导中的说明进行操作。该向导将引导您完成创建爬网程序所需的步骤。如果要添加自定义分类器来定义架构，请参阅在 AWS Glue 中向爬网程序添加分类器。

步骤 1：设置爬网程序属性

为您的爬网程序输入名称和描述（可选）。（可选）您可以使用 Tag key (标签键) 和可选的 Tag value (标签值) 来标记爬网程序。创建之后，标签键处于只读状态。对某些资源使用标签可帮助您整理和标识资源。有关更多信息，请参阅 AWS Glue 中的 AWS 标签。

名称

名称可以包含字母 (A-Z)、数字 (0-9)、连字符 (-)、或下划线 (_)，且长度最多为 255 个字符。

描述

描述长度最多为 2048 个字符。

标签

使用标签来组织和标识资源。有关更多信息，请参阅下列内容：

- AWS Glue 中的 AWS 标签

步骤 2：选择数据源和分类器

数据源配置

为您的数据是否已映射到 AWS Glue 表？选择相应的选项，选择“尚未”或“是”。默认情况下已选择“尚未”。

爬网程序可以直接访问数据存储作为爬取的源，也可以使用数据目录中的现有表作为源。如果爬网程序使用现有目录表，它将爬取由这些目录表指定的数据存储。

- 还没有：选择一个或多个要爬取的数据源。一个爬网程序可以爬取多个不同类型的数据存储（Amazon S3、JDBC 等）。

  您一次只能配置一个数据存储。提供连接信息并包含路径和排除模式后，您可以选择添加另一个数据存储。

- 是：从您的 AWS Glue 数据目录中选择现有表。目录表指定要爬取的数据存储。爬网程序在单次运行中只能爬取目录表；它无法混用其他源类型。

  将目录表指定为源的常见原因是在您手动创建该表（因为您已经知道数据存储的结构）时，您希望爬网程序让表保持更新，包括添加新分区。有关其他原因的讨论，请参阅使用爬网程序更新手动创建的数据目录表。

  当您指定现有表作为爬网程序源类型时，以下条件适用：

  - 数据库名称是可选的。
  - 仅允许指定 Amazon S3 或 Amazon DynamoDB 数据存储的目录表。
  - 爬网程序运行时，不会创建新的目录表。根据需要更新现有表，包括添加新分区。
  - 将忽略在数据存储中找到的已删除对象；不删除目录表。而是由爬网程序写入日志消息。(`SchemaChangePolicy.DeleteBehavior=LOG`)
  - 默认情况下启用为每个 Amazon S3 路径创建单一架构的爬网程序配置选项，且不能禁用此选项。（`TableGroupingPolicy=CombineCompatibleSchemas`）有关更多信息，请参阅如何为每个 Amazon S3 包含路径创建单个架构。
  - 您不能将目录表与任何其他源类型（例如 Amazon S3 或 Amazon DynamoDB）混用来作为源。

数据来源

选择或添加要由爬网程序扫描的数据源列表。

（可选）如果选择 JDBC 作为数据来源，则在指定存储驱动程序信息的连接访问权限时，可以使用自己的 JDBC 驱动程序。

包含路径

在评估要在爬取中包含或排除的内容时，爬网程序首先评估所需的包含路径。对于 Amazon S3、MongoDB、MongoDB Atlas、Amazon DocumentDB（与 MongoDB 兼容）和关系数据存储，您必须指定包含路径。

对于 Amazon S3 数据存储

选择是在此账户还是其他账户中指定路径，然后浏览以选择 Amazon S3 路径。

对于 Amazon S3 数据存储，包含路径语法为 `bucket-name/folder-name/file-name.ext`。要网络爬取存储桶中的所有对象，只需在包含路径中指定存储桶名称。排除模式与包含路径是相对的

对于 Delta Lake 数据存储

将一个或多个指向 Delta 表的 Amazon S3 路径，格式为 s3://*bucket*/*prefix*/*object*。

适用于 Iceberg 或 Hudi 数据存储

指定一个或多个包含 Iceberg 或 Hudi 表元数据为 s3://*bucket*/*prefix* 的文件夹的 Amazon S3 路径。

对于 Hudi 数据存储，Hudi 文件夹可能位于根文件夹的子文件夹中。爬网程序将扫描路径下的所有文件夹，寻找 Hudi 文件夹。

对于 JDBC 数据存储

输入 *<database>*/*<schema>*/*<table>* 或 *<database>*/*<table>*，具体取决于数据库产品。Oracle Database 和 MySQL 不支持路径中的架构。您可以用百分比 (%) 字符替换 *<schema>* 或 *<table>*。例如，对于系统标识符 (SID) 为 `orcl` 的 Oracle 数据库，输入 `orcl/%` 以导入连接中指定的用户有权访问的所有表。

> **Important**
>
> 此字段区分大小写。

对于 MongoDB、MongoDB Atlas 或 Amazon DocumentDB 数据存储

输入 *database*/*collection*。

对于 MongoDB、MongoDB Atlas 和 Amazon DocumentDB（与 MongoDB 兼容），语法为 `database/collection`。

对于 JDBC 数据存储，语法为 `database-name/schema-name/table-name` 或 `database-name/table-name`。语法取决于数据库引擎是否支持数据库中的架构。例如，对于 MySQL 或 Oracle 等数据库引擎，请不要在包含路径中指定 `schema-name`。您可以用百分号 (%) 取代包含路径中的架构或表，以表示数据库中的所有架构或所有表。您不能用百分号 (%) 取代包含路径中的数据库。

最大遍历深度（仅适用于 Iceberg 或 Hudi 数据存储）

定义爬网程序可以遍历的最大 Amazon S3 路径深度，以发现 Amazon S3 路径中的 Iceberg 或 Hudi 元数据文件夹。此参数的目的是限制爬网程序的运行时间。默认值为 10；最大值为 20。

排除模式

您可以使用这些模式从爬取中排除某些文件或表。排除路径与包含路径是相对的。例如，要排除 JDBC 数据存储中的一个表，请在排除路径中键入该表的名称。

爬网程序使用包含 JDBC URI 连接字符串的 AWS Glue 连接连接到 JDBC 数据存储。该爬网程序只能使用 AWS Glue 连接中的 JDBC 用户名和密码来访问数据库引擎中的对象。爬网程序只能创建它通过 JDBC 连接可以访问的表。在爬网程序使用 JDBC URI 访问数据库引擎后，包含路径用于确定在数据目录中创建了数据库引擎中的哪些表。例如，对于 MySQL，如果您指定包含路径 `MyDatabase/%`，则 `MyDatabase` 中的所有表都是在数据目录中创建的。访问 Amazon Redshift 时，如果您指定包含路径 `MyDatabase/%`，则数据库 `MyDatabase` 的所有架构中的所有表都是在数据目录中创建的。如果您指定包含路径 `MyDatabase/MySchema/%`，则会创建数据库 `MyDatabase` 和架构 `MySchema` 中的所有表。

指定包含路径后，您可以通过指定一个或多个 Unix 样式 `glob` 排除模式，从网络爬取中排除对象 (否则您的包含路径会包括它)。这些模式应用于您的包含路径，以确定哪些对象被排除。这些模式也会存储为由爬网程序创建的表的属性。AWS GluePySpark 扩展（例如 `create_dynamic_frame.from_catalog`）读取表属性并排除由排除模式定义的对象。

AWS Glue 在排除模式中支持以下类型的 `glob` 模式。

排除模式	描述
`*.csv`	与表示当前文件夹中以 `.csv` 结尾的对象名称的 Amazon S3 路径匹配
`*.*`	与包含点的所有对象名称匹配
`*.{csv,avro}`	与以 `.csv` 或 `.avro` 结尾的对象名称匹配

排除模式	描述
foo.?	与以 foo. 开头，后跟单个字符扩展名的对象名称匹配
myfolder/*	与 myfolder 的一个级别子文件夹中的对象匹配，例如 /myfolder/mysource
myfolder/*/*	与 myfolder 的两个级别子文件夹中的对象匹配，例如 /myfolder/mysource/data
myfolder/**	与 myfolder 的所有子文件夹中的对象匹配，例如 /myfolder/mysource/mydata 和 /myfolder/mysource/data
myfolder**	匹配子文件夹 myfolder 以及 myfolder 下的文件，如 /myfolder 和 /myfolder/mydata.txt
Market*	与具有以 Market 开头的名称的 JDBC 数据库中的表匹配，例如 Market_us 和 Market_fr

AWS Glue 解释 glob 排除模式，如下所示：

- 斜杠 (/) 字符是将 Amazon S3 密钥分隔到文件夹层次结构中的分隔符。
- 星号 (*) 字符与不跨越文件夹边界的名称组分的零个或多个字符匹配。
- 双星号 (**) 与跨越文件夹或架构边界的零个或多个字符匹配。
- 问号 (?) 字符恰好匹配名称组分的一个字符。
- 反斜杠 (\) 字符用于对其他可以解释为特殊字符的字符进行转义。表达式 \\ 与单个反斜杠匹配，\{ 与左大括号匹配。
- 方括号 [ ] 创建一个与名称组分的单个字符 (来自一组字符) 匹配的方括号表达式。例如，[abc] 与 a、b 或 c 匹配。连字符 (-) 可用于指定范围，因此 [a-z] 指定与从 a 到 z (含) 的范围匹配。这些形式可以混合，因此 [abce-g] 与 a、b、c、e、f 或 g 匹配。如果方括号 ([) 后的字符是感叹号 (!)，则括号表达式是否定的。例如，[!a-c] 与 a、b 或 c 以外的任何字符匹配。

在方括号表达式内，*、？和 \ 字符与自身匹配。如果连字符 (-) 是方括号内的第一个字符，或者如果它在您否定时是 ！ 之后的第一个字符，则它与自身匹配。

- 大括号 ({ }) 将一组子模式 (如果组中的任何子模式匹配，则组匹配) 括起来。逗号 (,) 字符用于分隔子模式。不能对组进行嵌套。
- 在匹配操作中，文件名中的前导句点或点字符被视为正常字符。例如，* 排除与文件名 .hidden 匹配的模式。

Example Amazon S3 排除模式

将会根据包含路径计算每个排除模式。例如，假设您具有以下 Amazon S3 目录结构：

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

假定包含路径 s3://mybucket/myfolder/，下面是排除模式的一些示例结果：

排除模式	结果
departments/**	排除 departments 下面的所有文件和文件夹，并包括 employees 文件夹及其文件
departments/market*	排除 market-us.json 、market-emea.json 和 market-ap.json
**.csv	排除名称以 .csv 结尾的 myfolder 下面的所有对象
employees/*.csv	排除 employees 文件夹中的所有 .csv 文件

Example 排除 Amazon S3 分区的子集

假设数据按天进行分区，以便一年中的每一天都在单独的 Amazon S3 分区中。对于 2015 年 1 月，有 31 个分区。现在，若要仅网络爬取 1 月的第一周内的数据，您必须排除除第 1 到第 7 天之外的所有分区：

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

让我们来看一下此 glob 模式的几个部分。第一部分 `2015/01/{[!0],0[8-9]}**` 排除不以“0”开头的所有天，以及 2015 年 01 月中的第 08 和第 09 天。请注意，“**”用作日期数字模式的后缀，并穿越文件夹边界直到较低级别的文件夹。如果使用“*”，则不排除较低文件夹级别。

第二部分 `2015/0[2-9]/**` 排除 2015 年 02 月 09 月中的天。

第三部分 `2015/1[0-2]/**` 排除 2015 年 10、11 和 12 月中的天。

Example JDBC 排除模式

假设您使用以下架构结构网络爬取 JDBC 数据库：

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

假定包含路径 `MyDatabase/MySchema/%`，下面是排除模式的一些示例结果：

排除模式	结果
`HR*`	排除名称以 HR 开头的表
`Market_*`	排除名称以 `Market_` 开头的表
`**_Table`	排除名称以 `_Table` 结尾的所有表

其他爬网程序源参数

每种源类型都需要一组不同的附加参数。以下是一个不完整的列表：

Connection

选择或添加 AWS Glue 连接。有关连接的信息，请参阅 连接到数据。

其他元数据 — 可选（适用于 JDBC 数据存储）

选择爬网程序要爬取的其他元数据属性。

- 注释：爬取相关的表级和列级注释。
- 原始类型：在其他元数据中保留表列的原始数据类型。作为默认行为，爬网程序将原始数据类型转换为与 Hive 兼容的类型。

JDBC 驱动程序类名 - 可选（对于 JDBC 数据存储）

键入自定义 JDBC 驱动程序类名，以便爬网程序连接到数据来源：

- 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
- Oracle：oracle.jdbc.driver.OracleDriver
- SQL Server：com.microsoft.sqlserver.jdbc.SQLServerDriver

JDBC 驱动程序 S3 路径 - 可选（对于 JDBC 数据存储）

选择 `.jar` 文件的现有 Amazon S3 路径。使用自定义 JDBC 驱动程序让爬网程序连接到数据来源时，`.jar` 文件将存储在这里。

启用数据采样（仅适用于 Amazon DynamoDB、MongoDB、MongoDB Atlas 和 Amazon DocumentDB 数据存储）

选择是否仅对数据样本进行爬网。如果未选择该选项，则对整个表进行爬网。当表不是高吞吐量表时，扫描所有记录会花费很长时间。

创建用于查询的表（仅适用于 Delta Lake 数据存储）

选择要如何创建 Delta Lake 表：

- 创建原生表：允许与支持直接查询 Delta 事务日志的查询引擎集成。
- 创建符号链接表：根据指定的配置参数，使用由由分区键分区的清单文件创建符号链接清单文件夹。

扫描速率 — 可选（仅适用于 DynamoDB 数据存储）

指定爬网程序使用的 DynamoDB 表读取容量单位的百分比。读取容量单位是一个由 DynamoDB 定义的术语，它是一个数值，用作每秒可对表执行的读取次数的速率限制器。请输入介于 0.1 和 1.5 之间的值。如果未指定，则该值默认为 0.5（对于预配置的表）和配置的最大容量的 1/4（对于按需表）。请注意，只应将预置容量模式与 AWS Glue 网络爬取程序结合使用。

> **Note**
>
> 对于 DynamoDB 数据存储，请设置预置容量模式来处理表的读写。AWS Glue 网络爬取程序不应与按需容量模式结合使用。

网络连接 — 可选（仅适用于 Amazon S3 数据存储）

可以包括一个用于此 Amazon S3 目标的网络连接。请注意，每个爬网程序仅限于一个网络连接，因此任何其他 Amazon S3 目标也将使用相同的连接（如果留空，则不使用任何连接）。

有关连接的信息，请参阅 连接到数据。

仅对文件子集和样本大小进行采样（仅适用于 Amazon S3 数据存储）

指定爬取数据集中的示例文件时要网络爬取的每个叶文件夹中的文件数。启用此功能后，爬网程序会随机选择每个叶文件夹中的一些文件进行网络爬取，而不是网络爬取此数据集中的所有文件。

采样爬网程序最适合于先前了解其数据格式并知道其文件夹中的架构不会更改的客户。启用此功能将显著减少爬网程序运行时间。

有效值是介于 1 到 249 之间的整数。如果未指定，则对所有文件进行爬取。

随后爬网程序运行

此字段是影响所有 Amazon S3 数据源的全局字段。

- 爬取所有子文件夹：在后续每次爬取时再次爬取所有文件夹。
- 仅爬取新子文件夹：仅爬取上次爬取后添加的 Amazon S3 文件夹。如果架构兼容，则新分区将添加到现有表中。有关更多信息，请参阅 the section called “用于添加新分区的增量爬取”。
- 基于事件爬取：依靠 Amazon S3 事件控制要爬取的文件夹。有关更多信息，请参阅 the section called “使用 Amazon S3 事件通知加速网络爬取”。

自定义分类器 — 可选

您可以先定义自定义分类器，然后再定义爬网程序。分类器检查爬网程序是否可以处理给定文件的格式。如果可以处理，分类器将以与该数据格式匹配的 `StructType` 对象的形式创建一个模式。

有关更多信息，请参阅 在 AWS Glue 中向爬网程序添加分类器。

步骤 3：配置安全设置

IAM 角色

爬网程序将担任此角色。它必须具有类似于 AWS 托管式策略 AWSGlueServiceRole 的权限。对于 Amazon S3 和 DynamoDB 源，它还必须具有访问数据存储的权限。如果爬网程序读取使用 AWS Key Management Service（AWS KMS）加密的 Amazon S3 数据，则该角色必须具有 AWS KMS 密钥的解密权限。

对于 Amazon S3 数据存储，附加到角色的其他权限类似于以下内容：

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

对于 Amazon DynamoDB 数据存储，附加到角色的其他权限类似于以下内容：

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

```
      ]
    }
  ]
}
```

要添加自己的 JDBC 驱动程序，需要添加其他权限。

- 授予以下作业操作的权限：`CreateJob`、`DeleteJob`、`GetJob`、`GetJobRun`、`StartJobRun`。
- 授予 Amazon S3 操作的权限：`s3:DeleteObjects`、`s3:GetObject`、`s3:ListBucket`、`s3:PutObject`。

> **Note**
>
> 如果禁用 Amazon S3 存储桶策略，则不需要使用 `s3:ListBucket`。

- 在 Amazon S3 策略中授予服务主体访问存储桶/文件夹的权限。

Amazon S3 策略示例：

```
{
    "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创建以下文件夹（`_crawler` 和 `_glue_job_crawler`，级别与 Amazon S3 存储桶中的 JDBC 驱动程序相同）。例如，如果驱动程序路径为 `<s3-path/driver_folder/driver.jar>`，则将创建以下文件夹（如果这些文件夹尚不存在）：

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

（可选）您可以向爬网程序添加安全配置来指定静态加密选项。

有关更多信息，请参阅步骤 2：为 AWS Glue 创建 IAM 角色 和AWS Glue 的身份和访问管理。

Lake Formation 配置 — 可选

允许爬网程序使用 Lake Formation 凭证爬取数据存储。

选中 Use Lake Formation credentials for crawling S3 data source（使用 Lake Formation 凭证爬取 S3 数据源）将允许爬网程序使用 Lake Formation 凭证爬取数据源。如果数据源属于另一个账户，则必须提供注册的账户 ID。否则，爬网程序将仅爬取与该账户关联的数据源。仅适用于 Amazon S3 和数据目录数据源。

安全配置 — 可选

设置包括安全配置。有关更多信息，请参阅下列内容：

- 加密 AWS Glue 写入的数据

> **Note**
>
> 在爬网程序上设置安全配置后，您可以进行更改，但不能将其移除。要降低爬虫的安全性级别，请在配置中将安全功能显式设置为 `DISABLED`，或者创建一个新的爬网程序。

步骤 4：设置输出和计划

输出配置

选项包括爬网程序应如何处理检测到的架构更改、数据存储中的已删除对象等等。有关更多信息，请参阅 自定义爬网程序行为

爬网程序计划

您可以按需运行爬网程序，也可以在 AWS Glue 中为爬网程序和作业定义基于时间的计划。这些计划的定义使用类似于 Unix 的 cron 语法。有关更多信息，请参阅 计划 AWS Glue 爬网程序。

第 5 步：审核并创建

查看您配置的爬网程序设置，然后创建爬网程序。

在 AWS Glue 中向爬网程序添加分类器

分类器读取数据存储中的数据。如果它可识别数据的格式，它会生成一个架构。分类器还会返回确定性数字，以指示格式识别的确定程度。

AWS Glue 提供一组内置分类器，但您也可以创建自定义分类符。AWS Glue 首先按照您在爬网程序定义中指定的顺序调用自定义分类符。根据从自定义分类符返回的结果，AWS Glue 还可能调用内置分类符。如果分类器在处理期间返回 `certainty=1.0`，则表明 100% 确定它可以创建正确的架构。然后，AWS Glue 使用该分类器的输出。

如果没有分类器返回 `certainty=1.0`，则 AWS Glue 使用具有最高确定性的分类器的输出。如果没有分类器返回大于 `0.0` 的确定性，AWS Glue 会返回 `UNKNOWN` 的默认分类字符串。

何时使用分类器？

在网络爬取数据存储以定义 AWS Glue Data Catalog 中的元数据表时，您可以使用分类器。您可以为您的爬网程序设置一组有序的分类器。当爬网程序调用分类器时，分类器会确定数据是否可被识别。如果分类器无法识别数据或不是 100% 确定，则爬网程序会调用列表中的下一个分类器来确定它能否识别数据。

有关使用 AWS Glue 控制台创建分类器的更多信息，请参阅在 AWS Glue 控制台上使用分类器。

自定义分类器

分类器的输出包含一个指示文件的分类或格式 (例如 `json`) 以及文件的架构的字符串。对于自定义分类器，您根据分类器类型定义用于创建架构的逻辑。分类器类型包括根据 grok 模式、XML 标签和 JSON 路径定义架构。

如果您更改一个分类器定义，则之前使用该分类器爬网的任何数据均不会重新分类。爬网程序将跟踪之前爬网的数据。新数据将用更新的分类器来分类，这可能会产生更新的架构。如果数据架构发生了变化，请更新分类器来考虑爬网程序运行时的任何架构更改。要对数据重新分类以更正错误的分类器，请用更新的分类器创建新爬网程序。

有关在 AWS Glue 中创建自定义分类器的更多信息，请参阅编写自定义分类器。

> **Note**
>
> 如果您的数据格式可被由某个内置分类器识别，则不需要创建自定义分类器。

AWS Glue 中的内置分类器

AWS Glue 为各种格式（包括 JSON、CSV、Web 日志和许多数据库系统）提供内置分类器。

如果 AWS Glue 找不到符合 100% 确定性的输入数据格式的自定义分类器，它会按照下表中所示的顺序调用内置分类器。内置分类器返回结果以指示格式是否匹配 (`certainty=1.0`) 或不匹配 (`certainty=0.0`)。第一个具有 `certainty=1.0` 的分类器为您的数据目录中的元数据表提供分类字符串和架构。

分类器类型	分类字符串	注意事项
Apache Avro	`avro`	读取文件开头处的架构以确定格式。
Apache ORC	`orc`	读取文件元数据以确定格式。
Apache Parquet	`parquet`	读取文件结尾处的架构以确定格式。
JSON	`json`	读取文件的开头以确定格式。
二进制 JSON	`bson`	读取文件的开头以确定格式。
XML	`xml`	读取文件的开头以确定格式。AWS Glue 根据文档中的 XML 标记确定表架构。 有关创建自定义 XML 分类器以指定文档中的行的信息，请参阅编写 XML 自定义分类器。
Amazon Ion	`ion`	读取文件的开头以确定格式。
组合 Apache 日志	`combined_apache`	通过 grok 模式确定日志格式。
Apache 日志	`apache`	通过 grok 模式确定日志格式。
Linux 内核日志	`linux_kernel`	通过 grok 模式确定日志格式。
Microsoft 日志	`microsoft_log`	通过 grok 模式确定日志格式。
Ruby 日志	`ruby_logger`	读取文件的开头以确定格式。
Squid 3.x 日志	`squid`	读取文件的开头以确定格式。

分类器类型	分类字符串	注意事项
Redis 监控日志	`redismonlog`	读取文件的开头以确定格式。
Redis 日志	`redislog`	读取文件的开头以确定格式。
CSV	`csv`	检查以下分隔符：逗号 (,)、竖线 (|)、制表符 (\t)、分号 (;) 和 Ctrl-A (\u0001)。Ctrl-A 是 `Start Of Heading` 的 Unicode 控制字符。
Amazon Redshift	`redshift`	使用 JDBC 连接导入元数据。
MySQL	`mysql`	使用 JDBC 连接导入元数据。
PostgreSQL	`postgresql`	使用 JDBC 连接导入元数据。
Oracle 数据库	`oracle`	使用 JDBC 连接导入元数据。
Microsoft SQL Server	`sqlserver`	使用 JDBC 连接导入元数据。
Amazon DynamoDB	`dynamodb`	从 DynamoDB 表中读取数据。

以下压缩格式的文件可以分类：

- ZIP（在只包含单个文件的存档操作中支持此格式）。请注意，Zip 格式在其他服务中不太受支持（由于存档）。
- BZIP
- GZIP
- LZ4
- Snappy（支持标准和 Hadoop 本机 Snappy 格式）

内置的 CSV 分类器

内置的 CSV 分类器可分析 CSV 文件内容，以确定 AWS Glue 表的架构。此分类器会检查以下分隔符：

- 逗号 (,)
- 竖线 (|)
- 制表符 (\t)
- 分号 (;)
- Ctrl-A (\u0001)

  Ctrl-A 是 `Start Of Heading` 的 Unicode 控制字符。

要被分类为 CSV，表架构必须至少有两列和两行数据。CSV 分类器使用许多探试程序来确定给定中是否存在某一标头。如果分类器无法根据第一行数据确定标头，列标题将显示为 `col1`、`col2`、`col3`，以此类推。内置的 CSV 分类器确定是否通过评估文件的以下特性来推断标头：

- 潜在标头中的每列均分析为 STRING 数据类型。
- 除了最后一列外，潜在标头中的每列的内容均少于 150 个字符。要允许尾部的分隔符，在整个文件中最后一列均可为空。
- 潜在标头中的每列都必须满足列名称的 AWS Glue `regex` 要求。
- 标题行必须与数据行存在足够差别。为确定这一点，必须将一行或多行分析为 STRING 之外的类型。如果所有列均为 STRING 类型，则第一行数据与后续行差别不够大，无法用作标头。

> **Note**
>
> 如果内置的 CSV 分类器未创建您需要的 AWS Glue 表，您可能能够使用以下一种方法：
>
> - 更改数据目录中的列名称，将 `SchemaChangePolicy` 设置为 LOG，为未来的爬网程序运行将分区输出配置设置为 `InheritFromTable`。
> - 创建自定义 grok 分类器，以分析数据并按您所需来分配列。
> - 内置的 CSV 分类器会创建表，同时引用 `LazySimpleSerDe` 作为序列化库，这是进行类型推断的极好选择。但是，如果 CSV 数据包含带引号的字符串，请编辑表定义并将 SerDe 库更改为 `OpenCSVSerDe`。将任何推断类型调整为 STRING，将 `SchemaChangePolicy` 设置为 LOG，为未来的爬网程序运行将分区输出配置设置为 `InheritFromTable`。有关 SerDe 库的更多信息，请参阅 Amazon Athena 用户指南中的 SerDe 参考。

编写自定义分类器

您可以提供自定义分类器来对 AWS Glue 中的数据进行分类。您可以使用 grok 模式、XML 标签、JavaScript 对象表示法 (JSON) 或逗号分隔值 (CSV) 来创建自定义分类器。AWS Glue 爬网程序调用自定义分类器。如果分类器可识别数据，则它会将数据的分类和架构返回到爬网程序。如果数据与任何内置分类器不匹配，或者您希望自定义由爬网程序创建的表，则可能需要定义自定义分类器。

有关使用 AWS Glue 控制台创建分类器的更多信息，请参阅在 AWS Glue 控制台上使用分类器。

AWS Glue 按照您指定的顺序在内置分类器之前运行自定义分类器。当爬网程序找到与数据匹配的分类器时，分类字符串和架构将在写入 AWS Glue Data Catalog的表的定义中使用。

主题

- 编写 grok 自定义分类器
- 编写 XML 自定义分类器
- 编写 JSON 自定义分类器
- 编写 CSV 自定义分类器

编写 grok 自定义分类器

Grok 是一个工具，用于分析给定匹配模式的文本数据。grok 模式是一组命名的正则表达式 (regex)，用于一次匹配一行数据。AWS Glue 使用 grok 模式来推断数据的架构。当 grok 模式与您的数据匹配时，AWS Glue 使用该模式来确定数据的结构并将其映射到字段。

AWS Glue 提供许多内置模式，或者您也可以定义自己的模式。您可以使用自定义分类器定义中的内置模式和自定义模式来创建 grok 模式。您可以定制 grok 模式来对自定义文本文件格式进行分类。

> **Note**
>
> AWS Glue grok 自定义分类器对在 AWS Glue Data Catalog 中创建的表使用 `GrokSerDe` 序列化库。如果您要将 AWS Glue Data Catalog 与 Amazon Athena、Amazon EMR 或 Redshift Spectrum 结合使用，请参阅有关这些服务的文档以了解有关 `GrokSerDe` 支持的信息。目前，在从 Amazon EMR 和 Redshift Spectrum 查询使用 `GrokSerDe` 创建的表时，您可能会遇到问题。

以下是 grok 模式组件的基本语法：

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

与命名 PATTERN 匹配的数据映射到架构中的 `field-name` 列，默认数据类型为 `string`。或者，该字段的数据类型可以在生成的架构中强制转换为 `byte`、`boolean`、`double`、`short`、`int`、`long` 或 `float`。

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

例如，要将 num 字段强制转换为 `int` 数据类型，您可以使用此模式：

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

模式可以是由其他模式组成。例如，您可以为 SYSLOG 时间戳指定一个模式，该模式由月份、月份中的日期和时间的模式定义（例如，`Feb 1 06:25:43`）。对于此数据，您可能定义以下模式：

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

> **Note**
>
> Grok 模式一次只能处理一行。不支持多行模式。此外，不支持模式中的换行符。

AWS Glue 中的自定义分类器值

定义 grok 分类器时，可以向 AWS Glue 提供以下值以创建自定义分类器。

名称

分类器的名称。

分类

写入的文本字符串，用于描述分类数据的格式；例如，`special-logs`。

Grok 模式

应用于数据存储以确定是否存在匹配的模式集。这些模式来自 AWS Glue 内置模式和您定义的任何自定义模式。

以下是 grok 模式的示例：

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

当数据与 TIMESTAMP_ISO8601 匹配时，将会创建架构列 timestamp。此行为与示例中的其他命名模式类似。

自定义模式

您定义的可选自定义模式。这些模式由对您的数据进行分类的 grok 模式引用。您可以在应用于您的数据的 grok 模式中引用这些自定义模式。每个自定义组件模式必须位于不同的行上。正则表达式 (regex) 语法用于定义模式。

下面是如何使用自定义模式的示例：

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

第一个自定义命名模式 CRAWLERLOGLEVEL 在数据与其中一个枚举字符串匹配时是匹配项。第二个自定义模式 MESSAGEPREFIX 尝试匹配消息前缀字符串。

AWS Glue 跟踪分类器的创建时间、上次更新时间和版本。

AWS Glue 内置模式

AWS Glue 可提供许多常见模式，您可以用来构建自定义分类器。您将在分类器定义中向 grok pattern 添加命名模式。

以下列表为每个模式包含一行。在每个行中，模式名称遵循其定义。正则表达式 (regex) 语法用于定义模式。

```
#<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)?)
```

编写 XML 自定义分类器

XML 使用文件中的标签定义文档的结构。使用 XML 自定义分类器，您可以指定用于定义行的标签名称。

AWS Glue 中的自定义分类器值

定义 XML 分类器时，可以向 AWS Glue 提供以下值以创建分类器。此分类器的分类字段设置为 `xml`。

名称

分类器的名称。

行标签

在 XML 文档中定义表行的 XML 标记名称，不带尖括号 < >.. 名称必须符合标签的 XML 规则。

> **Note**
>
> 包含行数据的元素不能是自结束的空元素。例如， 不AWS Glue解析此空元素：
>
> ```
> <row att1=”xx” att2=”yy” />
> ```
>
> 可以如下所示编写空元素：
>
> ```
> <row att1=”xx” att2=”yy”> </row>
> ```

AWS Glue 跟踪分类器的创建时间、上次更新时间和版本。

例如，假设您具有以下 XML 文件。要创建仅包含作者和标题列的 AWS Glue 表，请在 AWS Glue 控制台中创建 Row tag (行标签) 为 AnyCompany 的分类器。然后，添加并运行使用此自定义分类器的爬网程序。

```
<?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>
```

编写 JSON 自定义分类器

JSON 一种数据交换格式。它使用名称-值对或值的有序列表来定义数据结构。使用 JSON 自定义分类器，您可以指定用于为表定义架构的数据结构的 JSON 路径。

AWS Glue 中的自定义分类器值

定义 JSON 分类器时，可以向 AWS Glue 提供以下值以创建分类器。此分类器的分类字段设置为 `json`。

名称

分类器的名称。

JSON 路径

一个 JSON 路径，指向用于定义表架构的对象。JSON 路径可以用点表示法或括号表示法编写。支持以下运算符：

描述
JSON 对象的根元素。这将启动所有路径表达式
通配符。在 JSON 路径中需要名称或数字的任何地方都可用。
点表示的子字段。指定 JSON 对象中的子字段。

描述
括号表示的子字段。指定 JSON 对象中的子字段。只能指定单个子字段。
数组索引。按索引指定数组的值。

AWS Glue 跟踪分类器的创建时间、上次更新时间和版本。

Example 使用 JSON 分类器从数组中提取记录

假设您的 JSON 数据是一组记录。例如，您的文件的前几行可能如下所示：

```
[
  {
    "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"
  }
]
```

使用内置 JSON 分类器运行爬网程序时，整个文件用于定义架构。由于您没有指定 JSON 路径，爬网程序将数据视为一个对象，即只是一个数组。例如，架构可能如下所示：

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

但是，要创建一个基于 JSON 数组中的每个记录的架构，请创建一个自定义 JSON 分类器，并将 JSON 路径指定为 `$[*]`。当您指定此 JSON 路径时，分类器会询问数组中的所有 12 个记录以确定架构。生成的架构包含每个对象的单独字段，类似于以下示例：

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

Example 使用 JSON 分类器仅检查文件的一部分

假设 JSON 数据遵循来自 http://everypolitician.org/ 的示例 JSON 文件 `s3://awsglue-datasets/examples/us-legislators/all/areas.json` 的模式。JSON 文件中的示例对象如下所示：

```
{
  "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"
}
```

使用内置 JSON 分类器运行爬网程序时，整个文件用于创建架构。您最后可能会得到一个如下所示的架构：

```
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
```

但是，要仅使用“id”对象创建架构，请创建自定义 JSON 分类器，并将 JSON 路径指定为 `$.id`。然后，该架构仅基于“id”字段：

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

用此架构提取的前几行数据如下所示：

```
{"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"}
```

要在 JSON 文件中基于深层嵌套对象（如“identifier”）创建架构，可以创建自定义 JSON 分类器，并将 JSON 路径指定为 `$.identifiers[*].identifier`。尽管该架构与上一个示例类似，但它基于 JSON 文件中的另一个对象。

此架构看上去与下类似：

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

如果列出表中的前几行数据，则会指示架构基于“identifier”对象中的数据：

```
{"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"}
```

要在 JSON 文件中基于另一个深层嵌套对象（如“name”数组中的“other_names”字段）创建表，可以创建自定义 JSON 分类器，并将 JSON 路径指定为 $.other_names[*].name。尽管该架构与上一个示例类似，但它基于 JSON 文件中的另一个对象。此架构看上去与下类似：

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

如果列出表中的前几行数据，则会指示它基于“name”数组中的“other_names”对象中的数据：

```
{"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": "#######"}
```

编写 CSV 自定义分类器

自定义 CSV 分类器允许您在自定义 CSV 分类器字段中为每列指定数据类型。您可以指定每列的数据类型，用逗号分隔。通过指定数据类型，您可以覆盖爬网程序推断出的数据类型，并确保对数据进行适当分类。

您可以设置用于在分类器中处理 CSV 的 SerDe，该分类器将应用于 Data Catalog。

创建自定义分类器时，您也可以将该分类器重复用于不同的爬网程序。

- 对于只有标题（无数据）的 CSV 文件，由于提供的信息不足，这些文件将被归类为 UNKNOWN。如果您在列标题选项中指定 CSV“有标题”，并提供数据类型，我们可以正确地对这些文件进行分类。

您可以使用自定义 CSV 分类器来推断各种类型的 CSV 数据的架构。您可以为分类器提供的自定义属性包括分隔符、CSV SerDe 选项、有关标题的选项以及是否对数据执行某些验证。

AWS Glue 中的自定义分类器值

定义 CSV 分类器时，可以向 AWS Glue 提供以下值以创建分类器。此分类器的分类字段设置为 `csv`。

分类器名称

分类器的名称。

CSV Serde

设置用于在分类器中处理 CSV 的 SerDe，该分类器将应用于 Data Catalog。选项包括“Open CSV SerDe”、“Lazy Simple SerDe”和“无”。当您想让爬网程序执行检测时，可以指定“无”值。

列分隔符

一个自定义符号，表示分隔行中每个列条目的内容。提供一个 Unicode 字符。如果您无法键入分隔符，则可以将其复制并粘贴。这适用于可打印字符，包括您的系统不支持的字符（通常显示为 □）。

引用符号

一个自定义符号，表示将内容组合为单个列值的内容。必须与列分隔符不同。提供一个 Unicode 字符。如果您无法键入分隔符，则可以将其复制并粘贴。这适用于可打印字符，包括您的系统不支持的字符（通常显示为 □）。

列标题

指示有关应如何在 CSV 文件中检测列标题的行为。如果您的自定义 CSV 文件包含列标题，请输入列标题的逗号分隔列表。

处理选项：允许具有单列的文件

允许处理仅包含一列的文件。

处理选项：在标识列值之前去除空格

指定是否在标识列值类型之前去除值。

自定义数据类型 - 可选

输入用逗号分隔的自定义数据类型。在 CSV 文件中指定自定义数据类型。自定义数据类型必须为受支持的数据类型。受支持的数据类型有：“BINARY”、“BOOLEAN”、“DATE”、“DECIMAL”、“DOUBLE”、“FLOAT”、“INT”、“LONG”、“SHORT”。不受支持的数据类型将显示错误。

在 AWS Glue 控制台上使用分类器

分类器确定您的数据架构。您可以编写一个自定义分类器并从 AWS Glue 指向该分类器。

查看分类器

要查看您创建的所有分类器的列表，请点击 https://console.aws.amazon.com/glue/ 打开 AWS Glue 控制台，然后选择 Classifiers (分类器) 选项卡。

列表显示了有关每个分类器的以下属性：

- 分类器 – 分类器名称。创建分类器时，您必须为其提供名称。
- 分类 – 由此分类器推断的表的分类类型。
- 上次更新 – 上次更新此分类器的时间。

管理分类器

从 控制台中的 ClassifiersAWS Glue (分类器) 列表中，您可以添加、编辑和删除分类器。要查看分类器的详细信息，请在列表中选择分类器名称。详细信息包括您创建分类器时定义的信息。

创建分类器

要在 AWS Glue 控制台中添加分类器，请选择 Add classifier (添加分类器)。在定义分类器时，您需要提供以下项目的值：

- 分类器名称 – 为您的分类器提供唯一名称。
- 分类器类型 – 由此分类器推断的表的分类类型。
- 上次更新 – 上次更新此分类器的时间。

分类器名称

为您的分类器提供唯一名称。

分类器类型

选择要创建的分类器的类型。

根据您选择的分类器类型，为分类器配置以下属性：

Grok

- 分类。

  描述分类的数据的格式或类型或提供自定义标签。

- Grok 模式

  此模式用于将您的数据解析为结构化架构。Grok 模式由描述您数据存储格式的命名模式组成。您使用 AWS Glue 提供的已命名内置模式和您编写且包含在 Custom patterns (自定义模式) 字段中

的自定义模式编写此 grok 模式。尽管 grok 调试程序结果可能不完全匹配 AWS Glue 中的结果，我们仍建议您使用一些示例数据与 grok 调试程序来尝试您的模式。您可以在 Web 上查找 grok 调试程序。AWS Glue 提供的已命名的内置模式通常可与 Web 上提供的 grok 模式兼容。

通过以迭代方式添加已命名的模式来构建您的 grok 模式，并在调试程序中检查您的结果。此活动将使您确信，当 AWS Glue 爬网程序运行 grok 模式时会解析您的数据。

- 自定义模式

  对于 grok 分类器，这些是您编写的 Grok pattern (Grok 模式) 的可选构建块。当内置模式无法解析您的数据时，您可能需要编写自定义模式。这些自定义模式在此字段中定义并在 Grok pattern (Grok 模式) 字段中引用。每个自定义模式在单独的行上定义。和内置模式一样，它包含一个已命名的模式定义，该定义使用正则表达式 (regex) 语法。

  例如，以下是具有名称 `MESSAGEPREFIX` 后跟要应用于您的数据以确定其是否符合该模式的正则表达式定义。

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

XML

- 行标签

  对于 XML 分类器，这是定义 XML 文档中的表行的 XML 标签的名称。键入名称，不带尖括号 < >。名称必须符合标签的 XML 规则。

  有关更多信息，请参阅 编写 XML 自定义分类器。

JSON

- JSON 路径

  对于 JSON 分类器，这是定义正在创建的表的行的对象、数组或值的 JSON 路径。使用 AWS Glue 支持的运算符在点或括号 JSON 语法中键入名称。

  有关更多信息，请参阅编写 JSON 自定义分类器中的运算符列表。

CSV

- 列分隔符

  单个字符或符号，表示分隔行中每个列条目的内容。从列表中选择分隔符，或选择 `Other` 来输入自定义分隔符。

- 引用符号

  单个字符或符号，表示将内容组合为单个列值的内容。必须与列分隔符不同。从列表中选择引用符号，或选择 `Other` 来输入自定义引用字符。

- 列标题

  指示有关应如何在 CSV 文件中检测列标题的行为。您可以选择 `Has headings`、`No headings` 或 `Detect headings`。如果您的自定义 CSV 文件包含列标题，请输入列标题的逗号分隔列表。

- 允许具有单列的文件

  要被分类为 CSV，数据必须至少有两列和两行数据。使用此选项处理仅包含一列的文件。

- 在标识列值之前去除空格

  此选项指定是否在标识列值类型之前去除值。

- 自定义数据类型

  （可选）- 在逗号分隔列表中输入自定义数据类型。受支持的数据类型有：“BINARY”、“BOOLEAN”、“DATE”、“DECIMAL”、“DOUBLE”、“FLOAT”、“INT”、“LONG”、“SHOR

- CSV Serde

  （可选）- 设置用于在分类器中处理 CSV 的 SerDe，该分类器将应用于 Data Catalog。可以选择 `Open CSV SerDe`、`Lazy Simple SerDe` 或 `None`。当您想让爬网程序执行检测时，可以指定 `None` 值。

有关更多信息，请参阅 编写自定义分类器。

计划 AWS Glue 爬网程序

您可以根据需要或定期计划运行 AWS Glue 爬网程序。爬网程序计划可以用 cron 格式表示。有关更多信息，请参阅 Wikipedia 中的 cron。

当您根据计划创建爬网程序时，您可以指定包含约束，如爬网程序运行的频率、在一周中的那些天运行以及具体时间。这些约束基于 cron 。当您为设置爬网程序计划时，您应该考虑 cron 的功能和限制。例如，如果您选择在每月第 31 天运行您的爬网程序，请记住，有些月份没有 31 天。

每个爬网程序的爬取有效期最长为 12 个月

有关使用 cron 安排作业和爬网程序的更多信息，请参阅用于作业和爬网程序的基于时间的计划。

查看爬网程序结果和详细信息

爬网程序成功运行后，它会在数据目录中创建表定义。在导航窗格中选择 Tables (表) 来查看爬网程序在您指定的数据库中创建的表。

您可以按如下方式查看与爬网程序本身相关的信息：

- AWS Glue 控制台上的 Crawlers (爬网程序) 页面显示爬网程序的以下属性：

属性	描述
名称	当您创建爬网程序时，您必须为其指定一个唯一名称。
状态	爬网程序状态可以为：准备就绪、正在启动、正在停止、已安排或计划已暂停。正在运行的爬网程序从正在启动前进到正在停止。您可以恢复或暂停附加到爬网程序的计划。
计划	您可以选择按需运行爬网程序或选择具有计划的频率。有关安排爬网程序的更多信息，请参阅计划爬网程序。
上次运行	爬网程序上次运行的日期和时间。
日志	来自上次运行的爬网程序的任何可用日志的链接。
上次运行后的表格变更	AWS Glue Data Catalog中由最近一次运行的爬网程序更新的表的数量。

- 要查看爬网程序的历史记录，请在导航窗格中选择 Crawlers（爬网程序）以查看您创建的爬网程序。从可用爬网程序列表中选择一个爬网程序。您可以在 Crawler runs（爬网程序运行）选项卡中查看爬网程序属性和爬网程序历史记录。

  “Crawler runs”（爬网程序运行）显示每次爬网程序运行时的相关信息，包括 Start time (UTC) [开始时间（UTC）]、End time (UTC) [结束时间（UTC）]、Duration（持续时间）、Status（状态）、DPU hours（DPU 小时）和 Table changes（表格变更）。

  “爬网程序运行”选项卡将仅显示自爬网程序历史记录功能启动之日以来发生的爬取操作，并且最长仅保留 12 个月的爬取操作。较早的爬取结果将不会被返回。
- 要查看其他信息，请在爬网程序详细信息页面中选择一个选项卡。每个选项卡都将显示与爬网程序相关的信息。
  - Schedule（计划）：为爬网程序创建的所有计划都将在此处显示。
  - Data sources（数据来源）：爬网程序扫描的所有数据来源都将在此处显示。
  - Classifiers（分类器）：分配给爬网程序的所有分类器都将在此处显示。
  - Tags（标记）：创建并分配给 AWS 资源的所有标记都将在此处显示。

爬网程序在数据目录表上设置的参数

这些表属性由 AWS Glue 爬网程序设置。我们希望用户使用 `classification` 和 `compressionType` 属性。其他属性（包括估算表大小）用于内部计算，我们无法保证其准确性或客户使用案例适用性。更改这些参数可能会改变爬网程序的行为，我们不支持此工作流程。

属性键	属性值
`UPDATED_BY_CRAWLER`	执行更新的爬网程序的名称。
`connectionName`	用于连接到数据存储的爬网程序在“数据目录”中的连接名称。
`recordCount`	根据文件大小和标题估算表中记录的数量。
`skip.header.line.count`	跳过行以跳过标题。在被归类为 CSV 的表上设置。

属性键	属性值
CrawlerSchemaSerializerVersion	供内部使用
classification	由爬网程序推断数据格式。更多有关 AWS Glue 爬网程序支持的数据格式的信息，请参阅 the section called “AWS Glue 中的内置分类器”。
CrawlerSchemaDeserializerVersion	供内部使用
sizeKey	已爬取的表中文件的组合大小。
averageRecordSize	表中行的平均大小（字节）。
compressionType	对表中的数据使用的压缩类型。更多有关 AWS Glue 爬网程序支持的压缩类型的信息，请参阅 the section called “AWS Glue 中的内置分类器”。
typeOfData	file、table 或 view。
objectCount	Amazon S3 表路径下的对象数量。

这些额外的表属性是由 AWS Glue 爬网程序为 Snowflake 数据存储设置的。

属性键	属性值
aws:RawTableLastAltered	记录 Snowflake 表上次修改的时间戳。
ViewOriginalText	查看 SQL 语句。
ViewExpandedText	查看以 Base64 格式编码的 SQL 语句。

属性键	属性值
`ExternalTable:S3Location`	Snowflake 外部表的 Amazon S3 位置。
`ExternalTable:FileFormat`	Snowflake 外部表的 Amazon S3 文件格式。

这些额外的表属性是由 AWS Glue 爬网程序为 Amazon Redshift、Microsoft SQL Server、MySQL、PostgreSQL 和 Oracle 等 JDBC 类型的数据存储设置的。

属性键	属性值
`aws:RawType`	当爬网程序将数据存储在数据目录中时，它会将数据类型转换为与 Hive 兼容的类型，这往往会导致有关本机数据类型的信息丢失。爬网程序输出 `aws:RawType` 参数，以提供原生级别的数据类型。
`aws:RawColumnComment`	如果注释与数据库中的列相关联，则爬网程序会在目录表中输出相应的注释。注释字符串被截断为 255 个字节。 Microsoft SQL Server 不支持注释。
`aws:RawTableComment`	如果注释与数据库中的列相关联，则爬网程序会在目录表中输出相应的注释。注释字符串被截断为 255 个字节。 Microsoft SQL Server 不支持注释。

自定义爬网程序行为

当爬网程序运行时，它可能会遇到导致架构或分区与以前的爬网不同的数据存储更改。您可以使用 AWS Management Console或 AWS Glue API 来配置爬网程序如何处理某些类型的更改。

主题

- 用于添加新分区的增量爬取
- 设置分区索引爬网程序配置选项

- 使用 Amazon S3 事件通知加速网络爬取
- 如何阻止爬网程序更改现有架构
- 如何为每个 Amazon S3 包含路径创建单个架构
- 如何指定表位置和分区级别
- 如何指定允许爬网程序创建的最大表数
- 如何为 Delta Lake 数据存储指定配置选项
- 如何将爬网程序配置为使用 Lake Formation 凭证

Console

当您使用 AWS Glue 控制台定义爬网程序时，有多个选项可用于配置爬网程序的行为。有关使用 AWS Glue 控制台添加爬网程序的更多信息，请参阅配置爬网程序。

当爬网程序针对以前已爬网的数据存储运行时，可能会发现架构已更改或数据存储中的一些对象已被删除。爬网程序将记录架构更改。根据爬网程序的源类型，可能创建新的表和分区，而不考虑架构更改策略。

要指定爬网程序在发现架构中的更改时执行什么操作，您可以在控制台上选择下列操作之一：

- Update the table definition in the Data Catalog (更新数据目录中的表定义) – 在 AWS Glue Data Catalog 中添加新列、删除缺少的列并修改现有列的定义。删除爬网程序未设置的任何元数据。这是默认设置。
- Add new columns only (仅添加新列) – 对于映射到 Amazon S3 数据存储的表，在发现新列时添加这些新列，但不删除或更改数据目录中的现有列的类型。当数据目录中的当前列正确且您不希望爬网程序删除或更改现有列的类型时，选择此选项。如果基本 Amazon S3 表属性（如分类、压缩类型或 CSV 分隔符）更改，将此表标记为已弃用。保留数据目录中存在的输入格式和输出格式。仅当 SerDe 参数是由爬网程序设置时，才更新该参数。对于所有其他数据存储，修改现有列定义。
- Ignore the change and don't update the table in the Data Catalog (忽略更改，不更新数据目录中的表) – 只创建新表和分区。

  这是增量爬网的默认设置。

爬网程序可能还发现新的或更改的分区。默认情况下，会添加新分区，如果现有分区已更改，则会更新它们。此外，您还可以在 AWS Glue 控制台上将爬网程序配置选项设置为 Update all new and existing partitions with metadata from the table (使用表中的元数据更新所有新的和现有的分区)。

设置此选项后，分区会继承其父表中的元数据属性，如分类、输入格式、输出格式、SerDe 信息和架构。对表中的这些属性进行的任何更改都将传播到其分区。当在现有爬网程序上设置此配置选项时，将会更新现有分区，以便在下次爬网程序运行时与父表的属性相匹配。

要指定爬网程序在数据存储中找到已删除对象时执行什么操作，请选择下列操作之一：

- 删除数据目录中的表和分区
- Ignore the change and don't update the table in the Data Catalog（忽略更改，不更新数据目录中的表）

  这是增量爬网的默认设置。
- Mark the table as deprecated in the Data Catalog (在数据目录中将表标记为已弃用) – 这是默认设置。

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

当使用 AWS Glue API 定义爬网程序时，您可以从多个字段中选择来配置爬网程序。爬网程序 API 中的 SchemaChangePolicy 确定爬网程序在发现已更改的架构或已删除的对象时执行什么操作。爬网程序在运行时记录架构更改。

显示爬网程序配置选项的示例 python 代码

```
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
```

当爬网程序运行时，无论架构更改策略如何，都始终会创建新的表和分区。您可以在 SchemaChangePolicy 结构中的 UpdateBehavior 字段中选择以下操作之一来确定爬网程序在发现更改的表架构时执行什么操作：

- UPDATE_IN_DATABASE – 更新 AWS Glue Data Catalog 中的表。添加新列、删除缺少的列并修改现有列的定义。删除爬网程序未设置的任何元数据。

- LOG – 忽略更改，不更新数据目录中的表

  这是增量爬网的默认设置。

您也可以使用爬网程序 API Configuration 字段中提供的 JSON 对象覆盖 SchemaChangePolicy 结构。此 JSON 对象可以包含键–值对以将策略设置为不更新现有列并仅添加新列。例如，以字符串形式提供以下 JSON 对象：

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

此选项对应于 AWS Glue 控制台上的 Add new columns only (仅添加新列) 选项。它仅覆盖通过对 Amazon S3 数据存储进行网络爬取生成的表的 SchemaChangePolicy 结构。如果您希望保留数据目录（可信来源）中存在的元数据，请选择此选项。在遇到新列时添加这些新列，包括嵌套数据类型。但是，不会删除现有列，且不会更改其类型。如果 Amazon S3 表属性显著更改，将此表标记为已弃用，并记录一条警告，指示需要解决不兼容的属性。此选项不适用于增量爬网程序。

当爬网程序针对以前爬取的数据存储运行时，它可能会发现新的或已更改的分区。默认情况下，会添加新分区，如果现有分区已更改，则会更新它们。此外，您还可以将爬网程序配置选项设置为 InheritFromTable（对应于 AWS Glue 控制台上的 Update all new and existing partitions with metadata from the table (使用表中的元数据更新所有新的和现有的分区) 选项）。设置此选项后，分区会继承其父表中的元数据属性，如分类、输入格式、输出格式、SerDe 信息和架构。对父表进行的任何属性更改都将传播到其分区。

当在现有爬网程序上设置此配置选项时，将会更新现有分区，以便在下次爬网程序运行时与父表的属性相匹配。在爬网程序 API Configuration 字段中设置此行为。例如，以字符串形式提供以下 JSON 对象：

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

爬网程序 API Configuration 字段可以设置多个配置选项。例如，要配置分区和表的爬网程序输出，可以提供以下 JSON 对象的字符串表示形式：

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

您可以选择下列操作之一来确定爬网程序在数据存储中找到已删除对象时执行什么操作。爬网程序 API 中的 SchemaChangePolicy 结构中的 DeleteBehavior 字段设置爬网程序在发现已删除对象时的行为。

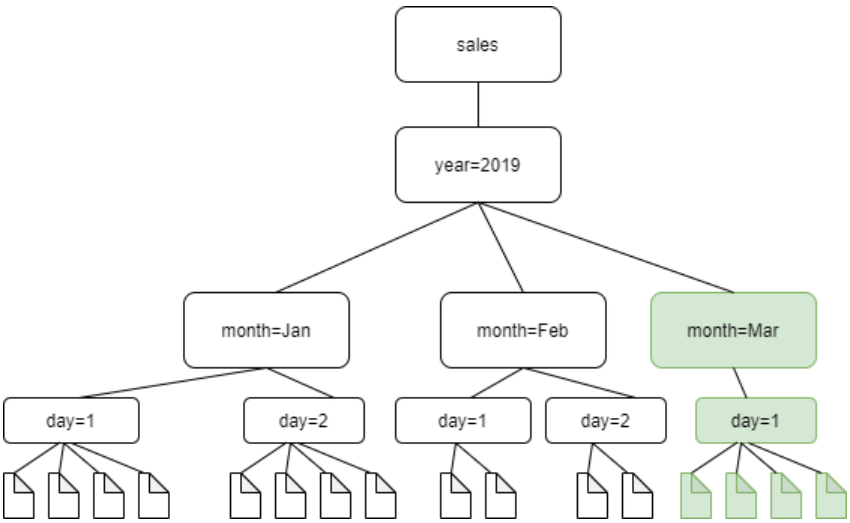
- DELETE_FROM_DATABASE – 删除数据目录中的表和分区。
- LOG – 忽略更改。请勿更新数据目录。而是写入日志消息。
- DEPRECATE_IN_DATABASE – 在数据目录中将表标记为已弃用。这是默认设置。

用于添加新分区的增量爬取

爬网程序提供了添加新分区的选项，从而使具有稳定表架构的增量数据集可以更快地爬取。典型用例是针对计划的爬网程序，在每次爬取期间都会添加新的分区。启用此选项后，它将先对目标数据集运行一次完整的爬取，以使爬网程序能够记录初始架构和分区结构。在重新爬取期间，只有当架构兼容时，新分区才会添加到现有表中。第一次爬取运行后，不会对架构进行任何更改，也不会向数据目录添加任何新表。

设置 Amazon S3 数据来源时，可以使用此选项。您可以在 CreateCrawler API 中将带有 RecrawlBehavior 的 RecrawlPolicy 设置为“Crawl_New_Folders”，或在控制台中将后续爬网程序运行设置为仅爬取新的子文件夹。

继续 the section called “爬网程序如何确定何时创建分区？” 中的示例，下图显示已添加三月份的文件。



如果您将 `RecrawlBehavior` 设置为“Crawl_New_Folders”选项，则爬取 `month=Mar`。

注释和限制

启用此选项后，您无法在编辑爬网程序时更改 Amazon S3 目标数据存储。此选项会影响某些爬网程序配置设置。启用后，它会将爬网程序的更新行为和删除行为强制为 `LOG`。这意味着：

- 如果发现架构不兼容的对象，爬网程序将不会在 Data Catalog 中添加这些对象，而是将此详细信息作为日志添加到 CloudWatch Logs 中。
- 它不会更新 Data Catalog 中的已删除对象。

有关更多信息，请参阅 the section called “自定义爬网程序行为”。

设置分区索引爬网程序配置选项

Data Catalog 支持分区索引，以提供对特定分区的有效查找。有关更多信息，请参阅 Working with partition indexes in AWS Glue。默认情况下，AWS Glue 爬网程序会为 Amazon S3 和 Delta Lake 目标创建分区索引。

定义爬网程序时，在设置输出和调度页面中的高级选项下，自动创建分区索引选项会默认处于启用状态。

要禁用此选项，可以在控制台中取消选择自动创建分区索引复选框。您也可以使用爬网程序 API 禁用此选项，具体需要在 `Configuration` 中设置 `CreatePartitionIndex` 。默认值为 true。

分区索引的使用说明

- 默认情况下，由爬网程序创建的表没有变量 `partition_filtering.enabled`。有关更多信息，请参阅 AWS Glue partition indexing and filtering。
- 不支持为加密分区创建分区索引。

使用 Amazon S3 事件通知加速网络爬取

您可以将爬网程序配置为使用 Amazon S3 事件来查找任何更改，而不是列出 Amazon S3 或 Data Catalog 目标中的对象。此功能使用 Amazon S3 事件，通过列出触发事件的子文件夹中的所有文件，而不是列出完整的 Amazon S3 或 Data Catalog 目标，来识别两次网络爬取之间的更改，从而缩短了重新爬网时间。

第一次网络爬取会列出目标中的所有 Amazon S3 对象。第一次成功爬取之后，您可以选择手动重新爬取或按设定计划重新爬取。爬网程序只会列出这些事件中的对象，而不会列出所有对象。

迁移到基于 Amazon S3 事件的爬网程序的优点包括：

- 不需要列出目标中的所有对象，而是在添加或删除对象的位置列出特定文件夹，从而重新爬网更快。
- 在添加或删除对象的位置列出特定文件夹，从而降低总体网络爬取成本。

Amazon S3 事件网络爬取基于爬网程序调度，从 SQS 队列中使用 Amazon S3 事件来运行。如果队列中没有事件，则无需支付费用。Amazon S3 事件可以配置为直接进入 SQS 队列，或者在多个用户需要相同事件的情况下，也可以配置为 SNS 和 SQS 的组合。有关更多信息，请参阅 the section called “为 Amazon S3 事件通知设置账户”。

在事件模式下创建和配置爬网程序之后，第一次网络爬取将在列表模式下通过执行 Amazon S3 或 Data Catalog 目标的完整列表来运行。第一次成功网络爬取后将使用 Amazon S3 事件，以下日志可确认网络爬取的运行：“网络爬取正在使用 Amazon S3 事件来运行。”

创建 Amazon S3 事件网络爬取并更新可能影响网络爬取的爬网程序属性后，网络爬取将在列表模式下运行，并添加以下日志：“网络爬取未在 S3 事件模式下运行”。

> **Note**
>
> 每次爬取使用的最大消息数为 10000。

目录目标

当目标为 Data Catalog 时，爬网程序会利用更改（例如，表中的额外分区）更新 Data Catalog 中的现有表。

主题

- 为 Amazon S3 事件通知设置账户
- 对 Amazon S3 事件爬网程序使用加密

为 Amazon S3 事件通知设置账户

本节介绍如何为 Amazon S3 事件通知设置账户，并提供使用脚本或 AWS Glue 控制台执行此操作的说明。

先决条件

完成以下设置任务。请注意，括号中的值引用了脚本中的可配置设置。

1. 创建 Amazon S3 存储桶 (s3_bucket_name)。
2. 识别爬网程序目标（folder_name，例如“test1”），即识别的存储桶中的路径。
3. 准备爬网程序名称 (crawler_name)
4. 准备 SNS 主题名称 (sns_topic_name)，该名称可能与爬网程序名称相同。
5. 准备要运行爬网程序且存在 S3 存储桶的 AWS 区域 (region)。
6. 如果使用电子邮件获取 Amazon S3 事件，则可以选择准备电子邮件地址 (subscribing_email)。

您也可以使用 CloudFormation 堆栈创建您的资源。完成以下步骤：

1. 在美国东部（弗吉尼亚州北部）启动您的 CloudFormation 堆栈：
2. 在 Parameters 下方中输入 Amazon S3 存储桶的名称（包括您的账号）。
3. 选择 I acknowledge that AWS CloudFormation might create IAM resources with custom names。
4. 选择Create stack。

限制:

- 无论是 Amazon S3 还是 Data Catalog 目标，爬网程序仅支持单个目标。
- 不支持私有 VPC 上的 SQS。
- 不支持 Amazon S3 采样。
- 爬网程序目标应为 Amazon S3 目标的文件夹，或者 Data Catalog 目标的一个或多个 AWS Glue Data Catalog 表。
- 不支持“所有”路径通配符：s3://%
- 对于 Data Catalog 目标，所有目录表都应指向 Amazon S3 事件模式的同一 Amazon S3 存储桶。
- 对于 Data Catalog 目标，目录表不应指向 Delta Lake 格式的 Amazon S3 位置（包含 _symlink 文件夹或检查目录表的 `InputFormat`）。

要使用基于 Amazon S3 事件的爬网程序，您应该在 S3 存储桶上启用事件通知，并使用从与 S3 目标相同的前缀中筛选的事件，并存储在 SQS 中。您可以按照演练：为存储桶配置通知中的步骤或使用 the section called “用于从目标生成 SQS 和配置 Amazon S3 事件的脚本” 通过控制台设置 SQS 和事件通知。

SQS 策略

添加以下 SQS 策略，需要附上爬网程序使用的角色。

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

用于从目标生成 SQS 和配置 Amazon S3 事件的脚本

确保满足先决条件后，可以运行以下 Python 脚本来创建 SQS。将可配置设置替换为根据先决条件准备的名称。

> **Note**
>
> 运行脚本后，登录到 SQS 控制台以查找创建的 SQS 的 ARN。

Amazon SQS 会设置可见性超时，即 Amazon SQS 阻止其他用户接收并处理消息的一段时间。将可见性超时设置为大致等于网络爬取运行时。

```
#!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 -----------------
```

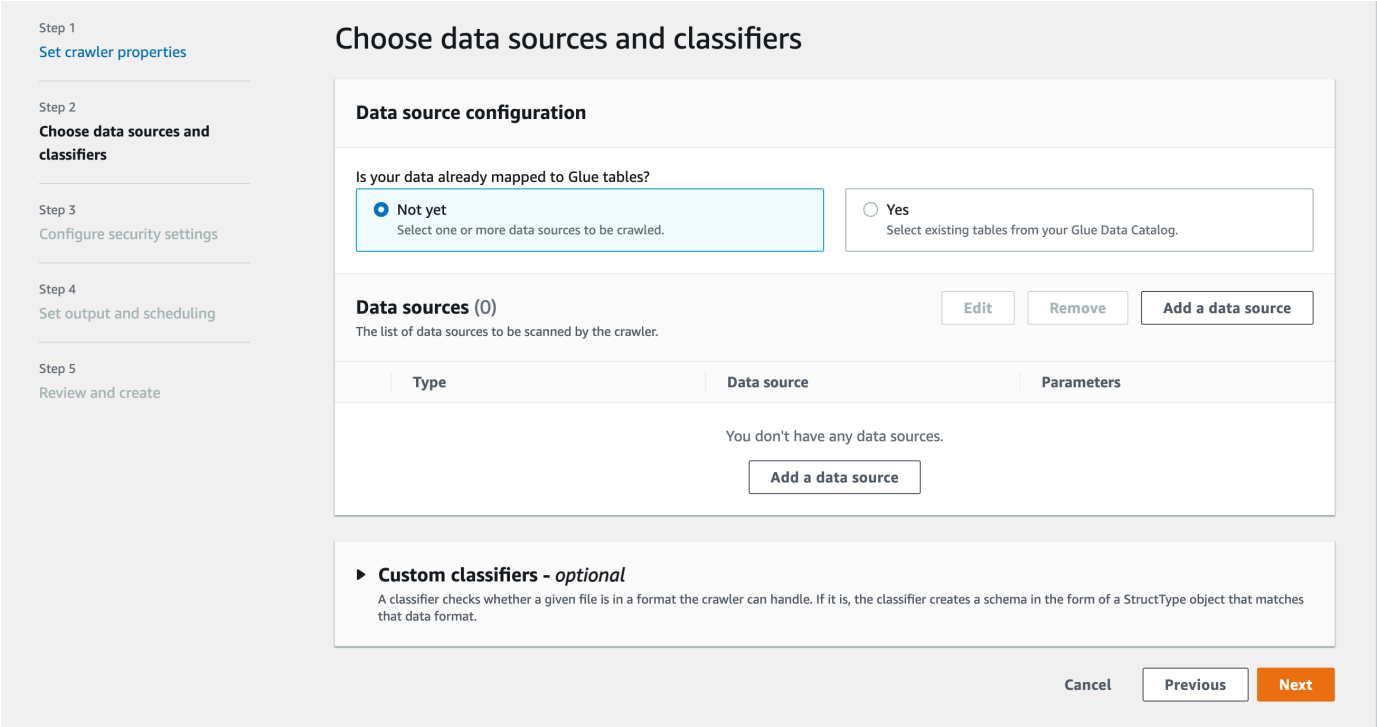
```
            print('\nSetting topic policy to allow s3 bucket %s to publish.' %
s3_bucket_name)
            if set_sns_topic_policy(topic_arn, sns, s3_bucket_name):
                print('SNS topic policy added successfully.')
                if set_s3_notification_sns(s3_bucket_name, s3, topic_arn):
                    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.')
```

使用控制台为 Amazon S3 事件通知设置爬网程序（Amazon S3 目标）

要针对 Amazon S3 目标使用 AWS Glue 控制台为 Amazon S3 事件通知设置爬网程序，请执行以下操作：

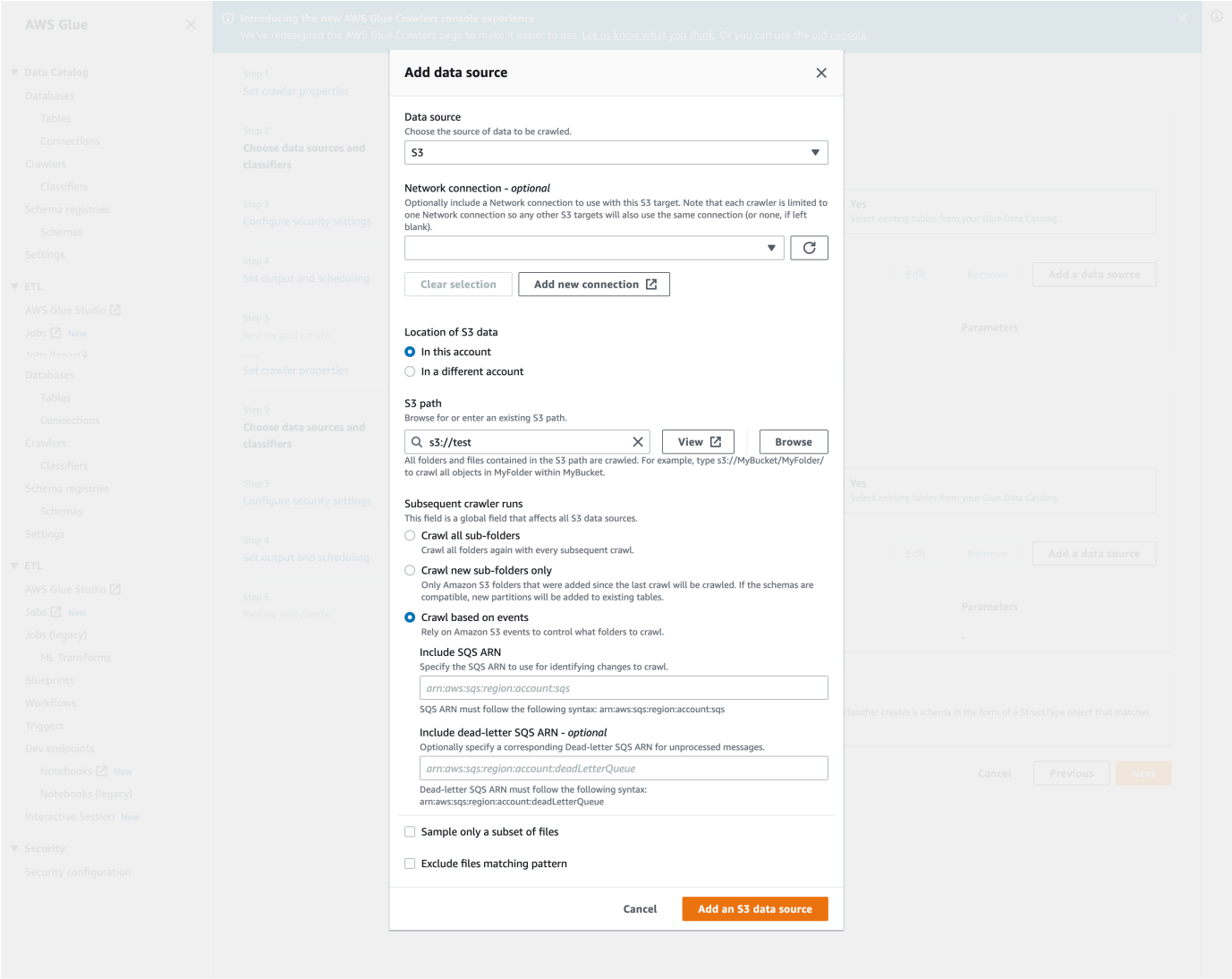
1. 设置爬网程序属性。有关更多信息，请参阅在 AWS Glue 控制台上设置爬网程序配置选项。
2. 在数据来源配置部分中，系统将询问您的数据是否已映射到 AWS Glue 表？

   默认情况下已选择 Not yet（尚未）。请将其保留为默认值，这是因为您使用的是 Amazon S3 数据来源，而该数据尚未映射到 AWS Glue 表。
3. 在 Data sources（数据来源）部分中，选择 Add a data source（添加数据来源）。



4. 在 Add data source（添加数据来源）模态中，配置 Amazon S3 数据来源：

   - Data source（数据来源）：默认选择 Amazon S3。
   - Network connection（网络连接）（可选）：选择 Add new connection（添加新连接）。
   - Location of Amazon S3 data（Amazon S3 数据位置）：默认选择 In this account（此账户中）。
   - Amazon S3 path（Amazon S3 路径）：指定在其中爬取文件夹和文件的 Amazon S3 路径。
   - Subsequent crawler runs（后续爬网程序运行）：选择 Crawl based on events（基于事件爬取）以对爬网程序使用 Amazon S3 事件通知。
   - Include SQS ARN（包含 SQS ARN）：指定数据存储参数，包括有效的 SQS ARN。（例如，`arn:aws:sqs:region:account:sqs`）。
   - Include dead-letter SQS ARN（包含死信 SQS ARN）（可选）：指定有效的 Amazon 死信 SQS ARN。（例如，`arn:aws:sqs:region:account:deadLetterQueue`）。
   - 选择 Add an Amazon S3 data source（添加 Amazon S3 数据来源）。



使用 AWS CLI 为 Amazon S3 事件通知设置爬网程序

以下是在 Amazon S3 目标存储桶上创建 SQS 队列和设置事件通知的 Amazon S3 AWS CLI 调用示例。

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

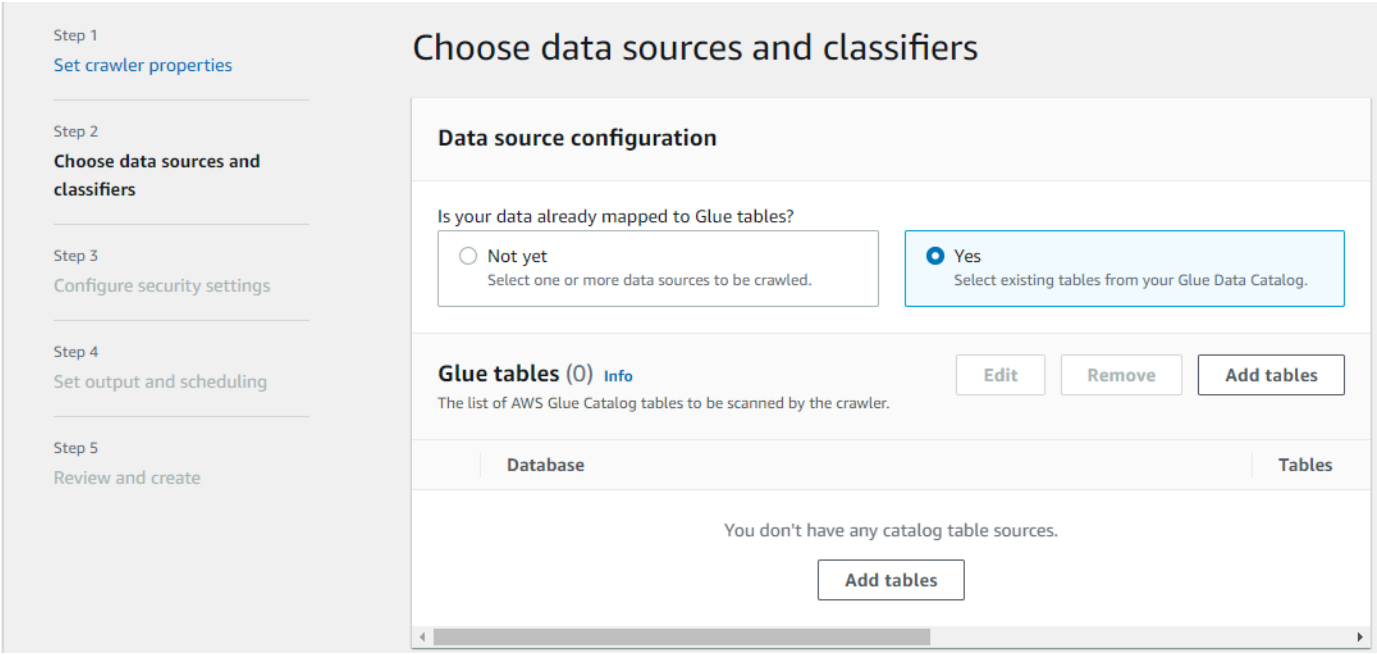
Create Crawler:

使用控制台为 Amazon S3 事件通知设置爬网程序（Data Catalog 目标）

当您有目录目标时，请使用 AWS Glue 控制台为 Amazon S3 事件通知设置爬网程序：

1. 设置爬网程序属性。有关更多信息，请参阅在 AWS Glue 控制台上设置爬网程序配置选项。
2. 在数据来源配置部分中，系统将询问您的数据是否已映射到 AWS Glue 表？

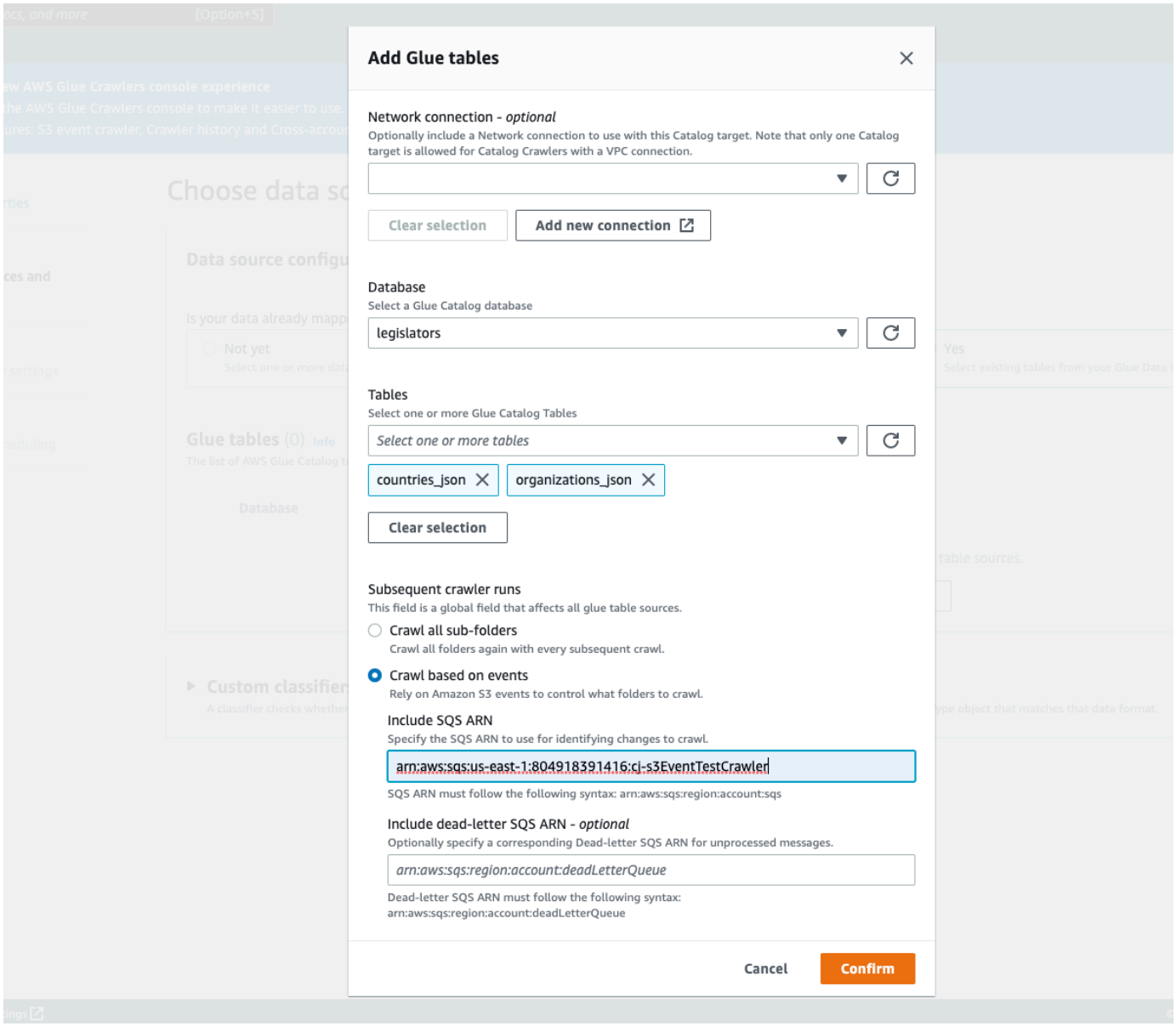
   选择 Yes（是），从 Data Catalog 中选择现有表作为数据来源。
3. 在 Glue tables（Glue 表）部分中，选择 Add tables（添加表）。



4. 在 Add table（添加表）模式中，配置数据库和表：

   - Network connection（网络连接）（可选）：选择 Add new connection（添加新连接）。

- Database（数据库）：在 Data Catalog 中选择数据库。
- Tables（表）：在 Data Catalog 中选择该数据库中的一个或多个表。
- Subsequent crawler runs（后续爬网程序运行）：选择 Crawl based on events（基于事件爬取）以对爬网程序使用 Amazon S3 事件通知。
- Include SQS ARN（包含 SQS ARN）：指定数据存储参数，包括有效的 SQS ARN。（例如，`arn:aws:sqs:region:account:sqs`）。
- Include dead-letter SQS ARN（包含死信 SQS ARN）（可选）：指定有效的 Amazon 死信 SQS ARN。（例如，`arn:aws:sqs:region:account:deadLetterQueue`）。
- 选择确认。



对 Amazon S3 事件爬网程序使用加密

本节介绍仅对 SQS 或同时对 SQS 和 Amazon S3 使用加密。

主题

- 仅对 SQS 启用加密
- 同时对 SQS 和 Amazon S3 启用加密
- 常见问题解答

仅对 SQS 启用加密

预设情况下，Amazon SQS 会在传输过程中提供加密。要向队列中添加可选的服务器端加密 (SSE)，可以在编辑面板中附上客户主密钥 (CMK)。这意味着 SQS 会加密 SQS 服务器上所有客户静态数据。

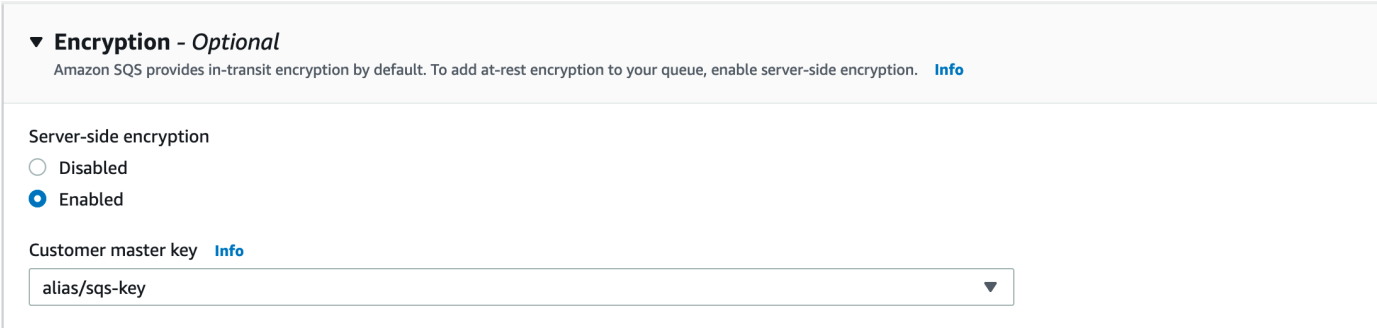
创建客户主密钥 (CMK)

1. 选择 Key Management Service (KMS)（密钥管理服务 (KMS)）> Customer Managed Keys（客户管理的密钥）> Create key（创建密钥）。
2. 按照以下步骤添加您自己的别名和说明。
3. 添加您希望能够使用此密钥的相应 IAM 角色。
4. 在密钥策略中，为“语句”列表添加另一条语句，以便自定义密钥策略可以为 Amazon SNS 提供足够的密钥使用权限。

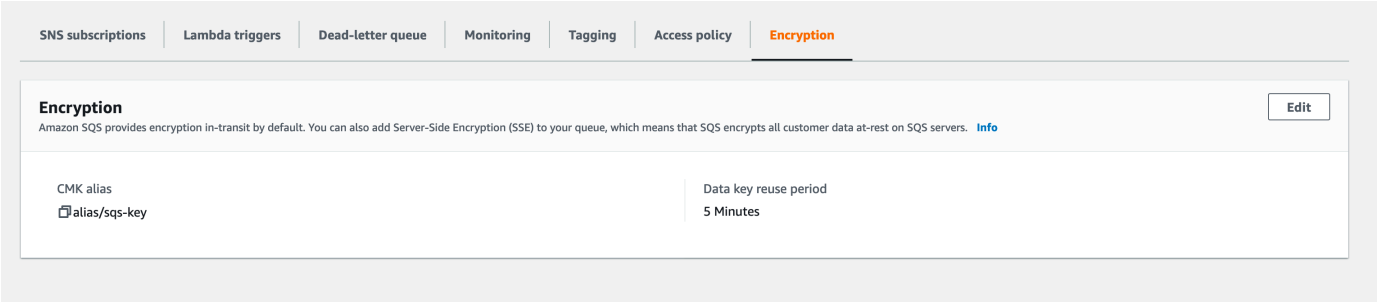
```
"Statement": [
       {
           "Effect": "Allow",
           "Principal": {
               "Service": "sns.amazonaws.com"
           },
           "Action": [
               "kms:GenerateDataKey",
               "kms:Decrypt"
           ],
           "Resource": "*"
       }
]
```

为队列启用服务器端加密 (SSE)

1. 选择 Amazon SQS > Queues（队列）> sqs_queue_name > Encryption（加密）选项卡。
2. 选择 Edit（编辑），然后向下滚动至 Encryption（加密）下拉列表。
3. 选择 Enabled（已启用）以添加 SSE。
4. 选择先前创建的 CMK，而不是名称为 `alias/aws/sqs` 的原定设置密钥。



添加此密钥后，加密选项卡更新为添加的密钥。



> **Note**
>
> Amazon SQS 会自动删除在队列中已过了最大消息保存期的消息。默认的消息保存期为 4 天。为避免事件丢失，请将 SQS MessageRetentionPeriod 更改为最大值 14 天。

同时对 SQS 和 Amazon S3 启用加密

为 SQS 启用服务器端加密 (SSE)

1. 按 the section called “仅对 SQS 启用加密” 中的步骤操作。
2. 在 CMK 设置的最后一个步骤中，为 Amazon S3 提供足够的密钥使用权限。

   将以下内容粘贴到“语句”列表中：

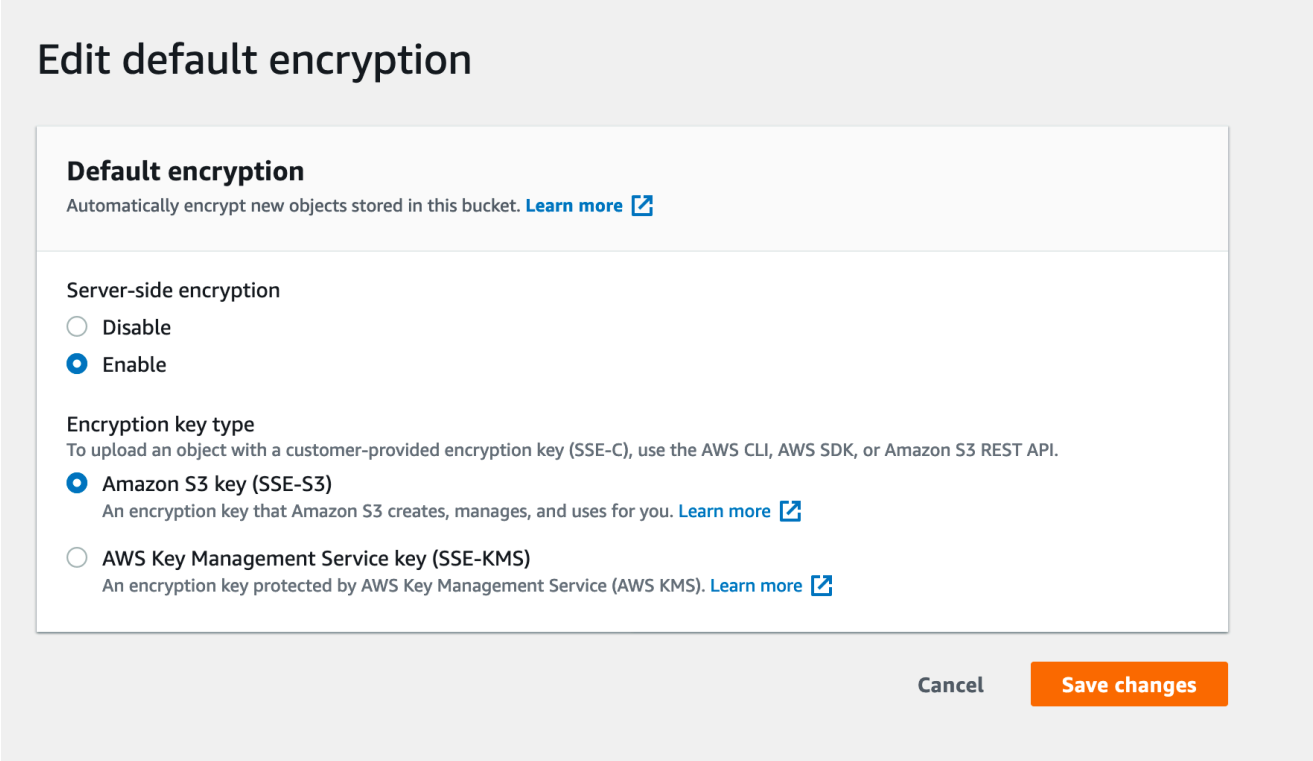
```
"Statement": [
      {
          "Effect": "Allow",
          "Principal": {
              "Service": "s3.amazonaws.com"
          },
```

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

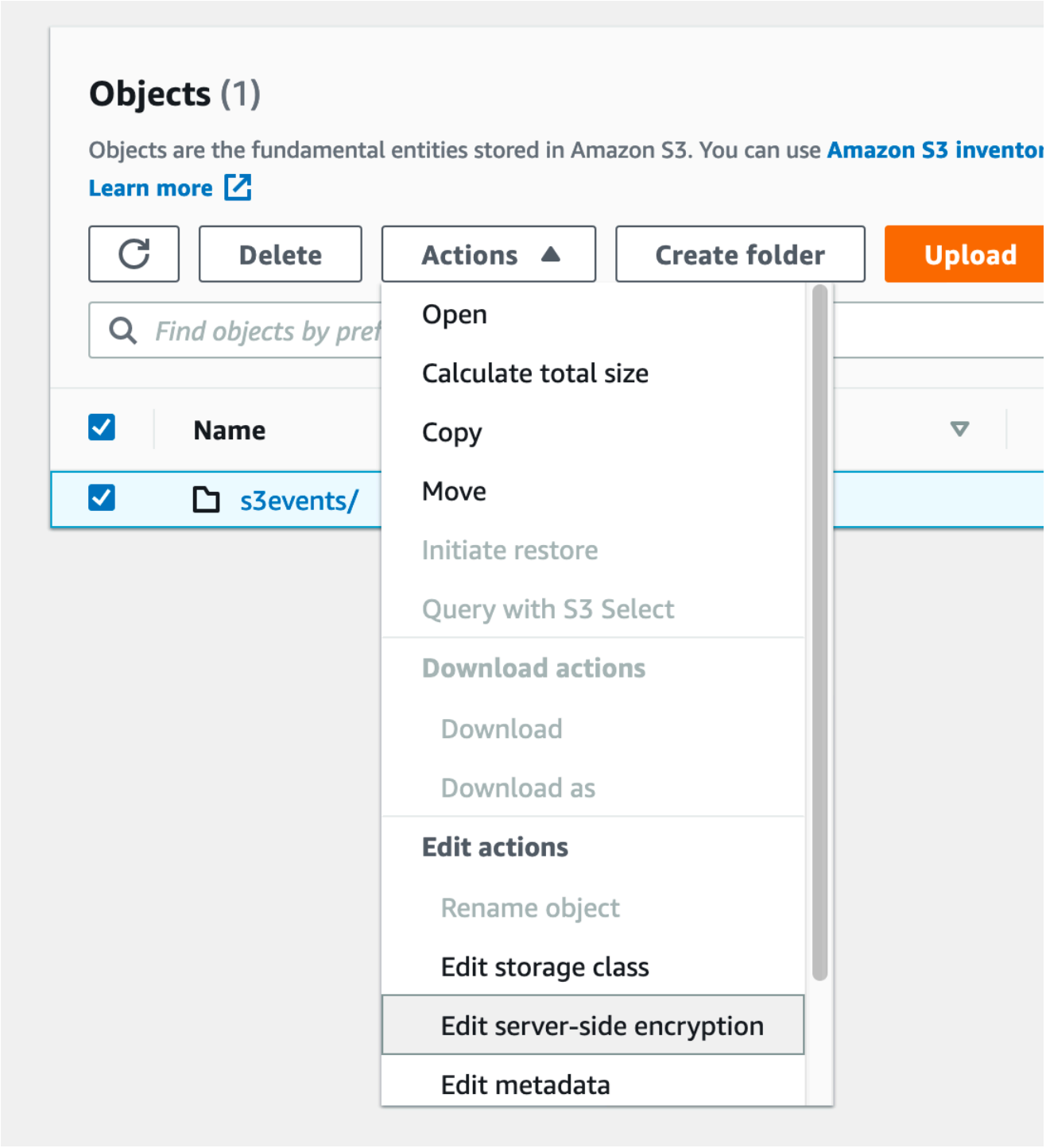
为 Amazon S3 存储桶启用服务器端加密（SSE）

1. 按 the section called “仅对 SQS 启用加密” 中的步骤操作。
2. 请执行以下操作之一：
   - 要为整个 S3 存储桶启用 SSE，请导航到目标存储桶中的 Properties（属性）选项卡。

     在这里，您可以启用 SSE 并选择要使用的加密类型。Amazon S3 提供了 Amazon S3 为您创建、管理和使用的加密密钥，您也可以从 KMS 中选择密钥。



   - 要对特定文件夹启用 SSE，请单击目标文件夹旁边的复选框，然后在 Actions（操作）下拉列表下选择 Edit server-side encryption（编辑服务器端加密）。



常见问题解答

为什么我发布到 Amazon SNS 主题的消息没有传递到我订阅的启用了服务器端加密 (SSE) 的 Amazon SQS 队列中？

仔细检查您的 Amazon SQS 队列是否正在使用：

1. 由客户管理的客户主密钥 (CMK)。不是 SQS 提供的原定设置密钥。
2. (1) 中的 CMK 包括一个自定义密钥策略，为 Amazon SNS 提供足够的密钥使用权限。

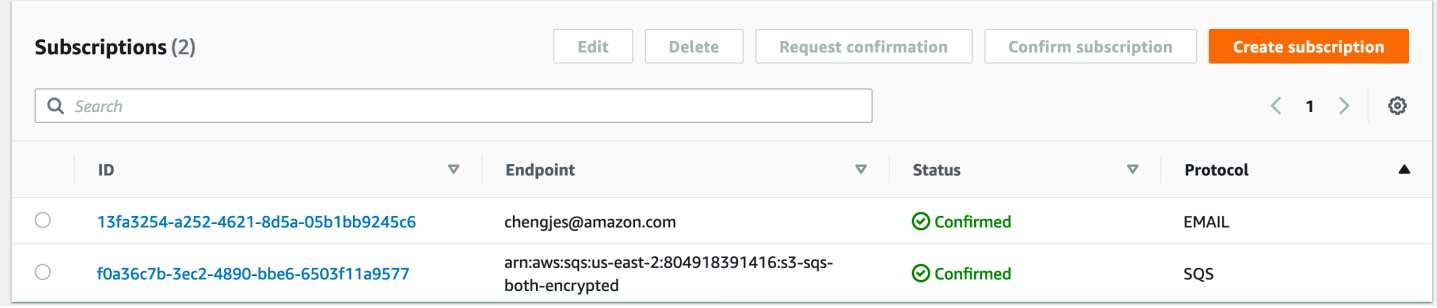
有关更多信息，请参阅知识中心中的 此文章。

我已经订阅了电子邮件通知，但在编辑 Amazon S3 存储桶时，我没有收到任何电子邮件更新。

单击电子邮件中的“Confirm Subscription”（确认订阅）链接，确保您已确认您的电子邮件地址。您可以检查 SNS 主题下的 Subscriptions（订阅）表来验证确认的状态。

选择 Amazon SNS > Topics（主题）> `sns_topic_name` > Subscriptions table（订阅表）。

如果您遵循了我们的先决条件脚本，您将发现 `sns_topic_name` 等于 `sqs_queue_name`。如下所示：



对我的 SQS 队列启用服务器端加密后，我添加的文件夹中只有一些显示在我的表中。为什么我缺少了一些 parquet？

如果对 SQS 队列启用 SSE 之前进行了 Amazon S3 存储桶更改，爬网程序可能无法获取这些更改。要确保已网络爬取 S3 存储桶的所有更新，请在列表模式下再次运行爬网程序（“Crawl All Folders”（网络爬取所有文件夹））。另一个选项是创建启用了 S3 事件的新爬网程序重新开启。

如何阻止爬网程序更改现有架构

如果您不希望爬网程序覆盖您对 Amazon S3 表定义中的现有字段进行的更新，请在控制台上选择选项 Add new columns only (仅添加新列) 或设置配置选项 `MergeNewColumns`。这适用于表和分区，除非 `Partitions.AddOrUpdateBehavior` 被覆盖为 `InheritFromTable`。

如果您不希望在爬网程序运行时更改表架构，请将架构更改策略设置为 `LOG`。您还可以设置将分区架构设置为从表继承的配置选项。

如果您在控制台上配置爬网程序，可以选择以下操作：

- Ignore the change and don't update the table in the Data Catalog（忽略更改，不更新数据目录中的表）
- Update all new and existing partitions with metadata from the table (使用表中的元数据更新所有新的和现有的分区)

当您使用 API 配置爬网程序时，请设置以下参数：

- 将 SchemaChangePolicy 结构中的 UpdateBehavior 字段设置为 LOG。
- 使用爬网程序 API 中的以下 JSON 对象的字符串表示形式设置 Configuration 字段；例如：

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

如何为每个 Amazon S3 包含路径创建单个架构

默认情况下，当爬网程序为 Amazon S3 中存储的数据定义表时，它会同时考虑数据兼容性和架构相似性。它考虑的数据兼容性因素包括数据是否具有相同的格式（例如，JSON），相同的压缩类型（例如，GZIP），Amazon S3 路径的结构以及其他数据属性。架构相似性衡量单独 Amazon S3 对象的架构的相似程度。

您可以在可能的情况下将 CombineCompatibleSchemas 的爬网程序配置为公用表定义。使用此选项，爬网程序仍会考虑数据兼容性，但在评估指定包含路径中的 Amazon S3 对象时会忽略特定架构的相似性。

如果要在控制台上配置爬网程序以组合架构，请选择爬网程序选项 Create a single schema for each S3 path (为每个 S3 路径创建单个架构)。

当您使用 API 配置爬网程序时，请设置以下配置选项：

- 使用爬网程序 API 中的以下 JSON 对象的字符串表示形式设置 Configuration 字段；例如：

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

```
}
```

为帮助说明此选项，假设您使用包含路径 s3://bucket/table1/ 定义了一个爬网程序。当该爬网程序运行时，它会找到两个具有以下特征的 JSON 文件：

- 文件 1 – S3://bucket/table1/year=2017/data1.json
- 文件内容 – {“A”: 1, “B”: 2}
- 架构 – A:int, B:int

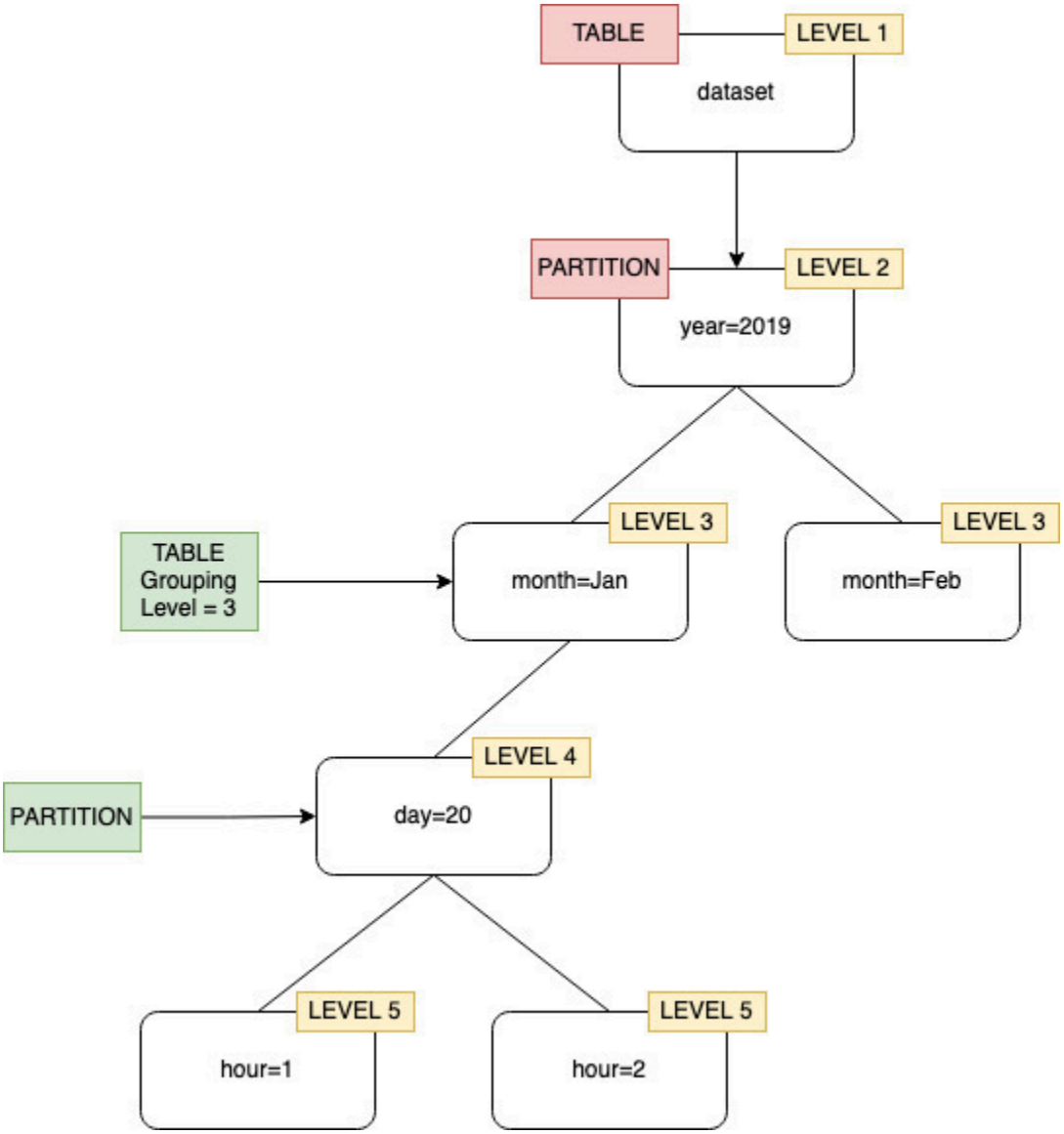
- 文件 2 – S3://bucket/table1/year=2018/data2.json
- 文件内容 – {“C”: 3, “D”: 4}
- 架构 – C: int, D: int

默认情况下，该爬网程序会创建两个名为 year_2017 和 year_2018 的表，因为架构不够相似。但是，如果选项 Create a single schema for each S3 path (为每个 S3 路径创建单个架构) 处于选中状态，并且数据是兼容的，则爬网程序会创建一个表。该表具有架构 A:int,B:int,C:int,D:int 和 partitionKey year:string。

如何指定表位置和分区级别

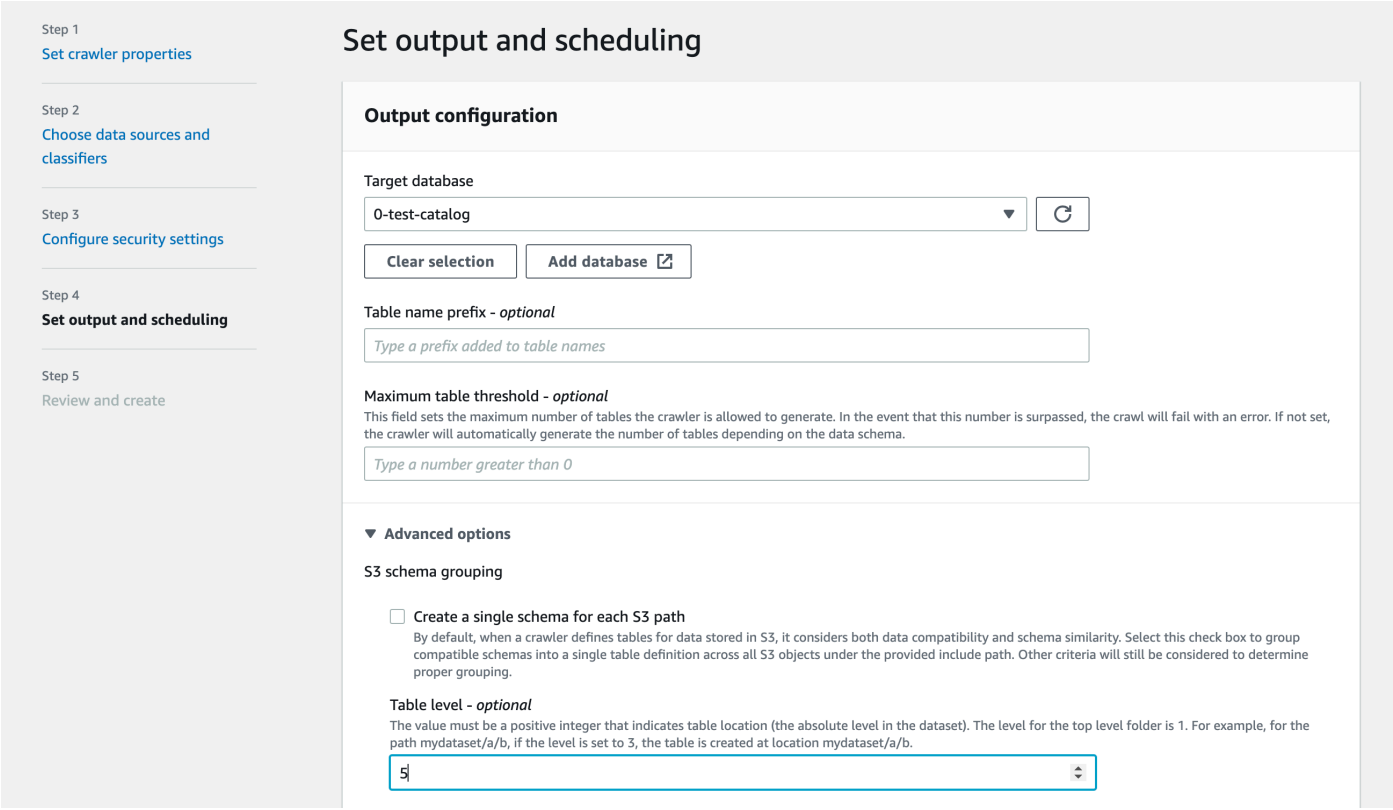
默认情况下，当爬网程序为 Amazon S3 中存储的数据定义表时，爬网程序会尝试将架构合并在一起并创建顶级表（year=2019）。在某些情况下，您可能希望爬网程序为文件夹 month=Jan 创建一个表，但由于同级文件夹（month=Mar）已合并到同一个表中，因此爬网程序会创建一个分区。

通过表级别爬网程序选项，您可以灵活地告诉爬网程序表的位置，以及您希望如何创建分区。当您指定 Table level (表级别) 时，则会从 Amazon S3 存储桶中以该绝对级别创建表。



当在控制台上配置爬网程序时，您可以为 Table level (表级别) 爬网程序选项指定一个值。该值必须是指示表位置（数据集中的绝对级别）的正整数。顶级文件夹的级别为 1。例如，对于路径 `mydataset/year/month/day/hour`，如果级别设置为 3，则在位置 `mydataset/year/month` 处创建表。

Console



API

使用 API 配置爬网程序时，请使用以下 JSON 对象的字符串表示形式设置 `Configuration` 字段；例如：

```
configuration = jsonencode(

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

CloudFormation

在本例中，您在 CloudFormation 模板的控制台中设置了可用的表级别选项：

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

```
}"
```

如何指定允许爬网程序创建的最大表数

您可以选择允许爬网程序创建的最大表数，方法是通过 AWS Glue 控制台或 CLI 指定 `TableThreshold`。如果爬网程序在其爬取过程中检测到的表数大于此输入值，则爬取失败且不会向 Data Catalog 写入任何数据。

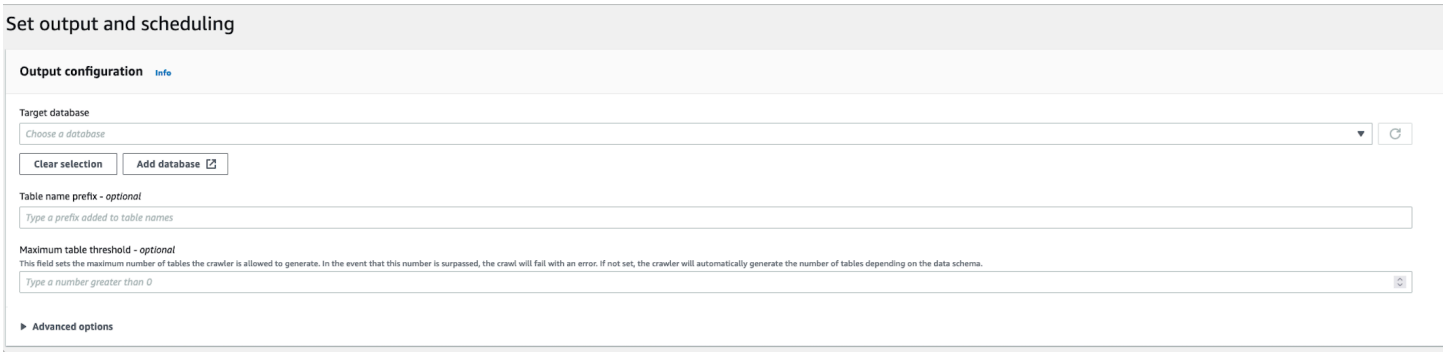
当爬网程序检测和创建的表数比预期表数要大得多时，此参数非常有用。这可能有多种原因，例如：

- 使用 AWS Glue 作业填充 Amazon S3 位置时，您最终可能会得到与文件夹相同级别的空文件。在这种情况下，当您在此 Amazon S3 位置运行爬网程序时，由于文件和文件夹位于同一级别，爬网程序会创建多个表。
- 如果没有配置 `"TableGroupingPolicy": "CombineCompatibleSchemas"`，您最终得到的表数可能比预期数量多。

您可以将 `TableThreshold` 指定为一个大于 0 的整数值。该值根据每个爬网程序进行配置。也就是说，每次爬取都会考虑该值。例如：爬网程序的 `TableThreshold` 值设置为 5。每次爬取时，AWS Glue 会将检测到的表数与此表阈值（5）进行比较。如果检测到的表数小于 5，AWS Glue 将表写入 Data Catalog；否则，爬取失败，将不会写入 Data Catalog。

控制台

使用 AWS 控制台设置 `TableThreshold`：



CLI

使用 AWS CLI 设置 `TableThreshold`：

```
"{"Version":1.0,
"CrawlerOutput":
```

```
{"Tables":{"AddOrUpdateBehavior":"MergeNewColumns",
"TableThreshold":5}}}";
```

记录错误消息以帮助您识别表路径和清理数据。爬网程序因表数大于提供的表阈值而失败时您账户中的日志示例：

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

在 CloudWatch 中，我们将检测到的所有表位置记录为 INFO 消息。将错误记录为失败原因。

```
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.
```

如何为 Delta Lake 数据存储指定配置选项

在为 Delta Lake 数据存储配置网络爬取程序时，可以指定以下配置参数：

Connection

可以选择或添加要用于此 Amazon S3 目标的网络连接。有关连接的信息，请参阅 连接到数据。

创建用于查询的表

选择要如何创建 Delta Lake 表：

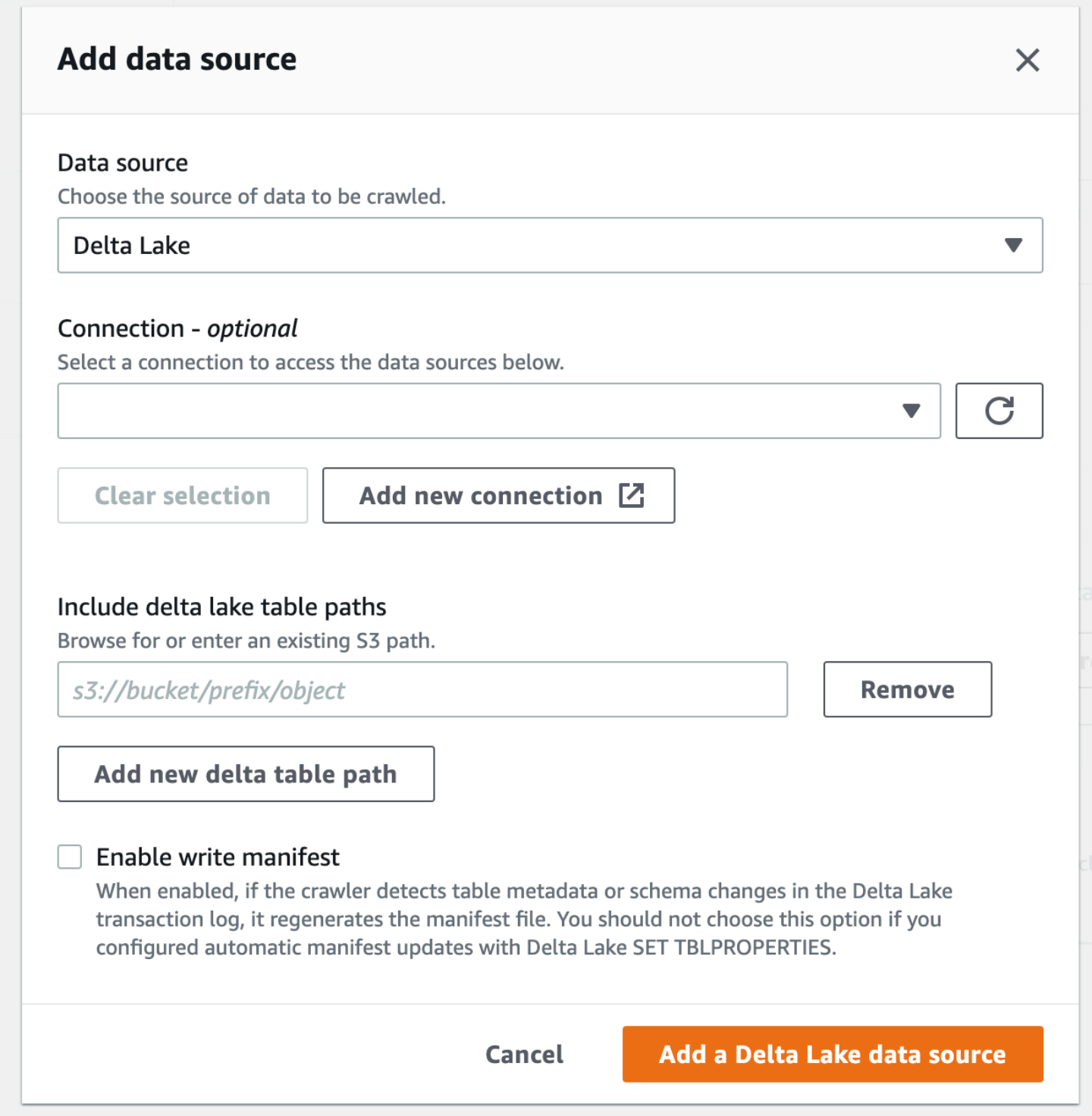
- 创建原生表：允许与支持直接查询 Delta 事务日志的查询引擎集成。
- 创建符号链接表：根据指定的配置参数，使用由由分区键分区的清单文件创建符号链接清单文件夹。

启用写入清单（仅限您选择为 Delta Lake 源创建符号链接表时才可配置）

选择是否检测 Delta Lake 事务处理日志中的表元数据或 Schema 更改；它会重新生成清单文件。如果已经使用 Delta Lake `SET TBLPROPERTIES` 配置了自动清单更新，则不应选择此选项。

包含 Delta Lake 表路径

将一个或多个指向 Delta 表的 Amazon S3 路径，格式为 s3://*bucket*/*prefix*/*object*。



如何将爬网程序配置为使用 Lake Formation 凭证

您可以将爬网程序配置为使用 AWS Lake Formation 凭证访问 Amazon S3 数据存储库或 Data Catalog 表的证书，该表包含相同 AWS 账户 或不同 AWS 账户 中的基础 Amazon S3 位置。如果爬网程序与

Data Catalog 表位于同一账户中，则可以将现有 Data Catalog 表配置为爬网程序的目标。使用 Data Catalog 表作为爬网程序的目标时，目前只允许具有单个目录表的单个目录目标。

> **Note**
>
> 将 Data Catalog 表定义为爬网程序目标时，请确保 Data Catalog 表的基础位置是 Amazon S3 位置。使用 Lake Formation 凭证的爬网程序仅支持具有基础 Amazon S3 位置的 Data Catalog 目标。

爬网程序与注册的 Amazon S3 位置或 Data Catalog 表位于同一账户（账户内爬取）时所需的设置

要允许爬网程序使用 Lake Formation 凭证访问数据存储或 Data Catalog 表，您需要向 Lake Formation 注册数据位置。此外，爬网程序的 IAM 角色必须有从 Amazon S3 桶的注册目标读取数据的权限。

您可以使用 AWS Management Console 或 AWS Command Line Interface（AWS CLI）完成以下配置步骤。

AWS Management Console

1. 在配置爬网程序以访问爬网程序源之前，向 Lake Formation 注册数据存储或 Data Catalog 的数据位置。在 Lake Formation 控制台（https://console.aws.amazon.com/lakeformation/）中，将 Amazon S3 位置注册为定义爬网程序的 AWS 账户 中数据湖的根位置。有关更多信息，请参阅 Registering an Amazon S3 location（注册 Amazon S3 位置）。
2. 向用于爬网程序运行的 IAM 角色授予 Data location（数据位置）权限，以便爬网程序可以从 Lake Formation 中的目标读取数据。有关更多信息，请参阅 Granting data location permissions (same account)（授予数据位置权限（同一账户））。
3. 授予爬网程序角色访问数据库的权限（`Create`），该数据库被指定为输出数据库。有关更多信息，请参阅 Granting database permissions using the Lake Formation console and the named resource method（使用 Lake Formation 控制台和指定的资源方法授予数据库权限）。
4. 在 IAM 控制台（https://console.aws.amazon.com/iam/）中，为爬网程序创建 IAM 角色。将 `lakeformation:GetDataAccess` 策略添加到该角色。
5. 在 AWS Glue 控制台（https://console.aws.amazon.com/glue/）中配置爬网程序时，选择选项 Use Lake Formation credentials for crawling Amazon S3 data source（使用 Lake Formation 凭证爬取 Amazon S3 数据来源）。

> **Note**
>
> accountId 字段对于账户内爬取是可选的。

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": ""
  }
}'
```

爬网程序与注册的 Amazon S3 位置位于不同账户（跨账户爬取）时所需的设置

要允许爬网程序使用 Lake Formation 凭证访问不同账户中的数据存储，您必须先向 Lake Formation 注册 Amazon S3 数据位置。然后，通过执行以下步骤向爬网程序的账户授予数据位置权限。

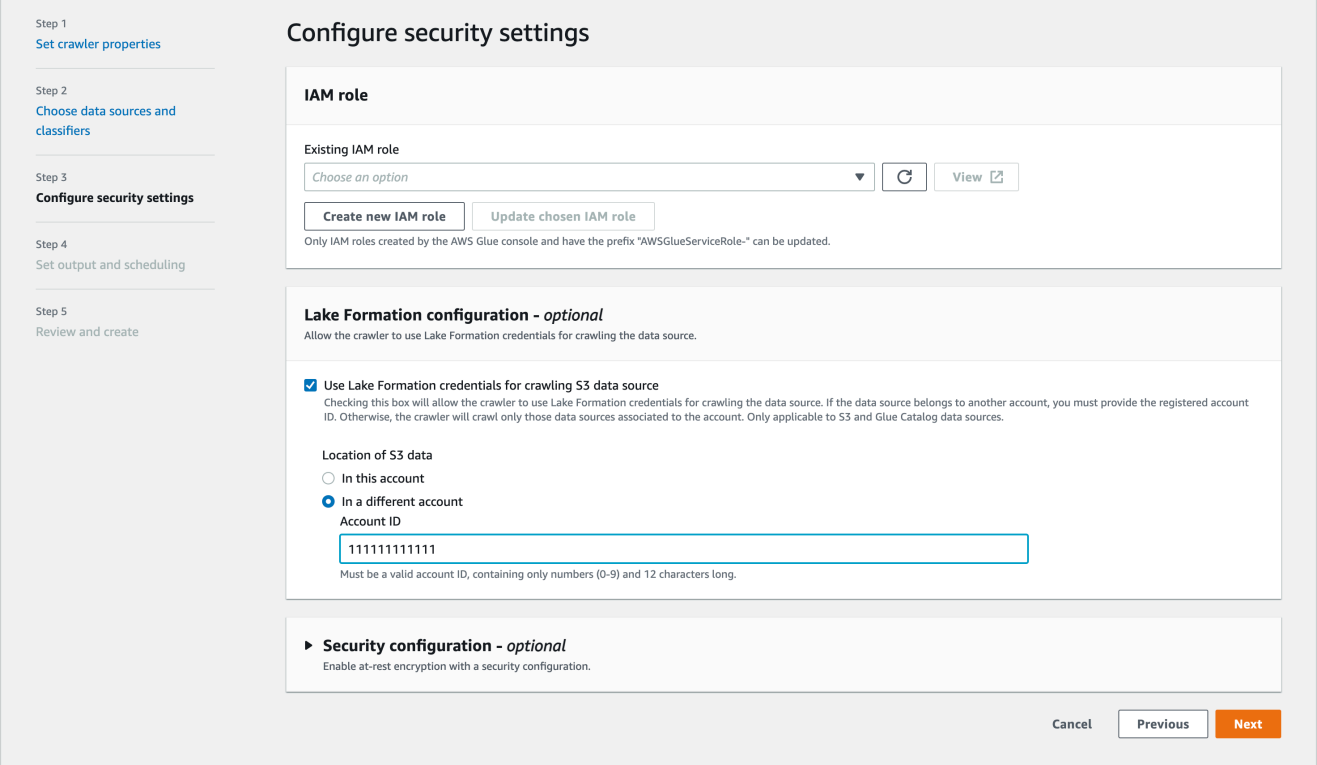
您可以使用 AWS Management Console 或 AWS CLI 完成以下步骤。

AWS Management Console

1. 在注册 Amazon S3 位置的账户（账户 B）中：
   a. 向 Lake Formation 注册 Amazon S3 路径。有关更多信息，请参阅注册 Amazon S3 位置。
   b. 向将运行爬网程序的账户（账户 A）授予 Data location（数据位置）权限。有关更多信息，请参阅授予数据位置权限。
   c. 在 Lake Formation 中创建一个空数据库，将基础位置作为目标 Amazon S3 位置。有关更多信息，请参阅创建数据库。
   d. 授予账户 A（将运行爬网程序的账户）访问您在上一步中创建的数据库的权限。有关更多信息，请参阅授予数据库权限。
2. 在创建并将运行爬网程序的账户（账户 A）中：
   a. 使用 AWS RAM 控制台，接受从外部账户（账户 B）共享的数据库。有关更多信息，请参阅 Accepting a resource share invitation from AWS Resource Access Manager（接受来自 RAMlong 的资源共享邀请）。
   b. 为爬网程序创建 IAM 角色。将 `lakeformation:GetDataAccess` 策略添加到该角色。
   c. 在 Lake Formation 控制台（https://console.aws.amazon.com/lakeformation/）中，授予用于爬网程序运行的 IAM 角色目标 Amazon S3 位置上的 Data location（数据位置）权限，以使爬网程序能够从 Lake Formation 中的目标读取数据。有关更多信息，请参阅授予数据位置权限。
   d. 在共享数据库上创建资源链接。有关更多信息，请参阅创建资源链接。
   e. 授予爬网程序角色对共享数据库和（`Describe`）资源链接的访问权限（`Create`）。资源链接在爬网程序的输出中指定。

f. 在 AWS Glue 控制台（https://console.aws.amazon.com/glue/）中配置爬网程序时，选择选项 Use Lake Formation credentials for crawling Amazon S3 data source（使用 Lake Formation 凭证爬取 Amazon S3 数据来源）。

对于跨账户爬取，请指定向 Lake Formation 注册的目标 Amazon S3 位置的 AWS 账户 ID。对于账户内爬取，accountId 字段是可选的。



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 Data Catalog 目标支持使用 Lake Formation 凭证的爬网程序。
- 对于使用 Lake Formation 凭证售卖的目标，基础 Amazon S3 位置必须属于同一个桶。例如，只要所有目标位置都在同一个桶（bucket1）下，客户就可以使用多个目标（s3://bucket1/folder1、s3://bucket1/folder2）。不允许指定不同的桶（s3://bucket1/folder1、s3://bucket2/folder2）。
- 目前，对于 Data Catalog 目标爬网程序，只允许具有单个目录表的单个目录目标。

教程：添加 AWS Glue 爬网程序

对于此 AWS Glue 场景，需要您分析主要航空公司的抵达数据，从而计算出发机场每月的受欢迎程度。您拥有以 CSV 格式存储在 Amazon S3 中的 2016 年的航班数据。在转换和分析数据之前，请您先将其元数据编录在 AWS Glue Data Catalog 中。

在本教程中，让我们添加一个能够根据 Amazon S3 中的这些飞行日志推断元数据，并且能在您的数据目录中创建表的爬网程序。

主题

先决条件

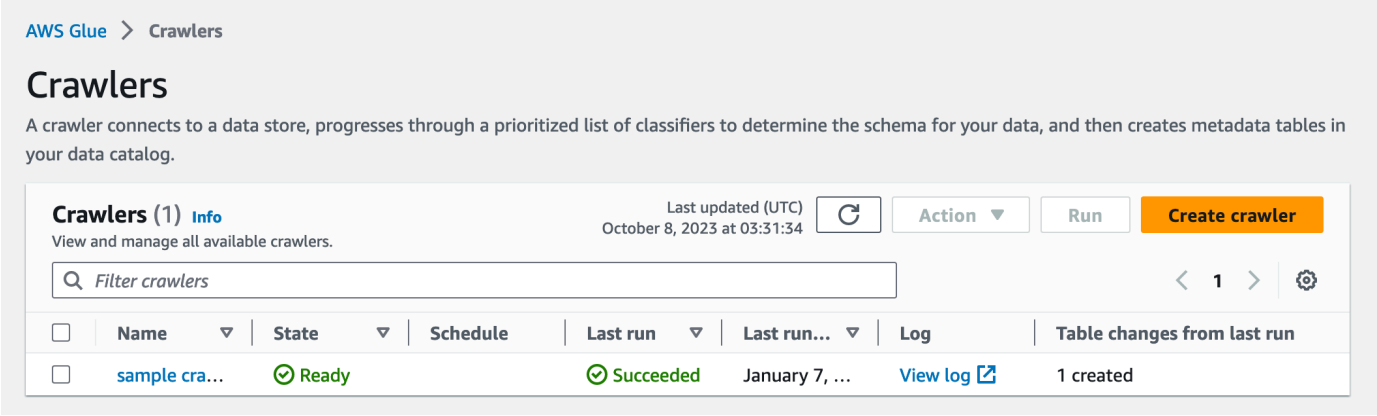
本教程假定您拥有 AWS 账户并具有对 AWS Glue 的访问权限。

步骤 1：添加爬网程序

使用以下步骤配置并运行能够从 Amazon S3 存储的 CSV 文件中提取元数据的爬网程序。

要创建能够读取 Amazon S3 存储的文件的爬网程序

1. 在 AWS Glue 服务控制台的左侧菜单，选择 Crawlers (爬网程序)。
2. 在“爬网程序”页面上，选择创建爬网程序。这将打开一系列页面，提示您输入爬网程序详细信息。



3. 在 Crawler name (爬网程序名称) 字段中，输入 **`Flights Data Crawler`**，然后选择 Next (下一步)。

爬网程序会调用分类器来推断您的数据架构。预设情况下，本教程使用 CSV 的内置分类器。

4. 对于爬网程序源类型，选择 Data stores (数据存储)，然后选择 Next (下一步)。
5. 现在，让我们将爬网程序指向您的数据。在 Add a data store (添加数据存储) 页面上，选择 Amazon S3 data store (Amazon S3 数据存储)。本教程不使用连接，因此请将 Connection (连接) 字段留空（如果可见）。

   对于 Crawl data in (对以下位置的数据进行爬网) 选项，选择 Specified path in another account (另一个账户中的指定路径)。然后，对于 Include path (包含路径)，输入爬网程序可以找到航班数据的路径，即 **`s3://crawler-public-us-east-1/flight/2016/csv`**。输入路径后，此字段的标题将更改为 Include path (包含路径)。选择下一步。
6. 您可以使用单个爬网程序对多个数据存储进行爬网。但是，在本教程中，我们只使用单个数据存储，因此选择 No (否)，然后选择 Next (下一步)。
7. 爬网程序需要权限才能访问数据存储并在 AWS Glue Data Catalog 中创建对象。要配置这些权限，请选择 Create an IAM role (创建 IAM 角色)。IAM 角色名称以 `AWSGlueServiceRole-` 开头，另外在字段中，请您输入角色名称的最后一部分。输入 **`CrawlerTutorial`**，然后选择 Next (下一步)。

   > **Note**
   >
   > 要创建 IAM 角色，您的亚马逊云科技用户必须具有对 `CreateRole`、`CreatePolicy` 和 `AttachRolePolicy` 的访问权限。

   向导将创建名为 `AWSGlueServiceRole-CrawlerTutorial` 的 IAM 角色，并为此角色附上 AWS 托管式策略 `AWSGlueServiceRole`，然后添加允许对 Amazon S3 位置 `s3://crawler-public-us-east-1/flight/2016/csv` 进行读取访问内联策略。
8. 创建爬网程序计划。对于 Frequency (频率)，选择 Run on demand (按需运行)，然后选择 Next (下一步)。
9. 爬网程序在数据目录中创建表。数据目录中的数据库包含元数据表。首先，选择 Add database (添加数据库) 来创建数据库。在弹出窗口中，输入 **`test-flights-db`** 作为数据库名称，然后选择 Create (创建)。

   接下来，输入 **`flights`** 作为添加到表中的前缀。使用其余选项的原定设置值，然后选择 Next (下一步)。

10. 验证您在 Add crawler (添加爬网程序) 向导中所做的选择。如果看到任何错误，您可以选择 Back (返回)，从而返回到之前的页面进行更改。

    查看信息后，选择 Finish (完成)，结束创建爬网程序。

步骤 2：运行爬网程序

创建爬网程序后，向导会将您引导至爬网程序视图页面。由于您使用按需计划创建爬网程序，因此将为您提供运行爬网程序的选项。

要运行爬网程序

1. 此页面顶部附近的横幅会告知您爬网程序已完成创建，并询问您是否要立即运行。选择 Run it now? (现在运行它？) 来运行爬网程序。

   该横幅更改为显示爬网程序的“尝试运行”和“正在运行”消息。爬完程序开始运行后，横幅消失，爬网程序显示更新，以显示启动状态。一分钟后，您可以单击 Refresh (刷新) 图标来更新表中显示的爬网程序的状态。
2. 爬网程序完成后，将出现一条描述爬网程序所做更改的新横幅。您可以选择 test-flights-db 链接来查看数据目录对象。

步骤 3：查看 AWS Glue Data Catalog 对象

爬网程序在源位置读取数据并在数据目录中创建表。表是代表您的数据及其架构的元数据定义。数据目录中的表不包含数据。相反，您可以使用这些表作为任务定义中的源或目标。

要查看爬网程序创建的数据目录对象

1. 在左侧导航窗格中的 Data catalog (数据目录) 下方，选择 Databases (数据库)。在这里，您可以查看爬网程序创建的 `flights-db` 数据库。
2. 在左侧导航窗格的 Data catalog (数据目录) 以及 Databases (数据库) 下方，选择 Tables (表)。在这里，您可以查看爬网程序创建的 `flightscsv` 表。如果您选择表名称，则可以查看表设置、参数和属性。在此视图中向下滚动，您可以查看关于表的列和数据类型信息的架构。
3. 如果在表视图页面上选择 View partitions (查看分区)，您可以看到为数据创建的分区。第一列将作为分区键。

手动定义元数据

AWS Glue Data Catalog 是一个中央存储库，用于存储有关您的数据来源和数据集的元数据。虽然爬网程序可以自动爬取和填充支持的数据来源的元数据，但在某些情况下，您可能需要在 Data Catalog 中手动定义元数据：

- 不支持的数据格式 – 如果您的数据来源不受爬网程序支持，则需要在 Data Catalog 中手动定义这些数据来源的元数据。
- 自定义元数据要求 – AWS Glue 爬网程序 根据预定义的规则和约定推断元数据。如果您有 AWS Glue 爬网程序 推断元数据未涵盖的特定元数据要求，则可以手动定义元数据以满足您的需求
- 数据治理和标准化 – 在某些情况下，出于数据治理、合规性或安全原因，您可能需要对元数据定义拥有更多控制。通过手动定义元数据，您可以确保元数据符合组织的标准和政策。
- 用于未来数据摄取的占位符 – 如果您的数据来源无法立即使用或无法立即访问，则可以创建空架构表作为占位符。数据来源可用后，您可以使用实际数据填充表，同时保持预定义的结构。

要手动定义元数据，您可以使用 AWS Glue 控制台、Lake Formation 控制台、AWS Glue API 或 AWS Command Line Interface（AWS CLI）。您可以创建数据库、表和分区，并指定元数据属性，例如列名称、数据类型、描述和其他属性。

创建数据库

数据库用于组织 AWS Glue 中的元数据表。在 AWS Glue Data Catalog 中定义表时，您将其添加到数据库。表只能位于一个数据库中。

您的数据库可以包含定义来自很多不同数据存储的数据的表。此数据可以包括 Amazon Simple Storage Service（Amazon S3）中的对象和 Amazon Relational Database Service 中的关系表。

> **Note**
>
> 当您从 AWS Glue 数据目录中删除数据库时，也会删除数据库中的所有表。

要查看数据库列表，请登录 AWS Management Console 并通过以下网址打开 AWS Glue 控制台：https://console.aws.amazon.com/glue/。选择 Databases (数据库)，然后在列表中选择一个数据库名称以查看详细信息。

从 AWS Glue 控制台中的 Databases (数据库) 选项卡上，您可以添加、编辑和删除数据库。

- 要创建新的数据库，请选择 Add database (添加数据库) 并提供名称和描述。为了与其他元数据存储 (如 Apache Hive) 兼容，名称会转换为小写字符。

  > **Note**
  >
  > 如果您计划从 Amazon Athena 访问数据库，请提供只包含字母数字和下划线字符的名称。有关更多信息，请参阅 Athena 名称。

- 要编辑数据库的说明，请选中数据库名称旁边的复选框，然后选择 Edit（编辑）。
- 要删除数据库，请选中数据库名称旁边的复选框，然后选择 Remove（删除）。
- 要显示数据库中包含的表列表，请选择数据库名称，数据库属性将显示数据库中的所有表。

要更改爬网程序写入的数据库，必须更改爬网程序定义。有关更多信息，请参阅 使用爬网程序填充 Data Catalog 。

数据库资源链接

AWS Glue 控制台最近已更新。当前版本的控制台不支持数据库资源链接。

数据目录还可以包含到数据库的资源链接。数据库资源链接是指向本地或共享数据库的链接。目前，您只能在 AWS Lake Formation 中创建资源链接。创建到数据库的资源链接后，您可以在需要使用数据库名称的任何位置使用资源链接名称。与您拥有的或与您共享的数据库一起，数据库资源链接由 `glue:GetDatabases()` 返回，并在 AWS Glue 控制台的 Databases (数据库) 页面上显示为条目。

数据目录还可以包含表资源链接。

有关资源链接的更多信息，请参阅《AWS Lake Formation 开发人员指南》中的创建资源链接。

创建表

尽管运行爬网程序是清点数据存储中数据的推荐方法，但您也可以手动将元数据表添加到 AWS Glue Data Catalog 中。通过这种方法，您可以更好地控制元数据定义，并根据您的特定要求对其进行自定义。

您可以通过以下方式向 Data Catalog 中添加表：

- 使用 AWS Glue 控制台在 AWS Glue Data Catalog 中手动创建一个表。有关更多信息，请参阅 在 AWS Glue 控制台上处理表。

- 在 AWS Glue API 中使用 `CreateTable` 操作在 AWS Glue Data Catalog 中创建表。有关更多信息，请参阅 CreateTable 动作（Python：create_table）。
- 使用 AWS CloudFormation 模板。有关更多信息，请参阅 适用于 AWS Glue 的 AWS CloudFormation。

当您使用控制台或 API 手动定义表时，您需要指定表架构和分类字段的值（指示数据源中数据的类型和格式）。如果爬网程序创建表，则数据格式和架构由内置分类器或自定义分类器确定。有关使用 AWS Glue 控制台创建表的更多信息，请参阅在 AWS Glue 控制台上处理表。

主题

- 表分区
- 表资源链接
- 使用爬网程序更新手动创建的数据目录表
- 数据目录表属性
- 在 AWS Glue 控制台上处理表
- 在 AWS Glue 中使用分区索引

表分区

Amazon Simple Storage Service（Amazon S3）文件夹的 AWS Glue 表定义可以描述分区表。例如，要提高查询性能，分区表可以使用月份的名称作为键将每月数据分隔为不同的文件。在 AWS Glue 中，表定义包含表的分区键。当 AWS Glue 评估 Amazon S3 文件夹中的数据以编目表时，它确定是否添加了单个表或分区表。

您可以在表上创建分区索引以获取分区的子集，而不是加载表中的所有分区。有关使用分区索引的信息，请参阅在 AWS Glue 中使用分区索引。

以下 AWS Glue 所有条件都必须为 true，才能为 Amazon S3 文件夹创建分区表：

- 文件的架构类似，由 AWS Glue 确定。
- 文件的数据格式是相同的。
- 文件的压缩格式是相同的。

例如，您可能拥有一个名为 `my-app-bucket` 的 Amazon S3 存储桶，在其中您存储了 iOS 和 Android 应用程序销售数据。该数据按年、月和日分区。适用于 iOS 和 Android 销售的数据文件具有相

同的架构、数据格式和压缩格式。在 AWS Glue Data Catalog 中，AWS Glue 爬网程序使用年、月和日的分区键创建一个表定义。

my-app-bucket 的以下 Amazon S3 列表显示某些分区。= 符号用于分配分区键值。

```
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
```

表资源链接

> AWS Glue 控制台最近已更新。当前版本的控制台不支持表资源链接。

数据目录还可以包含表资源链接。表资源链接是指向本地或共享表的链接。目前，您只能在 AWS Lake Formation 中创建资源链接。创建到表的资源链接后，您可以在需要使用表名称的任何位置使用资源链接名称。与您拥有的或与您共享的表一起，表资源链接由 glue:GetTables() 返回，并在 AWS Glue 控制台的表页面上显示为条目。

数据目录还可以包含数据库资源链接。

有关资源链接的更多信息，请参阅《AWS Lake Formation 开发人员指南》中的创建资源链接。

使用爬网程序更新手动创建的数据目录表

您可能需要手动创建 AWS Glue Data Catalog 表，然后将它们保持为通过 AWS Glue 爬网程序进行更新。按计划运行的爬网程序可以添加新分区，并使用任何架构更改来更新表。这同样适用于从 Apache Hive 元存储中迁移的表。

要执行此操作，当您定义爬网程序时，不是指定一个或多个数据存储作为爬取源，而是指定一个或多个现有数据目录表。然后，爬网程序爬取由目录表指定的数据存储。在这种情况下，不会创建新表；而是更新手动创建的表。

以下是您可能需要手动创建目录表并将目录表指定为爬网程序源的其他原因：

- 您想要选择目录表名称，但不依赖于目录表命名算法。

- 在将其格式可能损坏分区检测的文件错误地保存在数据源路径的情况下，您可能希望阻止创建新表。

有关更多信息，请参阅 步骤 2：选择数据源和分类器。

数据目录表属性

AWS CLI 中已知的表属性或参数是未经验证的键值字符串。您可以在表上设置自己的属性，以支持在 AWS Glue 之外使用数据目录。其他使用数据目录的服务也可以执行该操作。AWS Glue 会在运行作业或爬网程序时设置一些表属性。除非另有说明，否则这些属性仅供内部使用，我们不支持它们继续以其当前形式存在，也不支持手动更改这些属性的产品行为。

更多有关 AWS Glue 爬网程序设置的表属性的信息，请参阅 the section called “爬网程序在数据目录表上设置的参数”。

在 AWS Glue 控制台上处理表

AWS Glue Data Catalog中的表是表示数据存储中的数据的元数据定义。您可以在运行爬网程序时创建表，也可以在 AWS Glue 控制台中手动创建表。AWS Glue 控制台中的 Tables (表) 列表显示表的元数据值。您可以在创建 ETL (提取、转换和加载) 作业时使用表定义来指定源和目标。

> **Note**
>
> 随着最近对 AWS 管理控制台的更改，您可能需要修改现有的 IAM 角色才能获得 `SearchTables` 权限。对于创建新角色，已将 `SearchTables` API 权限添加为默认。

要查看现有任务，请登录 AWS Management Console 并通过以下网址打开 AWS Glue 控制台：https://console.aws.amazon.com/glue/。选择 Tables (表) 选项卡，然后使用 Add tables (添加表) 按钮来通过爬网程序或通过手动键入属性来创建表。

在控制台上添加表

要使用爬网程序添加表，请依次选择 Add tables (添加表) 和 Add tables using a crawler (使用爬网程序添加表)。然后按照 Add crawler (添加爬网程序) 向导中的说明操作。当爬网程序运行时，会将表添加到 AWS Glue Data Catalog。有关更多信息，请参阅 使用爬网程序填充 Data Catalog 。

如果您知道在数据目录中创建 Amazon Simple Storage Service（Amazon S3）表定义所需的属性，则可以使用表向导创建它。请依次选择 Add tables (添加表) 和 Add table manually (手动添加表)，然后按照 Add tables (添加表) 向导中的说明操作。

在通过控制台手动添加表时，请考虑以下各项：

- 如果您计划从 Amazon Athena 访问表，请提供只包含字母数字和下划线字符的名称。有关更多信息，请参阅 Athena 名称。
- 源数据的位置必须是 Amazon S3 路径。
- 数据的数据格式必须与向导中列出的格式之一匹配。将基于所选的格式自动填充相应的分类、SerDe 和其他表属性。您可以使用以下格式定义表：

  Avro

  Apache Avro JSON 二进制格式。

  CSV

  字符分隔值。您还可以指定逗号、竖线、分号、制表符或 Ctrl-A 等分隔符。

  JSON

  JavaScript 对象表示法。

  XML

  可扩展标记语言格式。指定定义数据中的行的 XML 标签。在行标签中定义列。

  Parquet

  Apache Parquet 列式存储。

  ORC

  优化的行列式（ORC）格式。一种旨在高效存储 Hive 数据的格式。
- 您可以为表定义分区键。
- 目前，使用控制台创建的分区表不能用于 ETL 作业。

表属性

以下是表的一些重要属性：

名称

在创建表时确定名称，并且您无法更改它。您在许多 AWS Glue 操作中引用表名称。

数据库

表所在的容器对象。此对象包含 AWS Glue Data Catalog中存在的表组织，并且可能与您的数据存储中的组织不同。当您删除数据库时，也会从数据目录中删除数据库中包含的所有表。

描述

表的描述。您可以编写描述以帮助您了解表的内容。

表格式

指定创建标准 AWS Glue 表或 Apache Iceberg 格式的表。

启用压缩

选择启用压缩，将表中的小 Amazon S3 对象压缩成较大的对象。

IAM 角色

为了运行压缩，该服务会代表您代入一个 IAM 角色。您可以使用下拉列表选择一个 IAM 角色。确保该角色具有启用压缩所需的权限。

要了解该 IAM 角色所需的权限，请参阅 表优化的先决条件 。

位置

指向此表定义表示的数据存储中的数据位置的指针。

分类

在创建表时提供的分类值。通常，在爬网程序运行并指定源数据格式时写入它。

上次更新

在数据目录中更新此表的日期和时间（UTC）。

日期已添加

此表添加到数据目录中的日期和时间（UTC）。

已弃用

如果 AWS Glue 发现数据目录中的表不再存在于其原始数据存储中，则会在数据目录中将此表标记为已淘汰。如果您运行的作业引用已淘汰的表，则此作业可能会失败。编辑引用已淘汰的表的作业，以从源和目标中删除这些表。我们建议您删除不再需要的已淘汰的表。

Connection

如果 AWS Glue 需要连接到您的数据存储，则连接的名称与表相关联。

查看和编辑表详细信息

要查看现有表的详细信息，请在列表中选择表名称，然后选择 Action, View details (操作 -> 查看详细信息)。

表详细信息包括表的属性和架构。此视图显示表的架构，包括按为表定义的顺序排列的列名称、数据类型和分区的键列。如果列是复杂类型，您可以选择 View properties (查看属性) 来显示该字段的结构的详细信息，如以下示例所示：

```
{
"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"
    }
}
```

有关表的属性的更多信息，如 `StorageDescriptor`，请参阅 StorageDescriptor 结构。

要更改表的架构，请选择 Edit schema (编辑架构) 来添加和删除列、更改列名称以及更改数据类型。

要比较不同版本的表，包括其架构，请选择 Compare versions (比较版本) 来查看表架构的两个版本之间的逐项对照比较。有关更多信息，请参阅 比较表架构版本 。

要显示组成 Amazon S3 分区的文件，请选择 View partition (查看分区)。对于 Amazon S3 表，Key (键) 列显示用于对源数据存储中的表进行分区的分区键。分区是根据键列 (如日期、位置或部门) 的值将一个表划分为多个相关部分的方法。有关分区的更多信息，请在 Internet 上搜索有关“hive 分区”的信息。

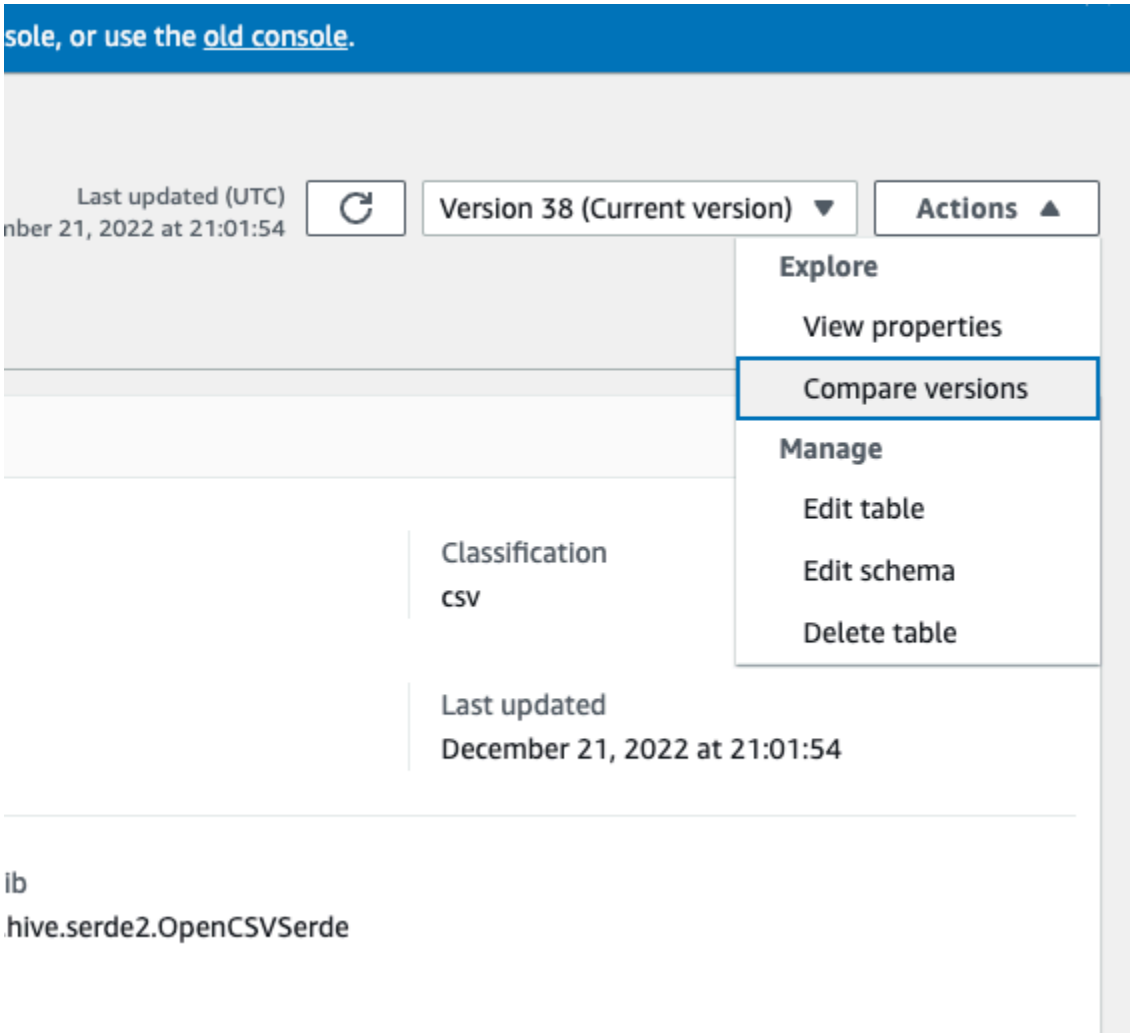
> **Note**
>
> 要获取查看表详细信息的分步指导，请参阅控制台中的 Explore table (浏览表) 教程。

比较表架构版本

比较两个版本的表架构时，可以通过展开和折叠嵌套行来比较嵌套行更改，并排比较两个版本的架构，并排查看表属性。

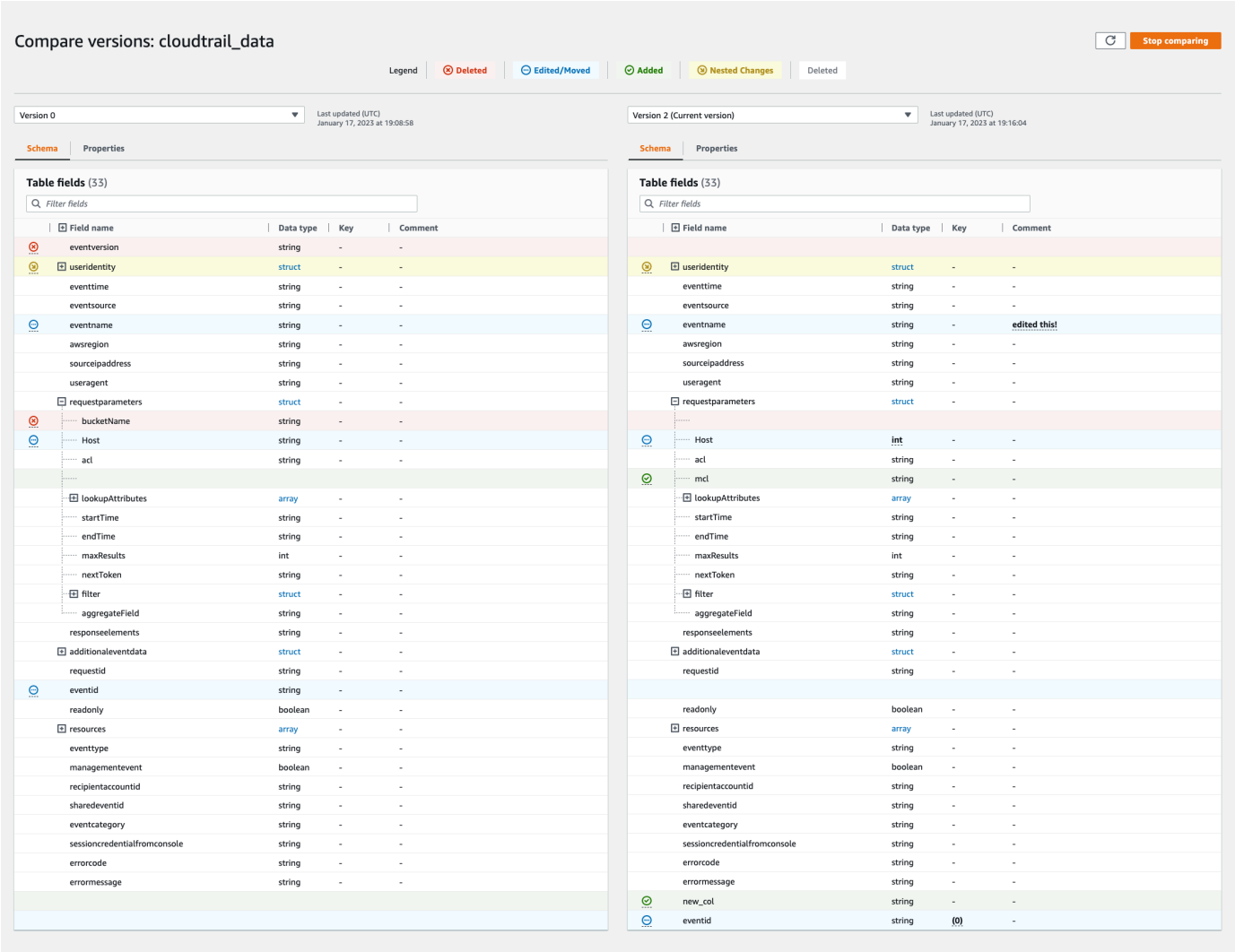
要比较版本，请执行以下操作。

1. 在 AWS Glue 控制台中选择表，然后选择操作，再选择比较版本。



2. 通过选择版本下拉菜单来选择要比较的版本。比较架构时，架构选项卡突出显示为橙色。
3. 比较两个版本之间的表时，将在屏幕的左侧和右侧分别显示表架构。便于您通过并排比较列名称、数据类型、键和注释字段来直观地确定更改。对于有更改的地方，彩色图标会显示所做更改的类型。

   - 已删除：红色图标表示该列已从先前版本的表架构中删除。
   - 已编辑或移动：蓝色图标表示在较新版本的表架构中被修改或移动的列。
   - 已添加：绿色图标表示被添加到新版表架构中的列。
   - 嵌套更改：黄色图标表示包含更改的嵌套列。选择要展开的列并查看已删除、编辑、移动或添加的列。



4. 使用筛选字段搜索栏根据您在此处输入的字符显示字段。如果您在任一表版本中输入列名，筛选后的字段将显示在两个表版本中，以显示发生更改的位置。
5. 要比较属性，请选择属性选项卡。
6. 要停止比较版本，请选择停止比较返回表列表。

在 AWS Glue 中使用分区索引

随着时间的推移，数十万个分区被添加到一个表中。GetPartitions API 用于获取表中的分区。API 返回与请求中提供的表达式匹配的分区。

让我们以 sales_data 表为例，该表依据 Country、Category、Year、Month 和 creationDate 键进行分区。如果您想获得 2020 年 2020-08-15 以后 Books 类别所有已售出商品的销售数据，则必须使用表达式 "Category = 'Books' and creationDate > '2020-08-15'" 向数据目录发出 `GetPartitions` 请求。

如果表中没有分区索引，则 AWS Glue 加载表的所有分区，然后使用用户在 GetPartitions 请求中提供的查询表达式筛选加载的分区。随着没有索引的表上的分区数量的增加，查询需要更多的时间来运行。借助索引，GetPartitions 查询将尝试获取分区的子集，而不是加载表中的所有分区。

主题

关于分区索引

当您创建分区索引时，请指定给定表中已存在的分区键列表。分区索引是表中定义的分区键的子列表。可以在表中定义的分区键的任何排列上创建分区索引。对于上面的 sales_data 表，可能的索引有 (country, category, year, creationDate)、(country, category, year)、(country, category)、(country)、(category, country, year, month) 等。

数据目录将按照创建索引时提供的顺序连接分区值。随着分区被添加到表中，索引的构建是一致的。可以为字符串（string、char 和 varchar）、数值（int、bigint、long、tinyint 和 smallint）和日期 (yyyy-MM-dd) 列类型创建索引。

支持的数据类型

- 日期 – ISO 格式的日期，例如 YYYY-MM-DD。例如，date 2020-08-15。该格式使用连字符（-）来分隔年月日。允许的日期索引范围为 0000-01-01 到 9999-12-31。
- 字符串 – 用单引号或双引号括起的字符串文本。
- Char – 固定长度字符数据，指定长度介于 1 到 255 之间，例如 char(10)。
- Varchar – 可变长度字符数据，具有介于 1 和 65535 之间的指定长度，例如 varchar(10)。
- 数值 – int、bigint、long、tinyint 和 smallint

数值、字符串和日期数据类型的索引支持 =、>、>=、<、<= 以及 between（介于）运算符。
索引解决方案目前仅支持 AND 逻辑运算符。在使用索引进行筛选的表达式中，将忽略带有运算

符“LIKE”、“IN”、“OR”和“NOT”的子表达式。对被忽略的子表达式的筛选是在应用索引筛选后获取的分区上完成的。

对于添加到表中的每个分区，都会创建一个相应的索引项。对于具有“n”个分区的表，1 个分区索引将生成“n”个分区索引项。同一个表上的“m”个分区索引将生成“m*n”个分区索引项。每个分区索引项将根据数据目录存储的当前 AWS Glue 定价策略收费。有关存储对象定价的详细信息，请参阅 AWS Glue 定价。

使用分区索引创建表

您可以在表创建过程中创建分区索引。CreateTable 请求将 PartitionIndex 对象列表作为输入。在给定的表上最多可以创建 3 个分区索引。每个分区索引都需要一个名称和一个为表定义的 partitionKeys 列表。可以使用 GetPartitionIndexes API 获取在表上创建的索引

将分区索引添加到现有表

要将分区索引添加到现有表，请使用 CreatePartitionIndex 操作。您只能为每个 CreatePartitionIndex 操作创建一个 PartitionIndex。添加索引不会影响表的可用性，因为在创建索引期间该表继续可用。

添加的分区的索引状态设置为“CREATING (正在创建)”，并开始创建索引数据。如果创建索引的过程成功，则 indexStatus 将更新为“ACTIVE (活跃)”，对于不成功的过程，索引状态将更新为“FAILED (失败)”。索引创建可能会因多种原因而失败，您可以使用 GetPartitionIndexes 操作检索失败详细信息。可能出现的故障包括：

- ENCRYPTED_PARTITION_ERROR – 不支持在具有加密分区的表上创建索引。
- NVALID_PARTITION_TYPE_DATA_ERROR – 当 partitionKey 值不是相应 partitionKey 数据类型的有效值时观察到。例如：具有“int”数据类型的 partitionKey 的值为“foo”。
- MISSING_PARTITION_VALUE_ERROR – 当 indexedKey 的 partitionValue 不存在时观察到。当表的分区不一致时可能会发生这种情况。
- UNSUPPORTED_PARTITION_CHARACTER_ERROR – 当索引分区键的值包含字符 \u0000、\u0001 或 \u0002 时观察到的
- INTERNAL_ERROR – 在创建索引时发生内部错误。

描述表上的分区索引

要获取在表上创建的分区索引，请使用 GetPartitionIndexes 操作。响应返回表上的所有索引，以及每个索引的当前状态（IndexStatus）。

分区索引的 IndexStatus 将是以下之一：

- CREATING – 索引当前正在创建中，尚不能使用。
- ACTIVE – 索引已准备就绪，可供使用。请求可以使用索引执行优化查询。
- DELETING – 索引当前正在被删除，无法再使用。可以使用 DeletePartitionIndex 请求删除处于活动状态的索引，这会将状态从“ACTIVE (活跃)”移至“DELETING (正在删除)”。
- FAILED – 在现有表上创建索引失败。每个表存储最后 10 个失败的索引。

在现有表上创建的索引的可能状态转换包括：

- CREATING (正在创建) → ACTIVE (活跃) → DELETING (正在删除)
- CREATING (正在创建) → FAILED (失败)

使用分区索引的限制

创建分区索引后，请注意对表和分区功能的以下更改：

创建新分区（添加索引后）

在表上创建分区索引后，添加到表中的所有新分区都将针对索引键的数据类型检查进行验证。索引键的分区值将针对数据类型格式进行验证。如果数据类型检查失败，则创建分区操作将失败。对于 sales_data 表，如果为类别为类型 string 且年份为类型 int 的键 (category, year) 创建索引，则创建 YEAR 值为“foo”的新分区将失败。

启用索引后，添加具有字符 U+0000、U+00001 和 U+0002 的索引键值的分区将开始失败。

表更新

在表上创建分区索引后，您将无法修改现有分区键的分区键名称，也无法更改向该索引注册的键的类型或顺序。

使用索引进行优化的 GetPartitions 调用

当您在具有索引的表上调用 GetPartitions 时，您可以包含一个表达式，如果可能，数据目录将使用索引。索引的第一个键应传递到要用于筛选的索引的表达式中。筛选中的索引优化基于最佳效果应用。数据目录尝试尽可能多地使用索引优化，但在缺少索引或不支持的运算符的情况下，它会回退到加载所有分区的现有实现。

对于上面的 sales_data 表，让我们添加索引 [Country, Category, Year]。如果表达式中未传递“Country”，则注册的索引将无法使用索引筛选分区。您最多可以添加 3 个索引来支持各种查询模式。

让我们举一些示例表达式，看看索引是如何处理它们的：

Expressions	如何使用索引
Country = 'US'	索引将用于筛选分区。
Country = 'US' and Category = 'Shoes'	索引将用于筛选分区。
Category = 'Shoes'	不会使用索引，因为表达式中未提供“country”。将加载所有分区以返回响应。
Country = 'US' and Category = 'Shoes' and Year > '2018'	索引将用于筛选分区。
Country = 'US' and Category = 'Shoes' and Year > '2018' and month = 2	索引将用于获取 country = "US" 和 category = "shoes" 且 year > 2018 的所有分区。然后，将执行月份表达式的筛选。
Country = 'US' AND Category = 'Shoes' OR Year > '2018'	由于表达式中存在 OR 运算符，因此不会使用索引。
Country = 'US' AND Category = 'Shoes' AND (Year = 2017 OR Year = '2018')	索引将用于获取所有 country = "US" 和 category = "shoes" 的分区，然后对年份表达式进行筛选。
Country in ('US', 'UK') AND Category = 'Shoes'	索引不会用于筛选，因为当前不支持 IN 运算符。
Country = 'US' AND Category in ('Shoes', 'Books')	索引将用于获取所有 country = "US" 的分区，然后对 Category 表达式进行筛选。
Country = 'US' AND Category in ('Shoes', 'Books') AND (creationDate > '2023-9-01'	索引将用于获取所有 country = "US" 且 creationDate > '2023-9-01' 的分区，然后对 Category 表达式进行筛选。

与引擎集成

Redshift Spectrum、Amazon EMR 和 AWS Glue ETL Spark DataFrames 能够在 AWS Glue 中的索引处于 ACTIVE 状态后利用索引来获取分区。Athena 和 AWS Glue ETL Dynamic frames 要求您执行额外的步骤以利用索引来改进查询。

启用分区筛选

要在 Athena 中启用分区筛选，您需要按如下方式更新表属性：

1. 在 AWS Glue 控制台中，选择 Data Catalog 下的表。
2. 选择一个 表。
3. 在操作下，选择编辑表。
4. 在表属性下，添加以下内容：
   - 键 – `partition_filtering.enabled`
   - 值 – `true`
5. 选择 应用。

您还可以通过在 Athena 中运行 ALTER TABLE SET PROPERTIES 查询来设置此参数。

```
ALTER TABLE partition_index.table_with_index
SET TBLPROPERTIES ('partition_filtering.enabled' = 'true')
```

与其他 AWS 服务集成

虽然您可以使用 AWS Glue 爬网程序 来填充 AWS Glue Data Catalog，但有几种 AWS 服务可以自动与目录集成并为您填充目录。以下各节提供了有关可填充 Data Catalog 的特定用例（由 AWS 服务提供支持）的更多信息。

主题

- AWS Lake Formation
- Amazon Athena

AWS Lake Formation

AWS Lake Formation 是一项服务，让用户能够在 AWS 中更轻松地设置安全数据湖。Lake Formation 建立在 AWS Glue 之上，而 Lake Formation 与 AWS Glue 共享相同的 AWS Glue Data Catalog。您可以在 Lake Formation 中注册您的 Amazon S3 数据位置，然后使用 Lake Formation 控制台在 AWS Glue Data Catalog 中创建数据库和表、定义数据访问策略，并从一个中央位置审核数据湖中的数据访问。您可以使用 Lake Formation 细粒度访问控制来管理现有的数据目录资源和 Amazon S3 数据位置。

凭借在 Lake Formation 中注册的数据，您可以在 IAM 主体、AWS 账户、AWS 组织和组织单位之间安全地共享 Data Catalog 资源。

有关使用 Lake Formation 创建 Data Catalog 资源的更多信息，请参阅《AWS Lake Formation Developer Guide》中的 Creating Data Catalog tables and databases。

Amazon Athena

Amazon Athena 使用 Data Catalog 在 AWS 账户中存储和检索 Amazon S3 数据的表元数据。通过表元数据，Athena 查询引擎可以了解如何查找、读取和处理您要查询的数据。

您可以直接使用 Athena `CREATE TABLE` 语句填充 AWS Glue Data Catalog。无需运行爬网程序即可在 Data Catalog 中手动定义和填充架构和分区元数据。

1. 在 Athena 控制台中创建一个数据库，将表元数据存储在 Data Catalog 中。
2. 使用 `CREATE EXTERNAL TABLE` 语句定义数据来源的架构。
3. 使用 `PARTITIONED BY` 子句定义任何分区键（前提是您的数据已分区）。
4. 使用 `LOCATION` 子句指定存储实际数据文件的 Amazon S3 路径。
5. 运行 `CREATE TABLE` 语句。

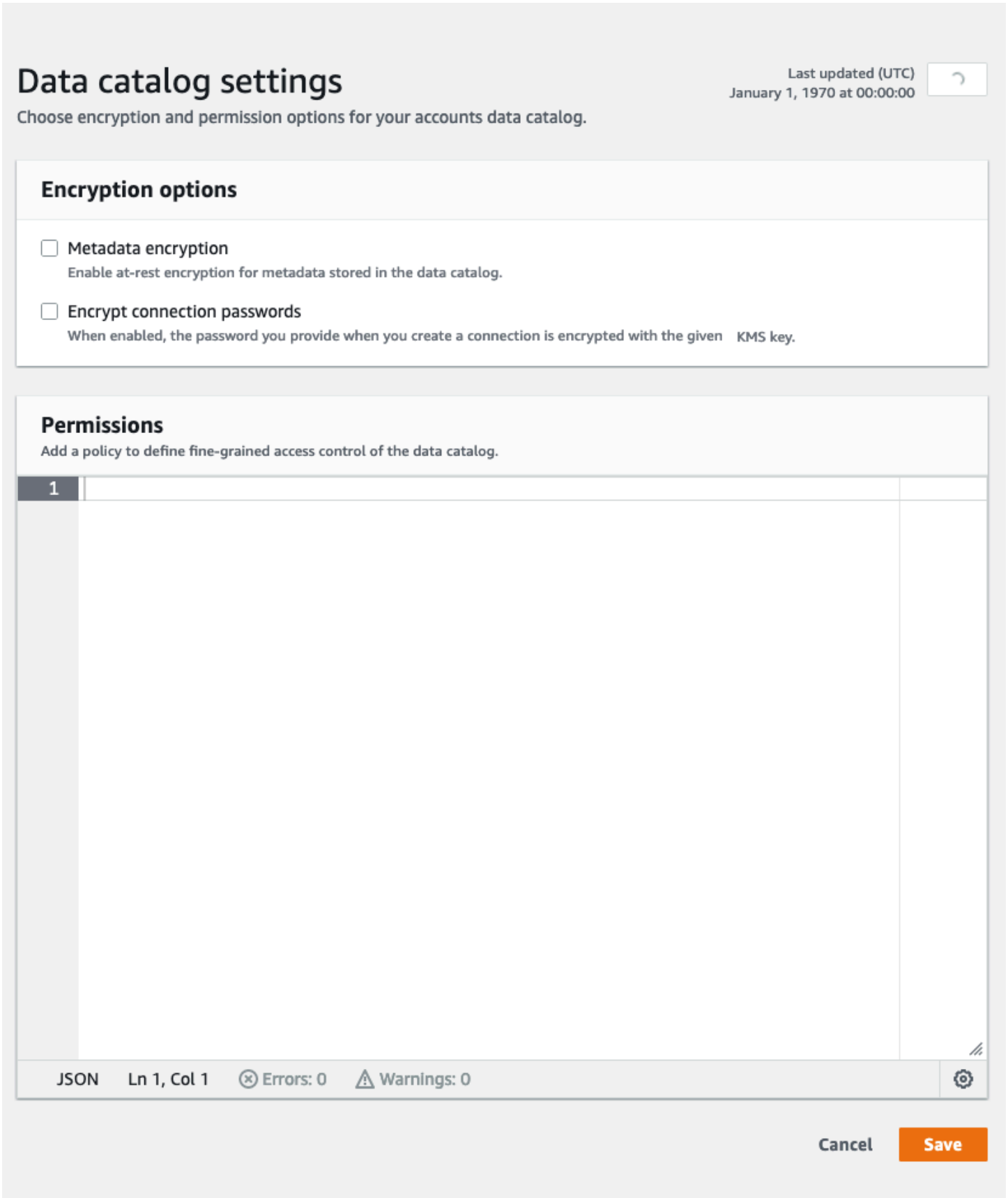
   此查询根据您定义的架构和分区在 Data Catalog 中创建表元数据，而无需实际爬取数据。

您可以在 Athena 中查询表，该表将使用 Data Catalog 中的元数据来访问和查询 Amazon S3 中的数据文件。

有关更多信息，请参阅《Amazon Athena 用户指南》中的创建数据库和表。

Data Catalog 设置

Data Catalog 设置中包含用于为账户中的 Data Catalog 设置加密和权限的选项。


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

更改数据目录的精细访问控制

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 选择加密选项。

   - 元数据加密：选中此复选框可加密 Data Catalog 中的元数据。系统使用您指定的 AWS Key Management Service (AWS KMS) 密钥对元数据进行静态加密。有关更多信息，请参阅 加密数据目录。
   - 加密连接密码：选中此复选框可在创建或更新连接时加密 AWS Glue 连接对象中的密码。系统使用您指定的 AWS KMS 密钥对密码进行加密。返回密码时，会对其进行加密。此选项是数据目录中所有 AWS Glue 连接的全局设置。如果清除此复选框，则先前加密的密码会保持加密状态，而密钥就是在以前创建或更新这些密码时所使用的密钥。有关 AWS Glue 连接的更多信息，请参阅 连接到数据。

     启用此选项后，请选择 AWS KMS 密钥，或者选择输入密钥 ARN 并提供密钥的 Amazon Resource Name（ARN）。输入 `arn:aws:kms:region:account-id:key/key-id` 格式的 ARN。您也可以提供密钥别名形式的 ARN，例如 `arn:aws:kms:region:account-id:alias/alias-name` 。

     > **Important**
     >
     > 如果选择此选项，则创建或更新连接的任何用户或角色必须对指定的 KMS 密钥具有 `kms:Encrypt` 权限。

     有关更多信息，请参阅 加密连接密码。
3. 选择 Settings (设置)，然后在 Permissions (权限) 编辑器中添加策略声明来为您的账户更改数据目录的精细访问控制。一次只能将一个策略附加到数据目录。您可以将 JSON 资源策略粘贴到此控件中。有关更多信息，请参阅 Glue 中 AWS 基于资源的策略。
4. 选择 Save (保存)，以用您所做的任何更改更新您的数据目录。

您还可以使用 AWS Glue API 操作来放置、获取和删除资源策略。有关更多信息，请参阅 AWS Glue 中的安全性 API。

填充和管理事务表

Apache Iceberg、Apache Hudi 和 Linux Foundation Delta Lake 是开源表格式，专为 Apache Spark 中的大规模数据分析和数据湖工作负载而设计。

您可以使用以下方法在 AWS Glue Data Catalog 中填充 Iceberg、Hudi 和 Delta Lake 表：

- AWS Glue 爬网程序 – AWS Glue 爬网程序 可以自动发现 Iceberg、Hudi 和 Delta Lake 表元数据并在 Data Catalog 中填充这些数据。有关更多信息，请参阅 使用爬网程序填充 Data Catalog 。
- AWS Glue ETL 任务 – 您可以创建 ETL 任务，以便将数据写入 Iceberg、Hudi 和 Delta Lake 表，并将其元数据填充到 Data Catalog 中。有关更多信息，请参阅 Using data lake frameworks with AWS Glue ETL jobs。
- AWS Glue 控制台、AWS Lake Formation 控制台、AWS CLI 或 API – 您可以使用 AWS Glue 控制台、Lake Formation 控制台或 API 在 Data Catalog 中创建和管理 Iceberg 表定义。

主题

创建 Apache Iceberg 表

您可以使用驻留在 Amazon S3 中的数据在 AWS Glue Data Catalog 中创建使用 Apache Parquet 数据格式的 Apache Iceberg 表。该数据目录中的表是表示数据存储中数据的元数据定义。默认情况下，AWS Glue 会创建 Iceberg v2 表。有关 v1 和 v2 表之间的区别，请参阅 Apache Iceberg 文档中的格式版本更改。

Apache Iceberg 是适用于超大型分析数据集的开放表格式。Iceberg 允许轻松更改架构，也称为架构发展，这意味着用户可以在不破坏基础数据的情况下添加、重命名或删除数据表中的列。Iceberg 还支持数据版本控制，允许用户跟踪数据随时间的变化。这将启用时间旅行功能，该功能允许用户访问和查询数据的历史版本，并分析更新和删除之间的数据更改。

您可以使用 AWS Glue 或 Lake Formation 控制台或 AWS Glue API 中的 `CreateTable` 操作在 Data Catalog 中创建 Iceberg 表。有关更多信息，请参阅 CreateTable 操作 (Python: create_table)。

在数据目录中创建 Iceberg 表时，您必须在 Amazon S3 中指定表格式和元数据文件路径，以便能够执行读取和写入操作。

当您向 AWS Lake Formation 注册 Amazon S3 数据位置时，您可以使用 Lake Formation 通过精细访问控制权限来保护 Iceberg 表。对于 Amazon S3 中的源数据和未向 Lake Formation 注册的元数据，访问权限由 Amazon S3 和 AWS Glue 操作的 IAM 权限策略决定。有关更多信息，请参阅 Managing permissions。

> **Note**
>
> 数据目录不支持创建分区和添加 Iceberg 表属性。

先决条件

要在数据目录中创建 Iceberg 表并设置 Lake Formation 数据访问权限，您需要完成以下要求：

1. 在没有向 Lake Formation 注册数据的情况下创建 Iceberg 表所需的权限。

   除了在数据目录中创建表所需的权限外，表创建者还需要以下权限：

   - 针对资源 arn:aws:s3:::{bucketName} 的 `s3:PutObject`
   - 针对资源 arn:aws:s3:::{bucketName} 的 `s3:GetObject`
   - 针对资源 arn:aws:s3:::{bucketName} 的 `s3:DeleteObject`
2. 使用向 Lake Formation 注册的数据创建 Iceberg 表所需的权限：

   要使用 Lake Formation 管理和保护数据湖中的数据，请向 Lake Formation 注册包含表数据的 Amazon S3 位置。这样，Lake Formation 就可以向 Athena、Redshift Spectrum 和 Amazon EMR 等 AWS 分析服务提供凭证以访问数据。有关注册 Amazon S3 位置的更多信息，请参阅 Adding an Amazon S3 location to your data lake。

   读取和写入向 Lake Formation 注册的基础数据的主体需要以下权限：

   - `lakeformation:GetDataAccess`
   - `DATA_LOCATION_ACCESS`

     对某个位置具有数据位置权限的主体也对所有子位置具有位置权限。

     有关数据位置权限的更多信息，请参阅 Underlying data access control Ulink。

要启用压缩，该服务需要代入有权更新数据目录中的表的 IAM 角色。有关详细信息，请参阅 表优化的先决条件

创建 Iceberg 表

您可以使用 AWS Glue 或 Lake Formation 控制台或 AWS Command Line Interface 创建 Iceberg v1 和 v2 表，如本页所述。您也可以使用 AWS Glue 爬网程序 创建 Iceberg 表。有关更多信息，请参阅《AWS Glue 开发人员指南》中的数据目录和爬网程序。

创建 Iceberg 表

Console

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在数据目录下，选择表，然后使用创建表按钮指定以下属性：
   - 表名称 – 输入表的唯一名称。如果您使用 Athena 访问表，请使用《Amazon Athena 用户指南》中的这些命名提示。
   - 数据库 – 选择现有数据库或创建新数据库。
   - 描述 – 表的描述。您可以编写描述以帮助您了解表的内容。
   - 表格式 – 对于表格式，请选择 Apache Iceberg。
   - 启用压缩 – 选择启用压缩，将表中较小的 Amazon S3 对象压缩成较大对象。
   - IAM 角色 – 为了运行压缩，该服务会代表您代入一个 IAM 角色。您可以使用下拉列表选择一个 IAM 角色。确保该角色具有启用压缩所需的权限。

     要了解有关所需权限的更多信息，请参阅 表优化的先决条件 。
   - 位置 – 指定 Amazon S3 中存储元数据表的文件夹的路径。Iceberg 需要数据目录中的元数据文件和位置才能执行读取和写入。
   - 架构 – 选择添加列以添加列和列的数据类型。您可以选择创建一个空表，然后稍后更新架构。数据目录支持 Hive 数据类型。有关更多信息，请参阅 Hive 数据类型。

     Iceberg 允许您在创建表后演变架构和分区。您可以使用 Athena 查询更新表架构，使用 Spark 查询更新分区。

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/"
          }
      }'
```

优化 Iceberg 表

使用 Apache Iceberg 等开放表格式的 Amazon S3 数据湖会将数据存储为 Amazon S3 对象。如果数据湖表中包含成千上万个小 Amazon S3 对象，则会增加 Iceberg 表的元数据开销并影响读取性能。为提高 AWS 分析服务（例如 Amazon Athena 和 Amazon EMR）和 AWS Glue ETL 作业的读取性能，AWS Glue Data Catalog 为数据目录中的 Iceberg 表提供了托管式压缩功能（一种将小 Amazon S3 对象压缩成较大对象的进程）。您可以使用 Lake Formation 控制台、AWS Glue 控制台、AWS CLI 或 AWS API 为数据目录中的单个 Iceberg 表启用或禁用压缩。

表优化器会持续监控表分区，并在超过文件数量和文件大小阈值时启动压缩进程。如果 write.target-file-size-bytes 属性中指定的文件大小在 128 MB 到 512 MB 的范围内，则 Iceberg 表符合压缩条件。在 Data Catalog 中，如果表中有超过五个文件，每个文件都小于 write.target-file-size-bytes 属性的 75%，则会开始压缩过程。

例如，在 write.target-file-size-bytes 属性中，表的文件大小阈值设置为 512 MB（在 128 MB 到 512 MB 的规定范围内），该表包含 10 个文件。如果这 10 个文件中有 6 个文件，每个文件均小于 384 MB（0.75*512），则 Data Catalog 会触发压缩。

数据目录会在不干扰并发查询的情况下执行压缩。数据目录仅支持对 Parquet 格式的表进行数据压缩。

有关支持的数据类型、压缩格式和限制，请参阅[托管式数据压缩的支持的格式和限制](#)。

主题

表优化的先决条件

表优化器会代入您在为表启用压缩时指定的 AWS Identity and Access Management (IAM) 角色的权限。该 IAM 角色必须具有读取数据和更新数据目录中元数据的权限。您可以创建一个 IAM 角色并附加以下内联策略：

- 添加以下内联策略，以向 Amazon S3 授予对未注册到 Lake Formation 的数据位置的读/写权限。此策略还包括更新数据目录中表的权限，以及允许 AWS Glue 在 Amazon CloudWatch 日志中添加日志并发布指标的权限。对于 Amazon S3 中未注册到 Lake Formation 的源数据，访问权限由 Amazon S3 和 AWS Glue 操作的 IAM 权限策略决定。

  请将以下内联策略中的 `bucket-name` 替换为您的 Amazon S3 存储桶名称，请将 `aws-account-id` 和 `region` 替换为有效的 AWS 账户和数据目录所在的区域，将 `database_name` 替换为数据库的名称，并将 `table_name` 替换为表的名称。

```
{
    "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:*"
        }
    ]
}
```

- 使用以下策略为注册到 Lake Formation 的数据启用压缩功能。

  如果该压缩角色不具有对表的 IAM_ALLOWED_PRINCIPALS 组权限，则该角色需要具有对该表的 Lake Formation ALTER、DESCRIBE、INSERT 和 DELETE 权限。

  有关向 Lake Formation 注册 Amazon S3 存储桶的更多信息，请参阅 Adding an Amazon S3 location to your data lake。

```
{
 "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:*"
        }
    ]
 }
```

- （可选）如果要压缩的 Iceberg 表包含使用服务器端加密进行加密的 Amazon S3 存储桶中数据，该压缩角色需要具有解密 Amazon S3 对象并生成新数据密钥以将对象写入加密存储桶的权限。将以下策略添加到需要的 AWS KMS 密钥。我们仅支持在存储桶级加密。

```
{
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::<aws-account-id>:role/<compaction-role-name>"
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*"
}
```

- （可选）对于注册到 Lake Formation 的数据位置，用于注册该位置的角色需要具有解密 Amazon S3 对象并生成新数据密钥以将对象写入加密存储桶的权限。有关更多信息，请参阅 Registering an encrypted Amazon S3 location。
- （可选）如果 AWS KMS 密钥存储在其他 AWS 账户中，则需要为该压缩角色添加以下权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": ["arn:aws:kms:<REGION>:<KEY_OWNER_ACCOUNT_ID>:key/<KEY_ID>" ]
        }
    ]
}
```

- 用于运行压缩的角色必须拥有该角色的 `iam:PassRole` 权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
```

```
            ],
            "Resource": [
                "arn:aws:iam::<account-id>:role/<compaction-role-name>"
            ]
        }
    ]
}
```

- 将以下信任策略添加到该角色，以便 AWS Glue 服务代入该 IAM 角色来运行压缩进程。

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

启用压缩

您可以使用 Lake Formation 控制台、AWS Glue 控制台、AWS CLI 或 AWS API 为数据目录中的 Apache Iceberg 表启用压缩。对于新表，您可以在创建表时选择 Apache Iceberg 表格式并启用压缩。新表会默认禁用压缩。

Console

启用压缩

1. 打开 AWS Glue 控制台（https://console.aws.amazon.com/glue/），然后以数据湖管理员、表创建者或已被授予表的 `glue:UpdateTable` 和 `lakeformation:GetDataAccess` 权限的用户身份登录。
2. 在导航窗格的数据目录下，请选择表。
3. 在表页面上，选择要启用压缩的开放表格式的表，然后在操作菜单下，选择启用压缩。

4. 您也可以通过选中该表并打开表详细信息页面来启用压缩。选择页面下半部的表优化选项卡，然后选择启用压缩。
5. 然后从下拉列表中选择一个具有 表优化的先决条件 部分所示权限的现有 IAM 角色。

   选择创建新的 IAM 角色选项后，服务会创建一个具有运行压缩所需权限的自定义角色。

   按照以下步骤更新一个现有的 IAM 角色：

   a. 要更新 IAM 角色的权限策略，请在 IAM 控制台中转到用于运行压缩的 IAM 角色。
   b. 在“添加权限”部分中，选择“创建策略”。在新打开的浏览器窗口中，创建将用于您的角色的新策略。
   c. 在“创建策略”页面上，选择 JSON 选项卡。将“先决条件”中显示的 JSON 代码复制到策略编辑器字段中。

AWS CLI

以下示例演示如何启用压缩。将账户 ID 替换为有效的 AWS 账户 ID。将数据库名称和表名称替换为实际的 Iceberg 表名称和数据库名称。将 `roleArn` 替换为 IAM 角色的 AWS 资源名称 (ARN) 以及具有运行压缩所需权限的 IAM 角色的名称。

```
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

调用 `CreateTableOptimizer` 操作为表启用压缩。

启用压缩后，表优化选项卡会显示以下压缩详细信息（大约 15–20 分钟后）：

开始时间

在 Lake Formation 中启动压缩进程的时间。该值是一个采用 UTC 时间格式的时间戳。

结束时间

数据目录中压缩进程结束的时间。该值是一个采用 UTC 时间格式的时间戳。

Status

压缩运行的状态。值为成功或失败。

已压缩的文件数

已压缩的文件总数。

已压缩的字节数

已压缩的字节总数。

禁用压缩

您可以使用 AWS Glue 控制台或 AWS CLI 来为特定 Apache Iceberg 表禁用自动压缩。

Console

1. 选择数据目录，然后选择表。从表列表中，选择要禁用压缩的开放表格式的表。
2. 您可以选择一个 Iceberg 表，然后在操作下选择禁用压缩。

   您也可以选择表详细信息页面下半部分的禁用压缩，从而为表禁用压缩。
3. 在确认消息页面选择禁用压缩。您可以在以后重新启用压缩。

   确认后，压缩将被禁用，并且表的压缩状态将恢复为 `Off`。

AWS CLI

将以下示例中的账户 ID 替换为有效的 AWS 账户 ID。将数据库名称和表名称替换为实际的 Iceberg 表名称和数据库名称。将 `roleArn` 替换为 IAM 角色的 AWS 资源名称 (ARN) 以及具有运行压缩所需权限的 IAM 角色的实际名称。

```
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

调用 UpdateTableOptimizer 操作以为特定的表禁用压缩。

查看压缩详细信息

您可以使用 Lake Formation 控制台、AWS CLI 或使用 AWS API 操作查看 Apache Iceberg 的压缩状态。

Console

查看 Iceberg 表的压缩状态（控制台）

- 您可以通过在 Lake Formation 控制台中选择数据目录下的表来查看 Iceberg 表的压缩状态。压缩状态字段将显示压缩运行的状态。您可以使用表首选项来显示表的格式和压缩状态。
- 要查看特定表的压缩运行历史记录，请选择 AWS Glue Data Catalog 下的表，然后选择一个表来查看该表的详细信息。表优化选项卡将显示表的压缩历史记录。

AWS CLI

您可以使用 AWS CLI 查看压缩详细信息。

请将以下示例中的账户 ID 替换为有效的 AWS 账户 ID，将数据库名称和表名称替换为实际的 Iceberg 表名称。

- 获取表的上次压缩详细信息

```
aws get-table-optimizer \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
  --type compaction
```

- 使用以下示例来检索特定表的优化器历史记录。

```
aws list-table-optimizer-runs \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
```

```
  --table-name iceberg_table \
  --type compaction
```

- 以下示例演示了如何检索多个优化器的压缩运行和配置详细信息。您最多可以指定 20 个优化器。

```
aws glue batch-get-table-optimizer \
--entries '[{"catalogId":"123456789012", "databaseName":"iceberg_db",
 "tableName":"iceberg_table", "type":"compaction"}]'
```

AWS API

- 使用 GetTableOptimizer 操作检索优化器的上次运行详细信息。
- 使用 ListTableOptimizerRuns 操作检索特定表上给定优化器的历史记录。您可以在单个 API 调用中指定 20 个优化器。
- 使用 BatchGetTableOptimizer 操作检索您账户中多个优化器的配置详细信息。此操作不支持跨账户调用。

查看 Amazon CloudWatch 指标

成功运行压缩后，服务会创建有关压缩作业性能的 Amazon CloudWatch 指标。您可以前往 CloudWatch 指标并选择指标、所有指标。您可以按特定的命名空间（例如 AWS Glue）、表名称或数据库名称筛选指标。

有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的查看可用的指标。

- 已压缩的字节数
- 已压缩的文件数
- 分配给作业的 DPU 数
- 作业持续时间（小时）

删除优化器

您可以使用 AWS CLI 或 AWS API 操作来删除表的优化器和关联的元数据。

运行以下 AWS CLI 命令删除表的压缩历史记录。

```
aws glue delete-table-optimizer \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
  --type compaction
```

使用 DeleteTableOptimizer 操作删除表的优化器。

托管式数据压缩的支持的格式和限制

为了提高诸如Amazon Athena、Amazon EMR和ETL AWS Glue Data Catalog 任务之类的 AWS 分析服务的读取性能 AWS Glue ，为数据目录中的冰山表提供了托管压缩（一种将小型Amazon S3对象压缩成较大对象的过程）。

数据压缩支持多种用于读取和写入数据的压缩格式，例如从加密表中读取数据。

数据压缩支持：

- 文件类型 – Parquet
- 数据类型 – 布尔值、整型、长整型、浮点数、双精度、字符串、十进制、日期、时间、时间戳、字符串、UUID、二进制
- 压缩 – zstd、gzip、snappy、未压缩
- 加密 – 数据压缩仅支持默认的 Amazon S3 加密（SSE-S3）和服务器端 KMS 加密（SSE-KMS）。
- 资源装箱压缩
- 架构演变
- 具有目标文件大小的表（写入。 target-file-size-bytes 冰山配置中的属性）在 128MB 到 512 MB 的包含范围内。
- 区域
  - Asia Pacific (Tokyo)
  - 亚太地区（首尔）
  - 亚太地区（孟买）
  - 亚太地区（新加坡）
  - 欧洲地区（爱尔兰）
  - 欧洲地区（伦敦）
  - 欧洲地区（法兰克福）
  - 美国东部（弗吉尼亚州北部）

  - 美国东部（俄亥俄州）
  - 美国西部（加利福尼亚北部）
  - 南美洲（圣保罗）
- 当存储基础数据的 Amazon S3 存储桶位于另一个账户中时，您可以从数据目录所在的账户运行压缩。要实现此目的，压缩角色需要具有访问 Amazon S3 存储桶的权限。

数据压缩目前不支持：

- 文件类型 – Avro、ORC
- 数据类型 – 固定
- 压缩 – brotli、lz4
- 随分区规格发展压缩文件。
- 常规排序或 Z-Order 排序
- 合并或删除文件 – 压缩进程会跳过拥有与之关联的删除文件的数据文件。
- 对跨账户表进行压缩 – 您无法对跨账户表进行压缩。
- 对跨区域表进行压缩 – 您无法对跨区域表进行压缩。
- 针对资源链接启用压缩
- Amazon S3 存储桶的 VPC 端点
- DynamoDB 锁定管理器 — 使用数据压缩时，不应将其他数据加载任务用作 org.apache.iceberg.aws.dy namodb。`lock-impl` DynamoDbLockManager。

管理 Data Catalog

AWS Glue Data Catalog 是一个中央元数据存储库，用于存储 Amazon S3 数据集的结构和操作元数据。高效管理 Data Catalog 对于维护数据质量、性能、安全性和治理至关重要。

通过了解和应用这些 Data Catalog 管理实践，您可以确保随着数据环境的发展，您的元数据仍能保持准确、高性能、安全且治理良好。

本节介绍 Data Catalog 管理的以下方面：

- 更新表架构和分区随着数据的发展，您可能需要更新 Data Catalog 中定义的表架构或分区结构。有关如何使用 AWS Glue ETL 以编程方式进行这些更新的更多信息，请参阅使用 AWS Glue ETL 任务在 Data Catalog 中更新架构并添加新分区。

- 管理列统计数据：准确的列统计数据有助于优化查询计划并提高性能。有关如何生成、更新和管理列统计数据的更多信息，请参阅使用列统计数据优化查询性能。
- 加密 Data Catalog 要保护敏感元数据，可以使用 AWS Key Management Service（AWS KMS）加密 Data Catalog。本节介绍如何启用和管理 Data Catalog 的加密。
- 使用 AWS Lake Formation 保护 Data Catalog Lake Formation 提供了一种全面的数据湖安全和访问控制方法。您可以使用 Lake Formation 来保护和治理对 Data Catalog 和底层数据的访问。

主题

使用 AWS Glue ETL 任务在 Data Catalog 中更新架构并添加新分区

您的提取、转换和加载 (ETL) 作业可能会在目标数据存储中创建新的表分区。随着时间推移，您的数据集架构可能演变和偏离 AWS Glue 数据目录架构。AWS GlueETL 任务现在提供了几个功能，您可以在 ETL 脚本中使用这些功能在数据目录中更新架构和分区。这些功能允许您在数据目录中查看 ETL 工作的结果，而无需重新运行爬网程序。

新分区

如果要在 AWS Glue Data Catalog 中查看新分区，可以执行以下操作之一：

- 在作业完成后，重新运行爬网程序，并在爬网程序完成后，在控制台上查看新分区。
- 一旦作业完成，即可在控制台上查看新分区，而无需重新运行爬网程序。可以通过向 ETL 脚本添加几行代码来启用此功能，如以下示例中所示。代码使用 `enableUpdateCatalog` 参数指示数据目录在任务运行期间将随着新分区的创建而更新。

方法 1：

在选项参数中传递 `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：

在 getSink() 中传递 enableUpdateCatalog 和 partitionKeys，并对 DataSink 对象调用 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)
```

现在，您可以使用 AWS Glue ETL 任务本身创建新的目录表、使用修改的架构更新现有表，并在数据目录中添加新的表分区，而无需重新运行爬网程序。

更新表架构

如果要覆盖数据目录表的架构，可以执行以下操作之一：

- 作业完成后，重新运行爬网程序，并确保您的爬网程序也已配置为更新表定义。当爬网程序完成时，在查控制台上看新分区以及任何架构更新。有关更多信息，请参阅使用 API 配置爬网程序。
- 一旦作业完成，即可在控制台上查看修改的架构，而无需重新运行爬网程序。可以通过向 ETL 脚本添加几行代码来启用此功能，如以下示例中所示。代码将 `enableUpdateCatalog` 设置为 true，也将 `updateBehavior` 设置为 `UPDATE_IN_DATABASE`，这表示在任务运行期间将覆盖架构并在数据目录中添加新分区。

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)
```

如果您希望防止表架构被覆盖，但仍希望添加新分区，也可以将该 updateBehavior 值设置为 LOG。updateBehavior 的默认值为 UPDATE_IN_DATABASE，因此，如果您没有明确定义它，则表架构将被覆盖。

如果 enableUpdateCatalog 未设置为 true，无论为 updateBehavior 选择哪个选项，ETL 任务都不会更新数据目录中的表。

创建新表

还可以使用相同的选项在数据目录中创建新表。您可以使用 setCatalogInfo 指定数据库和新表名。

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

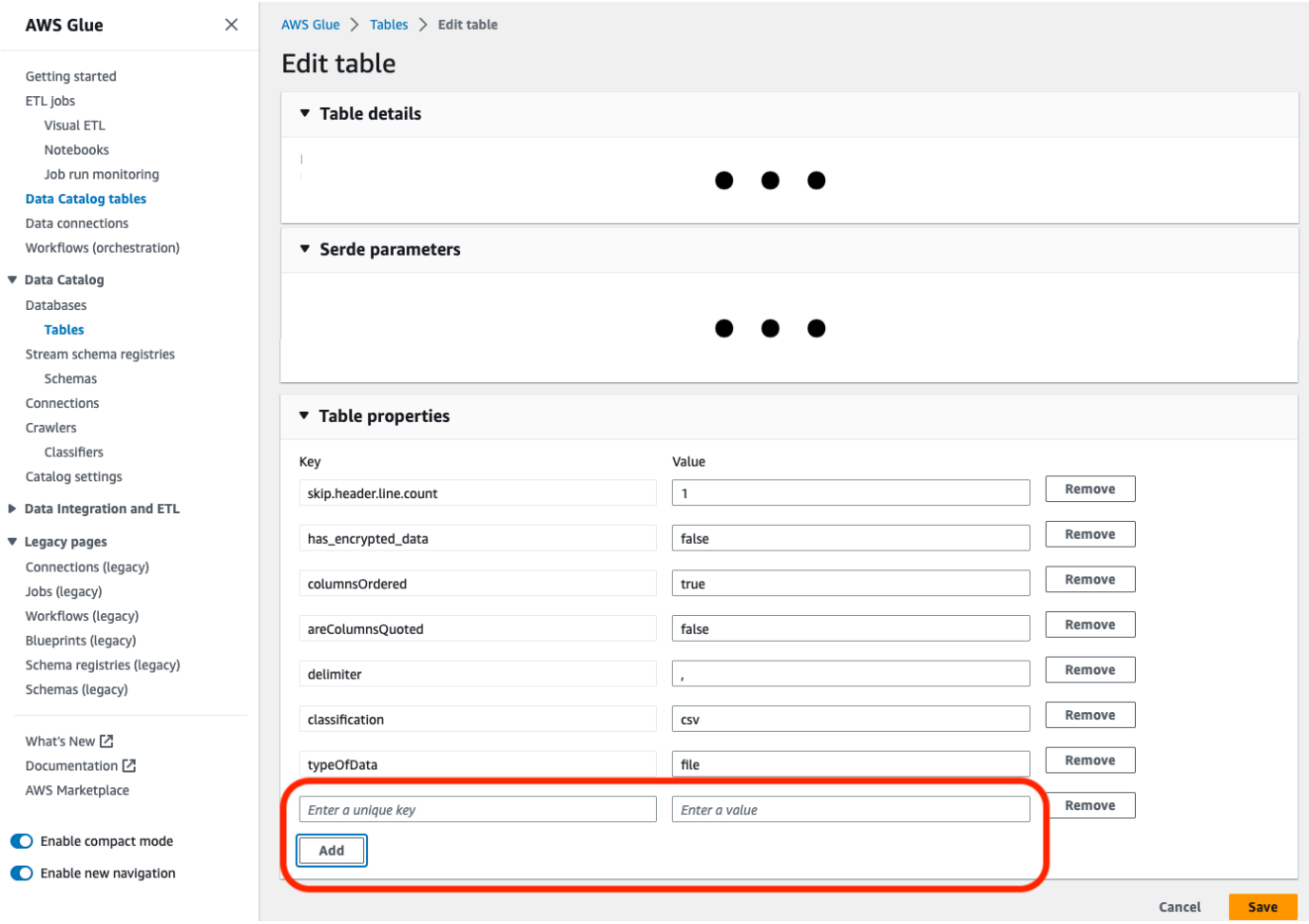
```
val options = JsonOptions(Map(
    "path" -> outputPath,
    "partitionKeys" -> Seq("<partition_1>", "<partition_2>"),
    "enableUpdateCatalog" -> true,
    "updateBehavior" -> "UPDATE_IN_DATABASE"))
val sink = glueContext.getSink(connectionType = "s3", connectionOptions =
 options).withFormat("<format>")
sink.setCatalogInfo(catalogDatabase = “<dst_db_name>”, catalogTableName =
 “<dst_tbl_name>”)
sink.writeDynamicFrame(df)
```

限制

请注意以下限制：

- 仅支持 Amazon Simple Storage Service（Amazon S3）目标。

- 受管辖的表不支持 enableUpdateCatalog 功能。
- 仅支持以下格式：json、csv、avro 和 parquet。
- 要使用 parquet 分类创建或更新表，您必须使用 DynamicFrames 针对 AWS Glue 优化的 Parquet 写入器。这可以通过以下一种方法实现：
  - 如果您要使用 parquet 分类更新目录中的现有表，则必须先将表的 "useGlueParquetWriter" 表属性设置为 true，然后才能对其进行更新。您可以通过 AWS Glue API/SDK、控制台或 Athena DDL 语句设置此属性。



设置目录表属性后，您可以使用以下代码片段使用新数据更新目录表：

```
glueContext.write_dynamic_frame.from_catalog(
    frame=frameToWrite,
    database="dbName",
    table_name="tableName",
    additional_options={
        "enableUpdateCatalog": True,
        "updateBehavior": "UPDATE_IN_DATABASE"
```

```
    }
)
```

- 如果目录中尚不存在该表，则可以在脚本中使用 `getSink()` 方法与 `connection_type="s3"` 将表及其分区添加到目录中，同时将数据写入 Amazon S3。为您的工作流程提供适当的 `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` 格式值是启用 AWS Glue Parquet 写入器的传统方法。
- 当 `updateBehavior` 设置为 `LOG` 时，只有当 `DynamicFrame` 架构等效于或包含在数据目录表的架构中定义的列子集时，才会添加新分区。
- 非分区表不支持架构更新（未使用“partitionKeys”选项）。
- 在 ETL 脚本中传递的参数与数据目录表架构中的 partitionKey 之间，您的 partitionKeys 必须是等效的且顺序相同。
- 此功能目前尚不支持更新/创建嵌套更新架构的表（例如，结构内的数组）。

有关更多信息，请参阅 the section called “AWS Glue for Spark”。

在 ETL 作业中使用 MongoDB 连接

您可以为 MongoDB 创建连接，然后在 AWS Glue 任务中使用该连接。有关更多信息，请参阅 AWS Glue 编程指南中的 the section called “MongoDB 连接”。连接 `url`、`username` 和 `password` 存储在 MongoDB 连接中。其他选项可以在 ETL 任务脚本中使用`additionalOptions` 参数 `glueContext.getCatalogSource` 指定。其他选项可包括：

- `database`：（必需）要从中读取数据的 MongoDB 数据库。
- `collection`：（必需）要从中读取数据的 MongoDB 集合。

通过将 `database` 和 `collection` 信息放在 ETL 任务脚本中，您可以在多个任务中使用相同的连接。

1. 为 MongoDB 数据源创建一个 AWS Glue Data Catalog 连接。请参阅“connectionType”：“mongodb”以获取连接参数的说明。您可以使用控制台、API 或 CLI 创建此连接。
2. 在 AWS Glue Data Catalog 中创建数据库以存储 MongoDB 数据的表定义。请参阅创建数据库了解更多信息。
3. 创建一个爬网程序，使用连接到 MongoDB 的连接中的信息对 MongoDB 中的数据进行爬网。该爬网程序将在 AWS Glue Data Catalog 中创建表，这些表描述您在任务中使用的 MongoDB 数据库中的表。请参阅使用爬网程序填充 Data Catalog 了解更多信息。
4. 使用自定义脚本创建任务。您可以使用控制台、API 或 CLI 创建此任务。有关更多信息，请参阅在 AWS Glue 中添加任务。
5. 为任务选择数据目标。表示数据目标的表可以在数据目录中定义，或者您的任务可以在运行时创建目标表。在编写作业时，您会选择目标位置。如果目标需要连接，连接也会在您的作业中被引用。如果您的作业需要多个数据目标，您可以稍后通过编辑脚本来添加这些数据目标。
6. 通过为任务和生成的脚本提供参数，自定义任务处理环境。

   以下是一个基于数据目录中定义的表结构，从 MongoDB 数据库创建 `DynamicFrame` 的示例。该代码使用 `additionalOptions` 来提供其他数据源信息：

   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. 按需运行任务或通过触发器运行任务。

使用列统计数据优化查询性能

无需设置其他数据管道，即可为 Parquet、ORC、JSON、ION、CSV 和 XML 等数据格式的 AWS Glue Data Catalog 表计算列级别的统计数据。借助列统计数据，您可以深入洞察列中的值，从而了解数据特征。Data Catalog 支持为列值生成统计数据，例如最小值、最大值、空值总计、非重复值总计、值的平均长度和真实值的总出现次数等。

Amazon Redshift 和 Amazon Athena 等 AWS 分析服务可以使用这些列统计数据来生成查询执行计划，并选择可提高查询性能的最优计划。

您可以使用 AWS Glue 控制台或 AWS CLI 来配置运行列统计数据生成任务。启动该进程时，AWS Glue 将在后台启动一个 Spark 作业并更新 Data Catalog 中的 AWS Glue 表元数据。您可以使用 AWS Glue 控制台、AWS CLI 或通过调用 GetColumnStatisticsForTable API 操作来查看列统计数据。

> **Note**
>
> 如果您使用 Lake Formation 权限来控制对表的访问权限，则列统计数据任务代入的角色需要拥有完全的表访问权限才能生成统计数据。

主题

生成列统计数据的先决条件

要生成或更新列统计数据，统计数据生成任务将代表您代入一个 AWS Identity and Access Management（IAM）角色。根据向该角色授予的权限，列统计数据生成任务可以读取 Amazon S3 数据存储中的数据。

> **Note**
>
> 要为由 Lake Formation 管理的表生成统计数据，用于生成统计数据的 IAM 角色需要具有完全的表访问权限。

要使用基于角色的访问控制，您必须使用以下策略中列出的权限创建一个 IAM 角色，并将该角色添加到列统计数据生成任务。

创建 IAM 角色以生成列统计数据

1. 要创建 IAM 角色，请参阅 为 AWS Glue 创建 IAM 角色。
2. 要更新现有的角色，进入 IAM 控制台后，请转到生成列统计数据进程正在使用的 IAM 角色。
3. 在添加权限选项卡中，选择附加策略。在新打开的浏览器窗口中，选择 `AWSGlueServiceRole` AWS 托管式策略。
4. 您还需要包含从 Amazon S3 数据位置读取数据所需的权限。

   在添加权限部分中，选择创建策略。在新打开的浏览器窗口中，创建将用于您的角色的新策略。
5. 在创建策略页面中，选择 JSON 选项卡。将以下 `JSON` 代码复制到策略编辑器字段中。

   > **Note**
   >
   > 请将以下策略中的 account ID 替换为有效的 AWS 账户，将 `region` 替换为表所在的区域，并将 `bucket-name` 替换为 Amazon S3 存储桶的名称。

```
{
    "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. （可选）如果您使用 Lake Formation 权限来提供对数据的访问权限，则该 IAM 角色需要具有 `lakeformation:GetDataAccess` 权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
            "Sid": "LakeFormationDataAccess",
            "Effect": "Allow",
            "Action": "lakeformation:GetDataAccess",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

如果该 Amazon S3 数据位置已注册到 Lake Formation，并且列统计数据生成任务所代入的 IAM 角色不具有对表的 `IAM_ALLOWED_PRINCIPALS` 组权限，则该角色需要具有对该表的 Lake Formation `ALTER` 和 `DESCRIBE` 权限。用于注册 Amazon S3 存储桶的角色需要具有对该表的 Lake Formation `INSERT` 和 `DELETE` 权限。

如果该 Amazon S3 数据位置尚未注册到 Lake Formation，并且该 IAM 角色不具有对表的 `IAM_ALLOWED_PRINCIPALS` 组权限，则该角色需要具有对该表的 Lake Formation `ALTER`、`DESCRIBE`、`INSERT` 和 `DELETE` 权限。

7. （可选）对于写入加密 Amazon CloudWatch Logs 的列统计数据生成任务，密钥政策中需要包含以下权限。

```
{
```

```
    "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. 用于运行列统计信息的角色必须拥有该角色的 iam:PassRole 权限。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "iam:PassRole"
```

```
        ],
        "Resource": [
            "arn:aws:iam::111122223333:role/<columnstats-role-name>"
        ]
    }]
}
```

9. 在您创建 IAM 角色以生成列统计数据时，该角色还必须具有以下的信任策略，以确保服务能够代入该角色。

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

生成列统计数据

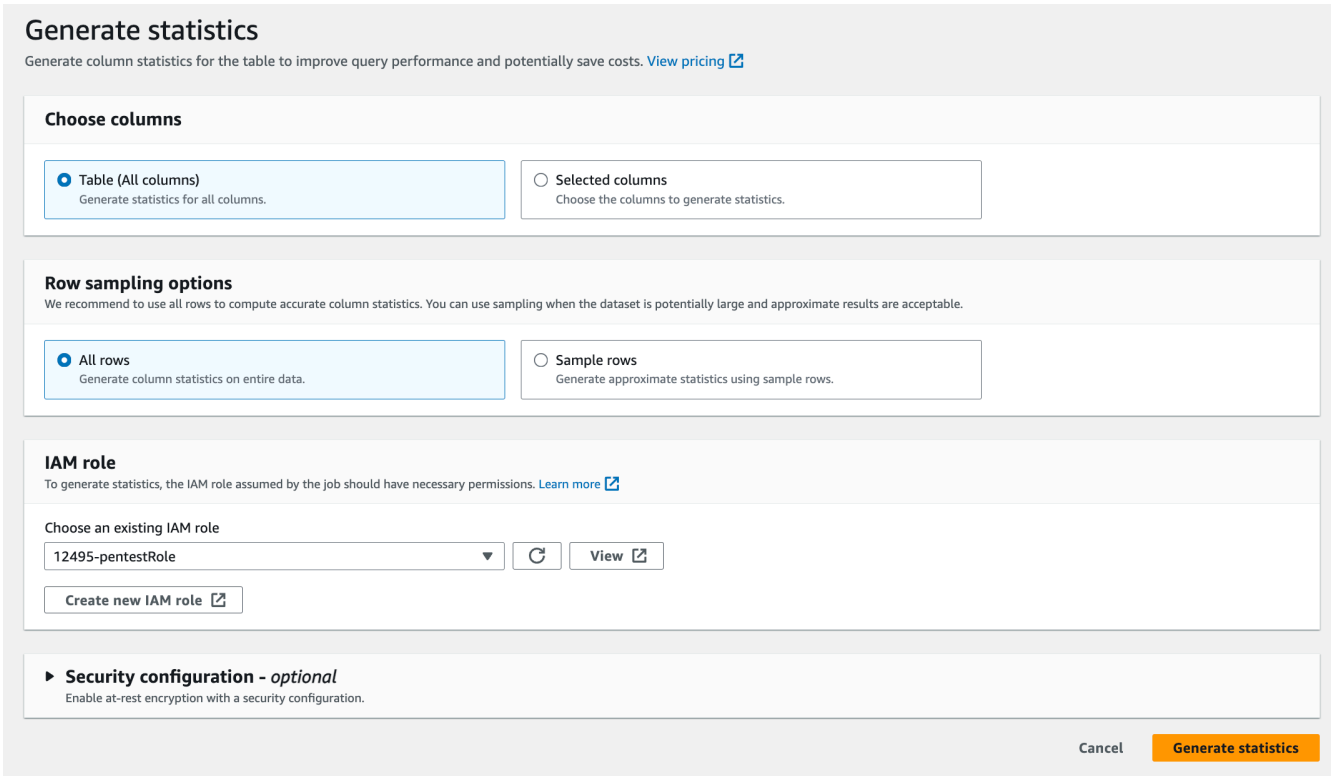
按照以下步骤，使用 AWS Glue 控制台或 AWS CLI 在 Data Catalog 中生成统计数据。

Console

使用控制台生成列统计数据

1. 通过以下网址登录到 AWS Glue 控制台：https://console.aws.amazon.com/glue/。
2. 选择 Data Catalog 表。
3. 从列表中选择一个表。
4. 在操作菜单下选择生成统计数据。

   您也可以选择表页面下半部分的列统计数据选项卡，然后选择生成统计数据按钮。
5. 在生成统计数据页面上，请指定以下选项：



- 表（所有列）– 选择此选项可生成表中所有列的统计数据。
- 选定列 – 选择此选项可生成特定列的统计数据。您可以从下拉列表中选择列。
- 所有行 – 选择表中的所有行以生成准确的统计数据。
- 样本行 – 仅从表中选择特定百分比的行来生成统计数据。默认值为所有行。使用向上和向下箭头可增加或减少百分比值。

> **Note**
>
> 我们建议选择表中的所有行，以计算出准确的统计数据。仅在可接受近似值时，才使用样本行来生成列统计数据。

6. （可选）然后选择一种安全配置来启用日志静态加密。
7. 选择生成统计数据以运行该进程。

AWS CLI

在以下示例中，请将 `DatabaseName`、`TableName` 和 `ColumnNameList` 的值替换为实际的数据库名、表名和列名。请将账户 ID 替换为有效的 AWS 账户，将角色名称替换为您用于生成统计数据的 IAM 角色名称。

```
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
}
```

您还可以通过调用 StartColumnStatisticsTaskRun 操作来生成列统计数据。

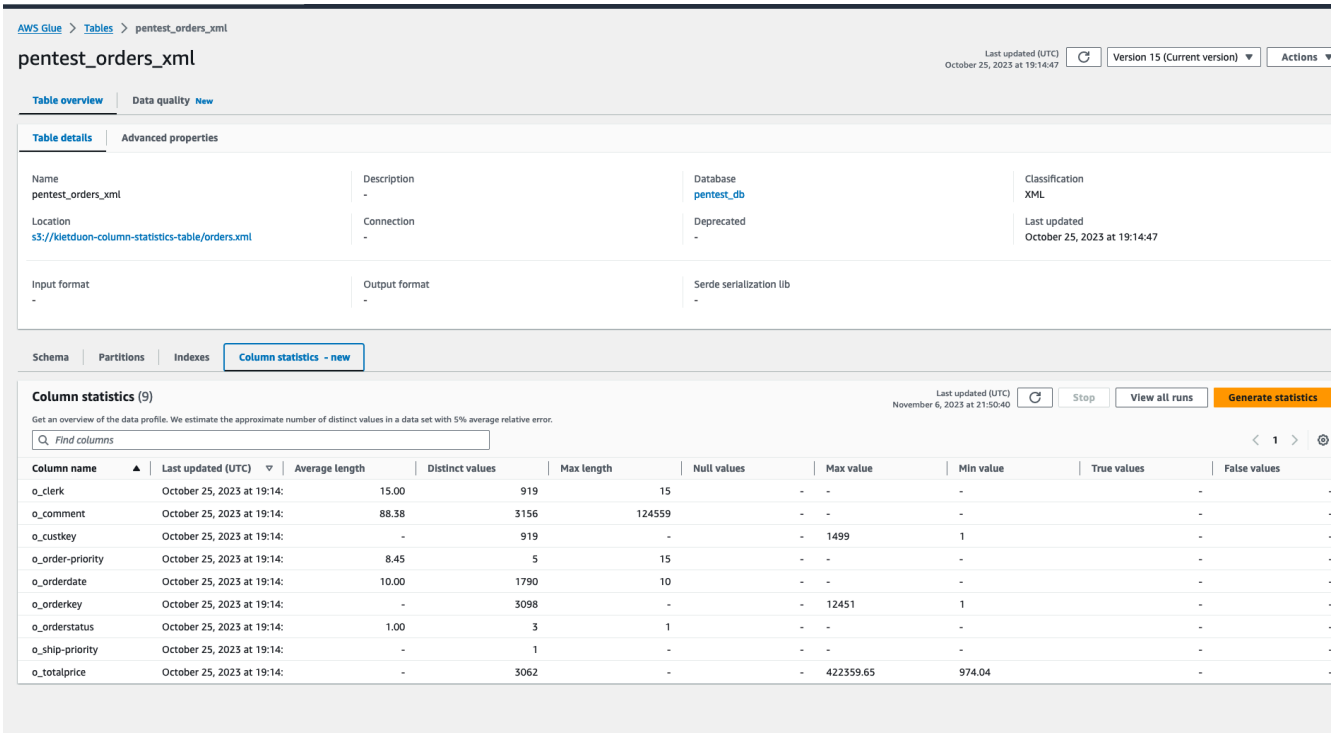
查看列统计数据

成功生成统计数据后，Data Catalog 会存储这一信息，以便 Amazon Athena 和 Amazon Redshift 中基于成本的优化器在运行查询时做出最佳选择。统计数据因列的类型而异。

AWS Management Console

查看表的列统计数据

- 运行列统计数据任务后，表详细信息页面上的列统计数据选项卡将显示表的统计数据。



将会提供以下统计数据：

- 列名：用于生成统计数据的列名
- 上次更新时间：生成统计数据的日期和时间
- 平均长度：列中值的平均长度
- 不重复值数：列中不重复的值总数。我们估算列中不重复的值的数量，相对误差为 5%。
- 最大值：列中最大的值。
- 最小值：列中最小的值。
- 最大长度：列中最大值的长度。
- 空值数：列中空值的总数。
- True 值数：列中 True 值的总数。
- False 值数：列中 False 值的总数。

AWS CLI

以下示例演示了如何使用 AWS CLI 检索列统计数据。

```
aws glue get-column-statistics-for-table \
    --database-name <test_db> \
```

```
    --table-name <test_tble> \
    --column-names <col1>
```

您还可以使用 GetColumnStatisticsForTable API 操作来查看列统计数据。

更新列统计数据

通过允许查询计划程序选择最佳计划，使统计数据保持最新可提高查询性能。您需要从 AWS Glue 控制台显式运行生成统计数据任务才能刷新列统计数据。Data Catalog 不会自动刷新统计数据。

如果您未使用控制台的使用 AWS Glue 统计数据生成功能，则可以使用 UpdateColumnStatisticsForTable API 操作或 AWS CLI 手动更新列统计数据。以下示例演示了如何使用 AWS CLI 更新列统计数据。

```
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
                }
            }
        }
    ]
}
```

删除列统计数据

您可以使用 DeleteColumnStatisticsForTable API 操作或 AWS CLI 来删除列统计数据。以下示例演示了如何使用 AWS Command Line Interface（AWS CLI）删除列统计数据。

```
aws glue delete-column-statistics-for-table \
    --database-name test_db \
    --table-name test_table \
    --column-name col1
```

查看列统计数据任务运行

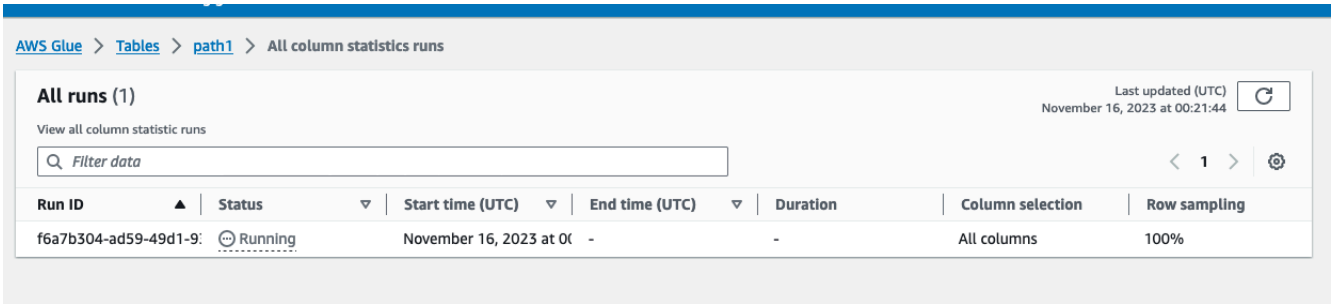
运行列统计任务后，您可以使用 AWS Glue 控制台、AWS CLI 或 GetColumnStatisticsTaskRuns 操作来浏览表的任务运行详细信息。

Console

查看列统计数据任务运行的详细信息

1. 在 AWS Glue 控制台上，选择 Data Catalog 下的表。
2. 选择包含列统计数据的表。
3. 在表详细信息页面上，选择列统计数据。
4. 选择查看运行。

   您可以看到与指定表关联的所有运行的信息。



AWS CLI

在以下示例中，请将 DatabaseName 和 TableName 的值替换为实际的数据库名和表名。

```
aws glue get-column-statistics-task-runs --input-cli-json file://input.json
{
    "DatabaseName": "<test-db>",
    "TableName": "<test-table>"
}
```

停止列统计数据任务运行

您可以使用 AWS Glue 控制台、AWS CLI 或 StopColumnStatisticsTaskRun 操作来停止某个表的列统计任务运行。

Console

停止列统计数据任务运行

1. 在 AWS Glue 控制台上，选择 Data Catalog 下的表。
2. 选择正在进行列统计数据任务运行的表。
3. 在表详细信息页面上，选择列统计数据。
4. 选择停止。

如果您在运行完成之前停止该任务，则不会生成该表的列统计数据。

AWS CLI

在以下示例中，请将 `DatabaseName` 和 `TableName` 的值替换为实际的数据库名和表名。

```
aws glue stop-column-statistics-task-run --input-cli-json file://input.json
{
    "DatabaseName": "<test-db>",
    "TableName": "<test-table>"
}
```

注意事项和限制

生成列统计数据时应注意以下因素和限制。

注意事项

- 使用采样方法生成统计数据可以减少运行时间，但生成的统计数据可能不准确。
- 每次列统计数据运行都需要处理整个数据集。
- Data Catalog 不会存储不同版本的统计数据。
- 每个表一次只能运行一个统计数据生成任务。

- 如果使用注册到 Data Catalog 的客户 AWS KMS 密钥对表进行加密，则 AWS Glue 将使用相同的密钥来加密统计数据。

在满足下列条件中的一个时，列统计数据任务才能生成统计数据：

- 该 IAM 角色拥有完整的表权限（IAM 或 Lake Formation）。
- 该 IAM 角色使用 Lake Formation 混合访问模式取得了对表的权限。

对于下列情况，列统计数据任务不支持生成统计数据：

- 表启用了基于 Lake Formation 单元格的访问控制。
- 事务处理数据湖 – Linux 基金会 Delta Lake、Apache Iceberg、Apache Hudi。
- 联合身份验证数据库中的表 – Hive 元数据存储、Amazon Redshift 数据共享
- 嵌套列、数组和结构数据类型。
- 其他账户共享给您的表。

加密数据目录

您可以使用由 AWS Key Management Service（AWS KMS）管理的加密密钥保护存储在 AWS Glue Data Catalog 中的静态元数据。您可以使用 Data Catalog 设置为新的 Data Catalog 启用 Data Catalog 加密。您可以根据需要启用或禁用现有 Data Catalog 的加密。启用后，AWS Glue 会加密写入目录的所有新元数据，而现有元数据将保持未加密状态。

有关加密 Data Catalog 的详细信息，请参阅加密数据目录。

使用 Lake Formation 保护您的 Data Catalog

AWS Lake Formation 是一项服务，让用户能够在 AWS 中更轻松地设置安全数据湖。它通过定义精细的访问控制权限，为创建和安全管理数据湖提供了一个中央位置。Lake Formation 使用 Data Catalog 来存储和检索有关数据湖的元数据，例如表定义、架构信息和数据访问控制设置。

您可以向 Lake Formation 注册元数据表或数据库的 Amazon S3 数据位置，并使用它来定义 Data Catalog 资源的元数据级别权限。您还可以使用 Lake Formation 代表集成分析引擎管理 Amazon S3 上存储的底层数据的存储访问权限。

有关更多信息，请参阅 What is AWS Lake Formation?。

访问 Data Catalog

您可以使用 AWS Glue Data Catalog 来发现和理解您的数据。Data Catalog 提供了一种一致的方法来维护架构定义、数据类型、位置和其他元数据。您可以使用如下方法访问 Data Catalog：

- AWS Glue 控制台 – 您可以通过 AWS Glue 控制台（基于 Web 的用户界面）访问和管理 Data Catalog。控制台允许您浏览和搜索数据库、表及其关联的元数据，以及创建、更新和删除元数据定义。
- AWS Glue 爬网程序 – 爬网程序是自动扫描您的数据来源并以元数据填充 Data Catalog 的程序。您可以创建和运行爬网程序来发现来自各种来源的数据并对其进行编目，例如 Amazon S3、Amazon RDS、Amazon DynamoDB、Amazon CloudWatch、兼容 JDBC 的关系数据库（例如 MySQL 和 PostgreSQL）以及一些非 AWS 的来源，例如 Snowflake 和 Google BigQuery。
- AWS Glue API – 您可以使用 AWS Glue API 以编程方式访问 Data Catalog。这些 API 允许您以编程方式与 Data Catalog 进行交互，从而实现自动化并与其他应用程序和服务集成。
- AWS Command Line Interface（AWS CLI）– 您可以使用 AWS CLI 从命令行访问和管理 Data Catalog。CLI 提供了用于创建、更新和删除元数据定义以及查询和检索元数据信息的命令。
- 与其他 AWS 服务集成 – Data Catalog 与其他各种 AWS 服务集成，使您能够访问和使用存储在目录中的元数据。例如，您可以使用 Data Catalog 中的元数据，利用 Amazon Athena 查询数据来源，并使用 AWS Lake Formation 来管理 Data Catalog 资源的数据访问和治理。

AWS Glue Data Catalog 最佳实践

本节介绍高效管理和利用 AWS Glue Data Catalog 的最佳实践。它强调了高效使用爬网程序、元数据组织、安全性、性能优化、自动化、数据治理以及与其他 AWS 服务的集成等实践。

- 高效使用爬网程序 – 定期运行爬网程序，以便 Data Catalog 与数据来源中的最新更改保持同步。对频繁更改的数据来源使用增量爬取以提高性能。将爬网程序配置为在检测到更改时自动添加新分区或更新架构。
- 组织和命名元数据表 – 为 Data Catalog 中的数据库和表建立一致的命名约定。将相关数据来源分组到逻辑数据库或文件夹中，以更好地进行组织。使用描述性名称来传达每个表格的目的和内容。
- 高效管理架构 – 利用 AWS Glue 爬网程序的架构推断功能。请先查看并更新架构更改再进行应用，以免破坏下游应用程序。使用架构发展功能来正常处理架构更改。
- 保护 Data Catalog – 为 Data Catalog 启用静态和传输中的数据加密。实施精细访问控制策略，以限制对敏感数据的访问。定期审核和审查 Data Catalog 权限和活动日志。

- 与其他 AWS 服务集成 Data Catalog 使用 Data Catalog 作为 Amazon Athena、Redshift Spectrum 和 AWS Lake Formation 等服务的集中化元数据层。利用 AWS Glue ETL 任务转换数据并将数据加载到各种数据存储中，同时在 Data Catalog 中维护元数据。
- 监控和优化性能Data Catalog 使用 Amazon CloudWatch 指标监控爬网程序和 ETL 任务的性能。对 Data Catalog 中的大型数据集进行分区以提高查询性能。对频繁访问的元数据实施性能优化。
- 随时了解 AWS Glue 文档和最佳实践的最新信息Data Catalog 会定期查看 AWS Glue 文档和 AWS Glue 资源，了解最新更新、最佳实践和建议。参加 AWS Glue 网络研讨会、讲习会和其他活动，向专家学习，随时了解新特性和功能。

AWS Glue 架构注册表

> **Note**
>
> AWS Glue 控制台中的以下区域不支持 AWS Glue 架构注册表：亚太地区（雅加达）和中东（阿联酋）。

AWS Glue 架构注册表是一项新功能，允许您集中发现、控制和演变数据流架构。架构定义了数据记录的结构和格式。借助 AWS Glue 架构注册表，您可以使用与 Apache Kafka、Amazon Managed Streaming for Apache Kafka、Amazon Kinesis Data Streams、适用于 Apache Flink 的亚马逊托管服务和 AWS Lambda 的方便集成，在数据流应用程序上管理和实施架构。

AWS Glue 架构注册表支持 AVRO (v1.10.2) 数据格式、采用适用于架构（规范 Draft-04、Draft-06 和 Draft-07）且已使用 Everit 库 验证的 JSON 架构的 JSON 架构格式的 JSON 数据格式、协议缓冲区 (Protobuf) 版本 proto2 和 proto3（不支持 `extensions` 或 `groups`）和 Java 语言支持以及其他即将推出的数据格式和语言。支持的功能包括兼容性、通过元数据获取架构、架构自动注册、IAM 兼容性，以及可选的用以减少存储的 ZLIB 压缩和数据传输。AWS Glue架构注册表无服务器，可以自由使用。

将架构用作创建者和使用者之间的数据格式合同，这样可以改进数据治理，提高数据质量，并使数据使用者能够适应兼容的上游更改。

架构注册表允许不同的系统共享序列化和反序列化的架构。例如，假设您拥有数据创建者和使用者。创建者在发布数据时知道架构。架构注册表为某些系统（如 Amazon MSK 或 Apache Kafka）提供序列化程序和反序列化程序。

有关更多信息，请参阅 架构注册表的工作原理。

主题

架构

架构定义了数据记录的结构和格式。架构是用于可靠数据发布、使用或存储的版本化规范。

在 Avro 的此示例架构中，格式和结构由布局和字段名称定义，字段名称的格式由数据类型（例如，`string`、`int`）定义。

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

在此示例 JSON 架构 Draft-07 中，格式由 JSON 架构组织定义。

```
{
 "$id": "https://example.com/person.schema.json",
 "$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
  }
 }
}
```

在此示例中，Protobuf 的格式由协议缓冲区语言版本 2（proto2）定义。

```
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;
}
```

注册表

注册表是架构的逻辑容器。注册表允许您组织架构以及管理应用程序的访问控制。注册表具有 Amazon Resource Name（ARN），允许您组织和设置注册表中架构操作的不同访问权限。

您可以根据需要使用默认注册表或创建任意数量的新注册表。

AWS Glue 架构注册表层次结构

- RegistryName: [string]
  - RegistryArn: [AWS ARN]
  - CreatedTime: [timestamp]
  - UpdatedTime: [timestamp]
- SchemaName: [string]

- SchemaArn: [AWS ARN]
- DataFormat：[Avro、Json 或 Protobuf]
- Compatibility: [eg。BACKWARD, BACKWARD_ALL, FORWARD, FORWARD_ALL, FULL, FULL_ALL, NONE, DISABLED]
- Status: [eg。PENDING, AVAILABLE, DELETING]
- SchemaCheckpoint: [integer]
- CreatedTime: [timestamp]
- UpdatedTime: [timestamp]

- SchemaVersion: [string]
  - SchemaVersionNumber: [integer]
  - Status: [eg。PENDING, AVAILABLE, DELETING, FAILURE]
  - SchemaDefinition: [string, Value: JSON]
  - CreatedTime: [timestamp]

- SchemaVersionMetadata: [list]
  - MetadataKey: [string]
  - MetadataInfo
  - MetadataValue: [string]
  - CreatedTime: [timestamp]

架构版本控制和兼容性

每个架构都可以有多个版本。版本控制由应用于架构的兼容性规则控制。在新架构版本的注册请求成功之前，架构注册表将根据此规则检查请求。

标记为检查点的架构版本用于确定注册架构新版本的兼容性。首次创建架构时，默认检查点将是第一个版本。随着架构发展为更多版本，您可以使用 CLI/SDK 将检查点更改为架构版本，使用遵守一组约束条件的 `UpdateSchema` API。在控制台中，编辑架构定义或兼容性模式会默认将检查点更改为最新版本。

兼容性模式允许您控制架构随时间发展或不能发展的方式。这些模式构成了生成和使用数据的应用程序之间的契约。将架构的新版本提交到注册表时，应用于

架构名称的兼容性规则将用于确定是否可以接受新版本。有 8 种兼容性模式：
NONE、DISABLED、BACKWARD、BACKWARD_ALL、FORWARD、FORWARD_ALL、FULL、FULL_ALL

在 Avro 数据格式中，字段可能是可选字段或必填字段。可选字段是指 Type 包含 null 的字段。必填字段不会将 null 视为 Type。

在 Protobuf 数据格式中，字段可以是可选字段（包括重复字段），也可以在 proto2 语法中为必填字段，而在 proto3 语法中，所有字段均为可选字段（包括重复字段）。所有兼容性规则均基于对协议缓冲区规范以及来自 Google 协议缓冲区文档指南的了解。

- NONE：不适用兼容模式。您可以在开发场景或者在不知道要应用于架构的兼容性模式时使用此选项。无需进行兼容性检查，接受添加的任何新版本。
- DISABLED：此兼容性选项可防止对特定架构进行版本控制。无法添加新版本。
- BACKWARD：建议使用此兼容性选项，因为它允许使用者读取架构的最新版本和上一个版本。当您需要删除字段或添加可选字段时，您可以使用此选项检查所有先前架构版本的兼容性。BACKWD 的典型使用案例是针对最近的架构创建应用程序时。

  AVRO

  例如，假定您有一个由名字（必填）、姓氏（必填）、电子邮件地址（必填）和电话号码（可选）定义的架构。

  如果您的下个架构版本删除了必填的电子邮件地址字段，这意味着注册成功。BACKWARD 兼容性要求使用者能够读取当前和以前的架构版本。您的使用者将能够读取新架构，因为旧邮件中的额外电子邮件地址字段将被忽略。

  如果您有建议的新架构版本添加了必填字段（例如邮政编码），这将无法成功注册 BACKWARD 兼容性。新版本的使用者将无法在架构更改之前读取旧邮件，因为他们缺少必需的邮政编码字段。但是，如果在新架构中将邮政编码字段设置为可选，则建议的版本将成功注册，因为使用者可以在没有可选邮政编码字段的情况下读取旧架构。

  JSON

  例如，假定您具有由名字（可选）、姓氏（可选）、电子邮件地址（可选）和电话号码（可选）定义的架构版本。

  如果您的下一个架构版本添加了可选的电话号码属性，则只要原始架构版本将 `additionalProperties` 字段设置为 False 以禁止任何附加属性，就能成功注册。BACKWARD 兼容性要求使用者能够读取当前和以前的架构版本。您的使用者将能够读取不存在电话号码属性的原始架构生成的数据。

如果您有建议的新架构版本添加了可选的电话号码属性，则当原始架构版本将 `additionalProperties` 字段设置为 True，即允许任何附加属性时，这样将无法成功注册 BACKWARD 兼容性。新版本的使用者无法在架构更改之前读取旧邮件，因为他们无法读取具有不同类型的电话号码属性的数据，例如字符串而不是数字。

PROTOBUF

例如，假设您有一个由 Message（邮件）`Person` 定义的架构版本，其中 proto2 语法下包含 `first name`（必填字段）、`last name`（必填字段）、`email`（必填字段）和 `phone number`（可选字段）。

与 AVRO 使用场景相似，如果您的下个架构版本删除了必填的 `email` 字段，这意味着注册成功。BACKWARD 兼容性要求使用者能够读取当前和以前的架构版本。您的使用者将能够读取新架构，因为旧邮件中的额外 `email` 字段将被忽略。

如果您有建议的新架构版本添加了必填字段（例如 `zip code`），这将无法成功注册 BACKWARD 兼容性。新版本的使用者将无法在架构更改之前读取旧邮件，因为他们缺少必需的 `zip code` 字段。但是，如果在新架构中将 `zip code` 字段设置为可选，则建议的版本将成功注册，因为使用者可以在没有可选 `zip code` 字段的情况下读取旧架构。

对于 gRPC 使用案例，添加新的 RPC 服务或 RPC 方法是一种向后兼容更改。例如，假设您拥有由 RPC 服务 `MyService`（包含两种 RPC 方法 `Foo` 和 `Bar`）定义的架构版本。

如果您的下一个架构版本添加了名为 `Baz` 的新 RPC 方法，则这会成功注册。您的使用者将能够读取由原始架构根据 BACKWARD 兼容性生成的数据，因为新添加的 RPC 方法 `Baz` 是可选的。

如果您有建议的新架构版本删除了现有的 PC 方法 `Foo`，这将无法成功注册 BACKWARD 兼容性。新版本的使用者无法读取架构更改之前的旧邮件，因为他们无法通过 gRPC 应用程序中不存在的 RPC 方法 `Foo` 了解和读取数据。

- BACKWARD_ALL：此兼容性选项允许使用者读取架构的最新版本和所有先前版本。当您需要删除字段或添加可选字段时，您可以使用此选项检查所有先前架构版本的兼容性。
- FORWARD：此兼容性选项允许使用者读取当前和下一个架构版本，但不一定是更高版本。当您添加字段或删除可选字段时，您可以使用此选项检查上一个架构版本的兼容性。FORDE 的一个典型使用案例是，您的应用程序是为之前的架构创建的，并且应该能够处理最新架构。

AVRO

例如，假定您有一个由名字（必填）、姓氏（必填）、电子邮件地址（可选）定义的架构。

如果您有新的架构版本添加了必填字段（例如电话号码），这将成功注册。FORWARD 兼容性要求使用者能够使用先前版本读取新架构生成的数据。

如果您有建议的架构版本删除了必填名字字段，这将无法成功注册 FORWARD 兼容性。先前版本的使用者将无法读取建议的架构，因为他们缺少必填的名称字段。但是，如果名字字段最初是可选的，则建议的新架构将成功注册，因为使用者可以根据没有可选名字字段的新架构读取数据。

JSON

例如，假定您具有由名字（可选）、姓氏（可选）、电子邮件地址（可选）和电话号码（可选）定义的架构版本。

如果您的新架构版本删除了可选的电话号码属性，则只要新架构版本将 `additionalProperties` 字段设置为 False，不允许任何附加属性，则这样就会成功注册。FORWARD 兼容性要求使用者能够使用先前版本读取新架构生成的数据。

如果您有建议的架构版本删除了可选的电话号码属性，则当新架构版本将 `additionalProperties` 字段设置为 True，即允许任何附加属性时，这样将无法成功注册 FORWARD 兼容性。先前版本的使用者无法读取建议的架构，因为他们可能有不同类型的电话号码属性，例如字符串而不是数字。

PROTOBUF

例如，假设您有一个由 Message（邮件）`Person` 定义的架构版本，其中 proto2 语法下包含 `first name`（必填字段）、`last name`（必填字段）和 `email`（可选字段）。

与 AVRO 使用场景相似，如果您有新的架构版本添加了必填字段（例如 `phone number`），这将成功注册。FORWARD 兼容性要求使用者能够使用先前版本读取新架构生成的数据。

如果您有建议的架构版本删除了必填 `first name` 字段，这将无法成功注册 FORWARD 兼容性。先前版本的使用者将无法读取建议的架构，因为他们缺少必填的 `first name` 字段。但是，如果 `first name` 字段最初是可选的，则建议的新架构将成功注册，因为使用者可以根据没有可选 `first name` 字段的新架构读取数据。

对于 gRPC 使用案例，删除 RPC 服务或 RPC 方法是一种向前兼容更改。例如，假设您拥有由 RPC 服务 `MyService`（包含两种 RPC 方法 `Foo` 和 `Bar`）定义的架构版本。

如果您的下一个架构版本删除了名为 `Foo` 的现有 RPC 方法，这将根据 FORWARD 兼容性成功注册，因为使用者可以使用先前版本读取新架构生成的数据。如果您有建议的架构版本添加了 RPC 方

法 Baz，这将无法成功注册 FORWARD 兼容性。先前版本的使用者将无法读取建议的架构，因为他们缺少 RPC 方法 Baz。

- FORWARD_ALL：此兼容性选项允许使用者读取任何新注册架构的创建者写入的数据。当您需要添加字段或删除可选字段以及检查所有先前架构版本的兼容性时，您可以使用此选项。
- FULL：此兼容性选项允许使用者读取创建者使用先前版本或下一版本的架构写入的数据，但不是早期版本或更高版本的创建器写入的数据。当您添加或删除可选字段时，您可以使用此选项检查上一个架构版本的兼容性。
- FULL_ALL：此兼容性选项允许创建者读取使用所有架构先前版本的创建器写入的数据。当您添加或删除可选字段时，您可以使用此选项检查所有先前架构版本的兼容性。

开源 Serde 库

AWS 提供开源 Serde 库，作为序列化和反序列化数据的框架。这些库的开源设计允许通用的开源应用程序和框架，以在其项目中支持这些库。

有关 Serde 库工作原理的更多详细信息，请参阅架构注册表的工作原理。

架构注册表的配额

配额，也称为 AWS 中的限制，是 AWS 账户中资源、操作和项目的最大值。以下是 AWS Glue 中架构注册表的软限制。

架构版本的元数据键值对数

每个 AWS 区域的每个 SchemaVersion 最多可以有 10 个键值对。

您可以使用 QuerySchemaVersionMetadata 操作（Python：查询架构版本元数据）或者 PutSchemaVersionMetadata 操作（Python：put_schema_version_metadata）API 查看或设置键值元数据对。

以下是 AWS Glue 中架构注册表的硬限制。

注册表

此账户在每个 AWS 区域最多可以有 100 个注册表。

SchemaVersion

此账户在每个 AWS 区域最多可以有 1 万个架构版本。

每个新架构都会创建一个新的架构版本，因此假设每个架构只有一个版本，则理论上每个区域每个账户最多可有 1 万个架构。

架构负载

架构负载的大小限制为 170KB。

架构注册表的工作原理

本部分介绍了架构注册表中序列化和反序列化过程的工作原理。

1. 注册架构：如果注册表中尚不存在架构，则可以使用与目标名称类似的架构名称（例如，test_topic、test_stream、prod_firehose）注册架构，或者创建者可以为架构提供自定义名称。创建者还可以在创建架构时将键值对（例如 source: msk_kafka_topic_A）作为元数据添加到架构，或者将 AWS 标签添加到架构。注册架构后，架构注册表会将架构版本 ID 返回到序列化程序。如果架构存在，但序列化程序使用的是不存在的新版本，则架构注册表将检查架构引用兼容性规则，以确保兼容新版本，然后再将其注册为新版本。

   有两种架构注册方法：手动注册和自动注册。您可以通过 AWS Glue 控制台或 CLI/SDK 手动注册架构。

   在序列化程序设置中打开自动注册时，将执行架构自动注册。如果创建者配置中未提供 `REGISTRY_NAME`，则自动注册将在默认注册表（default-registry）下注册新架构版本。请参阅安装 SerDe 库，了解有关指定自动注册属性的信息。
2. 序列化程序根据架构验证数据记录：当生成数据的应用程序注册了其架构时，架构注册表序列化程序将验证应用程序生成的记录是否使用与已注册架构匹配的字段和数据类型进行结构化。如果记录架构与注册架构不匹配，则序列化程序将返回异常，并且应用程序将无法将记录传递到目标。

   如果不存在架构，并且架构名称不是通过创建者配置提供，则将使用与主题名称（如果是 Apache Kafka 或 Amazon MSK）或流名称（如果是 Kinesis Data Streams）类似的名称创建架构。

   每条记录都有架构定义和数据。根据架构注册表中的现有架构和版本查询架构定义。

   默认情况下，创建者缓存架构定义和已注册架构的架构版本 ID。如果记录的架构版本定义与缓存中的可用内容不匹配，则创建者将尝试使用架构注册表验证架构。如果架构版本有效，则其版本 ID 和定义将在创建器上本地缓存。

   您可以在安装 SerDe 库的步骤 3 中的可选创建器属性内调整默认缓存周期（24 小时）。

3. 序列化和传递记录：如果记录符合架构，则序列化程序将使用架构版本 ID 装饰每条记录，根据选定的数据格式（AVRO、JSON、Protobuf，或即将推出的其他格式）序列化记录，压缩记录（可选的创建器配置），并将其传送到目标。
4. 使用者反序列化数据：读取此数据的使用者使用架构注册表反序列化程序库，该库从记录负载解析架构版本 ID。
5. 反序列化程序可能会从模式注册表请求架构：如果这是反序列化程序第一次看到具有特定架构版本 ID 的记录，则使用架构版本 ID，反序列化程序将从架构注册表请求架构并在使用器上本地缓存架构。如果架构注册表无法反序列化记录，则使用者可以录入记录中的数据，然后继续或停止应用程序。
6. 反序列化程序使用架构反序列化记录：当反序列化程序从架构注册表中检索架构版本 ID 时，反序列化程序将解压缩记录（如果创建器发送的记录已压缩），并使用架构反序列化记录。现在，应用程序处理记录。

> **Note**
>
> 加密：您的客户端通过 API 调用与架构注册表通信，这些调用使用 HTTPS 上的 TLS 加密对传输中的数据进行加密。存储于架构注册表中的架构始终使用服务托管 AWS Key Management Service（AWS KMS）密钥。

> **Note**
>
> 用户授权：架构注册表支持基于身份的 IAM policy。

架构注册表入门

以下部分概述和演示了如何设置和使用架构注册表。有关架构注册表概念和组件的信息，请参阅AWS Glue 架构注册表。

主题

- 安装 SerDe 库
- 将 AWS CLI 用于 AWS Glue 架构注册表 API
- 创建注册表
- 处理 JSON 的特定记录（JAVO POJO）

- 创建架构
- 更新架构或注册表
- 删除架构或注册表
- 用于序列化程序的 IAM 示例
- 用于反序列化程序的 IAM 示例
- 使用 AWS PrivateLink 的私有连接
- 访问 Amazon CloudWatch 指标
- 架构注册表的示例 AWS CloudFormation 模板

安装 SerDe 库

> **Note**
>
> 先决条件：完成以下步骤之前，您需要先运行 Amazon Managed Streaming for Apache Kafka（Amazon MSK）或 Apache Kafka 集群。您的创建器和使用器需要在 Java 8 或更高版本上运行。

Serde 库提供用于序列化和反序列化数据的框架。

您将为生成数据的应用程序（统称为“序列化程序”）安装开源序列化程序。序列化程序处理序列化、压缩以及与架构注册表的交互。序列化程序自动从正在写入架构注册表兼容目标（如 Amazon MSK）的记录中提取架构。同样，您将在使用数据的应用程序上安装开源反序列化程序。

在创建器和使用器上安装库：

1. 在创建器和使用器的 pom.xml 文件中，通过下面的代码添加此依赖关系：

```
<dependency>
    <groupId>software.amazon.glue</groupId>
    <artifactId>schema-registry-serde</artifactId>
    <version>1.1.5</version>
</dependency>
```

   或者，您可以克隆 AWS Glue 架构注册表。

2. 使用这些必填属性设置您的创建器：

```
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"
```

如果没有现有架构，则需要打开自动注册（下一步）。如果您确实有要应用的架构，则将“my-schema”替换为您的架构名称。此外，如果架构自动注册处于关闭状态，则必须提供“registry-name”。如果架构在“default-registry”下创建，则可以省略注册表名称。

3. （可选）设置这些可选的创建器属性。有关详细属性说明，请参阅自述文件。

```
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
```

自动注册会在默认注册表（“default-registry”）下注册架构版本。如果上一步未指定 SCHEMA_NAME，则主题名称被推断为 SCHEMA_NAME。

有关兼容性模式的更多信息，请参阅架构版本控制和兼容性。

4. 使用这些必填属性设置您的使用器：

```
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. （可选）设置这些可选的使用器属性。有关详细属性说明，请参阅自述文件。

```
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
```

将 AWS CLI 用于 AWS Glue 架构注册表 API

要将 AWS CLI 用于 AWS Glue 架构注册表 API，请确保将 AWS CLI 更新到最新版本。

创建注册表

您可以使用默认注册表，也可以使用 AWS Glue API 或 AWS Glue 控制台创建任意数量的新注册表。

AWS Glue API

您可以按照以下步骤使用 AWS Glue API 执行此任务。

要添加新注册表，请使用 CreateRegistry 操作（Python：创建注册表）API。将 `RegistryName` 指定为要创建的注册表的名称，其最大长度为 255 个字符，且只能包含字母、数字、连字符、下划线、美元符号或哈希标记。

将 `Description` 指定为字符串，其长度不超过 2048 个字节，与 URI 地址多行字符串模式匹配。

（可选）为注册表指定一个或多个 `Tags`，作为键值对的映射数组。

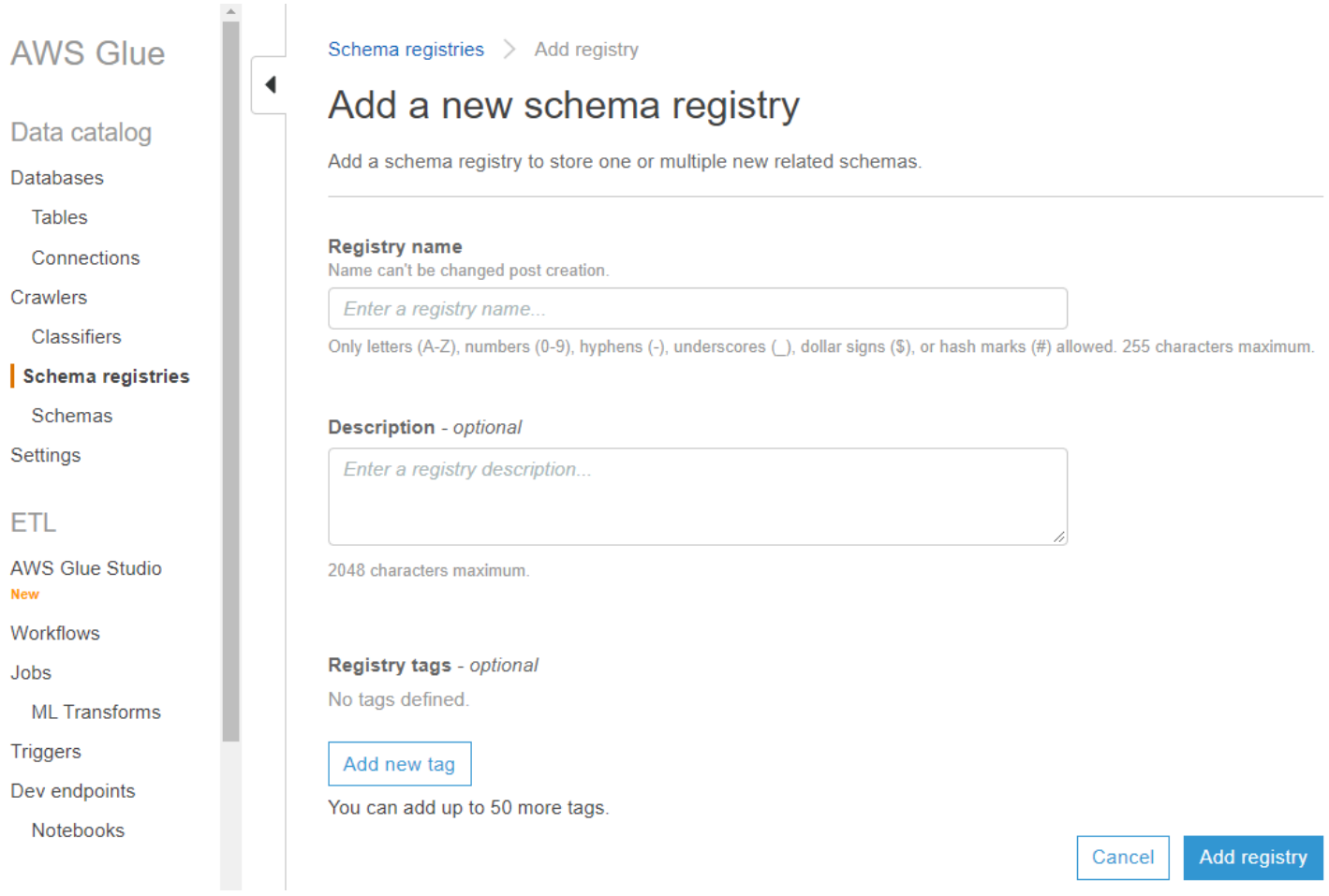
```
aws glue create-registry --registry-name registryName1 --description description
```

您的注册表创建后，会被分配一个 Amazon Resource Name（ARN），您可以在API 响应的 `RegistryArn` 中查看。现在，您已创建注册表，请为该注册表创建一个或多个架构。

AWS Glue 控制台

在 AWS Glue 控制台中添加新注册表：

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格中的 Data catalog (数据目录) 下面，选择 Schema registries (架构注册表)。
3. 选择 Add registry (添加注册表)。
4. 为注册表输入 Registry name (注册表名称)，包含字母、数字、连字符或下划线。此名称不能更改。
5. 为注册表输入 Description (说明)（可选）。
6. 或者，将一个或多个标签应用到您的注册表。选择 Add new tag (添加新标签)，指定 Tag key (标签键) 和 Tag value (标签值)（可选）。
7. 选择 Add registry (添加注册表)。



创建注册表后，系统会为其分配一个 Amazon Resource Name（ARN），您可以从 Schema registries (架构注册表) 列表中选择注册表进行查看。现在，您已创建注册表，请为该注册表创建一个或多个架构。

处理 JSON 的特定记录（JAVO POJO）

您可以使用普通旧 Java 对象（POJO）并将该对象作为记录传递。这类似于 AVRO 中特定记录的概念。mbknor-jackson-jsonschema 可以为传递的 POJO 生成 JSON 架构。此库还可以在 JSON 架构中注入其他信息。

AWS Glue 架构注册表库使用架构中注入的“className”字段提供完全分类的类名称。反序列化程序使用“className”字段，反序列化为该类的对象。

```
 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() {}
}
```

创建架构

您可以使用 AWS Glue API 或 AWS Glue 控制台创建架构。

AWS Glue API

您可以按照以下步骤使用 AWS Glue API 执行此任务。

要添加新架构，请使用 CreateSchema 操作（Python：创建架构）API。

指定 `RegistryId` 结构，指示架构的注册表。或者，忽略 `RegistryId`，使用默认注册表。

指定 `SchemaName`（包含字母、数字、连字符或下划线）和 `DataFormat` 为 **`AVRO`** 或者 **`JSON`**。`DataFormat` 一旦在架构上设置，便不可更改。

指定 `Compatibility` 模式：

- 向后兼容（推荐）– 使用者可以读取当前版本和先前版本。
- 向后兼容全部 – 使用者可以读取当前版本和所有先前版本。
- 向前兼容 – 使用者可以读取当前版本和后续版本。
- 向前兼容全部 – 使用者可以读取当前版本和所有后续版本。

- 完整兼容 – 向后兼容和向前兼容的组合。
- 完整兼容全部 – 向后兼容全部和向前兼容全部的组合。
- 无 – 不执行兼容性检查。
- 已禁用 – 防止对此架构进行任何版本控制。

（可选）为您的架构指定 Tags。

指定 SchemaDefinition，以 Avro、JSON 或 Protobuf 数据格式定义架构。请参阅示例。

对于 Avro 数据格式：

```
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\"} ]}"
```

对于 JSON 数据格式：

```
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\"}}}"
```

对于 Protobuf 数据格式：

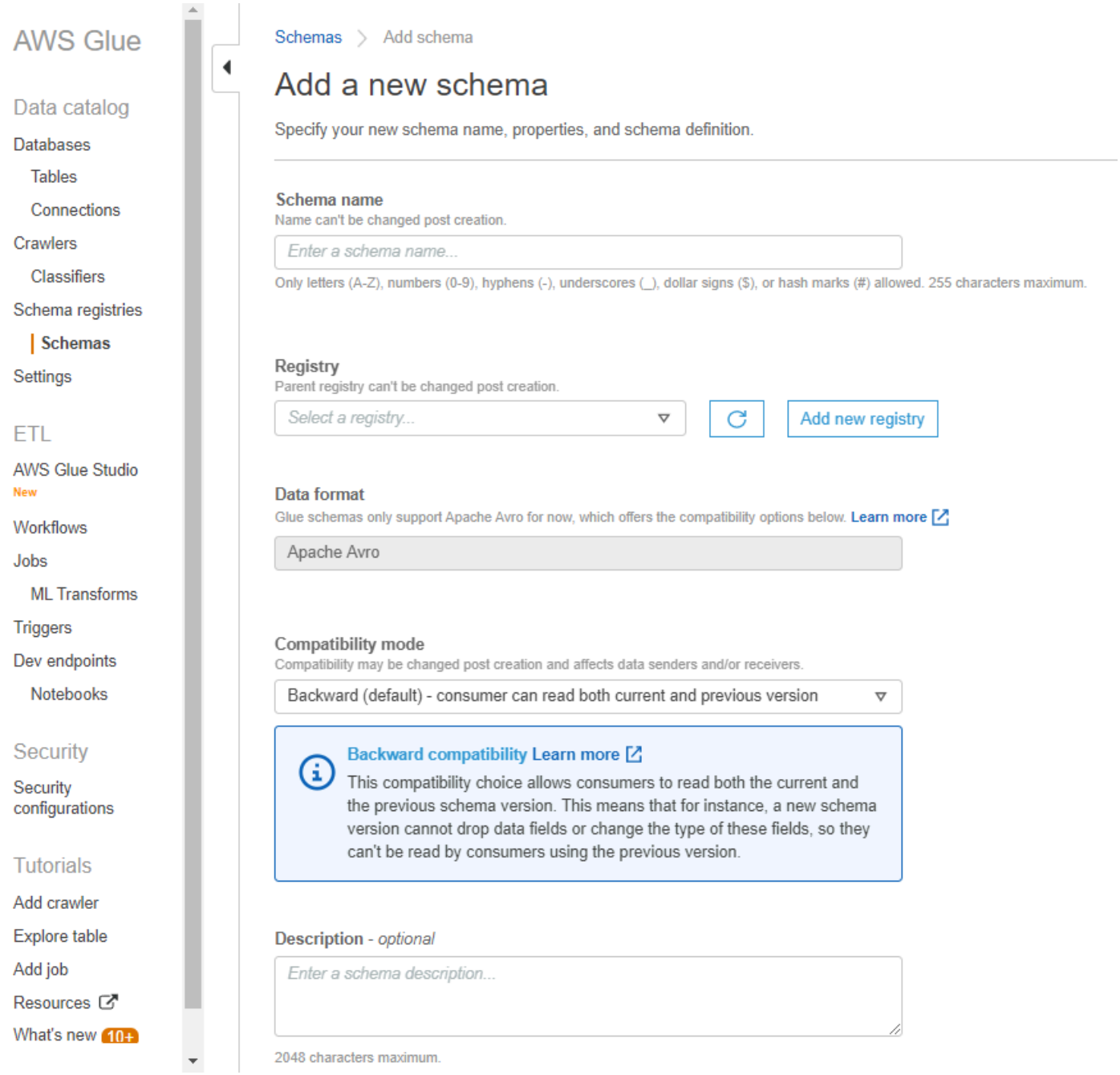
```
aws glue create-schema --registry-id RegistryName="registryName" --schema-name
 testSchemaProtobuf --compatibility NONE --data-format PROTOBUF --schema-definition
 "syntax = \"proto2\";package org.test;message Basic { optional int32 basic = 1;}"
```

```
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 控制台

使用 AWS Glue 控制台添加新架构

1. 登录 AWS 管理控制台，然后通过以下网址打开 AWS Glue 控制台：https://console.aws.amazon.com/glue/。
2. 在导航窗格的 Data catalog (数据目录) 下，选择 Schemas (架构)。
3. 选择 Add schema (添加架构)。
4. 输入 Schema name (架构名称)，包含字母、数字、连字符、下划线、美元符号或哈希标记。此名称不能更改。
5. 选择 Registry (注册表)，这是架构在下拉菜单中的存储位置。父注册表在创建后无法更改。
6. 将 Data format (数据格式) 保留为 Apache Avro 或者 JSON。此格式适用于此架构的所有版本。
7. 选择 Compatibility mode (兼容性模式)。
   - 向后兼容（推荐）– 接收者可以读取当前版本和先前版本。
   - 向后兼容全部 – 接收者可以读取当前版本和所有先前版本。
   - 向前兼容 – 发件人可以写入当前版本和先前版本。
   - 向前兼容全部 – 发件人可以写入当前版本和所有先前版本。
   - 完整兼容 – 向后兼容和向前兼容的组合。
   - 完整兼容全部 – 向后兼容全部和向前兼容全部的组合。
   - 无 – 不执行兼容性检查。
   - 已禁用 – 防止对此架构进行任何版本控制。
8. 为注册表输入可选 Description (说明)，最多 250 个字符。

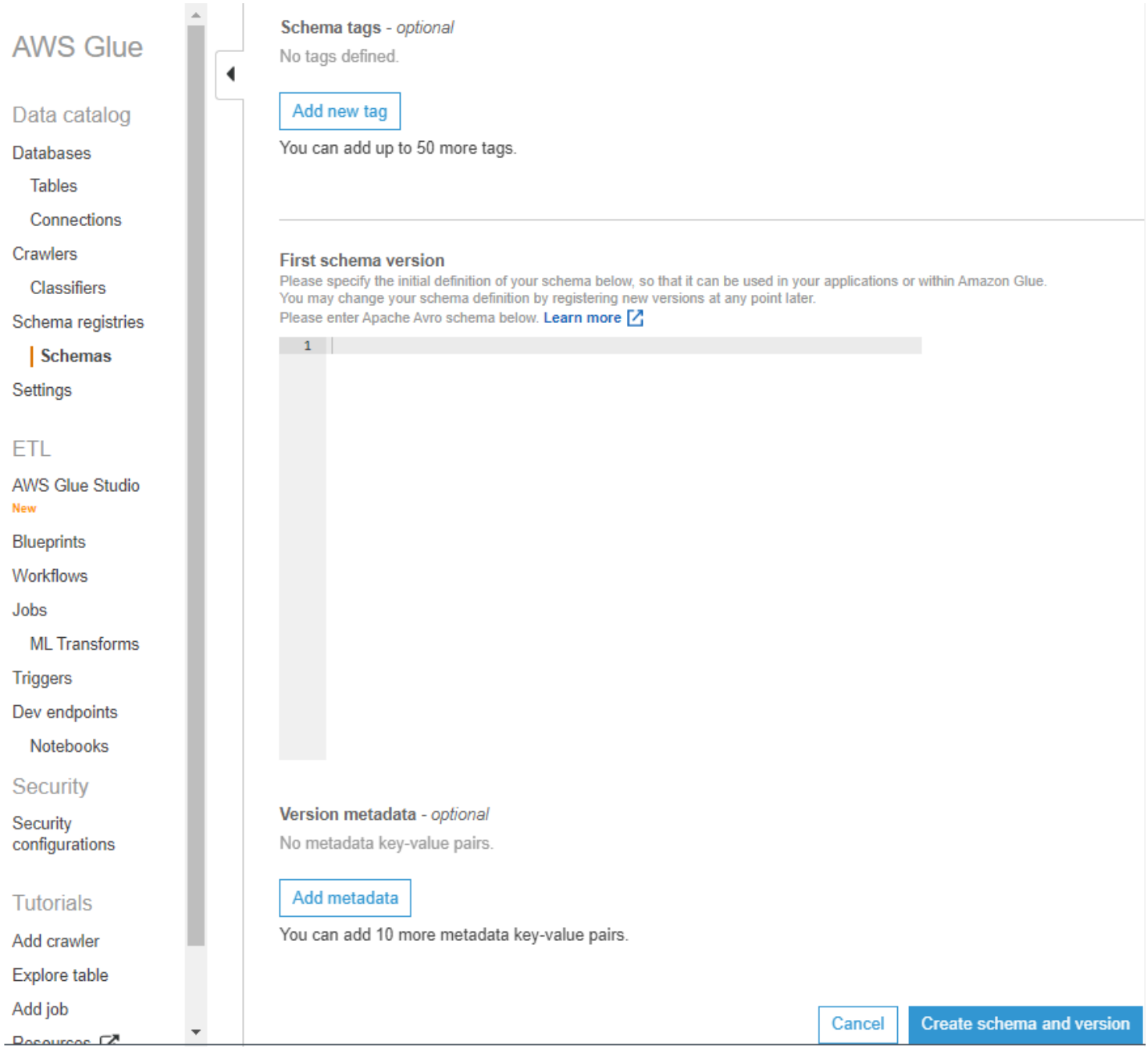


9. （可选）将一个或多个标签应用于架构。选择 Add new tag (添加新标签)，指定 Tag key (标签键) 和 Tag value (标签值)（可选）。
10. 在 First schema version (第一个架构版本) 框中，输入或粘贴您的初始架构。

    有关 Avro 格式的信息，请参阅使用 Avro 数据格式

    有关 JSON 格式的信息，请参阅使用 JSON 数据格式
11. （可选）选择 Add metadata (添加元数据) 添加版本元数据，对架构版本进行注释或分类。

12选择 Create schema and version (创建架构和版本)。



该架构在 Schemas (架构) 下面的列表中创建并显示。

使用 Avro 数据格式

Avro 提供数据序列化和数据交换服务。Avro 以 JSON 格式存储数据定义，使其易于阅读和解释。数据本身以二进制格式存储。

有关定义 Apache Avro 架构的信息，请参阅 Apache Avro 规范。

使用 JSON 数据格式

数据可以使用 JSON 格式进行序列化。JSON 架构格式定义了 JSON 架构格式的标准。

更新架构或注册表

创建后，您可以编辑架构、架构版本或注册表。

更新注册表

您可以使用 AWS Glue API 或 AWS Glue 控制台。无法编辑现有注册表的名称。您可以编辑注册表的说明。

AWS Glue API

要更新现有注册表，请使用 UpdateRegistry 操作（Python：update_registry） API。

指定 `RegistryId` 结构，指示要更新的注册表。传递 `Description`，更改注册表的说明。

```
aws glue update-registry --description updatedDescription --registry-id
 RegistryArn="arn:aws:glue:us-east-2:901234567890:registry/registryName1"
```

AWS Glue 控制台

使用 AWS Glue 控制台更新注册表

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格中的 Data catalog (数据目录) 下面，选择 Schema registries (架构注册表)。
3. 从注册表列表中选择一个注册表，方法是选中该注册表的复选框。
4. 在 Action (操作) 菜单中，选择 Edit registry (编辑注册表)。

更新架构

您可以更新架构的说明或兼容性设置。

要更新现有架构，请使用 UpdateSchema 操作（Python：更新架构） API。

指定 `SchemaId` 结构，指示要更新的架构。必须提供 `VersionNumber` 或 `Compatibility` 中的一个。

代码示例 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
```

添加架构版本

添加架构版本时，您需要比较版本，确保接受新架构。

要为现有架构添加新版本，请使用 RegisterSchemaVersion 操作（Python：register_schema_version）API。

指定 SchemaId 结构，指示要为其添加版本的架构，以及指定 SchemaDefinition 以定义架构。

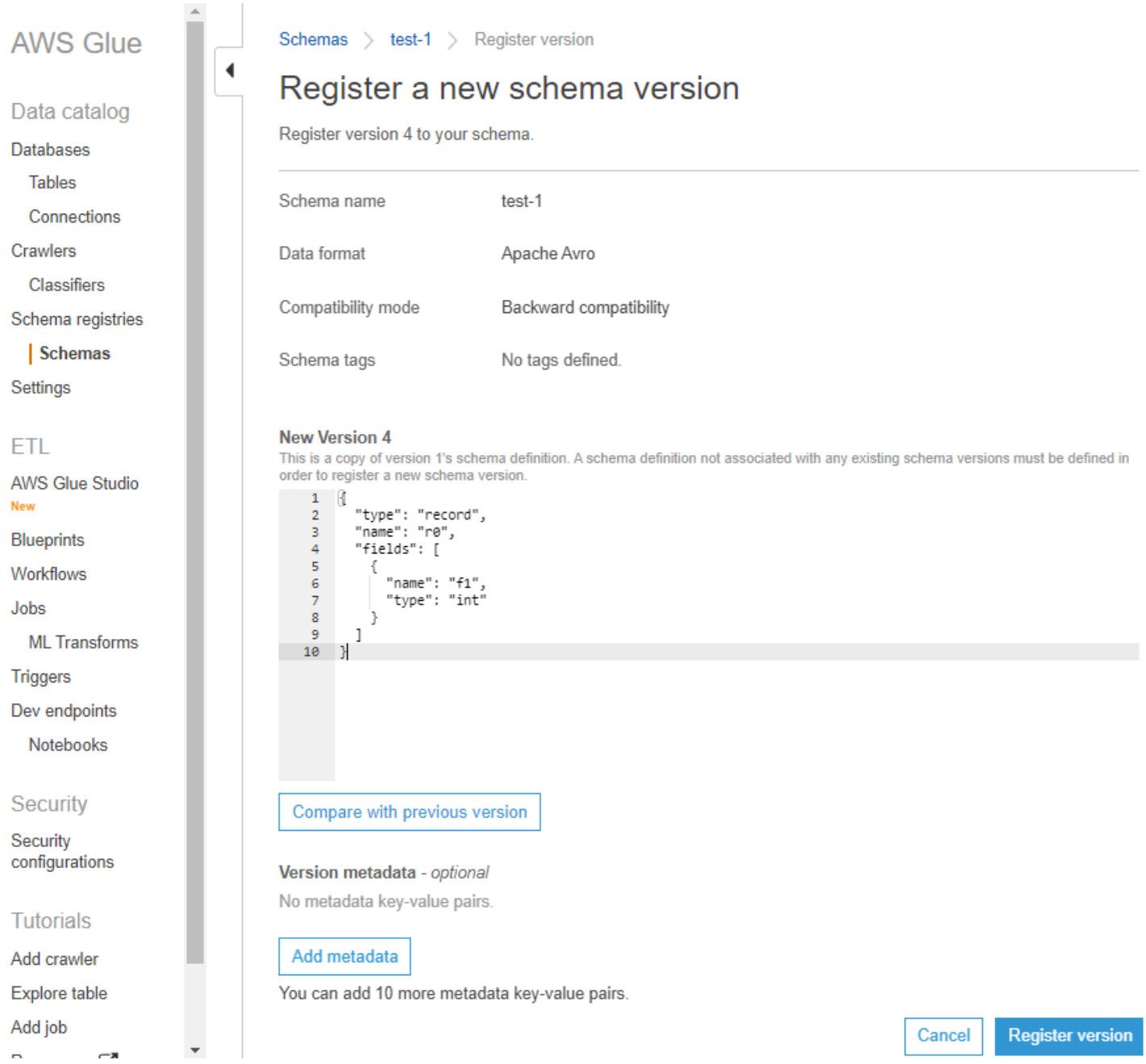
代码示例 12：

```
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. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格的 Data catalog (数据目录) 下，选择 Schemas (架构)。
3. 从方案列表中选择一个方案，方法是选择该方案对应的框。
4. 从列表中选择一个或多个方案，方法是选择相关方案对应的框。
5. 在 Action (操作) 菜单中，选择 Register new version (注册新版本)。
6. 在 New version (新版本) 框中，输入或粘贴新架构。
7. 选择 Compare with previous version (与先前版本比较)，查看与先前架构版本的差异。

8. （可选）选择 Add metadata (添加元数据) 添加版本元数据，对架构版本进行注释或分类。输入 Key (密钥) 和可选 Value (值)。
9. 选择 Register version (注册版本)。



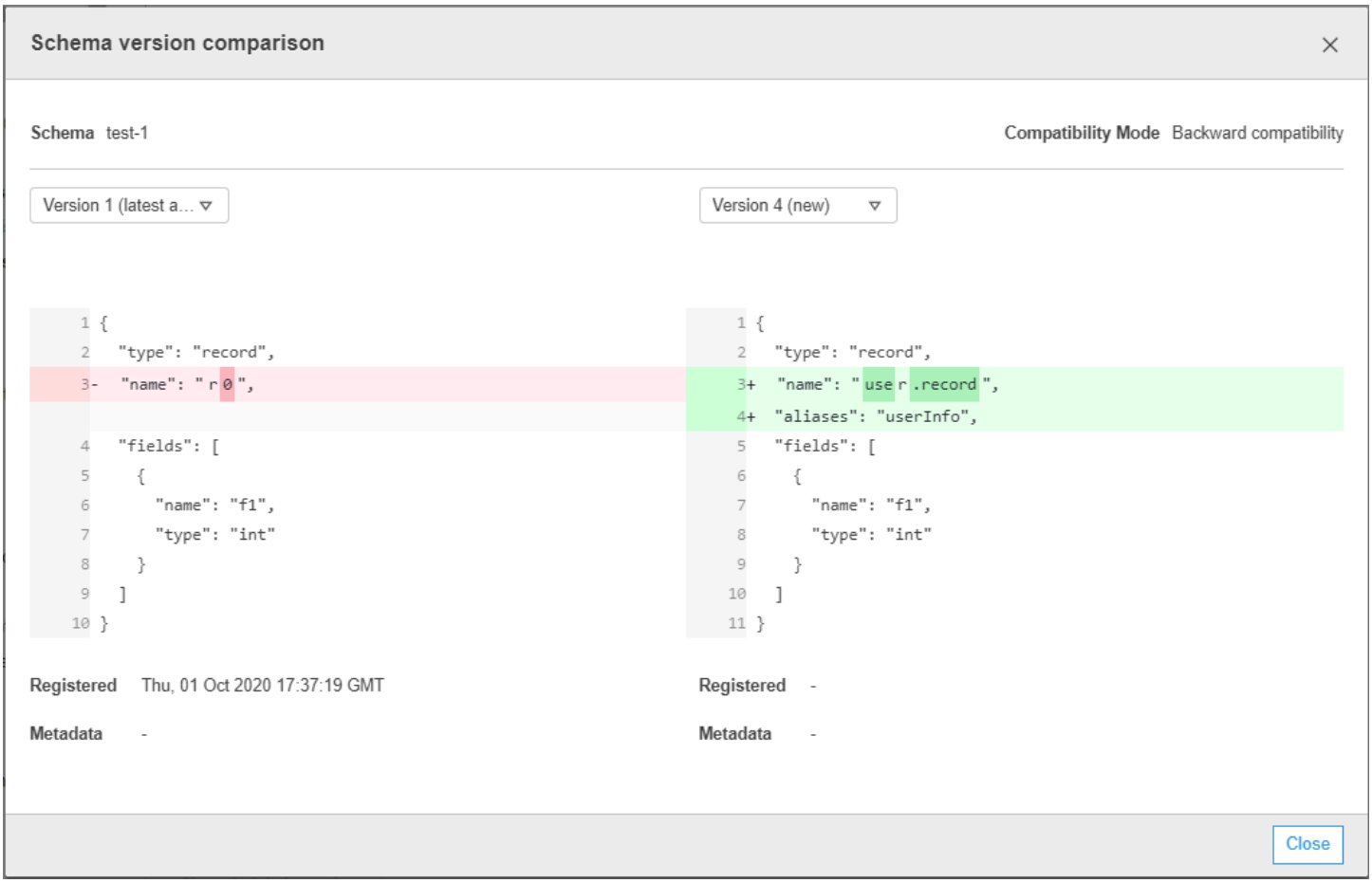
架构版本在版本列表中显示。如果版本更改了兼容性模式，则该版本将标记为检查点。

架构版本比较示例

当您选择 Compare with previous version (与先前版本比较) 时，您将看到先前版本和新版本一起显示。更改的信息将如下突出显示：

- 黄色：表示已更改的信息。
- 绿色：表示在最新版本中添加的内容。
- 红色：表示在最新版本中删除的内容。

您还可以与早期版本进行比较。



删除架构或注册表

删除架构、架构版本或注册表是永久性操作，无法撤消。

删除架构

如果架构不再在注册表中使用，您可能希望删除该架构，请使用 AWS Management Console 或 DeleteSchema 操作（Python：删除架构）API。

删除一个或多个架构是永久性操作，无法撤消。确保不再需要该架构。

要从注册表中删除架构，请调用 DeleteSchema 操作（Python：删除架构）API，指定 SchemaId 结构以标识架构。

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 控制台

从 AWS Glue 控制台删除架构：

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格中的 Data catalog (数据目录) 下面，选择 Schema registries (架构注册表)。
3. 从注册表列表中选择包含您的架构的注册表。
4. 从列表中选择一个或多个方案，方法是选择相关方案对应的框。
5. 在 Action (操作) 菜单中，选择 Delete schema (删除架构)。
6. 在字段中输入文本 **`Delete`** 以确认删除。
7. 选择 删除。

您指定的架构将从注册表中删除。

删除架构版本

随着架构在注册表中累积，您可能需要使用 AWS Management Console 或 DeleteSchemaVersions 操作（Python：删除架构版本）API 删除不需要的架构。删除一个或多个架构版本是永久性操作，无法撤消。确保不再需要该架构版本。

删除架构版本时，请注意以下约束：

- 您无法删除检查点版本。
- 连续版本的范围不能超过 25。

- 最新架构版本不得处于待处理状态。

指定 SchemaId 结构以标识架构，并指定 Versions 为要删除的版本范围。有关指定版本或版本范围的更多信息，请参阅DeleteRegistry 操作（Python：删除注册表）。您指定的架构版本将从注册表中删除。

此调用后调用的 ListSchemaVersions 操作（Python：列表架构版本） API 将列出已删除版本的状态。

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. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格中的 Data catalog (数据目录) 下面，选择 Schema registries (架构注册表)。
3. 从注册表列表中选择包含您的架构的注册表。
4. 从列表中选择一个或多个方案，方法是选择相关方案对应的框。
5. 在 Action (操作) 菜单中，选择 Delete schema (删除架构)。
6. 在字段中输入文本 **`Delete`** 以确认删除。
7. 选择 删除。

您指定的架构版本将从注册表中删除。

删除注册表

当注册表包含的架构不再在该注册表下进行组织时，您可能需要删除该注册表。您需要将这些架构重新分配给另一个注册表。

删除一个或多个注册表是永久性操作，无法撤消。确保不再需要该注册表。

默认注册表可以使用 AWS CLI 删除。

AWS Glue API

要删除整个注册表（包括架构及其所有版本），请调用 DeleteRegistry 操作（Python：删除注册表）API。指定 RegistryId 结构以标识注册表。

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"
```

要获取删除操作的状态，您可以在异步调用后调用 GetRegistry API。

AWS Glue 控制台

从 AWS Glue 控制台删除注册表：

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格中的 Data catalog (数据目录) 下面，选择 Schema registries (架构注册表)。
3. 从列表中选择一个注册表，方法是选中相应的框。
4. 在 Action (操作) 菜单中，选择 Delete registry (删除注册表)。
5. 在字段中输入文本 **`Delete`** 以确认删除。
6. 选择 删除。

您选择的注册表将从 AWS Glue 中删除。

用于序列化程序的 IAM 示例

> **Note**
>
> AWS 托管策略将授予针对常用使用案例的必要权限。有关使用托管策略管理模式注册表的信息，请参阅AWS 的托管（预定义）策略 AWS Glue。

对于序列化程序，您应该创建类似如下的最低策略，以便您能够找到针对指定架构定义的 schemaVersionId。请注意，您应该对注册表具有读取权限，以便读取注册表中的架构。您可以使用 Resource 子句，限制可以读取的注册表。

代码示例 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，您还可以允许创建者包括以下额外的方法，以创建新的架构和版本。请注意，您应该能够检查注册表，以便添加/删除/演变其中的架构。您可以使用 Resource 子句，限制可以检查的注册表。

代码示例 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"
                 ]
}
```

用于反序列化程序的 IAM 示例

对于反序列化程序（使用者端），您应创建类似如下的策略，以允许反序列化程序从架构注册表中获取架构以进行反序列化。请注意，您应该能够检查注册表，以便获取其中的架构。

代码示例 15：

```
{
    "Sid" : "GetSchemaVersion",
    "Effect" : "Allow",
    "Action" :
 [
        "glue:GetSchemaVersion"
    ],
    "Resource" : ["*"]
}
```

使用 AWS PrivateLink 的私有连接

您可以使用 AWS PrivateLink 将您的数据创建者的 VPC 连接到 AWS Glue，方法是为 AWS Glue 定义接口 VPC 终端节点。当您使用 VPC 接口端点时，您的 VPC 与 AWS Glue 之间的通信完全在 AWS 网络内进行。有关更多信息，请参阅将 AWS Glue 与接口 VPC 终端节点一起使用。

访问 Amazon CloudWatch 指标

Amazon CloudWatch 指标可作为 CloudWatch 免费套餐的一部分提供。您可以在 CloudWatch 控制台中查看这些指标。API 级指标包括 CreateSchema（成功和延迟）GetSchemaByDefinition（成功和延迟）、GetSchemaVersion（成功和延迟）、RegisterSchemaVersion（成功和延迟）、PutSchemaVersionMetadata（成功和延迟）。资源级指标包括 Registry.ThrottledByLimit、SchemaVersion.ThrottledByLimit、SchemaVersion.Size。

架构注册表的示例 AWS CloudFormation 模板

以下是一个示例模板，用于在 AWS CloudFormation 中创建架构注册表。要在您的账户中创建此堆栈，请将上面的模板复制到文件 `SampleTemplate.yaml`，并运行以下命令：

```
aws cloudformation create-stack --stack-name ABCSchemaRegistryStack --template-body
 "'cat SampleTemplate.yaml'"
```

此示例使用 AWS::Glue::Registry 创建注册表，使用 AWS::Glue::Schema 创建架构，使用 AWS::Glue::SchemaVersion 创建架构版本，使用 AWS::Glue::SchemaVersionMetadata 填充架构版本元数据。

```
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"
```

与 AWS Glue 架构注册表集成

这些部分介绍了与 AWS Glue 架构注册表的基础。本部分中的示例显示了具有 AVRO 数据格式的架构。有关更多示例（包括具有 JSON 数据格式的架构），请参阅 AWS Glue 架构注册表开源存储库。

主题

- 使用案例：将架构注册表连接到 Amazon MSK 或 Apache Kafka
- 使用案例：将 Amazon Kinesis Data Streams 与 AWS Glue 架构注册表集成
- 应用场景：适用于 Apache Flink 的亚马逊托管服务
- 使用案例：与 AWS Lambda 集成
- 使用案例：AWS Glue Data Catalog
- 使用案例：AWS Glue 流式传输
- 使用案例：Apache Kafka Streams
- 使用案例：Apache Kafka Connect

使用案例：将架构注册表连接到 Amazon MSK 或 Apache Kafka

假设您正在向 Apache Kafka 主题写入数据，您可以按照以下步骤操作。

1. 创建 Amazon Managed Streaming for Apache Kafka（Amazon MSK）或 Apache Kafka 集群，至少具有一个主题。如果创建 Amazon MSK 集群，您可以使用 AWS Management Console。遵循以下说明：请参阅《Amazon Managed Streaming for Apache Kafka 开发人员指南》中的开始使用 Amazon MSK。
2. 遵循上面的安装 SerDe 库步骤。
3. 要创建架构注册表、架构或架构版本，请按照本文档架构注册表入门部分下面的说明操作。
4. 启动您的创建器和使用器，使用架构注册表将记录写入 Amazon MSK 或 Apache Kafka，或者从其中读取记录。您可从 Serde 库的自述文件中找到示例创建器和使用器代码。创建器上的架构注册表库将自动序列化记录，并使用架构版本 ID 装饰记录。

5. 如果已输入此记录的架构，或者启用了自动注册，则该架构将已在架构注册表中注册。
6. 如果使用器从 Amazon MSK 或 Apache Kafka 主题读取数据，使用 AWS Glue 架构注册表库，则该使用器将自动从架构注册表中查找架构。

使用案例：将 Amazon Kinesis Data Streams 与 AWS Glue 架构注册表集成

此集成要求您拥有现有 Amazon Kinesis 数据流。有关更多信息，请参阅《Amazon Kinesis Data Streams 开发人员指南》中的 Amazon Kinesis Data Streams 入门。

您可以通过两种方式与 Kinesis 数据流中的数据进行交互。

- 通过 Java 中的 Kinesis Producer Library（KPL）和 Kinesis Client Library（KCL）库。不提供多语言支持。
- 通过 AWS SDK for Java 中的 `PutRecords`、`PutRecord` 和 `GetRecords` Kinesis Data Streams API。

如果您当前使用的是 KPL/KCL 库，我们建议您继续使用该方法。有更新的 KCL 和 KPL 版本集成了架构注册表，如示例所示。否则，您可以使用示例代码来利用 AWS Glue 架构注册表（如果直接使用 KDS API）。

架构注册表集成仅适用于 KPL v0.14.2 或更高版本以及 KCL v2.3 或更高版本。架构注册表与 JSON 数据格式的集成仅适用于 KPL v0.14.8 或更高版本以及 KCL v2.3.6 或更高版本。

使用 Kinesis SDK V2 与数据进行交互

本部分介绍如何使用 Kinesis SDK V2 与 Kinesis 进行交互

```
// 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 库与数据进行交互

本部分介绍如何使用 KPL/KCL 库将 Kinesis Data Streams 与架构注册表集成。有关 KPL/KCL 使用的更多信息，请参阅《Amazon Kinesis Data Streams 开发人员指南》中的 使用 Amazon Kinesis Producer Library 开发创建器。

在 KPL 中设置架构注册表

1. 定义 AWS Glue 架构注册表中编写的数据、数据格式和架构名称的架构定义。
2. （可选）配置 GlueSchemaRegistryConfiguration 对象。
3. 将架构对象传递到 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());
 }
```

设置 Kinesis 客户端库

您将在 Java 中开发 Kinesis Client Library 使用器 有关更多信息，请参阅《Amazon Kinesis Data Streams 开发人员指南》中的 在 Java 中开发 Kinesis Client Library 使用器。

1. 传递 `GlueSchemaRegistryConfiguration` 对象以创建 `GlueSchemaRegistryDeserializer` 的实例。

2. 将 GlueSchemaRegistryDeserializer 传递到 retrievalConfig.glueSchemaRegistryDeserializer。
3. 调用 kinesisClientRecord.getSchema()，访问传入消息的架构。

```
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 与数据交互

本部分介绍如何使用 Kinesis Data Streams API 将 Kinesis Data Streams 与架构注册表集成。

1. 更新这些 Maven 依赖关系：

```
<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. 在创建器中，使用 Kinesis Data Streams 中的 PutRecords 或者 PutRecord API 添加架构标头信息。

```
//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. 在创建器中，使用 PutRecords 或者 PutRecord API 将记录放入数据流。
4. 在使用器中，从标头中删除架构记录，然后序列化 Avro 架构记录。

```
//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 与数据交互

以下是使用 PutRecords 和 GetRecords API 的示例代码。

```
//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;
    }
}
```

应用场景：适用于 Apache Flink 的亚马逊托管服务

Apache Flink 是一个热门的开源框架和分布式处理引擎，用于对无界和有界数据流进行有状态计算。适用于 Apache Flink 的亚马逊托管服务是一项完全托管式的 AWS 服务，可让您构建和管理 Apache Flink 应用程序以处理流式传输数据。

开源 Apache Flink 提供了大量的源和接收器。例如，预定义的数据源包括从文件、目录和套接字读取数据，以及从集合和迭代器中提取数据。Apache Flink DataStream 连接器为 Apache Flink 提供与各种第三方系统（如作为源和/或接收器的 Apache Kafka 或 Kinesis）接口的代码。

有关更多信息，请参阅 Amazon Kinesis Data Analytics 开发人员指南。

Apache Flink Kafka 连接器

Apache Flink 提供 Apache Kafka 数据流连接器，用于从 Kafka 主题读取数据并将数据写入其中，确切具有一次保证。Flink 的 Kafka 使用器 `FlinkKafkaConsumer` 提供从一个或多个 Kafka 主题读取数据的访问权限。Apache Flink 的 Kafka 创建器 `FlinkKafkaProducer` 允许将记录流写入一个或多个 Kafka 主题。有关更多信息，请参阅 Apache Kafka 连接器。

Apache Flink Kinesis Streams 连接器

Kinesis 数据流连接器可让您访问 Amazon Kinesis Data Streams。`FlinkKinesisConsumer` 是确切一次的并行流数据源，它订阅同一 AWS 服务区域的多个 Kinesis 流，并且可以在任务运行时透明地处理流的重新分区。使用器的每个子任务负责从多个 Kinesis 分片中获取数据记录。随着分区关闭并由 Kinesis 创建，每个子任务获取的分片数量将发生变化。`FlinkKinesisProducer` 使用 Kinesis Producer Library（KPL）将来自 Apache Flink 流的数据放入 Kinesis 流。有关更多信息，请参阅 Amazon Kinesis Streams 连接器。

有关更多信息，请参阅 AWS Glue GitHub 存储库。

与 Apache Flink 集成

架构注册表提供的 SerDes 库与 Apache Flink 集成。要使用 Apache Flink，您需要实施 `SerializationSchema` 和 `DeserializationSchema` 接口（分别称为 `GlueSchemaRegistryAvroSerializationSchema` 和 `GlueSchemaRegistryAvroDeserializationSchema`），您可以将其插入 Apache Fink 连接器。

将 AWS Glue 架构注册表依赖关系添加到 Apache Flink 应用程序

将集成依赖项设置到 Apache Flink 应用程序中的 AWS Glue 架构注册表：

1. 将依赖项添加到 `pom.xml` 文件。

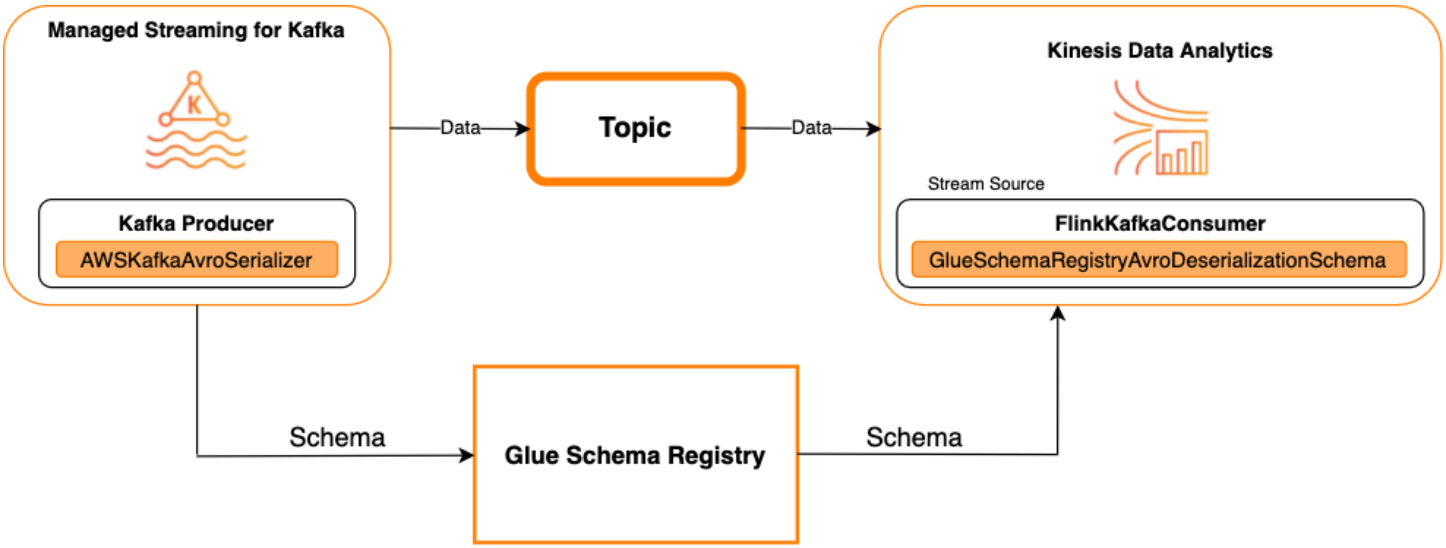
```
<dependency>
    <groupId>software.amazon.glue</groupId>
    <artifactId>schema-registry-flink-serde</artifactId>
    <version>1.0.0</version>
</dependency>
```

将 Kafka 或 Amazon MSK 与 Apache Fink 集成

您可以使用 Managed Service for Apache Flink for Apache Flink，以 Kafka 作为源或接收器。

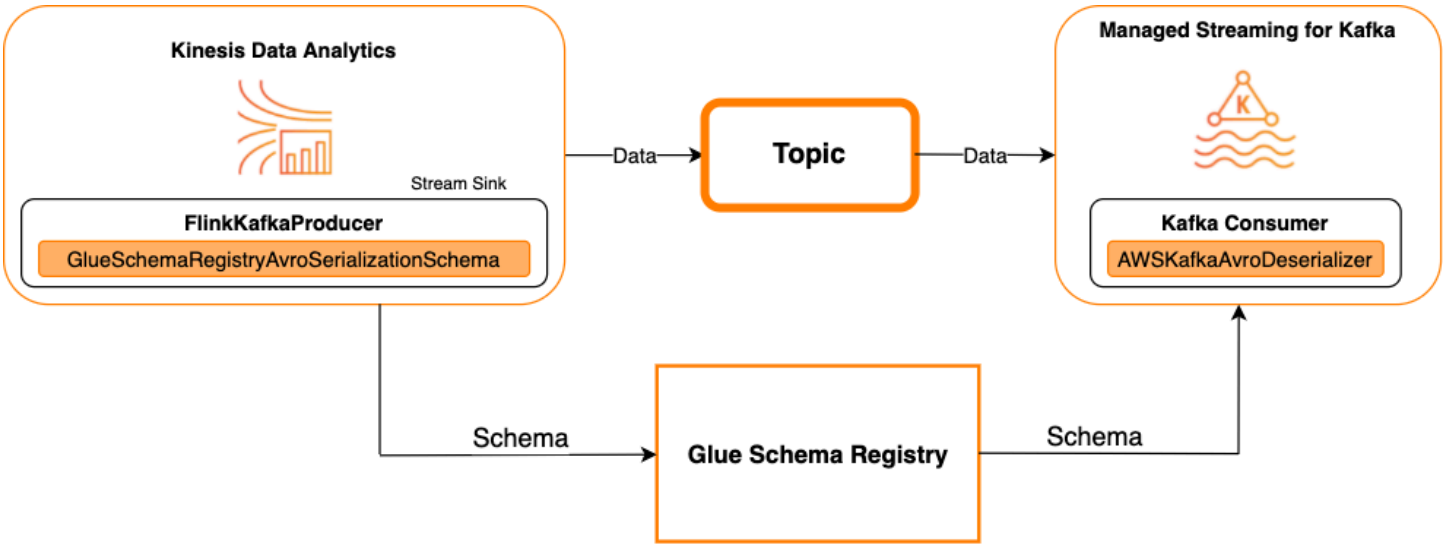
以 Kafka 为源：

下图显示了将 Kinesis Data Streams 与 Managed Service for Apache Flink for Apache Flink 集成，以 Kafka 作为源。



以 Kafka 为接收器

下图显示了将 Kinesis Data Streams 与 Managed Service for Apache Flink for Apache Flink 集成，以 Kafka 作为接收器。



要将 Kafka（或者 Amazon MSK）与 Managed Service for Apache Flink for Apache Flink 集成，以 Kafka 作为源或接收器，请在下面进行代码更改。将粗体代码数据块添加到类似部分中相应的代码。

如果 Kafka 是源，则使用反序列化程序代码（数据块 2）。如果 Kafka 是接收器，则使用序列化程序代码（数据块 3）。

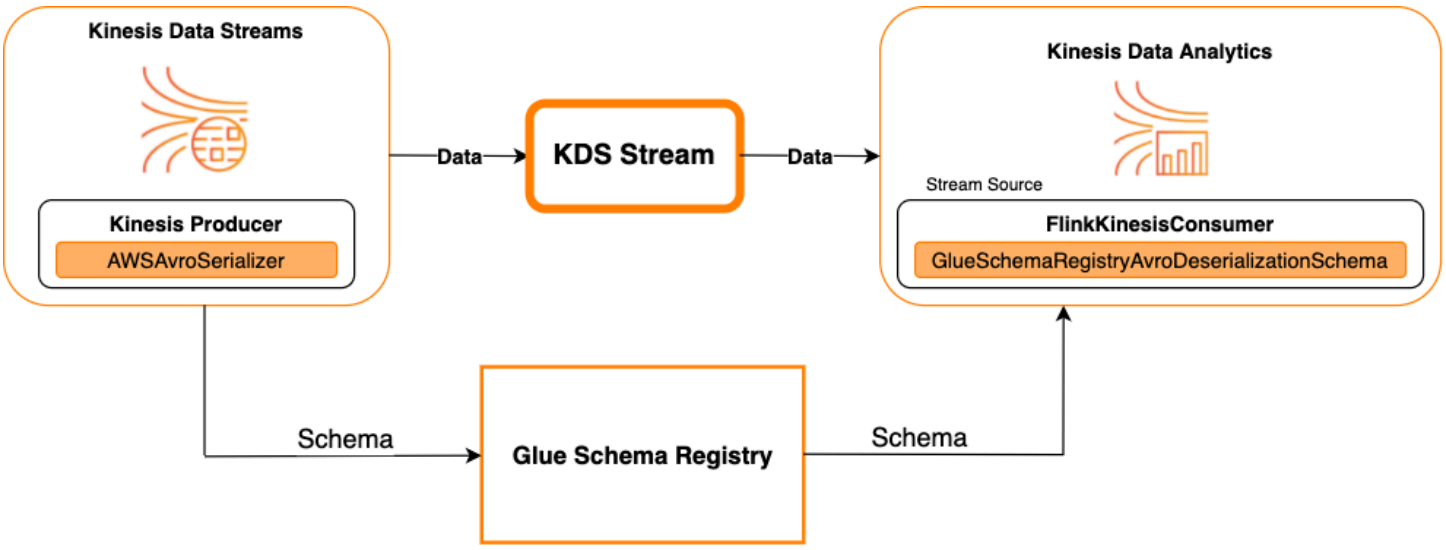
```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

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();
```

将 Kinesis Data Streams 与 Apache Flink 集成

您可以使用 Managed Service for Apache Flink for Apache Flink，以 Kinesis Data Streams 作为源或接收器。

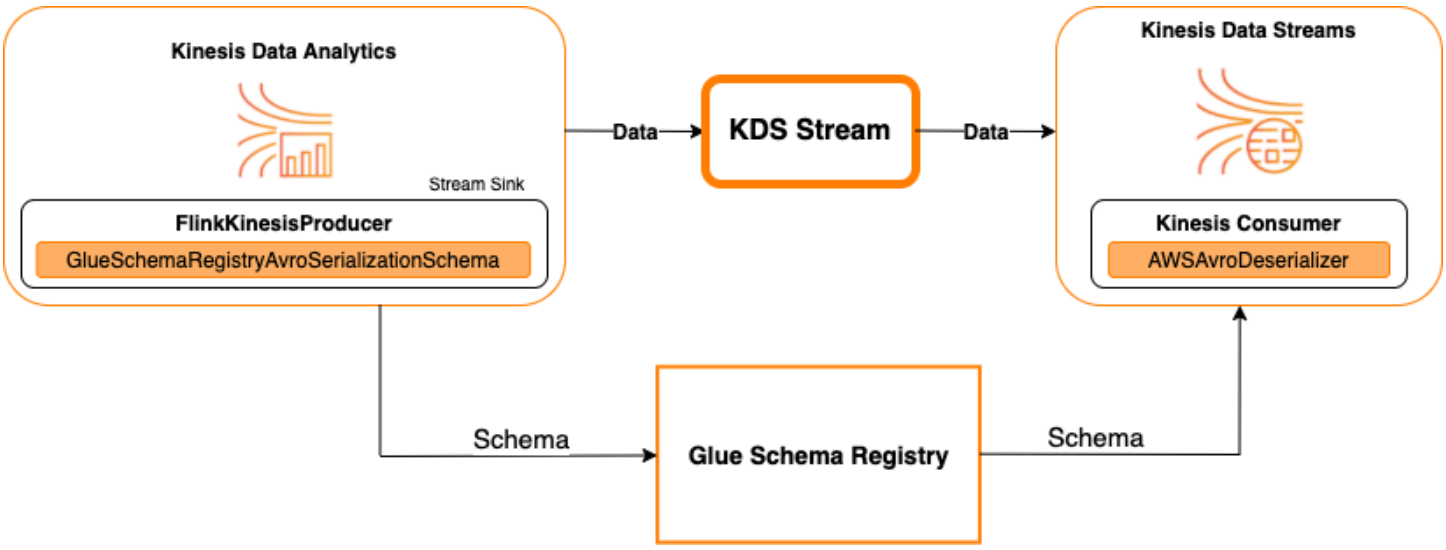
以 Kinesis Data Streams 为源

下图显示了将 Kinesis Data Streams 与 Managed Service for Apache Flink for Apache Flink 集成，以 Kinesis Data Streams 作为源。



以 Kinesis Data Streams 为接收器

下图显示了将 Kinesis Data Streams 与 Managed Service for Apache Flink for Apache Flink 集成，以 Kinesis Data Streams 作为接收器。



要将 Kinesis Data Streams 与 Managed Service for Apache Flink for Apache Flink 集成，以 Kinesis Data Streams 作为源或接收器，请在下面进行代码更改。将粗体代码数据块添加到类似部分中相应的代码。

如果 Kinesis Data Streams 是源，请使用反序列化程序代码（数据块 2）。如果 Kinesis Data Streams 是接收器，请使用序列化程序代码（数据块 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：与 AWS Lambda 集成

要将 AWS Lambda 函数用作 Apache Kafka/Amazon MSK 使用器，并使用 AWS Glue 架构注册表反序列化 Avro 编码消息，请访问 MSK Labs 页。

使用案例：AWS Glue Data Catalog

AWS Glue 表支持您可以手动指定或通过引用 AWS Glue 架构注册表的架构。架构注册表与数据目录集成，以允许您在创建或更新数据目录中 AWS Glue 表或分区时选择性地使用存储于架构注册表中的架构。要标识架构注册表中的架构定义，您至少需要知道它所属的架构的 ARN。架构的架构版本（包含架构定义）可以通过其 UUID 或版本号引用。总有一个架构版本，即“最新”版本，可以在不知道其版本号或 UUID 的情况下查找。

当调用 CreateTable 或者 UpdateTable 操作时，您将传递一个 TableInput 结构，其中包含 StorageDescriptor，它可能有指向架构注册表中现有架构的 SchemaReference。同样，当您调用 GetTable 或者GetPartition API 时，响应可能包含架构和 SchemaReference。使用架构引用创建表或分区时，数据目录将尝试为此架构引用提取架构。如果无法在架构注册表中找到架构，它会在 GetTable 响应中返回空架构；否则响应将具有架构和架构引用。

您还可以在 AWS Glue 控制台中执行操作。

要执行这些操作并创建、更新或查看架构信息，您必须为调用用户授予允许 GetSchemaVersion API 的 IAM 角色权限。

添加表或更新表的架构

从现有架构添加新表会将表绑定到特定架构版本。注册新架构版本后，您可以从 AWS Glue 控制台的 View table (查看表) 页面或使用 UpdateTable 动作（Python：update_table）API 更新表定义。

从现有架构添加表

您可以使用 AWS Glue 控制台或 CreateTable API 通过注册表中的架构版本创建 AWS Glue 表。

AWS Glue API

当调用 CreateTable API 时，您将传递一个 TableInput，其中包含 StorageDescriptor，它可能有指向架构注册表中现有架构的 SchemaReference。

AWS Glue 控制台

从 AWS Glue 控制台创建表：

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格的 Data catalog (数据目录) 中，请选择 Tables (表)。
3. 在 Add Tables (添加表) 菜单中，选择 Add table from existing schema (从现有架构添加表)。
4. 根据 AWS Glue 开发人员指南配置表属性和数据存储。
5. 在 Choose a Glue schema (选择 Glue 架构) 页面上，选择架构所在的 Registry (注册表)。
6. 选择 Schema name (架构名称)，然后选择要应用的架构的 Version (版本)。
7. 查看架构预览，然后选择 Next (下一步)。
8. 审核和创建表。

应用于表的架构和版本将在表列表的 Glue schema (Glue 架构)列中显示。您可以查看表以了解更多详细信息。

更新表架构

当有新架构版本时，您可能需要使用 UpdateTable 动作（Python：update_table） API 或 AWS Glue 控制台更新表架构。

> **Important**
>
> 当为已手动指定 AWS Glue 架构的现有表更新架构时，架构注册表中引用的新架构可能不兼容。这可能会导致您的任务失败。

AWS Glue API

当调用 `UpdateTable` API 时，您将传递一个 `TableInput`，其中包含 `StorageDescriptor`，它可能有指向架构注册表中现有架构的 `SchemaReference`。

AWS Glue 控制台

从 AWS Glue 控制台更新表的架构：

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格的 Data catalog (数据目录) 中，请选择 Tables (表)。
3. 从表列表中查看表。
4. 在告知您新版本的框中单击 Update schema (更新架构)。
5. 查看当前架构和新架构之间的差异。
6. 选择 Show all schema differences (显示所有架构差异)，查看更多详细信息。
7. 选择 Save table (保存表) 以接受新版本。

使用案例：AWS Glue 流式传输

AWS Glue 流式传输会首先使用流式传输源的数据并执行 ETL 操作，然后再写入输出接收器。可以使用数据表或直接通过指定源配置来指定输入流式传输源。

AWS Glue 流式传输支持将利用 AWS Glue Schema 注册表中存在的 Schema 创建的数据目录表作为流式传输源。您可以在 AWS Glue Schema 注册表中创建一个 Schema 并利用此 Schema 创建一个带

有流式传输源的 AWS Glue 表。此 AWS Glue 表可以用作 AWS Glue 流式传输任务的输入，以确保输入流中数据的反序列化。

这里需要注意的是，在 AWS Glue Schema 注册表中的 Schema 更改时，您需要重启 AWS Glue 流式传输任务以反映 Schema 中的更改。

使用案例：Apache Kafka Streams

Apache Kafka Streams API 是一个客户端库，用于处理和分析存储在 Apache Kafka 中的数据。本部分介绍 Apache Kafka Streams 与 AWS Glue 架构注册表的集成，允许您在数据流应用程序上管理和强制实施架构。有关 Apache Kafka Streams 的更多信息，请参阅 Apache Kafka。

与 SerDes 库集成

有一个 `GlueSchemaRegistryKafkaStreamsSerde` 类，您可以使用其配置 Streams 应用程序。

Kafka Streams 应用程序示例代码

在 Apache Kafka Streams 应用程序内使用 AWS Glue 架构注册表：

1. 配置 Kafka Streams 应用程序。

```
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. 通过主题 avro-input 创建流。

```
StreamsBuilder builder = new StreamsBuilder();
final KStream<String, GenericRecord> source = builder.stream("avro-input");
```

3. 处理数据记录（该示例会筛选出 favorite_color 值为 pink 或金额值为 15 的记录）。

```
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. 将结果写回主题 avro-output。

```
result.to("avro-output");
```

5. 启动 Apache Kafka Streams 应用程序。

```
KafkaStreams streams = new KafkaStreams(builder.build(), props);
streams.start();
```

实施结果

这些结果显示了记录筛选流程，这些记录在步骤 3 中筛选出，其 favorite_color 为“pink”或值为“15.0”。

筛选前的记录：

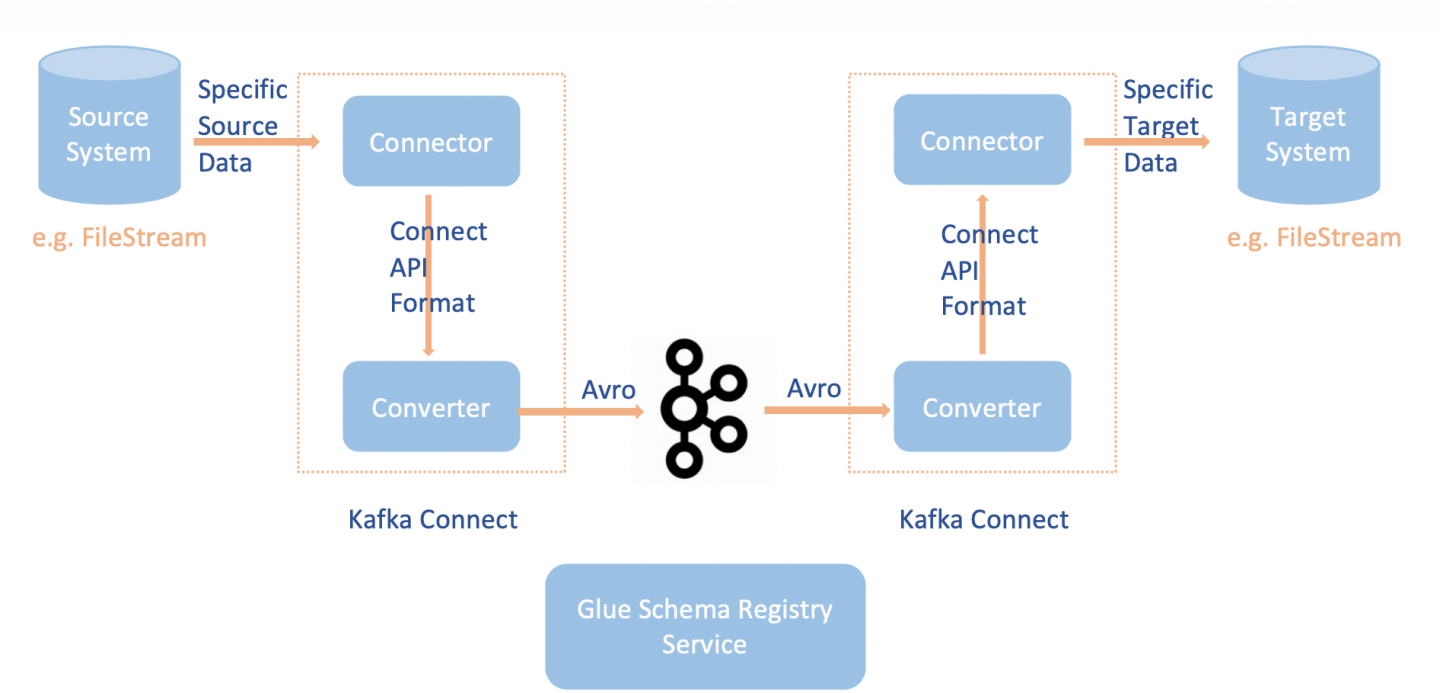
```
{"name": "Sansa", "favorite_number": 99, "favorite_color": "white"}
{"name": "Harry", "favorite_number": 10, "favorite_color": "black"}
{"name": "Hermione", "favorite_number": 1, "favorite_color": "red"}
{"name": "Ron", "favorite_number": 0, "favorite_color": "pink"}
{"name": "Jay", "favorite_number": 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}
```

筛选后的记录：

```
{"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

Apache Kafka Connect 与 AWS Glue 架构注册表集成，支持您从连接器获取架构信息。Apache Kafka 转换器指定 Apache Kafka 内数据的格式以及如何将其转换为 Apache Kafka Connect 数据。每个 Apache Kafka Connect 用户都需要配置这些转换器，基于从 Apache Kafka 加载数据或将数据存储到 Apache Kafka 时期望数据采用的格式。通过这种方式，您可以定义自己的转换器，将 Apache Kafka Connect 数据转换为 AWS Glue 架构注册表（例如：Avro），并使用我们的序列化程序注册其架构并执行序列化。然后，转换器还能够使用我们的反序列化程序来反序列化从 Apache Kafka 接收的数据，并将其转换回 Apache Kafka Connect 数据。下面提供了一个示例工作流图。



1. 克隆适用于 AWS Glue 架构注册表的 Github 存储库，安装 `aws-glue-schema-registry` 项目。

```
git clone git@github.com:awslabs/aws-glue-schema-registry.git
cd aws-glue-schema-registry
mvn clean install
mvn dependency:copy-dependencies
```

2. 如果您计划以独立模式使用 Apache Kafka Connect，请按照下面针对本步骤的说明更新 connect-standalone.properties。如果您计划以分布式模式使用 Apache Kafka Connect，请按照相同的说明更新 connect-avro-distributed.properties。

   a. 您还可以将这些属性添加到 Apache Kafka Connect 属性文件：

   ```
   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. 将下面的命令添加到 kafka-run-class.sh 下面的 Launch mode (启动模式)：

   ```
   -cp $CLASSPATH:"<your AWS GlueSchema Registry base directory>/target/dependency/*"
   ```

3. 将下面的命令添加到 kafka-run-class.sh 下面的 Launch mode (启动模式)

   ```
   -cp $CLASSPATH:"<your AWS GlueSchema Registry base directory>/target/dependency/*"
   ```

   它应如下所示：

   ```
   # 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，请运行以下命令以在 bash_profile 中设置您的 CLASSPATH。对于任何其他 Shell，请相应地更新环境。

```
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. （可选）如果要使用简单的文件源进行测试，请克隆文件源连接器。

```
git clone https://github.com/mmolimar/kafka-connect-fs.git
cd kafka-connect-fs/
```

   a. 在源连接器配置下，将数据格式编辑为 Avro，将文件读取器编辑为 AvroFileReader 并从您正在读取的文件路径中更新示例 Avro 对象。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. 安装源连接器。

```
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 下面的接收器属性，更新主题名称和输出文件名。

```
file=<output file full path>
topics=<my topic>
```

6. 启动源连接器（在本示例中，它是一个文件源连接器）。

```
$KAFKA_HOME/bin/connect-standalone.sh $KAFKA_HOME/config/connect-
standalone.properties config/kafka-connect-fs.properties
```

7. 运行接收器连接器（在本示例中，它是一个文件接收器连接器）。

```
$KAFKA_HOME/bin/connect-standalone.sh $KAFKA_HOME/config/connect-
standalone.properties $KAFKA_HOME/config/connect-file-sink.properties
```

有关 Kafka Connect 用法的示例，请查看适用于 AWS Glue 架构注册表的 Github 存储库中 integration-tests 文件夹下的 run-local-tests.sh 脚本。

从第三方架构注册表迁移到 AWS Glue 架构注册表

从第三方架构注册表到 AWS Glue 架构注册表的迁移依赖于现有的当前第三方架构注册表。如果 Apache Kafka 主题中存在使用第三方架构注册表发送的记录，则使用者需要第三方模式注册表来反序列化这些记录。`AWSKafkaAvroDeserializer` 支持指定辅助反序列化程序类，该类指向第三方反序列化程序并用于反序列化这些记录。

第三方架构的停用有两个条件。第一，仅当使用器不再需要使用第三方架构注册表的 Apache Kafka 主题中的记录，并且这些记录不再适用于这些使用器，则会发生停用。第二，当根据为这些主题指定的保留期限退出 Apache Kafka 主题时，会发生停用。请注意，如果您的主题具有无限保留期，您仍然可以迁移到 AWS Glue 架构注册表，但您将无法停用第三方架构注册表。作为一种解决方法，您可以使用应用程序或 Mirror Maker 2 从当前主题中读取并生成一个新主题，其中包含 AWS Glue 架构注册表。

从第三方架构注册表迁移到 AWS Glue 架构注册表

1. 在 AWS Glue 架构注册表中创建注册表，或者使用默认注册表。
2. 停止使用器。对其进行修改以包含 AWS Glue 架构注册表作为主要反序列化程序，第三方架构注册表作为辅助反序列化程序。
   - 设置使用器属性。在此示例中，secondary_deserializer 设置为不同的反序列化程序。行为如下所示：使用器从 Amazon MSK 检索记录，并首先尝试使用 `AWSKafkaAvroDeserializer`。如果无法读取包含 AWS Glue 架构注册表架构的 Avro 架构 ID 的幻字节，则 `AWSKafkaAvroDeserializer` 尝试使用 secondary_deserializer 中提供的反序列化程序类。使用器属性中还需要提供特定于辅助反序列化程序的属性，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. 重启使用器。
4. 停止创建器并将创建器指向 AWS Glue 架构注册表。

   a. 设置创建器属性。在此示例中，创建器将使用默认注册表和自动注册架构版本。

```
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. （可选）手动将现有架构和架构版本从当前第三方架构注册表移动到 AWS Glue 架构注册表，即 AWS Glue 架构注册表中的默认注册表或 AWS Glue 架构注册表中的特定非默认注册表。以 JSON 格式从第三方架构注册表导出架构并使用 AWS Management Console 或 AWS CLI 在 AWS Glue 架构注册表中创建新架构。

   如果您需要为使用 AWS CLI 和 AWS Management Console 新创建的架构版本启用与先前架构版本的兼容性检查，或者在启用架构版本自动注册的情况下发送包含新架构的消息时，则此步骤可能非常重要。
6. 启动创建器。

连接到数据

AWS Glue连接是一个数据目录对象，用于存储特定数据存储的登录凭据、URI 字符串、虚拟私有云 (VPC) 信息等。 AWS Glue 爬虫、作业和开发端点使用连接来访问某些类型的数据存储。您可以将连接用于源和目标，并在多个爬网程序或提取、转换、加载（ETL）作业中重复使用相同的连接。

AWS Glue 支持以下连接类型：

- Amazon DocumentDB
- 亚马逊 OpenSearch 服务，可与 for Spark AWS Glue 配合使用。
- Amazon Redshift
- Azure Cosmos，用于在 ETL 作业中使用 NoSQL 的 Azure Cosmos DB AWS Glue
- Azure SQL，与 Spark 一起 AWS Glue 使用。
- 谷歌 BigQuery，用 AWS Glue 于 Spark。
- JDBC
- Kafka
- MongoDB
- MongoDB Atlas
- Salesforce
- SAP HANA，与 Spark 搭 AWS Glue 配使用。
- Snowflake，用 AWS Glue 于 Spark。
- Teradata Vantage，用于 Spark 时。 AWS Glue
- Vertica，用 AWS Glue 于 Spark。
- 各种 Amazon Relational Database Service（Amazon RDS）产品。
- 网络（指定到 Amazon Virtual Private Cloud（Amazon VPC）中数据来源的连接）
- Aurora（如果使用原生 JDBC 驱动程序，则支持。并非所有驱动程序功能都可以利用）

使用 AWS Glue Studio，您还可以创建连接器的连接。连接器是一个可选代码包，可帮助访问 AWS Glue Studio中的数据存储。有关更多信息，请参阅 在 AWS Glue Studio 中使用连接器和连接

有关如何连接到本地数据库的信息，请参阅如何使用AWS GlueAWS 大数据博客网站访问和分析本地数据存储。

本节包含以下主题，可帮助您使用 AWS Glue 连接：

AWS Glue 连接属性

本主题包括有关 AWS Glue 连接属性的信息。

主题

- 谷歌 BigQuery 连接
- Vertica 连接

必需的连接属性

在 AWS Glue 控制台上定义连接时，您必须提供以下属性值：

连接名称

为连接输入一个唯一名称。

连接类型

选择 JDBC 或特定的连接类型之一。

有关 JDBC 连接类型的详细信息，请参阅the section called “JDBC 连接属性”

选择 Network (网络) 以连接到 Amazon Virtual Private Cloud 环境（Amazon VPC）中的数据源。

根据您选择的类型，AWS Glue 控制台会显示其他必需字段。例如，如果选择 Amazon RDS，则必须选择数据库引擎。

需要 SSL 连接

选择此选项时，AWS Glue 必须验证数据存储连接是通过受信任的安全套接字层（SSL）进行连接的。

有关更多信息（包括选择此选项时可用的其他选项），请参阅the section called “SSL 连接属性”。

选择 MSK 集群（仅适用于 Amazon managed streaming for Apache Kafka（MSK））

指定来自另一个 AWS 账户的 MSK 集群。

Kafka 引导启动服务器 URL（仅限 Kafka）

指定引导服务器 URL 的逗号分隔的列表。包括端口号。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 连接属性

AWS Glue 可通过 JDBC 连接来连接到以下数据存储：

- Amazon Redshift
- Amazon Aurora
- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL
- Snowflake，使用爬行器 AWS Glue 时。
- Aurora（如果使用原生 JDBC 驱动程序，则支持。并非所有驱动程序功能都可以利用）
- Amazon RDS for MariaDB

> **Important**
>
> 目前，ETL 任务只能在一个子网内使用 JDBC 连接。如果一个作业中有多个数据存储，则它们必须在同一子网，或者可以从该子网访问。
> 如果您选择为 AWS Glue 爬网程序引入自己的 JDBC 驱动程序版本，则您的爬网程序将消耗 AWS Glue 作业 和 Amazon S3 中的资源，以确保您提供的驱动程序在您的环境中运行。额外的资源使用量将反映在您的账户中。此外，提供自己的 JDBC 驱动程序并不意味着爬网程序能够利用该驱动程序的所有功能。驱动程序仅限于在 Data Catalog 中定义连接中描述的属性。

以下是适用于 JDBC 连接类型的其他属性。

JDBC URL

输入 JDBC 数据存储的 URL。对于大多数数据库引擎，此字段将采用以下格式。在此格式中，将 *protocol####*、*host####*、*port####*和 *db_name#db_###*替换为您自己的信息。

```
jdbc:protocol://host:port/db_name
```

根据数据库引擎，可能需要不同的 JDBC URL 格式。此格式可能稍微不同地使用冒号 (:) 和斜杠 (/) 或不同的关键字来指定数据库。

要让 JDBC 连接到数据存储，数据存储中需要 db_name。db_name 用于与提供的 `username` 和 `password` 建立网络连接。当连接时，AWS Glue 可以访问数据存储中的其他数据库以运行爬网程序或运行 ETL 作业。

以下 JDBC URL 示例显示了多个数据库引擎的语法。

- 要连接到具有 dev 数据库的 Amazon Redshift 群集数据存储：

  ```
  jdbc:redshift://xxx.us-east-1.redshift.amazonaws.com:8192/dev
  ```

- 要连接到具有 employee 数据库的 Amazon RDS for MySQL 数据存储：

  ```
  jdbc:mysql://xxx-cluster.cluster-xxx.us-east-1.rds.amazonaws.com:3306/
  employee
  ```

- 要连接到具有 employee 数据库的 Amazon RDS for PostgreSQL 数据存储：

  ```
  jdbc:postgresql://xxx-cluster.cluster-xxx.us-
  east-1.rds.amazonaws.com:5432/employee
  ```

- 要连接到具有 employee 服务名称的 Amazon RDS for Oracle 数据存储：

  ```
  jdbc:oracle:thin://@xxx-cluster.cluster-xxx.us-
  east-1.rds.amazonaws.com:1521/employee
  ```

  Amazon RDS for Oracle 的语法可以遵循以下模式。在这些模式中，将 *host####*、*port####*、*service_name###_###*和 *SID* 替换为您自己的信息。

  - `jdbc:oracle:thin://@`*host*`:`*port*`/`*service_name*
  - `jdbc:oracle:thin://@`*host*`:`*port*`:`*SID*

- 要连接到具有 employee 数据库的 Amazon RDS for Microsoft SQL Server 数据存储：

  ```
  jdbc:sqlserver://xxx-cluster.cluster-xxx.us-
  east-1.rds.amazonaws.com:1433;databaseName=employee
  ```

  Amazon RDS for SQL Server 的语法可以遵循以下模式：在这些模式中，将 *server_name*、*port* 和 *db_name* 替换为您自己的信息。

  - `jdbc:sqlserver://`*server_name*`:`*port*`;database=`*db_name*
  - `jdbc:sqlserver://`*server_name*`:`*port*`;databaseName=`*db_name*

- 要连接到employee数据库的 Amazon Aurora PostgreSQL 实例，请指定数据库实例的终端节点、端口和数据库名称：

  ```
  jdbc:postgresql://employee_instance_1.xxxxxxxxxxxx.us-
  east-2.rds.amazonaws.com:5432/employee
  ```

- 要使用数据库连接到 Amazon RDS for MariaDB 数据存储，请指定employee数据库实例的终端节点、端口和数据库名称：

```
jdbc:mysql://xxx-cluster.cluster-xxx.aws-
region.rds.amazonaws.com:3306/employee
```

- > **Warning**
  >
  > 只有爬虫支持 Snowflake JDBC 连接。 AWS Glue 在 AWS Glue 作业中使用 Snowflake 连接器时，请使用 Snowflake 连接类型。

  要连接到 sample 数据库的 Snowflake 实例，请指定 Snowflake实例的端点、用户、数据库名称和角色名称。您可以选择添加 warehouse 参数。

  ```
  jdbc:snowflake://account_name.snowflakecomputing.com/?
  user=user_name&db=sample&role=role_name&warehouse=warehouse_name
  ```

  > **Important**
  >
  > 对于通过 JDBC 进行的 Snowflake 连接，将强制执行 URL 中参数的顺序，并且必须按照 user、db、role_name 和 warehouse 进行排序。

- 要使用 AWS 专用链接连接到sample数据库的 Snowflake 实例，请按如下方式指定 snowflake JDBC 网址：

  ```
  jdbc:snowflake://account_name.region.privatelink.snowflakecomputing.com/?
  user=user_name&db=sample&role=role_name&warehouse=warehouse_name
  ```

用户名

> **Note**
>
> 我们建议您使用 AWS 密钥来存储连接凭证，而不是直接提供您的用户名和密码。有关更多信息，请参阅 在 AWS Secrets Manager 中存储连接凭证。

提供有权访问 JDBC 数据存储的用户名。

密码

输入对 JDBC 数据存储具有访问权限的用户名的密码。

端口

输入 JDBC URL 中使用的端口以连接到 Amazon RDS Oracle 实例。只有在为 Amazon RDS Oracle 实例选择 Require SSL connection (需要 SSL 连接) 时会显示此字段。

VPC

选择包含您的数据存储的 Amazon Virtual Private Cloud (VPC) 的名称。AWS Glue 控制台会列出当前区域的所有 VPC。

> **Important**
>
> 在使用托管的 JDBC 连接（例如来自 Snowflake 的数据）时 AWS，您的 VPC 应该有一个 NAT 网关，用于将流量分成公有子网和私有子网。公有子网用于连接外部源，内部子网用于处理 AWS Glue。有关为外部连接配置 Amazon VPC 的信息，请阅读使用 NAT 设备连接到互联网或其他网络以及 设置 Amazon VPC 以通过建立从 AWS Glue 到 Amazon RDS 数据存储的 JDBC 连接。

子网

选择包含您的数据存储的 VPC 内的子网。AWS Glue 控制台列出了您的 VPC 中的数据存储的所有子网。

安全组

选择与您的数据存储关联的安全组。AWS Glue 需要一个或多个安全组且其入站源规则允许 AWS Glue 进行连接。AWS Glue 控制台列出了所有被授权对您的 VPC 进行入站访问的安全组。AWS Glue 将这些安全组与连接到您的 VPC 的子网的弹性网络接口关联。

JDBC 驱动程序类名 - 可选

提供自定义 JDBC 驱动程序类名：

- 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 — oracle.jdbc.driver。 OracleDriver
-

SQL Server — com.microsoft.sqlserver.jdbc.S ServerDriver

JDBC 驱动程序 S3 路径 - 可选

向自定义 JDBC 驱动程序提供 Amazon S3 位置。这是 .jar 文件的绝对路径。如果您想提供自己的 JDBC 驱动程序，连接到您的爬网程序支持的数据库的数据源，则可以为参数 `customJdbcDriverS3Path` 和 `customJdbcDriverClassName` 指定值。
使用客户提供的 JDBC 驱动程序仅限于所需的 必需的连接属性。

AWS Glue MongoDB 和 MongoDB Atlas 连接属性

以下是适用于 MongoDB 或 MongoDB Atlas 连接类型的其他属性。

MongoDB URL

输入 MongoDB 或 MongoDB Atlas 数据存储的 URL：

- 对于 MongoDB：mongodb://host:port/database。主机可以是主机名、IP 地址或 UNIX 域套接字。如果连接字符串未指定端口，则使用默认的 MongoDB 端口 27017。
- 对于 MongoDB Atlas：mongodb+srv://server.example.com/database。主机可以是后面对应于 DNS SRV 记录的主机名。SRV 格式不需要端口，将使用默认的 MongoDB 端口 27017。

用户名

> **Note**
>
> 我们建议您使用 AWS 密钥来存储连接凭证，而不是直接提供您的用户名和密码。有关更多信息，请参阅 在 AWS Secrets Manager 中存储连接凭证。

提供有权访问 JDBC 数据存储的用户名。

密码

输入具有 MongoDB 或 MongoDB Atlas 数据存储访问权限的用户名的密码。

Salesforce 连接属性

以下是 Salesforce 连接类型的其他属性。

- `ENTITY_NAME`（字符串）-（必填）用于读/写。你在 Salesforce 中的对象的名称。

- API_VERSION（字符串）-（必填）用于读/写。你要使用的 Salesforce Rest API 版本。
- SELECTED_FIELDS（列表<String>）-默认：空（选择*）。用于读取。要为对象选择的列。
- FILTER_PREDICATE（字符串）-默认：空。用于读取。它应该采用 Spark SQL 格式。
- QUERY（字符串）-默认：空。用于读取。完整的 Spark SQL 查询。
- PARTITION_FIELD（字符串）-用于读取。用于分区查询的字段。
- LOWER_BOUND（字符串）-用于读取。所选分区字段的包含下限值。
- UPPER_BOUND（字符串）-用于读取。所选分区字段的唯一上限值。
- NUM_PARTITIONS（整数）-默认：1。用于读取。要读取的分区数。
- IMPORT_DELETED_RECORDS（字符串）-默认：假。用于读取。在查询时获取删除记录。
- WRITE_OPERATION（字符串）-默认：插入。用于写入。值应为 INSERT、UPDATE、UPSERT、DELETE。
- ID_FIELD_NAMES（字符串）-默认：空。仅用于 UPSERT。

Snowflake 连接

以下属性用于设置在 AWS Glue ETL 作业中使用的 Snowflake 连接。爬取 Snowflake 时，请使用 JDBC 连接。

Snowflake URL

Snowflake 端点的 URL。有关 Snowflake 端点 URL 的更多信息，请参阅 Snowflake 文档中的 Connecting to Your Accounts。

AWS 秘密

中密钥的秘密名称 AWS Secrets Manager。 AWS Glue 将使用你的`sfPassword`密钥`sfUser`和密钥连接到 Snowflake。

Snowflake 角色（可选）

连接时 AWS Glue 将使用 Snowflake 安全角色。

配置与托管在 Amazon VPC 中的 Snowflake 端点的连接时，请使用以下属性。 AWS PrivateLink

VPC

选择包含您的数据存储的 Amazon Virtual Private Cloud (VPC) 的名称。AWS Glue 控制台会列出当前区域的所有 VPC。

子网

选择包含您的数据存储的 VPC 内的子网。AWS Glue 控制台列出了您的 VPC 中的数据存储的所有子网。

安全组

选择与您的数据存储关联的安全组。AWS Glue 需要一个或多个安全组且其入站源规则允许 AWS Glue 进行连接。AWS Glue 控制台列出了所有被授权对您的 VPC 进行入站访问的安全组。AWS Glue 将这些安全组与连接到您的 VPC 的子网的弹性网络接口关联。

Vertica 连接

使用以下属性为 AWS Glue ETL 作业设置垂直连接。

Vertica 主机

Vertica 安装的主机名。

Vertica 端口

可用于访问 Vertica 安装的端口。

AWS 秘密

中密钥的秘密名称 AWS Secrets Manager。 AWS Glue 将使用你的密钥连接到 Vertica。

配置与 Amazon VPC 中的 Vertica 端点的连接时，请使用以下属性。

VPC

选择包含您的数据存储的 Amazon Virtual Private Cloud (VPC) 的名称。AWS Glue 控制台会列出当前区域的所有 VPC。

子网

选择包含您的数据存储的 VPC 内的子网。AWS Glue 控制台列出了您的 VPC 中的数据存储的所有子网。

安全组

选择与您的数据存储关联的安全组。AWS Glue 需要一个或多个安全组且其入站源规则允许 AWS Glue 进行连接。AWS Glue 控制台列出了所有被授权对您的 VPC 进行入站访问的安全组。AWS Glue 将这些安全组与连接到您的 VPC 的子网的弹性网络接口关联。

SAP HANA 连接

使用以下属性为 AWS Glue ETL 作业设置 SAP HANA 连接。

SAP HANA URL

一个 SAP JDBC URL。

SAP HANA JDBC URL 的格式为 `jdbc:sap://saphanaHostname:saphanaPort/?databaseName=saphanaDBname,ParameterNa`

AWS Glue 需要以下 JDBC 网址参数：

- `databaseName` – SAP HANA 中要连接的默认数据库。

AWS 秘密

中密钥的秘密名称 AWS Secrets Manager。 AWS Glue 将使用你的密钥连接到 SAP HANA。

配置与托管在 Amazon VPC 中的 SAP HANA 端点的连接时，请使用以下属性：

VPC

选择包含您的数据存储的 Amazon Virtual Private Cloud (VPC) 的名称。AWS Glue 控制台会列出当前区域的所有 VPC。

子网

选择包含您的数据存储的 VPC 内的子网。AWS Glue 控制台列出了您的 VPC 中的数据存储的所有子网。

安全组

选择与您的数据存储关联的安全组。AWS Glue 需要一个或多个安全组且其入站源规则允许 AWS Glue 进行连接。AWS Glue 控制台列出了所有被授权对您的 VPC 进行入站访问的安全组。AWS Glue 将这些安全组与连接到您的 VPC 的子网的弹性网络接口关联。

Azure SQL 连接

使用以下属性为 AWS Glue ETL 作业设置 Azure SQL 连接。

Azure SQL URL

Azure SQL 端点的 JDBC URL。

该 URL 必须为以下格式：`jdbc:sqlserver://databaseServerName:databasePort;databaseName=azuresqlDBna`

AWS Glue 需要以下 URL 属性：

- `databaseName` – Azure SQL 中要连接的默认数据库。

有关 Azure SQL 托管实例的 JDBC URL 的更多信息，请参阅 Microsoft 文档。

AWS 秘密

中密钥的秘密名称 AWS Secrets Manager。 AWS Glue 将使用你的密钥连接到 Azure SQL。

Teradata Vantage 连接

使用以下属性为 ETL 作业设置 Teradata Vantage 连接。 AWS Glue

Teradata URL

要连接到 Teradata 实例，请指定数据库实例的主机名和相关的 Teradata 参数：

`jdbc:teradata://teradataHostname/ParameterName=ParameterValue,ParameterName=Pa`

AWS Glue 支持以下 JDBC 网址参数：

- `DATABASE_NAME`— Teradata 中要连接的默认数据库。
- `DBS_PORT` – 如果属于非标准端口，则请指定 Teradata 端口。

AWS 秘密

中密钥的秘密名称 AWS Secrets Manager。 AWS Glue 将使用你的密钥连接到 Teradata Vantage。

配置与托管在 Amazon VPC 中的 Teradata Vantage 端点的连接时，请使用以下属性：

VPC

选择包含您的数据存储的 Amazon Virtual Private Cloud (VPC) 的名称。AWS Glue 控制台会列出当前区域的所有 VPC。

子网

选择包含您的数据存储的 VPC 内的子网。AWS Glue 控制台列出了您的 VPC 中的数据存储的所有子网。

安全组

选择与您的数据存储关联的安全组。AWS Glue 需要一个或多个安全组且其入站源规则允许 AWS Glue 进行连接。AWS Glue 控制台列出了所有被授权对您的 VPC 进行入站访问的安全组。AWS Glue 将这些安全组与连接到您的 VPC 的子网的弹性网络接口关联。

OpenSearch 服务连接

使用以下属性为 AWS Glue ETL 作业设置 OpenSearch 服务连接。

域端点

亚马逊 OpenSearch 服务域终端节点将采用以下默认格式 https://search-*#unstructuredIdContent#*-。 ## .es.amazonaws.com。有关识别域终端节点的更多信息，请参阅亚马逊 OpenSearch 服务文档中的创建和管理亚马逊 OpenSearch 服务域名。

端口

端点上的端口已打开。

AWS 秘密

中密钥的秘密名称 AWS Secrets Manager。 AWS Glue 将使用您的密钥连接到 OpenSearch 服务。

配置与 Amazon VPC 中托管的 OpenSearch 服务终端节点的连接时，请使用以下属性：

VPC

选择包含您的数据存储的 Amazon Virtual Private Cloud (VPC) 的名称。AWS Glue 控制台会列出当前区域的所有 VPC。

子网

选择包含您的数据存储的 VPC 内的子网。AWS Glue 控制台列出了您的 VPC 中的数据存储的所有子网。

安全组

选择与您的数据存储关联的安全组。AWS Glue 需要一个或多个安全组且其入站源规则允许 AWS Glue 进行连接。AWS Glue 控制台列出了所有被授权对您的 VPC 进行入站访问的安全组。AWS Glue 将这些安全组与连接到您的 VPC 的子网的弹性网络接口关联。

Azure Cosmos 连接

使用以下属性为 AWS Glue ETL 作业设置 Azure Cosmos 连接。

Azure Cosmos DB 账户端点 URI

用于连接到 Azure Cosmos 的端点。有关更多信息，请参阅 Azure 文档。

AWS 秘密

中密钥的秘密名称 AWS Secrets Manager。 AWS Glue 将使用你的密钥连接到 Azure Cosmos。

AWS Glue SSL 连接属性

下面是有关 Require SSL connection（需要 SSL 连接）属性的详细信息。

如果不需要 SSL 连接，则 AWS Glue 在使用 SSL 加密与数据存储的连接时会忽略失败。有关配置说明，请参阅数据存储的文档。当您选择此选项时，如果 AWS Glue 无法连接，则开发端点中的作业运行、爬网程序或 ETL 语句将失败。

> **Note**
>
> Snowflake 默认支持 SSL 连接，因此此属性不适用于 Snowflake。

此选项在 AWS Glue 客户端进行验证。对于 JDBC 连接，AWS Glue 仅通过 SSL 连接并进行证书和主机名验证。SSL 连接支持适用于：

- Oracle 数据库
- Microsoft SQL Server
- PostgreSQL
- Amazon Redshift
- MySQL（仅限 Amazon RDS 实例）
- Amazon Aurora MySQL（仅限 Amazon RDS 实例）
- Amazon Aurora PostgreSQL （仅限亚马逊 RDS 实例）
- Kafka，其中包括 Amazon Managed Streaming for Apache Kafka
- MongoDB

> **Note**
>
> 要使 Amazon RDS Oracle 数据存储能够使用 Require SSL connection (需要 SSL 连接)，您必须创建一个选项组并将其附加到 Oracle 实例。
>
> 1. 登录 AWS Management Console 并打开 Amazon RDS 控制台，网址为 https://console.aws.amazon.com/rds/。
> 2. 将 Option group (选项组) 添加到 Amazon RDS Oracle 实例。有关如何在 Amazon RDS 控制台上添加选项组的更多信息，请参阅创建选项组
> 3. 将 Option (选项) 添加到 SSL 的选项组。您为 SSL 指定的端口稍后在您为 Amazon RDS Oracle 实例创建 AWS Glue JDBC 连接 URL 时使用。有关如何在 Amazon RDS 控制台上添加选项的更多信息，请参阅 Amazon RDS 用户指南中的向选项组添加选项。有关 Oracle SSL 选项的更多信息，请参阅 Amazon RDS 用户指南中的 Oracle SSL。
> 4. 在 AWS Glue 控制台中，创建到 Amazon RDS Oracle 实例的连接。在连接定义中，选择 Require SSL connection (需要 SSL 连接)。在请求时，输入您在 Amazon RDS Oracle SSL 选项中使用的端口。

当为连接选择了 Require SSL connection (需要 SSL 连接) 时，可使用以下其他可选属性。

S3 中的自定义 JDBC 证书

如果您的证书当前用于与本地或云数据库进行 SSL 通信，则可以将该证书用于与 AWS Glue 数据源或目标的 SSL 连接。输入包含自定义根证书的 Amazon Simple Storage Service（Amazon S3）位置。AWS Glue 使用此证书建立与数据库的 SSL 连接。AWS Glue 仅处理 X.509 证书。该证书必须经过 DER 编码，并以 base64 编码 PEM 格式提供。

如果将此字段留空，则使用默认证书。

自定义 JDBC 证书字符串

输入 JDBC 数据库特定的证书信息。此字符串用于域匹配或可分辨名称 (DN) 匹配。对于 Oracle 数据库，此字符串会映射到 `tnsnames.ora` 文件中安全性部分中的 `SSL_SERVER_CERT_DN` 参数。对于 Microsoft SQL Server，此字符串将用作 `hostNameInCertificate`。

下面是适用于 Oracle 数据库 `SSL_SERVER_CERT_DN` 参数的示例。

```
cn=sales,cn=OracleContext,dc=us,dc=example,dc=com
```

Kafka 私有 CA 证书位置

如果您拥有当前用于与 Kafka 数据存储进行 SSL 通信的证书，则可以在 AWS Glue 连接中使用该证书。此选项对于 Kafka 数据存储是必需的，对于 Amazon Managed Streaming for Apache Kafka 数据存储是可选的。输入包含自定义根证书的 Amazon Simple Storage Service（Amazon S3）位置。AWS Glue 使用此证书建立与 Kafka 数据存储的 SSL 连接。AWS Glue 仅处理 X.509 证书。该证书必须经过 DER 编码，并以 base64 编码 PEM 格式提供。

Skip certificate validation (跳过证书验证)

选中 Skip certificate validation (跳过证书验证) 复选框，可跳过 AWS Glue 对自定义证书的验证。如果您选择验证，则 AWS Glue 会验证证书的签名算法和主题公有密钥算法。如果证书验证失败，则使用该连接的任何 ETL 作业或爬网程序都将失败。

唯一允许的签名算法是 SHA256withRSA、SHA384withRSA 或 SHA512withRSA。对于主题公有密钥算法，密钥长度必须至少为 2048 位。

Kafka 客户端密钥库位置

用于 Kafka 客户端身份验证的客户端密钥库文件的 Amazon S3 位置。路径必须采用 s3://bucket/prefix/filename.jks 的形式。它必须以文件名和 .jks 扩展名结尾。

Kafka 客户端密钥库密码（可选）

用于访问提供的密钥库的密码。

Kafka 客户端密钥密码（可选）

密钥库可以由多个密钥组成，因此这是访问要与 Kafka 服务器端密钥一起使用的客户端密钥的密码。

适用于客户端身份验证的 Apache Kafka 连接属性

AWS Glue 支持简单身份验证和安全层（SASL）框架，用于在创建 Apache Kafka 连接时进行身份验证。SASL 框架支持各种身份验证机制，且 AWS Glue 提供 SCRAM（用户名和密码）、GSSAPI（Kerberos 协议）和 PLAIN 协议。

AWS Glue Studio 用于配置以下客户机身份验证方法之一。有关更多信息，请参阅 AWS Glue Studio 用户指南中的为连接器创建连接。

- 无 – 不进行身份验证。如果是为进行测试而创建连接，这非常有用。
- SASL/SCRAM-SHA-512 – 选择此身份验证方法将允许您指定身份验证凭证。有两个可用的选项：

  - 使用 S AWS ecrets Manager（推荐）-如果您选择此选项，则可以将您的用户名和密码存储在 S AWS ecrets Manager 中，并在需要时允许AWS Glue访问它们。指定存储 SSL 或 SASL 身份验证凭证的密钥。有关更多信息，请参阅 在 AWS Secrets Manager 中存储连接凭证。
  - 直接提供用户名和密码。
- SASL/GSSAPI (Kerberos) – 如果选择此选项，则可以选择 keytab 文件、krb5.conf 文件的位置，然后输入 Kerberos 主体名称和 Kerberos 服务名称。keytab 文件和 krb5.conf 文件的位置必须位于 Amazon S3 位置。由于 MSK 尚不支持 SASL/GSSAPI，所以此选项仅适用于客户管理的 Apache Kafka 集群。有关更多信息，请参阅 MIT Kerberos 文档：keytab。
- SASL/PLAIN - 选择此身份验证方法以指定身份验证凭证。有两个可用的选项：
  - 使用 S AWS ecrets Manager（推荐）-如果您选择此选项，则可以将您的凭据存储在 S AWS ecrets Manager 中，并在需要时允许 AWS Glue 访问这些信息。指定存储 SSL 或 SASL 身份验证凭证的密钥。
  - 直接提供用户名和密码。
- SSL 客户端身份验证 – 如果选择此选项，则可以通过浏览 Amazon S3 来选择 Kafka 客户端密钥库的位置。或者，您可以输入 Kafka 客户端密钥库密码和 Kafka 客户端密钥密码。

谷歌 BigQuery 连接

以下属性用于设置在 AWS Glue ETL 作业中使用的 Google BigQuery 连接。有关更多信息，请参阅 the section called “BigQuery 连接”。

AWS 秘密

中密钥的秘密名称 AWS Secrets Manager。 AWS Glue ETL 作业将 BigQuery 使用您的密`credentials`钥连接到 Google。

Vertica 连接

以下属性用于设置在 AWS Glue ETL 作业中使用的垂直连接。有关更多信息，请参阅 the section called “Vertica 连接”。

在 AWS Secrets Manager 中存储连接凭证

我们建议您使用 AWS Secrets Manager 提供数据存储的连接凭证。以这种方式使用 Secrets Manager 可让 AWS Glue 在运行时访问您的密钥，以运行 ETL 任务和爬网程序，并帮助确保您的凭证安全。

先决条件

要将 Secrets Manager 与 AWS Glue 一起使用，您必须授予 AWS Glue 的 IAM 角色权限才能检索密钥值。AWS 托管策略 `AWSGlueServiceRole` 不包含 AWS Secrets Manager 权限。有关示例 IAM 策略，请参阅《AWS Secrets Manager 用户指南》中的 Example: Permission to retrieve secret values（示例：检索密钥值的权限）。

根据网络设置，您可能还需要创建一个 VPC 端点，以在 VPC 与 Secrets Manager 之间建立私有连接。有关更多信息，请参阅使用 AWS Secrets Manager VPC 端点。

为 AWS Glue 创建密钥

1. 请遵循《AWS Secrets Manager 用户指南》中的创建和管理密钥中的说明。以下示例 JSON 展示了在为 AWS Glue 创建密钥时如何在 Plaintext（明文）选项卡中指定凭证。

```
{
  "username": "EXAMPLE-USERNAME",
  "password": "EXAMPLE-PASSWORD"
}
```

2. 使用 AWS Glue Studio 界面将密钥与连接相关联。有关详细信息，请参阅《AWS Glue Studio 用户指南》中的 Creating connections for connectors。

添加 AWS Glue 连接

您可以通过编程方式连接到 AWS Glue for Spark 中的数据来源。有关更多信息，请参阅 AWS Glue for Spark 中适用于 ETL 的连接类型和选项。

您还可以使用 AWS Glue 控制台来添加、编辑、删除和测试连接。有关 AWS Glue 连接的信息，请参阅连接到数据。

添加 AWS Glue 连接

1. 登录 AWS Management Console 并打开 AWS Glue 控制台，网址为 https://console.aws.amazon.com/glue/。
2. 在导航窗格的 Data catalog (数据目录) 下，选择 Connections (连接)。
3. 选择 Add connection (添加连接)，然后完成向导，并输入the section called “AWS Glue 连接属性”中所述的连接属性。

正在连接亚马逊 Redshift AWS Glue Studio

> **Note**
>
> 你可以使用 f AWS Glue or Spark 对外部 Amazon Redshift 数据库中的表进行读取和写入AWS Glue Studio。要以编程方式配置 Amazon Redshift AWS Glue作业，请参阅Redshift 连接。

AWS Glue提供对的内置支持 Amazon Redshift。 AWS Glue Studio提供了一个可视化界面 Amazon Redshift，用于连接、创作数据集成作业并在AWS Glue Studio无服务器 Spark 运行时上运行这些作业。

主题

创建 Amazon Redshift 连接

所需权限

使用 Amazon Redshift 集群和 Amazon Redshift 无服务器环境需要额外的权限。有关如何为 ETL 作业添加权限的更多信息，请参阅 Review IAM permissions needed for ETL jobs。

- 红移：DescribeClusters
- redshift-serverl ListWorkgroups
- redshift-serverl ListNamespaces

概述

添加连接时，您可以选择现有 Amazon Redshift Amazon Redshift 连接，也可以在中添加数据源-Redshift 节点时创建新连接。AWS Glue Studio

AWS Glue同时支持 Amazon Redshift 集群和 Amazon Redshift 无服务器环境。创建连接时， Amazon Redshift 无服务器环境会在连接选项旁边显示无服务器标签。

有关如何创建 Amazon Redshift 连接的更多信息，请参阅将数据移入和移出 Amazon Redshift。

创建 Amazon Redshift 源节点

所需权限

使用 Amazon Redshift 数据来源的 AWS Glue Studio 作业需要额外的权限。有关如何为 ETL 作业添加权限的更多信息，请参阅 Review IAM permissions needed for ETL jobs。

要使用 Amazon Redshift 连接，需要以下权限。

- redshift-data:ListSchemas
- redshift-data:ListTables
- redshift-data:DescribeTable
- redshift-data:ExecuteStatement
- redshift-data:DescribeStatement
- redshift-data:GetStatementResult

添加 Amazon Redshift 数据来源

要添加数据来源 - Amazon Redshift 节点，请执行以下操作：

1. 选择 Amazon Redshift 访问类型：
   - 直接数据连接（推荐）- 如果您想直接访问 Amazon Redshift 数据，请选择此选项。这是推荐的选项，也是默认选项。
   - Data Catalog tables - 如果您有要使用的 Data Catalog 表，请选择此选项。
2. 如果您选择直接数据连接，请为 Amazon Redshift 数据来源选择连接。这假设该连接已经存在，并且您可以从现有连接中进行选择。如果需要创建连接，请选择创建 Redshift 连接。有关更多信息，请参阅 Overview of using connectors and connections。

   选择连接后，您可以通过单击查看属性来查看连接属性。可以看到有关连接的信息，包括 URL、安全组、子网、可用区、描述以及创建时间（UTC）和上次更新时间（UTC）时间戳。
3. 选择 Amazon Redshift 来源选项：
   - 选择单个表 - 该表包含您要从单个 Amazon Redshift 表中访问的数据。
   - 输入自定义查询 - 允许您根据自定义查询访问多个 Amazon Redshift 表中的数据集。

4. 如果您选择了单个表，请选择 Amazon Redshift 架构。可供选择的可用架构列表由所选表决定。

   或者，选择输入自定义查询。选择此选项可访问多个 Amazon Redshift 表中的自定义数据集。选择此选项后，输入 Amazon Redshift 查询。

   连接到 Amazon Redshift 无服务器环境时，请向自定义查询添加以下权限：

   ```
   GRANT SELECT ON ALL TABLES IN <schema> TO PUBLIC
   ```

   您可以选择推断架构，根据您输入的查询来读取架构。您也可以选择打开 Redshift 查询编辑器来输入 Amazon Redshift 查询。有关更多信息，请参阅 Querying a database using the query editor。
5. 在性能和安全中，选择 Amazon S3 暂存目录和 IAM 角色。
   - Amazon S3 暂存目录 — 选择用于临时暂存数据的 Amazon S3 位置。
   - IAM 角色 - 选择可以写入您选择的 Amazon S3 位置的 IAM 角色。
6. 在自定义 Redshift 参数 - 可选中，输入参数和值。

创建 Amazon Redshift 目标节点

所需权限

AWS Glue Studio 使用 Amazon Redshift 数据目标的作业需要额外的权限。有关如何为 ETL 作业添加权限的更多信息，请参阅 Review IAM permissions needed for ETL jobs。

要使用 Amazon Redshift 连接，需要以下权限。

- redshift-data ListSchemas
- redshift-data ListTables

添加 Amazon Redshift 目标节点

要创建 Amazon Redshift 目标节点，请执行以下操作：

1. 选择现有 Amazon Redshift 表作为目标，或输入新的表名。
2. 使用数据目标 - Redshift 目标节点时，可以从以下选项中进行选择：

- 附加 - 如果表已经存在，则将所有新数据作为插入内容转储到表中。如果表不存在，请创建表并插入所有新数据。

  此外，如果要更新（更新插入）目标表中的现有记录，请选中该复选框。该表必须先存在，否则操作将失败。

- 合并 - AWS Glue 将根据您指定的条件更新数据或将数据附加到目标表。

  > **Note**
  >
  > 要在中使用合并操作AWS Glue，必须启用 Amazon Redshift 合并功能。有关如何为您的 Amazon Redshift 实例启用合并的说明，请参阅合并（预览）。

  选择选项：

  - 选择键和简单操作 - 选择要用作源数据和目标数据集之间匹配键的列。

    匹配时指定以下选项：

    - 使用源数据更新目标数据集中的记录。
    - 删除目标数据集中的记录。

    如果不匹配，请指定以下选项：

    - 将源数据作为新行插入目标数据集。
    - 不执行任何操作。
  - 输入自定义 MERGE 语句 - 然后，您可以选择验证合并语句来验证该语句是有效还是无效。

- 截断 - 如果表已经存在，请先清除目标表的内容，从而截断表数据。如果截断成功，则插入所有数据。如果表不存在，请创建表并插入所有数据。如果截断不成功，操作将失败。
- 丢弃 - 如果表已存在，则删除表的元数据和数据。如果删除成功，则插入所有数据。如果表不存在，请创建表并插入所有数据。如果丢失不成功，操作将失败。
- 创建 - 使用默认名称创建新表。如果表名已经存在，请创建一个名称后缀为 `job_datetime` 的新表以保持唯一性。这会将所有数据插入到新表中。如果该表存在，则最终的表名将附加后缀。如果该表不存在，则将创建一个表。无论哪种情况，都将创建一个新表。

高级选项

请参阅 Using the Amazon Redshift Spark connector on AWS Glue。

在 AWS Glue Studio 中连接到 Azure Cosmos DB

AWS Glue 提供了对 Azure Cosmos DB 的内置支持。AWS Glue Studio 提供了直观的界面，以用于连接到 Azure Cosmos DB for NoSQL、编写数据集成作业以及在 AWS Glue Studio 无服务器 Spark 运行时系统上运行这些作业。

主题

创建 Azure Cosmos DB 连接

先决条件：

- 在 Azure 中，您需要确定或生成一个 Azure Cosmos DB 密钥 `cosmosKey`，以供 AWS Glue 使用。有关更多信息，请参阅 Azure 文档中的 保护对 Azure Cosmos DB 中数据的访问。

配置 Azure Cosmos DB 连接：

1. 在 AWS Secrets Manager 中，使用您的 Azure Cosmos DB 密钥创建一个密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *`secretName`*，以供下一步使用。
   - 在选择键/值对时，请使用键 `spark.cosmos.accountKey` 和值 *`cosmosKey`* 创建一个键值对。
2. 在 AWS Glue 控制台中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建一个连接。创建连接后，保留连接名为 *`connectionName`*，以供未来在 AWS Glue 中使用。
   - 选择连接类型时，请选择 Azure Cosmos DB。
   - 选择 AWS 密钥时，请提供 *`secretName`*。

创建 Azure Cosmos DB 源节点

所需的先决条件

- 使用 AWS Secrets Manager 密钥配置的 AWS Glue Azure Cosmos DB 连接，如上一节“the section called “创建 Azure Cosmos DB 连接””中所述。
- 对您的作业具有读取连接使用的密钥的相应权限。
- 您要读取的 Azure Cosmos DB for NoSQL 容器。您将需要该容器的标识信息。

  Azure Cosmos DB for NoSQL 容器由其数据库和容器来标识。在连接到 Azure Cosmos for NoSQL API 时，您必须提供数据库 *cosmosDBName* 和容器 *cosmosContainerName* 的名称。

添加 Azure Cosmos DB 数据来源

添加数据来源 – Azure Cosmos DB 节点：

1. 选择 Azure Cosmos DB 数据来源的连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 Azure Cosmos DB 连接。有关更多信息，请参阅之前的 the section called “创建 Azure Cosmos DB 连接” 部分。

   选择连接后，您可以通过单击查看属性来查看连接属性。
2. 选择 Cosmos DB 数据库名称 – 提供您要读取的数据库的名称 *cosmosDBName*。
3. 选择 Azure Cosmos DB 容器 – 提供您要读取的容器的名称 *cosmosContainerName*。
4. 或者，选择 Azure Cosmos DB 自定义查询 – 提供 SQL SELECT 查询以检索 Azure Cosmos 数据库中的特定信息。
5. 在自定义 Azure Cosmos 属性中，根据需要输入相关参数和值。

创建 Azure Cosmos DB 目标节点

所需的先决条件

- 使用 AWS Secrets Manager 密钥配置的 AWS Glue Azure Cosmos DB 连接，如上一节“the section called “创建 Azure Cosmos DB 连接””中所述。
- 对您的作业具有读取连接使用的密钥的相应权限。
- 您要写入的 Azure Cosmos DB 表。您将需要该容器的标识信息。必须首先创建容器，然后才能调用连接方法。

Azure Cosmos DB for NoSQL 容器由其数据库和容器来标识。在连接到 Azure Cosmos for NoSQL API 时，您必须提供数据库 *cosmosDBName* 和容器 *cosmosContainerName* 的名称。

添加 Azure Cosmos DB 数据目标

添加数据目标 – Azure Cosmos DB 节点：

1. 选择 Azure Cosmos DB 数据来源的连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 Azure Cosmos DB 连接。有关更多信息，请参阅之前的 the section called “创建 Azure Cosmos DB 连接” 部分。

   选择连接后，您可以通过单击查看属性来查看连接属性。
2. 选择 Cosmos DB 数据库名称 – 提供您要读取的数据库的名称 *cosmosDBName*。
3. 选择 Azure Cosmos DB 容器 – 提供您要读取的容器的名称 *cosmosContainerName*。
4. 在自定义 Azure Cosmos 属性中，根据需要输入相关参数和值。

高级选项

您可以在创建 Azure Cosmos DB 节点时提供高级选项。这些选项与编程 Spark 脚本的 AWS Glue 时可用的选项相同。

请参阅 the section called “Azure Cosmos DB 连接”。

在 AWS Glue Studio 中连接到 Azure SQL

AWS Glue 提供了对 Azure SQL 的内置支持。AWS Glue Studio 提供了直观的界面，以用于连接到 Azure SQL、编写数据集成作业以及在 AWS Glue Studio 无服务器 Spark 运行时系统上运行这些作业。

主题

- 创建 Azure SQL 连接
- 创建 Azure SQL 源节点
- 创建 Azure SQL 目标节点
- 高级选项

创建 Azure SQL 连接

要从 AWS Glue 连接到 Azure SQL，您需要创建 Azure SQL 凭证并将其存储在一个 AWS Secrets Manager 密钥中，然后将该密钥关联到某个 Azure SQL AWS Glue 连接。

配置 Azure SQL 连接：

1. 在 AWS Secrets Manager 中，使用您的 Azure SQL 凭证创建密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *secretName*，以供下一步使用。

   - 在选择键/值对时，请使用键 `user` 和值 *azuresqlUsername* 创建一个键值对。
   - 在选择键/值对时，请使用键 `password` 和值 *azuresqlPassword* 创建一个键值对。

2. 在 AWS Glue 控制台中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建一个连接。创建连接后，保留连接名为 *connectionName*，以供未来在 AWS Glue 中使用。

   - 选择连接类型时，请选择 Azure SQL。
   - 在提供 Azure SQL URL 时，请提供 JDBC 端点的 URL。

     该 URL 必须为以下格式：`jdbc:sqlserver://`*databaseServerName*`:`*databasePort*`;databaseName=`*azuresqlDB*

     AWS Glue 需要以下 URL 属性：

     - `databaseName` – Azure SQL 中要连接的默认数据库。

     有关 Azure SQL 托管实例的 JDBC URL 的更多信息，请参阅 Microsoft 文档。
   - 选择 AWS 密钥时，请提供 *secretName*。

创建 Azure SQL 源节点

所需的先决条件

- 使用 AWS Secrets Manager 密钥配置的 AWS Glue Azure SQL 连接，如上一节“the section called “创建 Azure SQL 连接””中所述。
- 对您的作业具有读取连接使用的密钥的相应权限。
- 您要读取的 Azure SQL 表 *tableName*。

Azure SQL 表由其数据库、Schema 和表名来标识。连接到 Azure SQL 时，必须提供数据库名和表名。如果 Schema 不是默认值“public”，则还必须提供 Schema。数据库通过 *connectionName* 中的 URL 属性来提供，Schema 和表名通过 `dbtable` 来提供。

添加 Azure SQL 数据来源

添加数据来源 – Azure SQL 节点：

1. 选择 Azure SQL 数据来源的连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 Azure SQL 连接。有关更多信息，请参阅之前的 the section called “创建 Azure SQL 连接” 部分。

   选择连接后，您可以通过单击查看属性来查看连接属性。

2. 选择 Azure SQL 源选项：

   - 选择单个表 – 访问单个表中的所有数据。
   - 输入自定义查询 - 允许您根据自定义查询访问多个表中的数据集。

3. 如果您选择单个表，请输入 *tableName*。

   如果选择输入自定义查询，请输入一个 TransactSQL SELECT 查询。

4. 在自定义 Azure SQL 属性中，根据需要输入相关参数和值。

创建 Azure SQL 目标节点

所需的先决条件

- 使用 AWS Secrets Manager 密钥配置的 AWS Glue Azure SQL 连接，如上一节“the section called “创建 Azure SQL 连接””中所述。
- 对您的作业具有读取连接使用的密钥的相应权限。
- 您要写入的 Azure SQL 表 *tableName*。

Azure SQL 表由其数据库、Schema 和表名来标识。连接到 Azure SQL 时，必须提供数据库名和表名。如果 Schema 不是默认值“public”，则还必须提供 Schema。数据库通过 *connectionName* 中的 URL 属性来提供，Schema 和表名通过 `dbtable` 来提供。

添加 Azure SQL 数据目标

添加数据目标 – Azure SQL 节点：

1. 选择 Azure SQL 数据来源的连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 Azure SQL 连接。有关更多信息，请参阅之前的 the section called “创建 Azure SQL 连接” 部分。

   选择连接后，您可以通过单击查看属性来查看连接属性。
2. 要配置表名，请提供 *tableName*。
3. 在自定义 Azure SQL 属性中，根据需要输入相关参数和值。

高级选项

您可以在创建 Azure SQL 节点时提供高级选项。这些选项与编程 Spark 脚本的 AWS Glue 时可用的选项相同。

请参阅 the section called “Azure SQL 连接”。

正在连接谷歌 BigQuery AWS Glue Studio

> **Note**
>
> 在 AWS Glue 4.0 及AWS Glue更高版本中，你可以使用 Spark 在 Google BigQuery 中读取和写入表。要以编程方式为 G BigQuery oogle 配置AWS Glue作业，请参阅 BigQuery 连接。

AWS Glue Studio提供了一个可视化界面 BigQuery，用于连接、创作数据集成作业并在AWS Glue Studio无服务器 Spark 运行时上运行这些作业。

主题

- 创建 BigQuery 连接
- 创建 BigQuery 源节点
- 创建 BigQuery 目标节点
- 高级选项

创建 BigQuery 连接

要从 AWS Glue 中连接到 Google BigQuery，您需要创建 Google Cloud Platform 凭证并将其存储在 AWS Secrets Manager 密钥中，然后将该密钥与 Google BigQuery AWS Glue 连接关联。

要配置与 BigQuery 的连接：

1. 在 Google Cloud Platform，创建并识别相关资源：
   - 创建或标识包含您想要连接的 BigQuery 表的 GCP 项目。
   - 启用 BigQuery API。有关更多信息，请参阅使用 BigQuery 存储读取 API 读取表数据。
2. 在 Google Cloud Platform 中，创建和导出服务账户凭证：

   您可以使用 BigQuery 凭证向导来加快此步骤：创建凭证。

   要在 GCP 中创建服务账户，请按照创建服务账户中的教程进行操作。

   - 选择项目时，请选择包含您的 BigQuery 表的项目。
   - 为您的服务账户选择 GCP IAM 角色时，请添加或创建一个角色，该角色将授予运行 BigQuery 作业的相应权限，以读取、写入或创建 BigQuery 表。

   要为您的服务账户创建凭证，请按照创建服务账户密钥中的教程进行操作。

   - 在选择密钥类型时，请选择 JSON。

   现在，您应该已经下载了包含服务账户凭证的 JSON 文件。如下所示：

```
{
  "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. base64 对您下载的凭证文件进行编码。在 AWS CloudShell 会话或类似会话中，您可以通过运行 `cat credentialsFile.json | base64 -w 0` 从命令行执行此操作。保留此命令的输出 *credentialString*。
4. 在 AWS Secrets Manager 中，使用您的 Google Cloud Platform 凭证创建密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *secretName*，以供下一步使用。
   - 在选择键/值对时，请使用值 *credentialString* 为键 `credentials` 创建对。
5. 在 AWS Glue Data Catalog 中，按照 https://docs.aws.amazon.com/glue/latest/dg/console-connections.html 中的步骤创建连接。创建连接后，保留连接名称 *connectionName*，以供下一步使用。
   - 在选择连接类型时，请选择 Google BigQuery。
   - 选择 AWS 密钥时，请提供 *secretName*。
6. 向与您的 AWS Glue 作业关联的 IAM 角色授予读取 *secretName* 的权限。
7. 在 AWS Glue 作业配置中，提供 *connectionName* 作为附加网络连接。

创建 BigQuery 源节点

所需的先决条件

- BigQuery 类型 AWS Glue Data Catalog 连接
- 您的 Google BigQuery 凭证的 AWS Secrets Manager 密钥，供连接使用。
- 对您的作业具有读取连接使用的密钥的相应权限。
- 您要读取的表的名称和数据集以及相应的 Google Cloud 项目。

添加 BigQuery 数据来源

要添加数据来源 – BigQuery 节点，请执行以下操作：

1. 为您的 BigQuery 数据来源选择连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 BigQuery 连接。有关更多信息，请参阅 Overview of using connectors and connections。

   选择连接后，您可以通过单击查看属性来查看连接属性。
2. 确定您想要读取的 BigQuery 数据，然后选择 BigQuery 来源选项

- 选择单个表 – 允许您从表中提取所有数据。
- 输入自定义查询 – 允许您通过提供查询来自定义检索哪些数据。

3. 描述您想要读取的数据

   （必填）将父项目设置为包含您的表的项目，或计费父项目（如果相关）。

   如果您选择单个表，则请按以下格式将表设置为 Google BigQuery 表的名称：`[dataset].[table]`

   如果您选择了查询，则请将其提供给 Query。在查询中，引用具有完全限定表名的表，使用的格式为：`[project].[dataset].[tableName]`。

4. 提供 BigQuery 属性

   如果选择了单个表，则无需提供其他属性。

   如果选择了查询，则必须提供以下自定义 Google BigQuery 属性：

   - 将 `viewsEnabled` 设置为 true。
   - 将 `materializationDataset` 设置为数据集。通过 AWS Glue 连接提供的凭证进行身份验证的 GCP 主体必须能够在此数据集中创建表。

创建 BigQuery 目标节点

所需的先决条件

- BigQuery 类型 AWS Glue Data Catalog 连接
- 您的 Google BigQuery 凭证的 AWS Secrets Manager 密钥，供连接使用。
- 对您的作业具有读取连接使用的密钥的相应权限。
- 您要写入到的表和相应的 Google Cloud 项目的名称和数据集。

添加 BigQuery 数据目标

要添加数据目标 – BigQuery 节点，请执行以下操作：

1. 为您的 BigQuery 数据目标选择连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 BigQuery 连接。有关更多信息，请参阅 Overview of using connectors and connections。

选择连接后，您可以通过单击查看属性来查看连接属性。

2. 确定您要写入的 BigQuery 表，然后选择写入方法。

   - 直接 – 使用 BigQuery 存储写入 API 直接写入 BigQuery。
   - 间接 – 写入 Google Cloud Storage，然后复制到 BigQuery。

   如果您想间接写入，请使用临时 GCS 存储桶提供目标 GCS 位置。您需要在 AWS Glue 连接中提供其他配置。有关更多信息，请参阅在 Google BigQuery 中使用间接写入。

3. 描述您想要读取的数据

   （必填）将父项目设置为包含您的表的项目，或计费父项目（如果相关）。

   如果您选择单个表，则请按以下格式将表设置为 Google BigQuery 表的名称：`[dataset].[table]`

高级选项

您可以在创建 BigQuery 节点时提供高级选项。这些选项与编程 AWS Glue Spark 脚本时可用的选项相同。

请参阅 AWS Glue 开发者指南中的 BigQuery 连接选项参考。

在 AWS Glue Studio 中连接到 MongoDB

AWS Glue 提供了对 MongoDB 的内置支持。AWS Glue Studio 提供了直观的界面，以用于连接到 MongoDB、编写数据集成作业以及在 AWS Glue Studio 无服务器 Spark 运行时系统上运行这些作业。

主题

创建 MongoDB 连接

先决条件：

- 如果您的 MongoDB 实例位于某个 Amazon VPC 中，请确保您的 Amazon VPC 配置允许您的 AWS Glue 作业与 MongoDB 实例进行通信，并且无需通过公共互联网路由流量。

  在 Amazon VPC 中，确定或创建 AWS Glue 将在执行作业时使用的 VPC、子网和安全组。此外，您的 Amazon VPC 配置需要允许您的 MongoDB 实例与该位置之间的网络流量。根据您的网络布局，这可能需要更改安全组规则、网络 ACL、NAT 网关和对等连接。

配置 MongoDB 连接：

1. 您还可以在 AWS Secrets Manager 中使用您的 MongoDB 凭证创建密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *secretName*，以供下一步使用。

   - 在选择键/值对时，请使用键 `username` 和值 *mongodbUser* 创建一个键值对。

     在选择键/值对时，请使用键 `password` 和值 *mongodbPass* 创建一个键值对。

2. 在 AWS Glue 控制台中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建一个连接。创建连接后，保留连接名为 *connectionName*，以供未来在 AWS Glue 中使用。

   - 选择连接类型时，请选择 MongoDB 或 MongoDB Atlas。
   - 选择 MongoDB URL 或 MongoDB Atlas URL 时，请提供 MongoDB 实例的主机名。

     MongoDB URL 的格式为 `mongodb://`*mongoHost*`:`*mongoPort*`/`*mongoDBname*。

     MongoDB Atlas URL 的格式为 `mongodb+srv://`*mongoHost*`:`*mongoPort*`/`*mongoDBname*。

     此外还可以选择提供连接的默认数据库 *mongoDBName*。
   - 如果您选择创建 Secrets Manager 密钥，请选择 AWS Secrets Manager 凭证类型。

     然后在 AWS密钥中提供 *secretName*。
   - *########## mongodbUser # mongodbPass*。

3. 对于下列情况，您可能需要添加额外的配置：

   - 对于通过 Amazon VPC 在 AWS 云端托管的 MongoDB 实例
     - 您需要向 AWS Glue 连接提供用于定义 MongoDB 安全凭证的 Amazon VPC 连接信息。创建或更新连接时，请在网络选项中设置 VPC、子网和安全组。

创建 AWS Glue MongoDB 连接后，您需要完成以下操作，然后才能运行 AWS Glue 作业：

- 在可视化编辑器中处理 AWS Glue 作业时，您必须提供作业的 Amazon VPC 连接信息，才能连接到 MongoDB。确定 Amazon VPC 中的适当位置并将其提供给您的 AWS Glue MongoDB 连接。
- 如果您选择创建 Secrets Manager 密钥，请向与您的 AWS Glue 作业关联的 IAM 角色授予读取 *secretName* 的权限。

创建 MongoDB 源节点

所需的先决条件

- 一个 AWS Glue MongoDB 连接，如上一节“the section called “创建 MongoDB 连接””所述。
- 如果您选择创建 Secrets Manager 密钥，则需要提供对您的作业具有读取连接使用的密钥的相应权限。
- 您要读取的 MongoDB 集合。您将需要该集合的标识信息。

  MongoDB 集合由数据库名 *mongodbName* 和集合名 *mongodbCollection* 来标识。

添加 MongoDB 数据来源

添加数据来源 - MongoDB 节点：

1. 选择 MongoDB DB 数据来源的连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 MongoDB 连接。有关更多信息，请参阅之前的 the section called “创建 MongoDB 连接” 部分。

   选择连接后，您可以通过单击查看属性来查看连接属性。
2. 选择一个数据库。输入 *mongodbName*。
3. 选择一个集合。进入 *mongodbCollection*。
4. 选择分区程序、分区大小（MB）和分区键。有关分区参数的更多信息，请参阅 the section called “"connectionType": "mongodb" as Source”。
5. 在自定义 Azure MongoDB 属性中，根据需要输入相关参数和值。

创建 MongoDB 目标节点

所需的先决条件

- 使用 AWS Secrets Manager 密钥配置的 AWS Glue MongoDB 连接，如上一节“the section called “创建 MongoDB 连接””中所述。

- 对您的作业具有读取连接使用的密钥的相应权限。
- 您要写入的 MongoDB 表 *tableName*。

添加 MongoDB 数据目标

添加数据目标 - MongoDB 节点：

1. 选择 MongoDB DB 数据来源的连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 MongoDB 连接。有关更多信息，请参阅之前的 the section called “创建 MongoDB 连接” 部分。

   选择连接后，您可以通过单击查看属性来查看连接属性。
2. 选择一个数据库。输入 *mongodbName*。
3. 选择一个集合。进入 *mongodbCollection*。
4. 选择分区程序、分区大小（MB）和分区键。有关分区参数的更多信息，请参阅 the section called “"connectionType": "mongodb" as Source”。
5. 在需要时选择重试写入。
6. 在自定义 Azure MongoDB 属性中，根据需要输入相关参数和值。

高级选项

您可以在创建 MongoDB 节点时提供高级选项。这些选项与编程 Spark 脚本的 AWS Glue 时可用的选项相同。

请参阅 the section called “MongoDB 连接”。

在 AWS Glue Studio 中连接到 OpenSearch Service

AWS Glue 提供了对 Amazon OpenSearch Service 的内置支持。AWS Glue Studio 提供了直观的界面，以用于连接到 Amazon OpenSearch Service、编写数据集成作业以及在 AWS Glue Studio 无服务器 Spark 运行时系统上运行这些作业。此功能与 OpenSearch Service 无服务器不兼容。

主题

- 创建 OpenSearch Service 连接
- 创建 OpenSearch Service 源节点
- 创建 OpenSearch Service 目标节点

- 高级选项

创建 OpenSearch Service 连接

先决条件：

- 确定您要从中读取的域端点 *aosEndpoint* 和端口 *aosPort*，或者按照 Amazon OpenSearch Service 文档中的说明创建资源。有关更多信息，请参阅《Amazon OpenSearch Service 开发人员指南》中的 Creating and managing Amazon OpenSearch Service domains。

  Amazon OpenSearch Service 域端点的默认格式为 https://search-*domainName*-*unstructuredIdContent*.*region*.es.amazonaws.com。有关如何确定域端点的更多信息，请参阅《Amazon OpenSearch Service 开发人员指南》中的 Creating and managing Amazon OpenSearch Service domains。

  确定或生成域的 HTTP 基本身份验证凭证（*aosUser* 和 *aosPassword*）。

配置 OpenSearch Service 连接：

1. 在 AWS Secrets Manager 中，使用您的 OpenSearch Service 凭证创建一个密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *secretName*，以供下一步使用。

   - 在选择键/值对时，请使用键 `opensearch.net.http.auth.user` 和值 *aosUser* 创建一个键值对。
   - 在选择键/值对时，请使用键 `opensearch.net.http.auth.pass` 和值 *aosPassword* 创建一个键值对。

2. 在 AWS Glue 控制台中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建一个连接。创建连接后，保留连接名为 *connectionName*，以供未来在 AWS Glue 中使用。

   - 选择连接类型时，请选择 OpenSearch Service。
   - 选择域端点时，请提供 *aosEndpoint*。
   - 选择端口时，请提供 *aosPort*。
   - 选择 AWS 密钥时，请提供 *secretName*。

创建 OpenSearch Service 源节点

所需的先决条件

- 使用 AWS Secrets Manager 密钥配置的 AWS Glue OpenSearch Service 连接，如上一节“the section called “创建 OpenSearch Service 连接””中所述。
- 对您的作业具有读取连接使用的密钥的相应权限。
- 您要读取的 OpenSearch Service 索引 *aosIndex*。

添加 OpenSearch Service 数据来源

添加数据来源 – OpenSearch Service 节点：

1. 选择 OpenSearch Service 数据来源的连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 OpenSearch Service 连接。有关更多信息，请参阅之前的 the section called “创建 OpenSearch Service 连接” 部分。

   选择连接后，您可以通过单击查看属性来查看连接属性。
2. 提供索引，即您要读取的索引。
3. （可选）提供查询，即用于提供更多具体结果的 OpenSearch 查询。有关编写 OpenSearch 查询的更多信息，请参阅 the section called “从 OpenSearch Service 读取”。
4. 在自定义 OpenSearch Service 属性中，根据需要输入相关参数和值。

创建 OpenSearch Service 目标节点

所需的先决条件

- 使用 AWS Secrets Manager 密钥配置的 AWS Glue OpenSearch Service 连接，如上一节“the section called “创建 OpenSearch Service 连接””中所述。
- 对您的作业具有读取连接使用的密钥的相应权限。
- 您要写入的 OpenSearch Service 索引 *aosIndex*。

添加 OpenSearch Service 数据目标

添加数据目标 – OpenSearch Service 节点：

1. 选择 OpenSearch Service 数据来源的连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 OpenSearch Service 连接。有关更多信息，请参阅之前的 the section called “创建 OpenSearch Service 连接” 部分。

   选择连接后，您可以通过单击查看属性来查看连接属性。
2. 提供索引，即您要读取的索引。
3. 在自定义 OpenSearch Service 属性中，根据需要输入相关参数和值。

高级选项

您可以在创建 Amazon OpenSearch Service 节点时提供高级选项。这些选项与编程 Spark 脚本的 AWS Glue 时可用的选项相同。

请参阅 the section called “OpenSearch Service 连接”。

连接到 Salesforce AWS Glue Studio

Salesforce 提供客户关系管理 (CRM) 软件，可在销售、客户服务、电子商务等方面为您提供帮助。如果你是 Salesforce 用户，你可以 AWS Glue 连接到你的 Salesforce 账户。然后，你可以使用 Salesforce 作为 ETL 任务中的数据源或目标。运行这些任务可在 Salesforce 和 AWS 服务或其他支持的应用程序之间传输数据。

主题

- AWS Glue 支持 Salesforce
- 包含用于创建和使用连接的 API 操作的策略
- 配置销售队伍
- 配置 Salesforce 连接
- 从 Salesforce 实体读取
- 写信 Salesforce
- Salesforce 连接选项
- Salesforce 连接器的限制
- 为 Salesforce 设置 JWT 持有人 OAuth 流程

AWS Glue 支持 Salesforce

AWS Glue 按如下方式支持 Salesforce：

是否支持作为来源？

是。你可以使用 AWS Glue ETL 任务来查询 Salesforce 的数据。

支持作为目标吗？

是。你可以使用 AWS Glue ETL 将记录写入 Salesforce。

支持 Salesforce API 版本

支持以下 Salesforce API 版本

- v58.0
- v59.0
- v60.0

包含用于创建和使用连接的 API 操作的策略

以下示例策略描述了创建和使用连接所需 AWS 的 IAM 权限。如果您要创建新角色，请创建包含以下内容的策略：

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

您还可以使用以下 IAM 策略来允许访问：

- AWSGlueServiceRole— 授予访问代表您运行各种 AWS Glue 进程所需的资源的权限。这些资源包括 AWS Glue Amazon S3、IAM、 CloudWatch 日志和亚马逊 EC2。如果您遵循此策略中指定的资源的命名约定，则 AWS Glue 进程将具有所需的权限。此策略通常附加到在定义爬网程序、作业和开发终端节点时指定的角色。
- AWSGlueConsoleFullAccess— 当策略所关联的身份使用 AWS 管理控制台时，授予对 AWS Glue 资源的完全访问权限。如果遵循此策略中指定的资源的命名约定，则用户具有完全控制台功能。此策略通常附加到 AWS Glue 控制台的用户。

配置销售队伍

在使用 AWS Glue 向 Salesforce 传输数据或从中传输数据之前，必须满足以下要求：

最低要求

以下是最低要求：

- 你有一个 Salesforce 账户。
- 您的 Salesforce 账户已启用 API 访问权限。默认情况下，企业版、无限版、开发版和性能版的 API 访问处于启用状态。
- 您的 Salesforce 帐户允许您安装关联的应用程序。如果您无法使用此功能，请联系您的 Salesforce 管理员。有关更多信息，请参阅 Salesforce 帮助中的关联应用程序。

如果您满足这些要求，就可以 AWS Glue 连接到您的 Salesforce 帐户了。 AWS Glue 使用 AWS 托管互联应用程序处理其余需求。

适用于 Salesforce 的 AWS 托管互联应用程序

AWS 托管连接应用程序可帮助您以更少的步骤创建 Salesforce 连接。在 Salesforce 中，联网应用程序是一个框架，用于授权外部应用程序（例如 AWS Glue）访问您的 Salesforce 数据。

- 使用 AWS Glue 控制台创建 Salesforce 连接。
- 配置连接时，将 OAuth 授权类型设置为授权码。

配置 Salesforce 连接

配置 Salesforce 连接，请执行以下操作：

1. 在 S AWS ecrets Manager 中，创建一个包含以下详细信息的密钥：
   a. 对于 JWT_TOKEN 授权类型，密钥应包含 JWT_TOKEN 密钥及其值。
   b. 对于 AuthorizationCode 授权类型：对于客户托管的互联应用程序，密钥应包含`USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET`作为密钥的关联应用程序 Consumer Secret。对于 AWS 托管连接的应用程序，为空密或具有临时值的密钥。
   c. 注意：您必须在中为每个连接创建一个密钥 AWS Glue。
2. 在 AWS Glue 数据目录中，按照以下步骤创建连接：
   a. 选择连接类型时，请选择 Salesforce。
   b. 提供你要连接的 Salesforce 的 INSTANCE_URL。
   c. 提供 Salesforce 环境。
   d. 选择 AWS Glue 可以代入并有权执行以下操作的 AWS IAM 角色：
   e. 选择要用于连接的 OAuth2 授权类型。授权类型决定了如何与 Salesforce AWS Glue 通信以请求访问您的数据。您的选择会影响您在创建连接之前必须满足的要求。您可以选择以下任一类型：
      - JWT_BEARER 授权类型：此授权类型非常适合自动化场景，因为它允许使用Salesforce实例中特定用户的权限预先创建 JSON Web 令牌 (JWT)。创建者可以控制 JWT 的有效期。 AWS Glue 能够使用 JWT 获取用于调用 Salesforce API 的访问令牌。

        此流程要求用户在其 Salesforce 实例中创建连接的应用程序，以便为用户发放基于 JWT 的访问令牌。

        有关为 JWT 承载 OAuth 流程创建互联应用程序的信息，请参阅 OAuth 2.0 JWT 承载流程进行集成。 server-to-server 要使用连接 Salesforce 的应用程序设置 JWT 承载流程，请参阅。为 Salesforce 设置 JWT 持有人 OAuth 流程
      - AUTH@@ ORIZATION_CODE 授权类型：此授权类型被视为 “三足型” OAuth，因为它依赖于将用户重定向到第三方授权服务器来对用户进行身份验证。它用于通过 AWS Glue 控制台创建连接。默认情况下，创建连接的用户可以依赖 AWS Glue 已连接的应用程序（AWS Glue 托管客户端应用程序），在该应用程序中，除了 Salesforce 实例 URL 之外，他们无需提供任何与 OAuth 相关的信息。 AWS Glue 控制台会将用户重定向到 Salesforce，用户必须登录并允许 AWS Glue 请求的权限才能访问他们的 Salesforce 实例。

        用户仍然可以选择在 Salesforce 中创建自己的联网应用程序，并在通过 AWS Glue 控制台创建连接时提供自己的客户端 ID 和客户端密钥。在这种情况下，他们仍会被重定向到 Salesforce 以登录并授权 AWS Glue 访问其资源。

        此授权类型会生成刷新令牌和访问令牌。访问令牌的有效期很短，无需用户使用刷新令牌进行交互即可自动刷新。

有关为授权码 OAuth 流程创建关联应用程序的信息，请参阅为授权码和凭证流程配置已连接的应用程序。

f. 选择要用于此连接secretName的，然后放 AWS Glue 入令牌。

g. 如果要使用网络，请选择网络选项。向与您的 AWS Glue 任务关联的 IAM 角色授予读取权限secretName。

3. 向与您的 AWS Glue 任务关联的 IAM 角色授予读取权限secretName。

4. 在您的 AWS Glue 任务配置中，提供connectionName作为附加网络连接。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

从 Salesforce 实体读取

先决条件

你想读的 Salesforce SobJect。您将需要对象名称，例如Account或Case或Opportunity。

示例：

```
salesforce_read = glueContext.create_dynamic_frame.from_options(
    connection_type="salesforce",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "Account",
```

```
        "API_VERSION": "v60.0"
    }
```

对查询进行分区

NUM_PARTITIONS如果你想在 Spark 中使用并发LOWER_BOUND性PARTITION_FIELDUPPER_BOUND，你可以提供其他 Spark 选项、、和。使用这些参数，原始查询将被拆分为NUM_PARTITIONS多个子查询，这些子查询可以由 Spark 任务同时执行。

- PARTITION_FIELD：用于对查询进行分区的字段的名称。
- LOWER_BOUND：所选分区字段的包含下限值。

  对于时间戳字段，我们接受 Spark SQL 查询中使用的 Spark 时间戳格式。

  有效值的示例：

  ```
  "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：所选分区字段的唯一上限值。
- NUM_PARTITIONS：分区的数量。

例如：

```
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"
    }
```

写信 Salesforce

先决条件

你想写信给 Salesforce SobJect。您将需要对象名称，例如`Account`或`Case`或`Opportunity`。

Salesforce 连接器支持四种写入操作：

- INSERT
- UPSERT
- UPDATE
- 删除

使用`UPSERT`写入操作时，`ID_FIELD_NAMES`必须提供以指定记录的外部 ID 字段。

示例

```
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"
    }
```

Salesforce 连接选项

以下是 Salesforce 的连接选项：

- `ENTITY_NAME`（字符串）-（必填）用于读/写。你在 Salesforce 中的对象的名称。
- `API_VERSION`（字符串）-（必填）用于读/写。你要使用的 Salesforce Rest API 版本。
- `SELECTED_FIELDS`（列表<String>）-默认：空（选择*）。用于读取。要为对象选择的列。
- `FILTER_PREDICATE`（字符串）-默认：空。用于读取。它应该采用 Spark SQL 格式。
- `QUERY`（字符串）-默认：空。用于读取。完整的 Spark SQL 查询。
- `PARTITION_FIELD`（字符串）-用于读取。用于分区查询的字段。

- LOWER_BOUND（字符串）-用于读取。所选分区字段的包含下限值。
- UPPER_BOUND（字符串）-用于读取。所选分区字段的唯一上限值。
- NUM_PARTITIONS（整数）-默认：1。用于读取。要读取的分区数。
- IMPORT_DELETED_RECORDS（字符串）-默认：假。用于读取。在查询时获取删除记录。
- WRITE_OPERATION（字符串）-默认：插入。用于写入。值应为 INSERT、UPDATE、UPSERT、DELETE。
- ID_FIELD_NAMES（字符串）-默认：空。仅用于 UPSERT。

Salesforce 连接器的限制

以下是 Salesforce 连接器的限制：

- 我们只支持 Spark SQL，不支持 Salesforce SOQL。
- 不支持作业书签。

为 Salesforce 设置 JWT 持有人 OAuth 流程

有关启用与 OAuth 2.0 JSON 网络令牌 server-to-server 集成的信息，请参阅 Salesforce 公共文档。

创建 PEM 文件的证书/密钥对

创建 PEM 文件的证书/密钥对

```
openssl req -newkey rsa:4096 -new -nodes -x509 -days 3650 -keyout key.pem -out cert.pem
```

使用 JWT 创建与 Salesforce 关联的应用程序

1. 登录 Salesforce，点击右上角的设置齿轮，然后选择设置。
2. 在左侧，导航到应用程序管理器。（平台工具 > 应用程序 > 应用程序管理器）
3. 选择 “新建连接应用程序”。
4. 提供应用名称，让其余名称自动填充。
5. 选中 “启用 OAuth 设置” 复选框。
6. 设置回调网址。它不会用于 JWT，所以你可以使用 https://localhost。
7. 选中 “使用数字签名” 复选框。

8. 上传之前创建的 cert.pem 文件。
9. 添加所需的权限：
   a. 通过 API (api) 管理用户数据。
   b. 访问自定义权限（自定义权限）。
   c. 访问身份 URL 服务（ID、个人资料、电子邮件、地址、电话）。
   d. 访问唯一的用户标识符 (openid)。
   e. 随时执行请求（refresh_token、offline_access）。
10选中为指定用户发放基于 JSON 网络令牌 (JWT) 的访问令牌复选框。
11选择保存。
12选择继续。
13选择管理使用者详细信息。
14复制使用者密钥（客户端 ID）。
15复制消费者密钥（客户端密钥）。
16单击 Cancel (取消)。

生成 JSON 网络令牌 (JWT)

1. 将 key pair 转换为 pkcs12（出现提示时设置导出密码）。

```
openssl pkcs12 -export -in cert.pem -inkey key.pem -name jwtcert > jwtcert.p12
```

2. 从 pkcs12 创建 Java 密钥库（出现提示时设置目标密钥库密码，并为源密钥库密码提供以前的导出密码）。

```
keytool -importkeystore -srckeystore jwtcert.p12 -destkeystore keystore.jks -
srcstoretype pkcs12 -alias jwtcert
```

3. 确认 keystore.jks 包含 jwtcert 别名（出现提示时输入之前的目标密钥库密码）。

```
keytool -keystore keystore.jks -list
```

4. 使用 Salesforce 文档中提供的 Java 类 jwteXample 生成签名令牌。
   a. 根据需要编辑 ClaimArray 中的值：
      - ClaimArray [0] = 客户端 ID
      - ClaimArray [1] = salesforce 用户 ID

- ClaimArray [2] = 销售人员登录网址
- ClaimArray [4] = 自纪元以毫秒为单位的到期日期。3660624000000 是 2085-12-31。

b. 将 path/to/keystore 替换为密钥库的正确路径.jks。

c. 将密钥库密码替换为您输入的目标密钥库密码

d. 用您输入的源密钥库密码替换私钥密码

e. 编译该代码。该代码依赖于 Apache 共享资源编解码器进行 base6 4 编码。

```
javac -classpath ".:./commons-codec-1.16.1.jar" JWTExample.java
```

f. 运行该代码。

```
java -classpath ".:commons-codec-1.16.1.jar" JWTExample
```

5. 创建关联的应用程序和 JWT 后，用户仍需要获得应用程序的授权。有关两种方法，请参阅 https://mannharleen.github.io/2020-03-03-salesforce-jwt/ 中的步骤 3。

完成上述步骤后，这将输出一个 JSON 网络令牌 (JWT)，该令牌可用于从 Salesforce 获取访问令牌。

示例输入：

```
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
```

示例输出：

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0Ij
```

有用的链接：

- 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 中连接到 SAP HANA

AWS Glue 提供了对 SAP HANA 的内置支持。AWS Glue Studio 提供了直观的界面，以用于连接到 SAP HANA、编写数据集成作业以及在 AWS Glue Studio 无服务器 Spark 运行时系统上运行这些作业。

主题

- 创建 SAP HANA 连接
- 创建 SAP HANA 源节点
- 创建 SAP HANA 目标节点
- 高级选项

创建 SAP HANA 连接

要从 AWS Glue 连接到 SAP HANA，您需要创建 SAP HANA 凭证并将其存储在某个 AWS Secrets Manager 密钥中，然后将该密钥关联到某个 SAP HANA AWS Glue 连接。您需要配置 SAP HANA 服务与 AWS Glue 之间的网络连接。

先决条件：

- 如果您的 SAP HANA 服务位于某个 Amazon VPC 中，请确保您的 Amazon VPC 配置允许您的 AWS Glue 作业与 SAP HANA 服务进行通信，并且无需通过公共互联网路由流量。

  在 Amazon VPC 中，确定或创建 AWS Glue 将在执行作业时使用的 VPC、子网和安全组。此外，您的 Amazon VPC 配置需要允许您的 SAP HANA 端点与该位置之间的网络流量。您的作业需要与您的 SAP HANA JDBC 端口建立 TCP 连接。有关 SAP HANA 端口的更多信息，请参阅 SAP HANA 文档。根据您的网络布局，这可能需要更改安全组规则、网络 ACL、NAT 网关和对等连接。

配置 SAP HANA 连接：

1. 在 AWS Secrets Manager 中，使用您的 SAP HANA 凭证创建一个密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *secretName*，以供下一步使用。

   - 在选择键/值对时，请使用键 `user` 和值 *saphanaUsername* 创建一个键值对。
   - 在选择键/值对时，请使用键 `password` 和值 *saphanaPassword* 创建一个键值对。
2. 在 AWS Glue 控制台中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建一个连接。创建连接后，保留连接名为 *connectionName*，以供未来在 AWS Glue 中使用。

   - 选择连接类型时，请选择 SAP HANA。
   - 在提供 SAP HANA URL 时，请提供您的实例的 URL。

     SAP HANA JDBC URL 的格式为 `jdbc:sap://`*saphanaHostname*`:`*saphanaPort*`/?`*databaseName*`=`*saphanaDBname*`,`*Paramete*

     AWS Glue 需要以下 JDBC URL 参数：

     - `databaseName` – SAP HANA 中要连接的默认数据库。
   - 选择 AWS 密钥时，请提供 *secretName*。

创建 AWS Glue SAP HANA 连接后，您需要完成以下操作，然后才能运行 AWS Glue 作业：

- 向与您的 AWS Glue 作业关联的 IAM 角色授予读取 *secretName* 的权限。

创建 SAP HANA 源节点

所需的先决条件

- 使用 AWS Secrets Manager 密钥配置的 AWS Glue SAP HANA 连接，如上一节“the section called “创建 SAP HANA 连接””中所述。
- 对您的作业具有读取连接使用的密钥的相应权限。
- 您要读取的 SAP HANA 表 *tableName* 或查询 *targetQuery*。

  可以使用 SAP HANA 表名和 Schema 名（格式为 *schemaName*.*tableName*）来指定表。如果表的 Schema 为默认的“public”，则不需要指定 Schema 名和“.”分隔符。调用此 *tableIdentifier* 操作。请注意，数据库在 connectionName 中是以 JDBC URL 参数的形式提供的。

添加 SAP HANA 数据来源

添加数据来源 - SAP HANA 节点：

1. 选择 SAP HANA 数据来源的连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 SAP HANA 连接。有关更多信息，请参阅之前的 the section called “创建 SAP HANA 连接” 部分。

   选择连接后，您可以通过单击查看属性来查看连接属性。
2. 选择 SAP HANA 源选项：
   - 选择单个表 – 访问单个表中的所有数据。
   - 输入自定义查询 - 允许您根据自定义查询访问多个表中的数据集。
3. 如果您选择单个表，请输入 *tableName*。

   如果选择输入自定义查询，请输入一个 SQL SELECT 查询。
4. 在自定义 SAP HANA 属性中，根据需要输入相关参数和值。

创建 SAP HANA 目标节点

所需的先决条件

- 使用 AWS Secrets Manager 密钥配置的 AWS Glue SAP HANA 连接，如上一节“the section called “创建 SAP HANA 连接””中所述。
- 对您的作业具有读取连接使用的密钥的相应权限。
- 您要写入的 SAP HANA 表 *tableName*。

  可以使用 SAP HANA 表名和 Schema 名（格式为 *schemaName*.*tableName*）来指定表。如果表的 Schema 为默认的“public”，则不需要指定 Schema 名和“.”分隔符。调用此 *tableIdentifier* 操作。请注意，数据库在 connectionName 中是以 JDBC URL 参数的形式提供的。

添加 SAP HANA 数据目标

添加数据目标 - SAP HANA 节点：

1. 选择 SAP HANA 数据来源的连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 SAP HANA 连接。有关更多信息，请参阅之前的 the section called “创建 SAP HANA 连接” 部分。

   选择连接后，您可以通过单击查看属性来查看连接属性。
2. 要配置表名，请提供 *tableName*。
3. 在自定义 Teradata 属性中，根据需要输入相关参数和值。

高级选项

您可以在创建 SAP HANA 节点时提供高级选项。这些选项与编程 Spark 脚本的 AWS Glue 时可用的选项相同。

请参阅 the section called “SAP HANA 连接”。

在 AWS Glue Studio 连接到 Snowflake

> **Note**
>
> 在 AWS Glue 4.0 及更高版本中，您可以使用 AWS Glue for Spark 在 Snowflake 中读取和写入表。要以编程方式为 Snowflake 连接配置 AWS Glue 作业，请参阅 Redshift 连接。

AWS Glue 提供对 Snowflake 的内置支持。AWS Glue Studio 提供一个可视化界面，用于连接到 Snowflake、创作数据集成作业并在 AWS Glue Studio 无服务器 Spark 运行时上运行这些作业。

主题

创建 Snowflake 连接

在 AWS Glue Studio 中添加数据来源 - Snowflak 节点时，您可以选择现有的 AWS Glue Snowflake 连接或创建新连接。必须选择 SNOWFLAKE 类型连接，而不是配置为连接到 Snowflake 的 JDBC 类型连接。请按照以下过程创建 AWS Glue Snowflake 连接：

创建 Snowflake 连接

1. 在 Snowflake 中，生成一个用户 *snowflakeUser* 和密码 *snowflakePassword*。
2. 确定 Snowflake 仓库，该用户将与哪个 *snowflakeWarehouse* 互动。要么将其设置为 *snowflakeUser* 的 DEFAULT_WAREHOUSE，要么在下一步中记住它。
3. 在 AWS Secrets Manager 中，使用您的 Snowflake 凭证创建密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *secretName*，以供下一步使用。
   - 选择键/值对时，请使用键 sfUser 为 *snowflakeUser* 创建一对。
   - 选择键/值对时，请使用键 sfPassword 为 *snowflakePassword* 创建一对。
   - 选择键/值对时，请使用键 sfWarehouse 为 *snowflakeWarehouse* 创建一对。如果在 Snowflake 中设置了默认值，则不需要这样做。
4. 在 AWS Glue Data Catalog 中，按照添加 AWS Glue 连接中的步骤创建连接。创建连接后，保留连接名称 *connectionName*，以供下一步使用。
   - 选择连接类型时，请选择 Snowflake。
   - 选择 Snowflake URL 时，请提供您的 Snowflake 实例的主机名。URL 将使用表单 *account_identifier*.snowflakecomputing.com 中的主机名。
   - 选择 AWS 密钥时，请提供 *secretName*。

创建 Snowflake 源节点

所需权限

使用 Snowflake 数据来源的 AWS Glue Studio 作业需要额外的权限。有关如何为 ETL 作业添加权限的更多信息，请参阅 Review IAM permissions needed for ETL jobs。

SNOWFLAKE AWS Glue 连接使用 AWS Secrets Manager 密钥来提供凭证信息。您在 AWS Glue Studio 中的作业和数据预览角色必须具有读取此密钥的权限。

添加 Snowflake 数据来源

先决条件：

- 您的 Snowflake 凭证的 AWS Secrets Manager 密钥
- Snowflake 类型 AWS Glue Data Catalog 连接

要添加数据来源 — Snowflake 节点，请执行以下操作：

1. 为您的 Snowflake 数据来源选择连接。这假设该连接已经存在，并且您可以从现有连接中进行选择。如果需要创建连接，请选择创建 Snowflake 连接。有关更多信息，请参阅 Overview of using connectors and connections。

   选择连接后，您可以通过单击查看属性来查看连接属性。可以看到有关连接的信息，包括 URL、安全组、子网、可用区、描述以及创建时间（UTC）和上次更新时间（UTC）时间戳。
2. 选择 Snowflake 源选项：
   - 选择单个表 - 该表包含您要从单个 Snowflake 表中访问的数据。
   - 输入自定义查询 - 允许您根据自定义查询访问多个 Snowflake 表中的数据集。
3. 如果您选择单个表，请输入 Snowflake 架构的名称。

   或者，选择输入自定义查询。选择此选项可访问多个 Snowflake 表中的自定义数据集。选择此选项后，输入 Snowflake 查询。
4. 在性能和安全选项（可选）中，
   - 启用查询下推 — 选择是否要将工作卸载到 Snowflake 实例。
5. 在自定义 Snowflake 属性（可选）中，根据需要输入参数和值。

创建 Snowflake 目标节点

所需权限

使用 Snowflake 数据来源的 AWS Glue Studio 作业需要额外的权限。有关如何为 ETL 作业添加权限的更多信息，请参阅 Review IAM permissions needed for ETL jobs。

SNOWFLAKE AWS Glue 连接使用 AWS Secrets Manager 密钥来提供凭证信息。您在 AWS Glue Studio 中的作业和数据预览角色必须具有读取此密钥的权限。

添加 Snowflake 数据目标

要创建 Snowflake 目标节点，请执行以下操作：

1. 选择现有 Snowflake 表作为目标，或输入新的表名。
2. 使用数据目标 - Snowflake 目标节点时，可以从以下选项中进行选择：

   - 附加 - 如果表已经存在，则将所有新数据作为插入内容转储到表中。如果表不存在，请创建表并插入所有新数据。
   - 合并 - AWS Glue 将根据您指定的条件更新数据或将数据附加到目标表。

     选择选项：

     - 选择键和简单操作 - 选择要用作源数据和目标数据集之间匹配键的列。

       匹配时指定以下选项：

       - 使用源数据更新目标数据集中的记录。
       - 删除目标数据集中的记录。

       如果不匹配，请指定以下选项：

       - 将源数据作为新行插入目标数据集。
       - 不执行任何操作。

     - 输入自定义 MERGE 语句 - 然后，您可以选择验证合并语句来验证该语句是有效还是无效。
   - 截断 - 如果表已经存在，请先清除目标表的内容，从而截断表数据。如果截断成功，则插入所有数据。如果表不存在，请创建表并插入所有数据。如果截断不成功，操作将失败。
   - 丢弃 - 如果表已存在，则删除表的元数据和数据。如果删除成功，则插入所有数据。如果表不存在，请创建表并插入所有数据。如果丢失不成功，操作将失败。

高级选项

请参阅《AWS Glue 开发人员者指南》中的 Snowflake connections。

在 AWS Glue Studio 中连接到 Teradata Vantage

AWS Glue 提供对 Teradata Vantage 的内置支持。AWS Glue Studio 提供一个可视化界面，用于连接到 Teradata、编写数据集成作业并在 AWS Glue Studio 无服务器 Spark 运行时系统上运行这些作业。

主题

创建 Teradata Vantage 连接

要从 AWS Glue 连接到 Teradata Vantage，您需要创建 Teradata 凭证并将其存储在某个 AWS Secrets Manager 密钥中，然后将该密钥关联到某个 AWS Glue Teradata 连接。

先决条件：

- 如果您通过 Amazon VPC 访问您的 Teradata 环境，您的 Amazon VPC 配置应允许您的 AWS Glue 作业与 Teradata 环境进行通信。我们不建议通过公共互联网访问 Teradata 环境。

  在 Amazon VPC 中，确定或创建 AWS Glue 将在执行作业时使用的 VPC、子网和安全组。此外，您的 Amazon VPC 配置需要允许您的 Teradata 实例与该位置之间的网络流量。您的作业需要与您的 Teradata 客户端端口建立 TCP 连接。有关 Teradata 端口的更多信息，请参阅 Teradata 文档。

  根据您的网络布局，可能需要更改 Amazon VPC 和其他联网服务以建立安全的 VPC 连接。有关 AWS 连接的更多信息，请参阅 Teradata 文档中的 AWS Connectivity Options。

配置 AWS Glue Teradata 连接：

1. 在您的 Teradata 配置中，确定或创建 AWS Glue 将用于连接的用户 *teradataUser* 和密码 *teradataPassword*。有关更多信息，请参阅 Teradata 文档中的 Vantage Security Overview。

2. 在 AWS Secrets Manager 中，使用您的 Teradata 凭证创建一个密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *secretName*，以供下一步使用。

   - 在选择键/值对时，请使用键 `user` 和值 *teradataUsername* 创建一个键值对。
   - 在选择键/值对时，请使用键 `password` 和值 *teradataPassword* 创建一个键值对。

3. 在 AWS Glue 控制台中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建一个连接。创建连接后，保留连接名称 *connectionName*，以供下一步使用。

   - 选择连接类型时，请选择 Teradata。
   - 在提供 JDBC URL 时，请提供您的实例的 URL。您还可以在 JDBC URL 中将某些连接参数进行硬编码，并以逗号分隔。该 URL 必须符合以下格式：`jdbc:teradata://`*teradataHostname*`/`*ParameterName*`=`*ParameterValue*`,`*ParameterN*

     支持的 URL 参数包括：

     - `DATABASE` – 主机上默认要访问的数据库的名称。
     - `DBS_PORT` – 在非标准端口上运行时要使用的数据库端口。
   - 选择凭证类型时，请选择 AWS Secrets Manager，然后将 AWS 密钥设置为 *secretName*。

4. 对于下列情况，您可能需要添加额外的配置：

   - 对于通过 Amazon VPC 在 AWS 云端托管的 Teradata 实例
     - 您需要向 AWS Glue 连接提供用于定义 Teradata 安全凭证的 Amazon VPC 连接信息。创建或更新连接时，请在网络选项中设置 VPC、子网和安全组。

创建 Teradata 源节点

所需的先决条件

- 使用 AWS Secrets Manager 密钥配置的 AWS Glue Teradata Vantage 连接，如上一节“the section called “创建 Teradata Vantage 连接””中所述。
- 对您的作业具有读取连接使用的密钥的相应权限。
- 您要读取的 Teradata 表 *tableName* 或查询 *targetQuery*。

添加 Teradata 数据来源

添加数据来源 - Teradata 节点：

1. 选择 Teradata 数据来源的连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建新连接。有关更多信息，请参阅之前的 the section called “创建 Teradata Vantage 连接” 部分。

   选择连接后，您可以通过单击查看属性来查看连接属性。
2. 选择一个 Teradata 源选项：
   - 选择单个表 – 访问单个表中的所有数据。
   - 输入自定义查询 - 允许您根据自定义查询访问多个表中的数据集。
3. 如果您选择单个表，请输入 *tableName*。

   如果选择输入自定义查询，请输入一个 SQL SELECT 查询。
4. 在自定义 Teradata 属性中，根据需要输入相关参数和值。

创建 Teradata 目标节点

所需的先决条件

- 使用 AWS Secrets Manager 密钥配置的 AWS Glue Teradata Vantage 连接，如上一节“the section called “创建 Teradata Vantage 连接””中所述。
- 对您的作业具有读取连接使用的密钥的相应权限。
- 您要写入的 Teradata 表 *tableName*。

添加 Teradata 数据目标

添加数据目标 - Teradata 节点：

1. 选择 Teradata 数据来源的连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 Teradata 连接。有关更多信息，请参阅 Overview of using connectors and connections。

   选择连接后，您可以通过单击查看属性来查看连接属性。
2. 要配置表名，请提供 *tableName*。

3. 在自定义 Teradata 属性中，根据需要输入相关参数和值。

高级选项

您可以在创建 Teradata 节点时提供高级选项。这些选项与编程 Spark 脚本的 AWS Glue 时可用的选项相同。

请参阅 the section called “Teradata Vantage 连接”。

在 AWS Glue Studio 中连接到 Vertica

AWS Glue 提供对 Vertica 的内置支持。AWS Glue Studio 提供一个可视化界面，用于连接到 Vertica、编写数据集成作业并在 AWS Glue Studio 无服务器 Spark 运行时系统上运行这些作业。

主题

- 创建 Vertica 连接
- 创建 Vertica 源节点
- 创建 Vertica 目标节点
- 高级选项

创建 Vertica 连接

先决条件：

- 读取和写入数据库时用于临时存储的 Amazon S3 存储桶或文件夹，也称为 *tempS3Path*。

> **Note**
>
> 在 AWS Glue 任务数据预览中使用 Vertica 时，临时文件可能不会自动从 *tempS3Path* 中删除。为确保删除临时文件，请在数据预览窗格中选择结束会话，以直接结束数据预览会话。如果无法保证数据预览会话直接结束，请考虑将 Amazon S3 生命周期配置设置为删除旧数据。我们建议根据最大作业运行时间加一定的裕度移除已存在超过 49 小时的数据。有关配置 Amazon S3 生命周期的更多信息，请参阅 Amazon S3 文档中的 管理存储生命周期。

- 对您的 Amazon S3 路径具有适当权限，并且您可以将其关联到您的 AWS Glue 作业角色的 IAM policy。
- 如果您的 Vertica 实例位于某个 Amazon VPC 中，请确保您的 Amazon VPC 配置允许您的 AWS Glue 作业与 Vertica 实例进行通信，并且无需通过公共互联网路由流量。

在 Amazon VPC 中，确定或创建 AWS Glue 将在执行作业时使用的 VPC、子网和安全组。此外，您的 Amazon VPC 配置需要允许您的 Vertica 实例与该位置之间的网络流量。您的作业需要与您的 Vertica 客户端端口（默认为 5433）建立 TCP 连接。根据您的网络布局，这可能需要更改安全组规则、网络 ACL、NAT 网关和对等连接。

配置 Vertica 连接：

1. 在 AWS Secrets Manager 中，使用您的 Vertica 凭证 *verticaUsername* 和 *verticaPassword* 创建一个密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *secretName*，以供下一步使用。
   - 在选择键/值对时，请使用键 `user` 和值 *verticaUsername* 创建一个键值对。
   - 在选择键/值对时，请使用键 `password` 和值 *verticaPassword* 创建一个键值对。
2. 在 AWS Glue 控制台中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建一个连接。创建连接后，保留连接名称 *connectionName*，以供下一步使用。
   - 选择连接类型时，请选择 Vertica。
   - 选择 Vertica 主机时，请提供您安装了 Vertica 的主机名。
   - 选择 Vertica 端口时，请提供可用于访问 Vertica 安装的端口。
   - 选择 AWS 密钥时，请提供 *secretName*。
3. 对于下列情况，您可能需要添加额外的配置：
   - 对于通过 Amazon VPC 在 AWS 云端托管的 Vertica 实例
     - 向 AWS Glue 连接提供用于定义 Vertica 安全凭证的 Amazon VPC 连接信息。创建或更新连接时，请在网络选项中设置 VPC、子网和安全组。

您需要首先完成以下步骤，然后才能运行 AWS Glue 作业：

- 向与您的 AWS Glue 作业关联的 IAM 角色授予对 *tempS3Path* 的权限。
- 向与您的 AWS Glue 作业关联的 IAM 角色授予读取 *secretName* 的权限。

创建 Vertica 源节点

所需的先决条件

- 一个 Vertica 类型的 AWS Glue Data Catalog 连接 *connectionName* 和一个临时 Amazon S3 位置 *tempS3Path*，如上一节“the section called “创建 Vertica 连接””所述。
- 您要读取的 Vertica 表 *tableName* 或查询 *targetQuery*。

添加 Vertica 数据来源

添加数据来源 - Vertica 节点：

1. 选择 Vertica 数据来源的连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 Vertica 连接。有关更多信息，请参阅之前的 the section called “创建 Vertica 连接” 部分。

   选择连接后，您可以通过单击查看属性来查看连接属性。
2. 选择包含您的表的数据库。
3. 选择 Amazon S3 中的暂存区域，然后在 *tempS3Path* 中输入一个 S3A URI。
4. 选择 Vertica 源。
   - 选择单个表 – 访问单个表中的所有数据。
   - 输入自定义查询 - 允许您根据自定义查询访问多个表中的数据集。
5. 如果您选择单个表，请输入 *tableName*，此外还可以选择一个 Schema。

   如果选择输入自定义查询，请输入一个 SQL SELECT 查询，此外还可以选择一个 Schema。
6. 在自定义 Vertica 属性中，根据需要输入相关参数和值。

创建 Vertica 目标节点

所需的先决条件

- 一个 Vertica 类型的 AWS Glue Data Catalog 连接 *connectionName* 和一个临时 Amazon S3 位置 *tempS3Path*，如上一节“the section called “创建 Vertica 连接””所述。

添加 Vertica 数据目标

添加数据目标 - Vertica 节点：

1. 选择 Vertica 数据来源的连接。由于您已经创建了它，它应该提供在下拉列表中。如果需要创建连接，请选择创建 Vertica 连接。有关更多信息，请参阅之前的 the section called “创建 Vertica 连接” 部分。

   选择连接后，您可以通过单击查看属性来查看连接属性。
2. 选择包含您的表的数据库。
3. 选择 Amazon S3 中的暂存区域，然后在 *tempS3Path* 中输入一个 S3A URI。
4. 输入 *tableName*，此外还可以选择一个 Schema。
5. 在自定义 Vertica 属性中，根据需要输入相关参数和值。

高级选项

您可以在创建 Vertica 节点时提供高级选项。这些选项与编程 Spark 脚本的 AWS Glue 时可用的选项相同。

请参阅 the section called “Vertica 连接”。

使用带有 AWS Glue Studio 的连接器和连接

AWS Glue 使用 JDBC 连接为常用数据存储（例如 Amazon Redshift、Amazon Aurora、Microsoft SQL Server、MySQL、MongoDB 和 PostgreSQL）提供内置支持。AWS Glue 还允许您在数据提取、转换和加载（ETL）任务中使用自定义 JDBC 驱动程序。对于本地不支持的数据存储（如 SaaS 应用程序），您可以使用连接器。

连接器是一个可选代码包，可帮助访问 AWS Glue Studio 中的数据存储。您可以订阅 AWS Marketplace中提供的几个连接器。

创建 ETL 作业时，您可以使用原生支持的数据存储 AWS Marketplace、来自的连接器或您自己的自定义连接器。如果您使用连接器，您必须首先为连接器创建连接。连接包含连接到特定数据存储所需的属性。您将使用与 ETL 任务中的数据源和数据目标的连接。连接器和连接协同工作，方便访问数据存储。

主题

- 连接器和连接使用概览

- 将连接器添加到 AWS Glue Studio
- 可用连接
- 为连接器创建连接
- 使用自定义连接器编写任务
- 管理连接器和连接
- 开发自定义连接器
- AWS Glue Studio 中连接器和连接的使用限制

连接器和连接使用概览

连接包含连接到特定数据存储所需的属性。当您创建连接时，它将存储于 AWS Glue Data Catalog 中。选择一个连接器，然后创建基于该连接器的连接。

您可以为中不支持的数据存储订阅连接器 AWS Marketplace，然后在创建连接时使用这些连接器。开发人员还可以创建自己的连接器，您可以在创建连接时使用它们。

> **Note**
>
> 使用自定义连接 AWS Marketplace 器或中的连接器创建的连接AWS Glue Studio将显示在 AWS Glue控制台中，类型设置为UNKNOWN。

以下步骤介绍了 AWS Glue Studio 中连接器的总体使用流程。

1. 在中订阅连接器 AWS Marketplace，或者开发自己的连接器并将其上传到AWS Glue Studio。有关更多信息，请参阅 将连接器添加到 AWS Glue Studio。
2. 查看连接器使用信息。您可以在连接器产品页面上的 Usage (使用) 选项卡上找到此类信息。例如，如果您点击此产品页面 “Google AWS Glue 连接器” 上的 “用法” 选项卡 BigQuery，则可以在 “其他资源” 部分看到有关使用此连接器的博客链接。其他连接器可能包含指向 Overview (概览) 部分中说明的链接，正如适用于 AWS Glue 的 Cloudwatch Logs 连接器的连接器产品页面上所示。
3. 创建连接。您可以选择要使用的连接器并为连接提供附加信息，例如登录凭证、URI 字符串和 Virtual Private Cloud（VPC）信息。有关更多信息，请参阅 为连接器创建连接。
4. 为您的任务创建 IAM 角色。作业代入您在创建它时指定的 IAM 角色的权限。此 IAM 角色必须具有对数据存储进行身份验证、从中提取数据和向其写入数据所需的权限。
5. 创建 ETL 任务并配置 ETL 任务的数据源属性。按照自定义连接器提供程序的指示提供连接选项和身份验证信息。有关更多信息，请参阅 使用自定义连接器编写任务。

6. 添加转换或其他数据存储以自定义 ETL 任务，如带有 AWS Glue Studio 的 Visual ETL中所示。
7. 如果为数据目标使用连接器，请为 ETL 任务配置数据目标属性。按照自定义连接器提供程序的指示提供连接选项和身份验证信息。有关更多信息，请参阅 the section called “使用自定义连接器编写任务”。
8. 配置任务属性以自定义任务运行环境，如修改任务属性中所示。
9. 运行作业。

将连接器添加到 AWS Glue Studio

连接器是一段代码，便于您在数据存储和 AWS Glue 之间通信。您可以订阅中提供的连接器 AWS Marketplace，也可以创建自己的自定义连接器。

主题

- 订阅连接器 AWS Marketplace
- 创建自定义连接器

订阅连接器 AWS Marketplace

AWS Glue Studio便于从中添加连接器 AWS Marketplace。

要添加连接器， AWS Marketplace 请从到 AWS Glue Studio

1. 在 AWS Glue Studio 控制台中，在导航窗格中选择 Connectors (连接器)。
2. 在 Connectors (连接器) 页面上，选择 Go to AWS Marketplace(转到 Amazon Web Services Marketplace)。
3. 在 AWS Marketplace精选产品中，选择要使用的连接器。您可以选择其中一个特色连接器，也可以进行搜索。您可以搜索连接器的名称或类型，也可以使用选项来优化搜索结果。

   如果您要使用其中一个特色连接器，请选择 View product (查看产品)。如果通过搜索来查找连接器，请选择连接器的名称。
4. 在连接器的产品页面上，使用选项卡查看有关连接器的信息。如果您决定购买此连接器，请选择 Continue to Subscribe (继续订阅)。
5. 提供付款信息，然后选择 Continue to Configure (继续配置)。
6. 在 Configure this software (配置此软件) 页面上，选择部署方法和要使用的连接器版本。然后选择 Continue to Launch (继续启动) 以继续。

7. 在 Launch this software (启动此软件) 页面上，您可以查看连接器提供程序提供的 Usage Instructions (使用说明)。准备就绪后，选择 Activate connection in AWS Glue Studio (激活 AWS Glue Studio Glue Studio 中的连接)。

   一小段时间后，控制台将显示 AWS Glue Studio 中的 Create marketplace connection (创建 Marketplace 连接) 页面。
8. 创建使用此连接器的连接，如为连接器创建连接中所述。

   或者，您可以选择 Activate connector only (仅激活连接器) 以跳过此时创建连接。您稍后必须创建连接才能使用该连接器。

创建自定义连接器

您还可以构建自己的连接器，然后将连接器代码上载到 AWS Glue Studio。

自定义连接器通过 AWS Glue Spark 运行时 API 集成到 AWS Glue Studio。AWS Glue Spark 运行时允许您插入兼容 Spark、Athena 或 JDBC 接口的连接器。它允许您传递自定义连接器可用的连接选项。

您可以使用 AWS Glue 连接封装所有连接属性并为 ETL 任务提供连接名称。与数据目录连接集成，您可以在单个 Spark 应用程序中或跨不同应用程序的多个调用使用相同的连接属性。

您可以指定连接的其他选项。AWS Glue Studio 生成的任务脚本包含 `Datasource` 条目，该条目使用连接通过指定的连接选项插入连接器。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")
```

将自定义连接器添加到 AWS Glue Studio

1. 为自定义连接器创建代码。有关更多信息，请参阅 开发自定义连接器。
2. 连接器增加了对 AWS Glue 功能的支持。下面举例说明了这些功能，以及如何在 AWS Glue Studio 生成的任务脚本内使用这些功能。

   - 数据类型映射 – 连接器可以在从基础数据存储中读取列的同时对列进行类型化。例如，解析记录并构造 `DynamicFrame` 时，`dataTypeMapping` 的 `{"INTEGER":"STRING"}` 会将所有类型为 `Integer` 的列转换为类型为 `String` 的列。这有助于用户将列转换为他们选择的类型。

```
DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type
= "custom.jdbc", connection_options = {"dataTypeMapping":{"INTEGER":"STRING"}",
connectionName":"test-connection-jdbc"}, transformation_ctx = "DataSource0")
```

- 分区以进行并行读取 – AWS Glue 允许对列上的数据进行分区，以便从数据存储中读取并行数据。您必须指定分区列、下分区界限、上分区界限和分区数。此功能使您能够利用数据并行性以及为 Spark 应用程序分配的多个 Spark 执行程序。

```
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")
```

- AWS Secrets Manager 用于存储凭据-数据目录连接还可以包含存储在中的密钥的 AWS Secrets Manager。`secretId`该 AWS 密钥可以安全地存储身份验证和凭据信息，并在运行 AWS Glue 时将其提供给。或者，您还可以在 Spark 脚本中指定 `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")
```

- 使用行谓词和列投影筛选源数据 – AWS Glue Spark 运行时还允许用户向下推送 SQL 查询，以使用行谓词和列投影在源处筛选数据。这样一来，ETL 任务能够更快地从支持推送的数据存储加载筛选的数据。向下推送到 JDBC 数据源的 SQL 查询示例如下：`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")
```

- 任务书签 – AWS Glue 支持从 JDBC 源增量加载数据。AWS Glue 从数据存储中跟踪上次处理的记录，并在后续 ETL 任务运行中处理新的数据记录。任务书签使用主键作为书签键的默认列，前提是此列按顺序增加或减少。有关任务书签的更多信息，请参阅《AWS Glue 开发人员指南》中的任务书签。

```
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. 将自定义连接器打包为 JAR 文件，然后将文件上载到 Amazon S3。
4. 测试您的自定义连接器。有关更多信息，请参阅 Glu GitHub e 自定义连接器：本地验证测试指南中的说明。
5. 在 AWS Glue Studio 控制台中，在导航窗格中选择 Connectors (连接器)。
6. 在 Connectors (连接器) 页面上，选择 Create custom connector (创建自定义连接器)。
7. 在 Create custom connector (创建自定义连接器) 页面上，输入以下信息：
   - 指向 Amazon S3 中自定义代码 JAR 文件位置的路径。
   - AWS Glue Studio 将使用的连接器的名称。
   - 您的连接器类型，可以是 JDBC、Spark 或者 Athena。
   - 自定义代码中入口点的名称，AWS Glue Studio 将调用以使用连接器。
     - 对于 JDBC 连接器，此字段应该是 JDBC 驱动程序的类名称。
     - 对于 Spark 连接器，此字段应该是完全限定的数据源类名称或其别名，您可以使用 `format` 运算符。
   - （仅限 JDBC）JDBC 连接用于数据存储的基本 URL。
   - （可选）自定义连接器的描述。
8. 选择 Create connector (创建连接器)。
9. 在 Connectors (连接器) 页面中，创建使用此连接器的连接，如为连接器创建连接中所示。

可用连接

在创建连接器连接时，可以使用以下连接：

- Amazon Aurora – 一种具有内置安全性、备份还原以及内存加速功能的可扩展、高性能关系数据库引擎。
- Amazon DocumentDB – 一种可扩展、高度可用且完全托管式的文档数据库服务，支持 MongoDB 和 SQL API。
- Amazon Redshift – 一种可扩展、高度可用且完全托管式的文档数据库服务，支持 MongoDB 和 SQL API。
- Azure SQL – 一种由 Microsoft Azure 提供的基于云的关系数据库服务，具有可扩展、可靠和安全的数据存储和管理功能。

- Cosmos DB – 一种由 Microsoft Azure 提供的全球分布式云数据库服务，具有可扩展、高性能的数据存储和查询功能。
- Google BigQuery — 一种无服务器云数据仓库，用于对大型数据集进行快速 SQL 查询。
- JDBC – 一种关系数据库管理系统（RDBMS），使用 Java API 来连接到数据连接并与之交互。
- Kafka – 一种用于实时数据流式传输和消息收发的开源流式处理平台。
- MariaDB – 一种由社区开发的 MySQL 分支，提供增强的性能、可扩展性和功能。
- MongoDB – 一种面向文档的跨平台数据库，具有高可扩展性、高灵活性、高性能等特点。
- MongoDB Atlas – 一种由 MongoDB 提供的基于云的数据库即服务（DBaaS）产品，可简化 MongoDB 部署的管理和扩展。
- Microsoft SQL Server – 一种由微软公司推出的关系数据库管理系统（RDBMS），具有强大的数据存储、分析和报告功能。
- MySQL – 一种广泛用于 Web 应用程序的开源关系数据库管理系统（RDBMS），并以可靠性和可扩展性闻名。
- 网络 – 网络数据来源是指数据集成平台可以访问并且可通过网络访问的资源或服务。
- OpenSearch— OpenSearch 数据源是一种 OpenSearch 可以连接到并从中提取数据的应用程序。
- Oracle – 一种由甲骨文公司推出的关系数据库管理系统（RDBMS），具有强大的数据存储、分析和报告功能。
- PostgreSQL – 一种开源关系数据库管理系统（RDBMS），具有强大的数据存储、分析和报告功能。
- Sales force — Salesforce 提供客户关系管理 (CRM) 软件，可在销售、客户服务、电子商务等方面为您提供帮助。如果你是 Salesforce 用户，你可以 AWS Glue 连接到你的 Salesforce 账户。然后，您可以在 ETL 作业中使用 Salesforce 作为数据源或目标。运行这些任务可在 Salesforce 和 AWS 服务或其他支持的应用程序之间传输数据。
- SAP HANA – 一种内存数据库和分析平台，提供快速数据处理、高级分析和实时数据集成等功能。
- Snowflake – 基于云的数据仓库，提供可扩展、高性能的数据存储和分析服务。
- Teradata – 一种开源关系数据库管理系统（RDBMS），提供高性能的数据存储、分析和报告功能。
- Vertica – 一种专门面向大数据分析的列式分析数据仓库，提供快速查询、高级分析和可扩展性等功能。

为连接器创建连接

AWS Glue 连接是存储特定数据存储的连接信息的 Data Catalog 对象。这些连接存储登录凭证、URI 字符串、Virtual Private Cloud (VPC) 等信息。在 Data Catalog 中创建连接，就可以不必在每次创建作业时都指定所有连接详细信息。

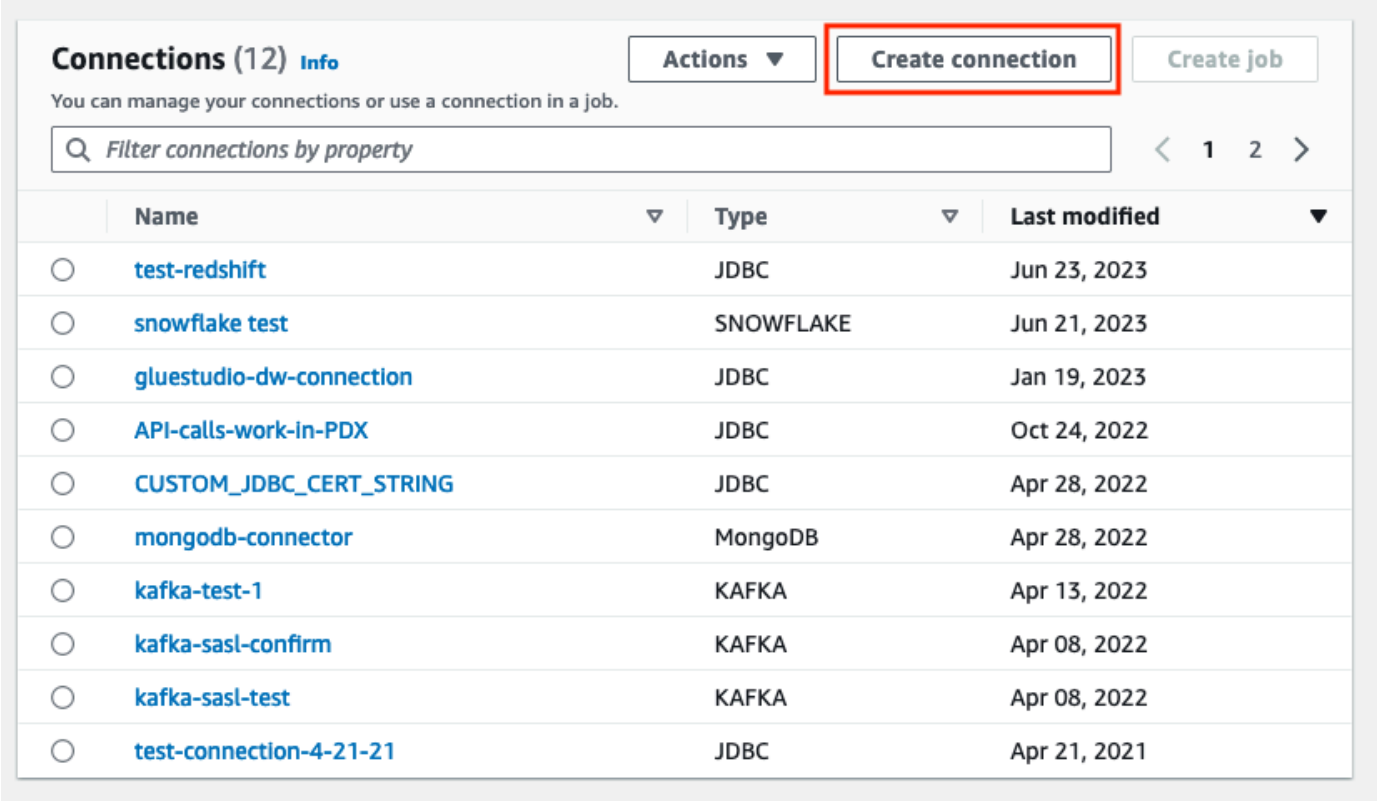
为连接器创建连接

1. 在 AWS Glue Studio 控制台中，在导航窗格中选择 Connectors (连接器)。在连接部分中，选择创建连接。
2. 在创建数据连接向导的第 1 步中，选择要为其创建连接的数据来源。可通过多种方式查看可用的数据来源，包括：

   - 通过选择选项卡筛选可用的数据来源。默认情况下，全部连接器处于选中状态。
   - 切换到列表可以列表形式查看数据来源，也可以切换回网格以在网格布局查看可用连接器。
   - 使用搜索栏缩小数据来源列表的范围。在键入过程中，系统将显示搜索匹配项，不匹配的来源将从视图中移除。

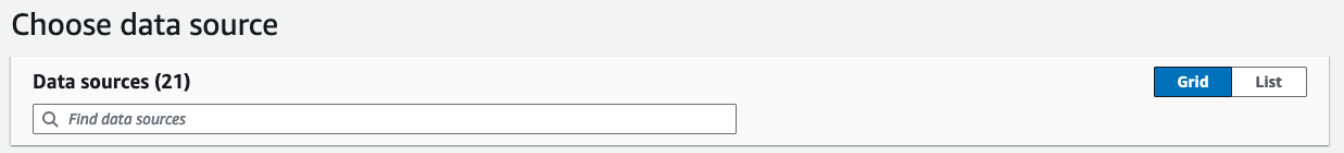
   选好数据来源后，选择下一步。
3. 在向导的第 2 步中配置连接。

   输入连接详细信息。根据所选连接器的类型，系统会提示您输入附加信息：



4. 在创建数据连接向导的第 1 步中，选择要为其创建连接的数据来源。可通过多种方式查看可用的数据来源。默认情况下，您将在网格布局中看到所有可用的数据来源。您也可以：

   - 切换到列表可以列表形式查看数据来源，也可以切换回网格以在网格布局查看可用连接器。
   - 使用搜索栏缩小数据来源列表的范围。在键入过程中，系统将显示搜索匹配项，不匹配的来源将从视图中移除。

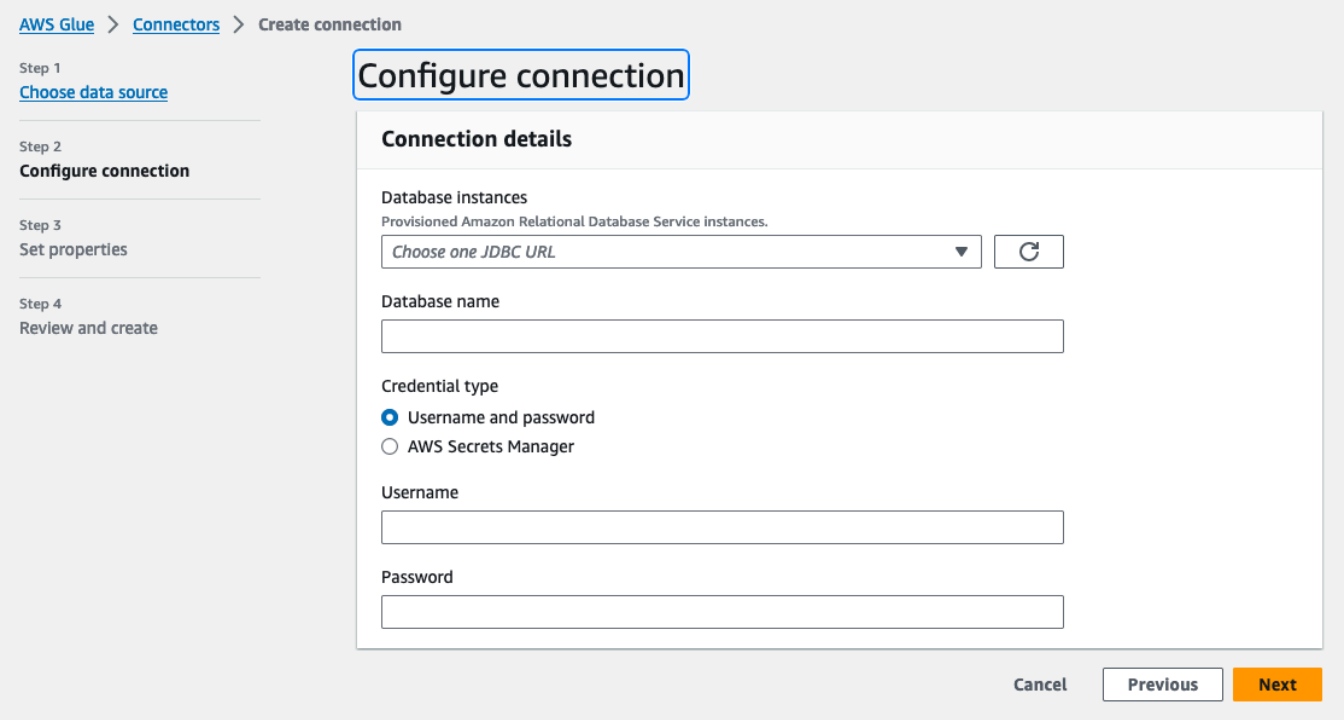
   

   选好数据来源后，选择下一步。

5. 在向导的第 2 步中配置连接。

   输入连接详细信息。根据所选连接器的类型，您可能会需要输入额外的连接信息。这可能包括：

   - 连接详细信息 – 这些字段将根据您要连接的数据来源而变化。例如，假设您要连接到 Amazon DocumentDB 数据库，则需要输入 Amazon DocumentDB URL。如果要连接到 Amazon Aurora，则需要选择数据库实例并输入数据库名称。以下是对于 Amazon Aurora 而言需要的连接详细信息：



- 凭证类型 – 可选择用户名和密码或者 AWS Secrets Manager。输入请求的身份验证信息。
- 对于使用 JDBC 的连接器，请输入为数据存储创建 JDBC URL 所需的信息。
- 如果您使用 Virtual Private Cloud（VPC），请为 VPC 输入网络信息。

6. 在向导的第 3 步中设置连接属性。在这一步中，您可以选择添加描述和标签。名称为必填项，且已预先填充默认值。选择下一步。
7. 查看连接源、详细信息和属性。如果要进行任何更改，请在向导的这一步选择编辑。准备就绪后，选择创建连接。

   选择创建连接。

   您将返回到 Connectors (连接器) 页面，信息性广告条会指示已创建的连接。您现在可以在您的 AWS Glue Studio 作业中使用连接。

创建 Kafka 连接

创建 Kafka 连接时，从下拉菜单中选择 Kafka 将显示要配置的其他设置：

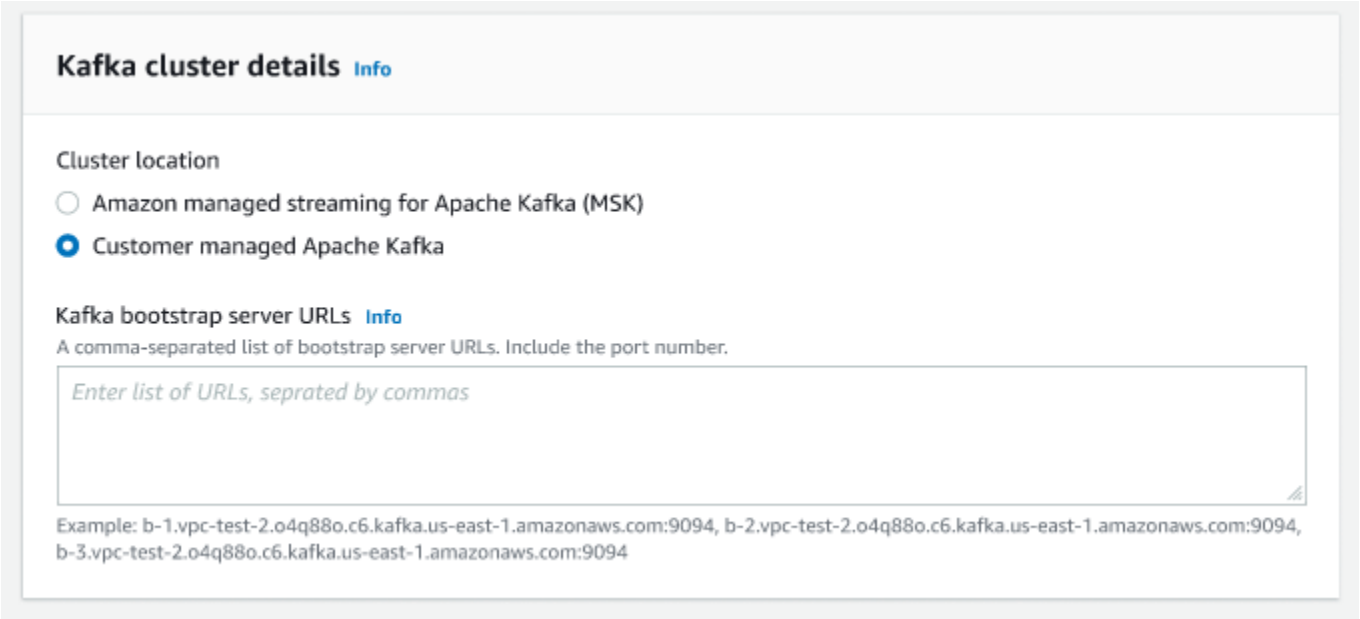
- Kafka 集群详细信息
- 身份验证
- 加密

- 网络操作

配置 Kafka 集群详细信息

1. 选择集群位置。可以从 Amazon Managed for Apache Kafka (MSK) 集群或 Customer managed Apache Kafka（客户管理的 Apache Kafka）集群中选择。有关 Amazon Managed Streaming for Apache Kafka 的更多信息，请参阅 Amazon Managed Streaming for Apache Kafka (MSK)。

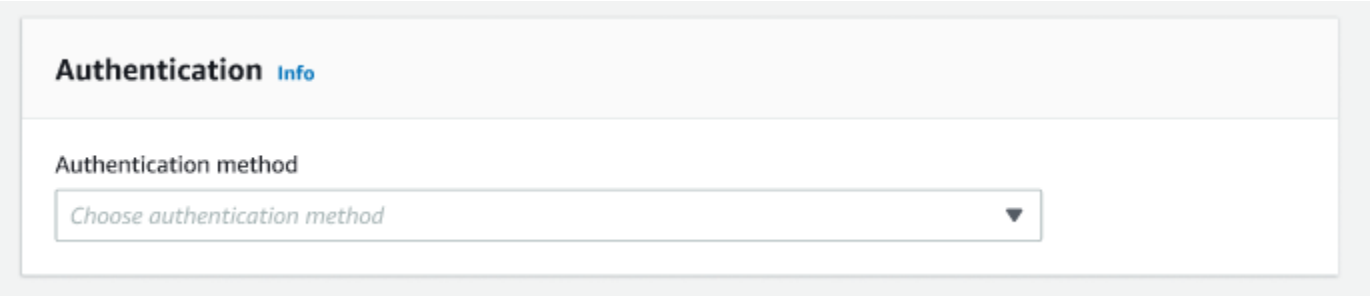
> **Note**
>
> Amazon Managed Streaming for Apache Kafka 仅支持 TLS 和 SASL/SCRAM-SHA-512 身份验证方法。



2. 输入 Kafka 引导服务器的 URL。通过用逗号分隔每台服务器，可以输入多个服务器。通过在 URL 末尾附加 `:<port number>`，包含端口号。

   例如：`b-1.vpc-test-2.034a88o.kafka-us-east-1.amazonaws.com:9094`

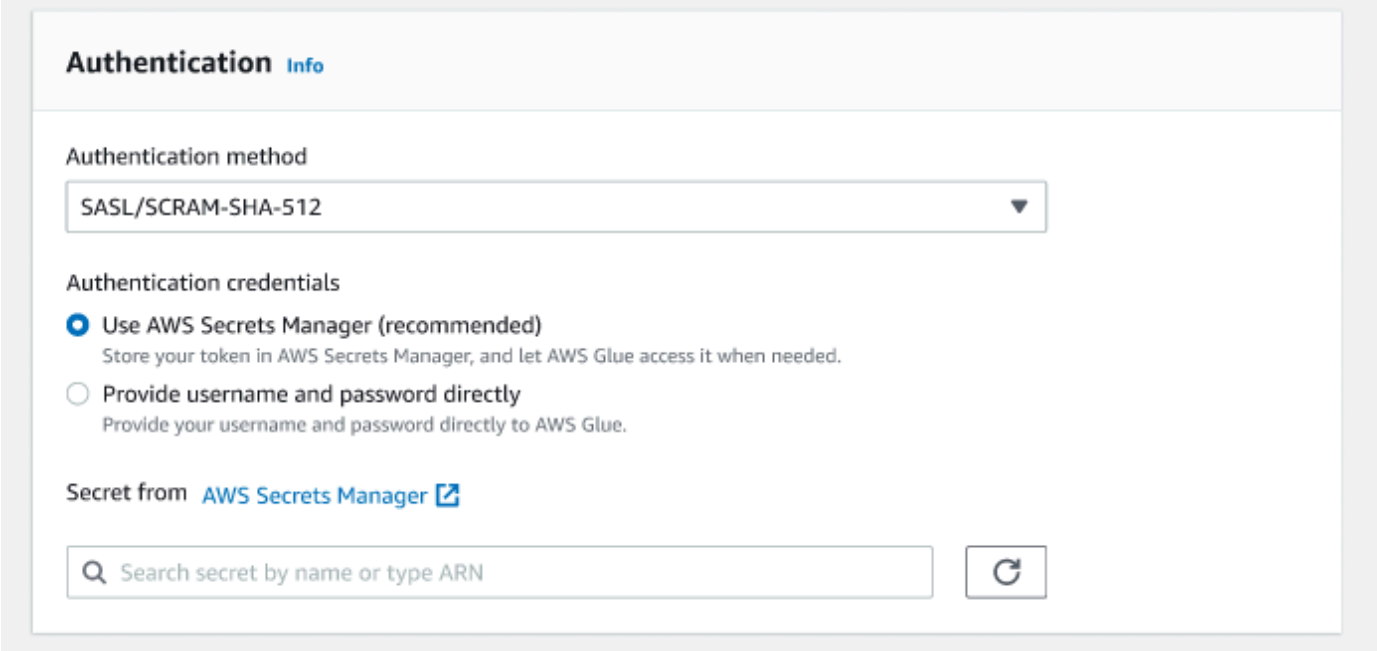
选择身份验证方法



AWS Glue 支持用于身份验证的简单身份验证和安全层 (SASL) 框架。SASL 框架支持各种身份验证机制，且 AWS Glue 提供 SCRAM（用户名和密码）、GSSAPI（Kerberos 协议）和 PLAIN （用户名和密码）协议。

从下拉菜单中选择身份验证方法时，可以选择以下客户端身份验证方法：

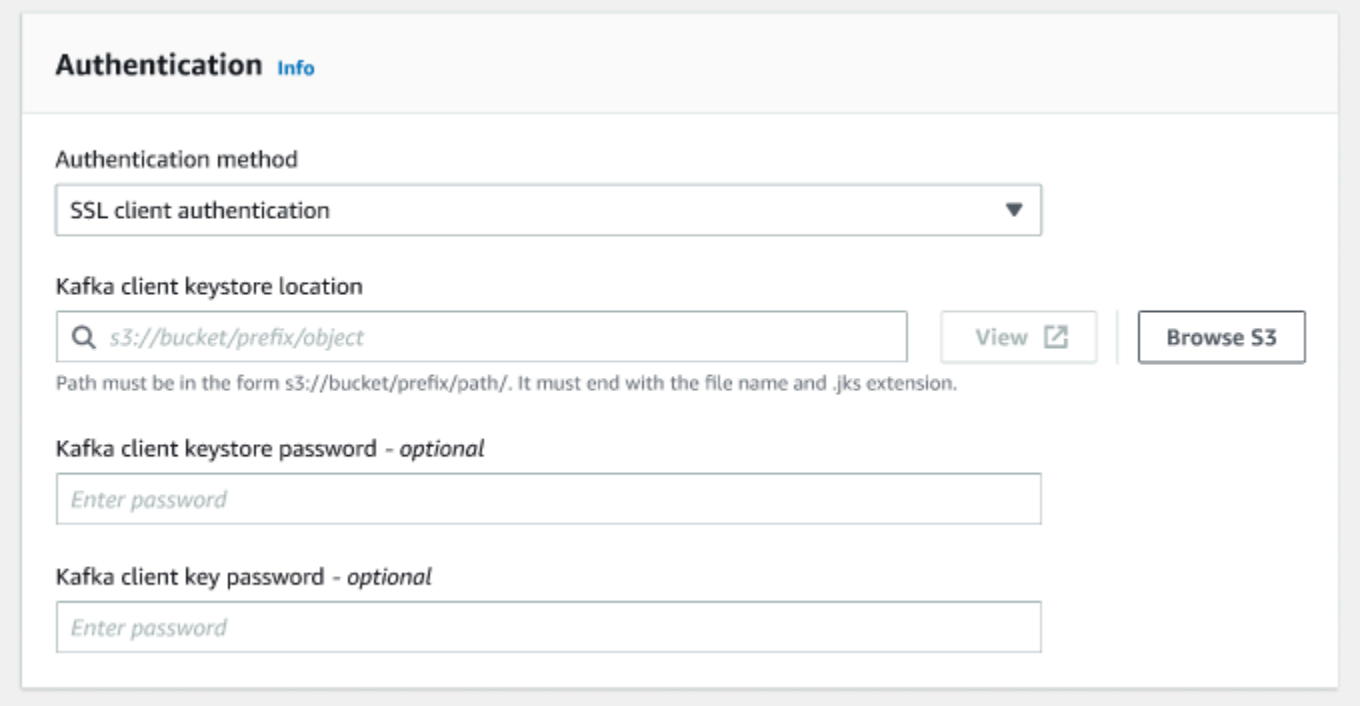
- 无 – 不进行身份验证。如果是为进行测试而创建连接，这非常有用。
- SASL/SCRAM-SHA-512 - 选择此身份验证方法以指定身份验证凭证。有两个可用的选项：
  - 使用 AWS Secrets Manager（推荐）- 如果选择此选项，则可以将凭证存储在 AWS Secrets Manager 中，然后让 AWS Glue 在需要时访问该信息。指定存储 SSL 或 SASL 身份验证凭证的密钥。



  - 直接提供用户名和密码。
- SASL/GSSAPI (Kerberos) – 如果选择此选项，则可以选择 keytab 文件、krb5.conf 文件的位置，然后输入 Kerberos 主体名称和 Kerberos 服务名称。keytab 文件和 krb5.conf 文件的位置必须位于

Amazon S3 位置。由于 MSK 尚不支持 SASL/GSSAPI，所以此选项仅适用于客户管理的 Apache Kafka 集群。有关更多信息，请参阅 MIT Kerberos 文档：keytab。

- SASL/PLAIN - 选择此身份验证方法以指定身份验证凭证。有两个可用的选项：
  - 使用 AWS Secrets Manager（推荐）- 如果选择此选项，则可以将凭证存储在 AWS Secrets Manager 中，然后让 AWS Glue 在需要时访问该信息。指定存储 SSL 或 SASL 身份验证凭证的密钥。
  - 直接提供用户名和密码。
- SSL 客户端身份验证 – 如果选择此选项，则可以通过浏览 Amazon S3 来选择 Kafka 客户端密钥库的位置。或者，您可以输入 Kafka 客户端密钥库密码和 Kafka 客户端密钥密码。



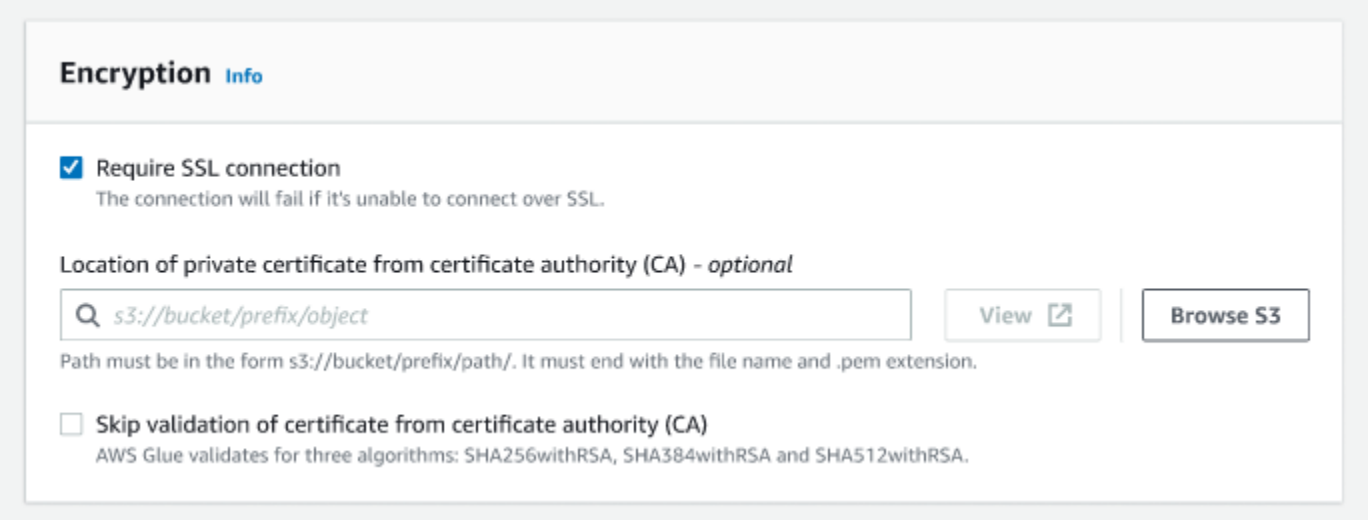
配置加密设置

1. 如果 Kafka 连接需要 SSL 连接，请选中 Require SSL connection（需要 SSL 连接）复选框。请注意，如果无法通过 SSL 连接，则连接将失败。用于加密的 SSL 可以与任何身份验证方法（SASL/SCRAM-SHA-512、SASL/GSSAPI、SASL/PLAIN、SSL 客户端身份验证）一起使用，并且是可选的。

   如果身份验证方法设置为 SSL client authentication（SSL 客户端身份验证），则系统将自动选择并禁用此选项，以防止出现任何更改。

2. （可选）。选择来自证书颁发机构 (CA) 的私有证书的位置。请注意，证书的位置必须在 S3 位置。选择 Browse（浏览），从连接的 S3 存储桶中选择文件。路径必须采用 `s3://bucket/prefix/filename.pem` 格式。它必须以文件名和 .pem 扩展名结尾。
3. 可以选择跳过验证证书颁发机构 (CA) 的证书。选择复选框 Skip validation of certificate from certificate authority (CA) [跳过验证证书颁发机构（CA）的证书]。如果未选中此框，则 AWS Glue 会验证三种算法的证书：

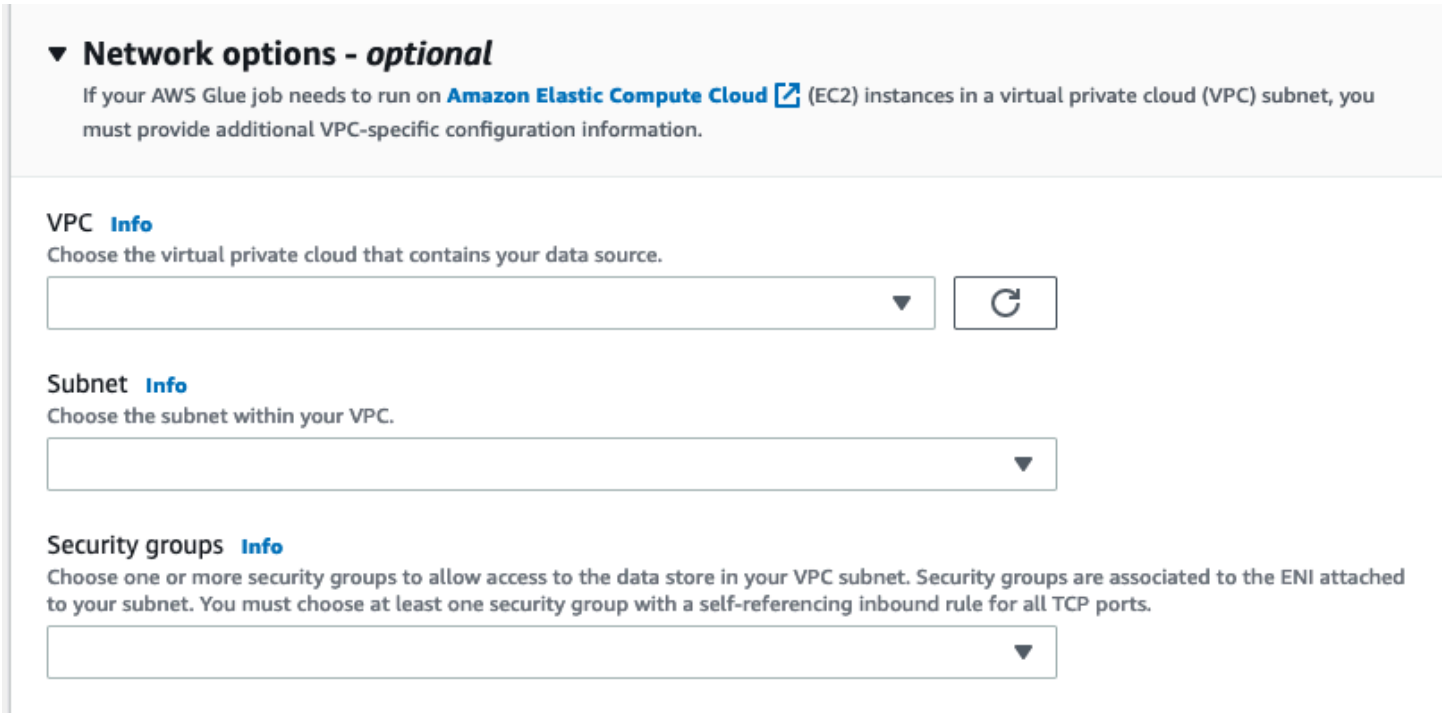
   - SHA256withRSA
   - SHA384withRSA
   - SHA512withRSA



（可选）网络选项

下面是配置 VPC、子网和安全组的可选步骤。如果您的 AWS Glue 任务需要在 virtual private cloud (VPC) 子网中的 Amazon EC2 实例上运行，必须提供其他特定于 VPC 的配置信息。

1. 选择包含您的数据源的 virtual private cloud (VPC)。
2. 选择您的 VPC 所在的子网。
3. 选择允许访问 VPC 子网中数据存储的一个或多个安全组。安全组与附加到子网的 ENI 相关联。必须为所有 TCP 端口选择至少一个具有自引用入站规则的安全组。



使用自定义连接器编写任务

您可以在 AWS Glue Studio 中为数据源节点和数据目标节点使用连接器和连接。

主题

- 创建任务，为数据源使用连接器
- 为使用连接器的节点配置源属性
- 为使用连接器的节点配置目标属性

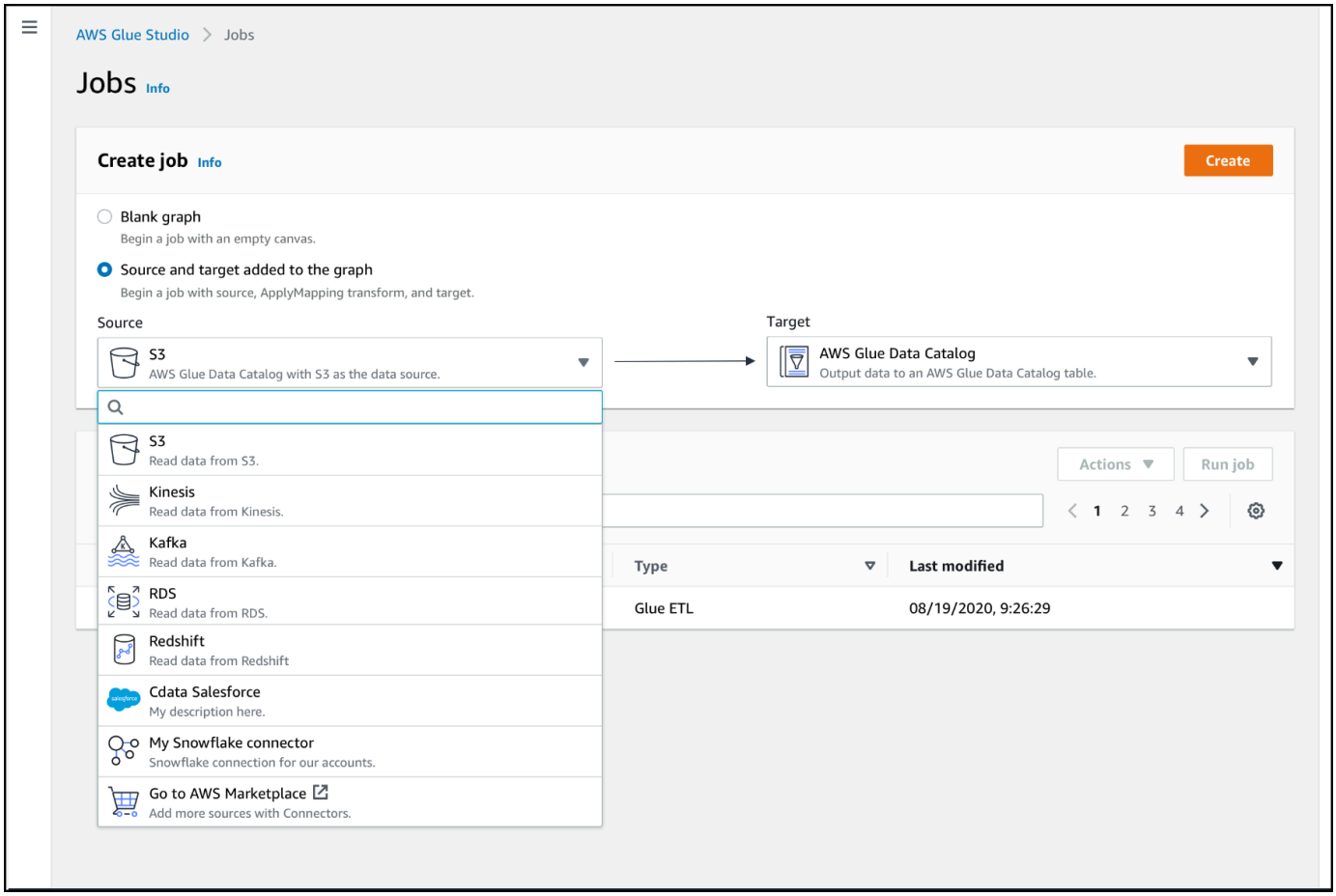
创建任务，为数据源使用连接器

创建新任务时，可以为数据源和数据目标选择连接器。

创建任务，为数据源或数据目标使用连接器

1. 登录 AWS Management Console 并打开AWS Glue Studio控制台，网址为 https://console.aws.amazon.com/gluestudio/。
2. 在 Connectors (连接器) 页面的 Your connections (您的连接) 资源列表中，选择要在任务中使用的连接，然后选择 Create job (创建任务)。

或者，在 AWS Glue Studio Jobs (任务) 页面的 Create job (创建任务) 下面，选择 Source and target added to the graph (已添加到图形中的源和目标)。在 Source (源) 下拉列表中，选择要在任务中使用的自定义连接器。还可以为 Target (目标) 选择连接器。



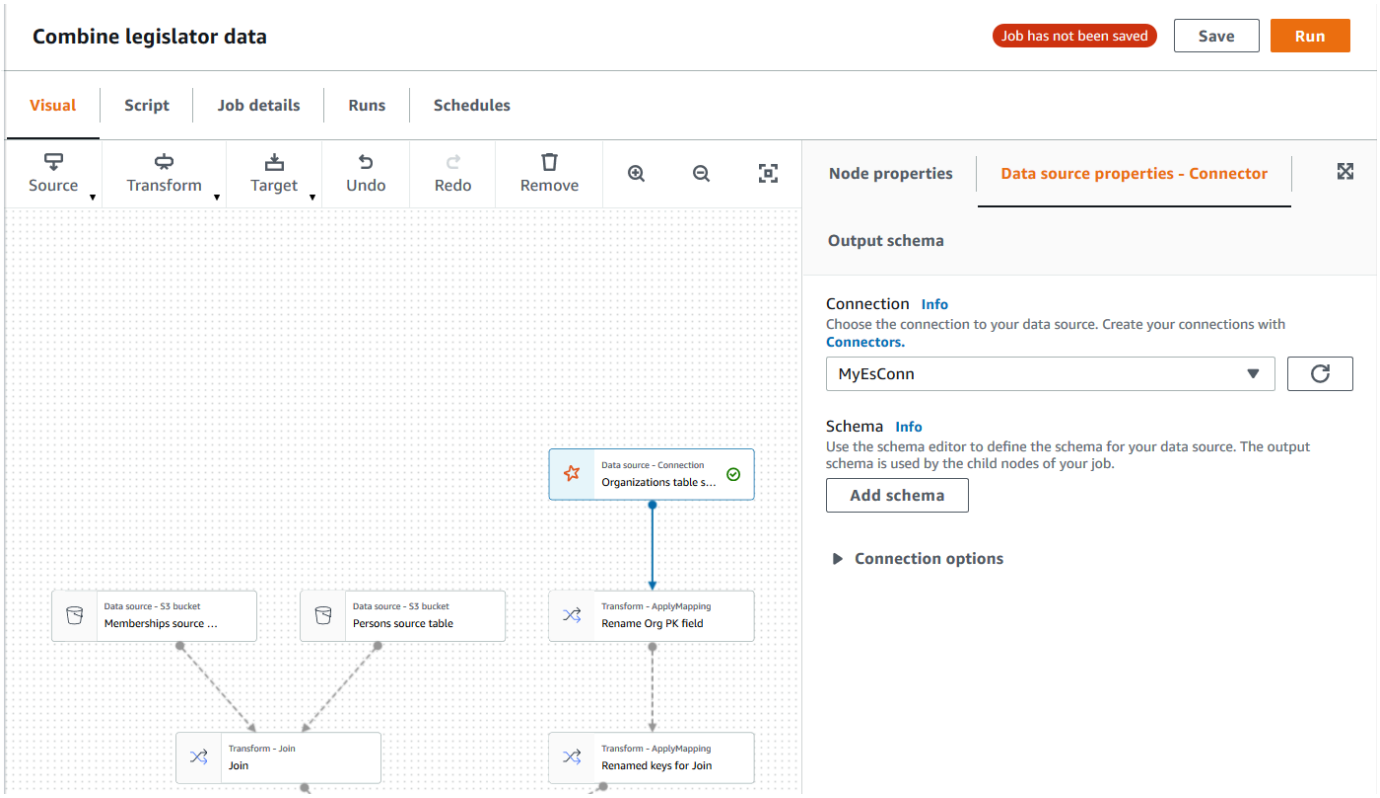
3. 选择 Create (创建) 以打开可视化任务编辑器。
4. 配置数据源节点，如为使用连接器的节点配置源属性中所示。
5. 添加转换、其他数据存储和数据目标以继续创建 ETL 任务，如带有 AWS Glue Studio 的 Visual ETL中所示。
6. 配置任务属性以自定义任务运行环境，如修改任务属性中所示。
7. 保存并运行任务。

为使用连接器的节点配置源属性

创建为数据源使用连接器的任务后，可视任务编辑器将显示任务图，其中包含为连接器配置的数据源节点。您必须为该节点配置数据源属性。

为使用连接器的数据源节点配置属性

1. 选择任务图中的连接器数据源节点，或者添加新节点，然后为 Node type (节点类型) 选择连接器。然后，在右侧的节点详细信息面板中，选择 Data source properties (数据源属性) 选项卡（如果尚未选择）。



2. 在 Data source properties (数据源属性) 选项卡上，选择要用于此任务的连接。

   输入每种连接类型所需的附加信息：

   JDBC

   - Data source input type (数据源输入类型)：选择以提供表名称或 SQL 查询作为数据源。根据您的选择，您需要提供以下附加信息：
     - Table name (表名称)：数据源中表的名称。如果数据源不使用术语表，则提供相应数据结构的名称，如自定义连接器使用信息（可在中找到 AWS Marketplace）所示。
     - Filter predicate (筛选条件谓词)：读取数据源时使用的条件子句，类似于 `WHERE` 子句，用于检索数据的子集。
     - Query code (查询代码)：输入用于从数据源检索特定数据集的 SQL 查询。基本 SQL 查询示例：

```
SELECT column_list FROM
                            table_name WHERE where_clause
```

- Schema (架构)：因为 AWS Glue Studio 使用存储在连接中的信息来访问数据源，而不是从数据目录表中检索元数据信息，所以您必须为数据源提供架构元数据。选择 Add schema (添加架构)，打开架构编辑器。

  有关如何使用架构编辑器的说明，请参阅编辑自定义转换节点的架构。

- Partition column (分区列)：（可选）您可以为 Partition Column (分区列)、Lower bound (下限)、Upper bound (上限) 和 Number of partitions (分区数) 提供值，对数据读取进行分区。

  `lowerBound` 和 `upperBound` 值用于确定分区步长，而不是用于筛选表中的行。对表中的所有行进行分区并返回。

  > **Note**
  >
  > 列分区为用于读取数据的查询添加额外的分区条件。使用查询（而不是表名称）时，您应验证查询是否适用于指定的分区条件。例如：
  >
  > - 如果您的查询格式为 `"SELECT col1 FROM table1"`，则在使用分区列的查询结尾附加 `WHERE` 子句，以测试查询。
  > - 如果您的查询格式为 `"SELECT col1 FROM table1 WHERE col2=val"`，则通过 `AND` 和使用分区列的表达式扩展 `WHERE` 子句，以测试查询。

- Data type casting (数据类型转换)：如果数据源使用 JDBC 中不可用的数据类型，请使用此部分指定如何将数据源中的数据类型转换为 JDBC 数据类型。您最多可指定 50 个不同的数据类型转换。数据源中使用相同数据类型的所有列都将以相同的方式进行转换。

  例如，如果数据源中有三列使用 `Float` 数据类型，并且您指示 `Float` 数据类型应转换为 JDBC `String` 数据类型，则使用 `Float` 数据类型的所有三列将转换为 `String` 数据类型。

- Job bookmark keys (任务书签键)：任务书签可帮助 AWS Glue 维护状态信息，并防止重新处理旧数据。指定一个或多个列为书签键。AWS Glue Studio 使用书签键跟踪上次运行 ETL 任务期间已处理的数据。用于自定义书签键的列都必须严格单调递增或递减，但是允许有间隙。

  如果您输入多个书签键，它们将合并成一个复合键。复合任务书签键不应包含重复的列。如果不指定书签键，则预设情况下，AWS Glue Studio 将使用主键作为书签键，前提是主键按

顺序递增或递减（没有间隙）。如果表没有主键，但任务书签属性已启用，则必须提供自定义任务书签键。否则，无法搜索用作默认值的主键，任务运行将失败。

- Job bookmark keys sorting order (任务书签键排序)：选择键值是按顺序递增还是递减。

Spark

- Schema (架构)：因为 AWS Glue Studio 使用存储在连接中的信息来访问数据源，而不是从数据目录表中检索元数据信息，所以您必须为数据源提供架构元数据。选择 Add schema (添加架构)，打开架构编辑器。

  有关如何使用架构编辑器的说明，请参阅编辑自定义转换节点的架构。

- Connection options (连接选项)：根据需要输入其他键值对，提供其他连接信息或选项。例如，您可以输入数据库名称、表名、用户名和密码。

  例如，对于 OpenSearch，您可以输入以下键值对，如中所述：the section called “ 教程：使用 AWS Glue Connector for Elasticsearch ”

  - `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`

有关要使用的最低连接选项的示例，请参阅示例测试脚本 MinimalSparkConnectorTest.scala on GitHub，该脚本显示了您通常在连接中提供的连接选项。

Athena

- Table name (表名称)：数据源中表的名称。如果您使用连接器从 Athena CloudWatch 日志中读取，则需要输入表名。`all_log_streams`
- Athena schema name (Athena 架构名称)：在 Athena 数据源中选择与包含该表的数据库相对应的架构。如果您使用连接器从 Athena CloudWatch 日志中读取，则需要输入类似于的架构名称。`/aws/glue/`*`name`*
- Schema (架构)：因为 AWS Glue Studio 使用存储在连接中的信息来访问数据源，而不是从数据目录表中检索元数据信息，所以您必须为数据源提供架构元数据。选择 Add schema (添加架构)，打开架构编辑器。

有关如何使用架构编辑器的说明，请参阅编辑自定义转换节点的架构。

- Additional connection options (其他连接选项)：根据需要输入其他键值对，提供其他连接信息或选项。

有关示例，请参阅 https://github.com/aws-samples/ aws-glue-samples /tree/master/ / Development/ GlueCustomConnectors Athena 的`README.md`文件。在本文档中的步骤中，示例代码显示了所需的最小连接选项，即 `tableName`、`schemaName` 和 `className`。代码示例将这些选项指定为 `optionsMap` 变量，但您可以为连接指定它们，然后使用连接。

3. （可选）提供所需信息后，您可以选择节点详细信息面板中的 Output schema (输出架构) 选项卡，查看生成的数据架构。此选项卡上显示的架构将由您添加到任务图的子节点使用。
4. （可选）配置节点属性和数据源属性后，您可以选择节点详细信息窗格中的 Data preview (数据预览) 选项卡来预览数据源的数据集。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。使用此功能会产生相关费用，并且一旦您提供 IAM 角色，则会立即开始计费。

为使用连接器的节点配置目标属性

如果将连接器用于数据目标类型，则必须配置数据目标节点的属性。

为使用连接器的数据目标节点配置属性

1. 在任务图中选择连接器数据目标节点。然后，在右侧的节点详细信息面板中，选择 Data target properties (数据目标属性) 选项卡（如果尚未选择）。
2. 在 Data target properties (数据目标属性) 选项卡上，选择用于写入目标的连接。

   输入每种连接类型所需的附加信息：

   JDBC

   - Connection (连接)：选择要与连接器一起使用的连接。有关如何创建连接的信息，请参阅为连接器创建连接。
   - Table name (表名称)：数据目标中表的名称。如果数据目标不使用术语表，则提供相应数据结构的名称，如自定义连接器使用信息（可在中找到 AWS Marketplace）所示。
   - Batch size (批处理大小)（可选）：在单个操作中输入要在目标表中插入的行数或记录数。默认值是 1000 行。

Spark

- Connection (连接)：选择要与连接器一起使用的连接。如果以前未创建连接，请选择 Create connection (创建连接) 创建一个。有关如何创建连接的信息，请参阅为连接器创建连接。
- Connection options (连接选项)：根据需要输入其他键值对，提供其他连接信息或选项。您可以输入数据库名称、表名、用户名和密码。

  例如，对于 OpenSearch，您可以输入以下键值对，如中所述：the section called “ 教程：使用 AWS Glue Connector for Elasticsearch ”

  - `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`

  有关要使用的最低连接选项的示例，请参阅示例测试脚本 MinimalSparkConnectorTest.scala on GitHub，该脚本显示了您通常在连接中提供的连接选项。

3. 提供所需信息后，您可以选择节点详细信息面板中的 Output schema (输出架构) 选项卡，查看生成的数据架构。

管理连接器和连接

您可以在 AWS Glue 中使用连接器页面管理您的连接器和连接。

主题

- 查看连接器和连接详细信息
- 编辑连接器和连接
- 删除连接器和连接
- 取消连接器的订阅

查看连接器和连接详细信息

您可以在 Connectors (连接) 页面上的 Your connectors (您的连接器) 和 Your connections (您的连接) 资源表中，查看关于您的连接器和连接的摘要信息。要查看详细信息，请执行以下步骤。

查看连接器或连接详细信息

1. 在 AWS Glue Studio 控制台中，在导航窗格中选择 Connectors (连接器)。
2. 选择要查看其详细信息的连接器或连接。
3. 选择 Actions (操作)，然后选择 View details (查看详细信息)，打开相应连接器或连接的详细信息页面。
4. 在详细信息页面上，您可以选择 Edit（编辑）或 Delete（删除）连接器或连接。
   - 对于连接器，可以选择 Create connection (创建连接)，创建使用连接器的新连接。
   - 对于连接，您可以选择 Create job (创建任务)，创建使用连接的任务。

编辑连接器和连接

您可以使用 Connectors (连接器) 页面更改您的连接器和连接中存储的信息。

修改连接器或连接

1. 在 AWS Glue Studio 控制台中，在导航窗格中选择 Connectors (连接器)。
2. 选择要更改的连接器或连接。
3. 选择操作，然后选择编辑。

   您还可以选择 View details (查看详细信息)，然后在连接器或连接详细信息页面上，选择 Edit (编辑)。
4. 在 Edit connector (编辑连接器) 或者 Edit connection (编辑连接) 页面上，更新信息，然后选择 Save (保存)。

删除连接器和连接

您可以使用 connector (连接器) 页面删除连接器和连接。如果删除某个连接器，还应删除为该连接器创建的所有连接。

要从 AWS Glue Studio 中删除连接器，请执行以下操作

1. 在 AWS Glue Studio 控制台中，在导航窗格中选择 Connectors (连接器)。
2. 选择要删除的连接器或连接。
3. 选择操作，然后选择删除。

   您还可以选择 View details (查看详细信息)，然后在连接器或连接详细信息页面上，选择 Delete (删除)。
4. 确认您要删除连接器或连接，方法是输入 **`Delete`**，然后选择 Delete (删除)。

   删除某个连接器时，还应删除为该连接器创建的所有连接。

使用已删除连接的任务将不再运行。您可以编辑任务以使用其他数据存储，也可以删除任务。有关如何删除任务的信息，请参阅删除任务。

如果您删除连接器，此操作不会取消 AWS Marketplace中该连接器的订阅。要删除已删除连接器的订阅，请按取消连接器的订阅中的说明操作。

取消连接器的订阅

从中删除连接和连接器后AWS Glue Studio， AWS Marketplace 如果您不再需要连接器，则可以在中取消订阅。

> **Note**
>
> 如果您取消对连接器的订阅，则不会从您的账户中删除连接器或连接。使用该连接器和相关连接的任务将无法再使用该连接器，并且会失效。
> 在取消订阅或重新订阅连接器之前 AWS Marketplace，应删除与该 AWS Marketplace 产品关联的现有连接和连接器。

要取消订阅连接器 AWS Marketplace

1. 登录 AWS Marketplace 主机，网址为 https://console.aws.amazon.com/marketplace。
2. 选择 Manage subscriptions (选择订阅)。
3. 在 Manage subscriptions (管理订阅) 页面上，选择要取消的连接器订阅旁边的 Manage (管理)。
4. 依次选择 Actions (操作) 和 Delete Subscriptions (删除订阅)。

5. 选中此复选框，确认正在运行的实例会向您的账户收取费用，然后选择 Yes, cancel subscription (是，取消订阅)。

开发自定义连接器

您可以编写从数据存储中读取数据或向数据存储写入数据的代码，并将数据格式化以用于 AWS Glue Studio 任务。您可以为 Spark、Athena 和 JDBC 数据存储创建连接器。上发布的示例代码 GitHub 概述了您需要实现的基本接口。

您需要用于创建连接器代码的本地开发环境。您可以使用任意 IDE，甚至只使用命令行编辑器来编写连接器。开发环境示例包括：

- 具有本地 AWS Glue ETL Maven 库的本地 Scala 环境，正如《AWS Glue 开发人员指南》中的使用 Scala 本地开发所述。
- IntelliJ IDE，可从 https://www.jetbrains.com/idea/ 下载 IDE。

主题

- 开发 Spark 连接器
- 开发 Athena 连接器
- 开发 JDBC 连接器
- 将自定义连接器与 AWS Glue Studio 结合使用的示例
- 开发用于的AWS Glue连接器 AWS Marketplace

开发 Spark 连接器

你可以使用 Spark DataSource API V2 (Spark 2.4) 创建 Spark 连接器来读取数据。

若要创建自定义 Spark 连接器

按照AWS Glue GitHub 示例库中的步骤开发 Spark 连接器，该库位于 https://github.com/aws-samples/ aws-glue-samples /tree/master/ /development/spark/READ GlueCustomConnectors ME.md。

开发 Athena 连接器

您可以创建 Athena 连接器，供 AWS Glue 和 AWS Glue Studio 查询自定义数据源。

若要创建自定义 Athena 连接器

按照AWS Glue GitHub 示例库中的步骤开发 Athena 连接器，该库位于 https://github.com/aws-samples/ /tree/master/ /development/Athena。aws-glue-samples GlueCustomConnectors

开发 JDBC 连接器

您可以创建使用 JDBC 访问数据存储的连接器。

创建自定义 JDBC 连接器

1. 在本地开发环境中安装 AWS Glue Spark 运行时库。请参阅AWS Glue GitHub 示例库中的说明，网址为 https://github.com/aws-samples/ aws-glue-samples /tree/master/ /developme/ / readme.m GlueCustomConnectors d。GlueSparkRuntime
2. 实施负责从数据源检索数据的 JDBC 驱动程序。请参阅适用于 Java SE 8 的 Java 文档。

   在您的代码内创建入口点，AWS Glue Studio 将其用于查找您的连接器。Class name (类名称) 字段应该是 JDBC 驱动程序的完整路径。
3. 借助连接器使用 `GlueContext` API 读取数据。用户可以在 AWS Glue Studio 控制台中添加更多输入选项，配置与数据源的连接（如有必要）。有关显示如何使用自定义 JDBC 连接器读取和写入 JDBC 数据库的代码示例，请参阅自定义值和连接类型值。 AWS Marketplace

将自定义连接器与 AWS Glue Studio 结合使用的示例

有关使用自定义连接器的示例，您可以参考以下博客：

- 为带有 AWS Glue 的数据存储开发、测试和部署自定义连接器
- Apache Hudi：使用 AWS Glue 自定义连接器写入 Apache Hudi 表
- 谷歌 BigQuery：使用AWS Glue自定义连接器将数据从谷歌迁移 BigQuery 到亚马逊 S3
- Snowflake（JDBC）：使用 Snowflake 和 AWS Glue 执行数据转换
- SingleStore: 使用和快速构建 ETL SingleStore AWS Glue
- Salesforce：使用 CData JDBC 自定义连接器和 AWS Glue将 Salesforce 数据提取到 Amazon S3 -
- MongoDB：使用 Amazon DocumentDB（与 MongoDB 兼容）和 MongoDB 构建 AWS Glue Spark ETL 任务
- Amazon Relational Database Service (Amazon RDS)：自带亚马逊 RDS 的 JDBC 驱动程序，构建 AWS Glue Spark ETL 任务
- MySQL (JDBC)：https://github.com/aws-samples/ aws-glue-samples /blob/master/ /development/ GlueCustomConnectors spark/ sql.Scala SparkConnectorMy

开发用于的AWS Glue连接器 AWS Marketplace

作为 AWS 合作伙伴，您可以创建自定义连接器并将其上传到以 AWS Marketplace 向AWS Glue客户销售。

开发连接器代码的过程与自定义连接器相同，但上载和验证连接器代码的过程更为详细。请参阅 GitHub 网站上的 “创建连接器” 中的说明。 AWS Marketplace

AWS Glue Studio 中连接器和连接的使用限制

当您使用来自的自定义连接器或连接器时 AWS Marketplace，请注意以下限制：

- 为自定义连接器创建的连接不支持 testConnection API。
- 自定义连接器不支持数据目录连接密码加密。
- 如果为使用 JDBC 连接器的数据源节点指定筛选条件谓语，则无法使用任务书签。
- 不支持在 AWS Glue Studio 用户界面之外创建 Marketplace 连接。

使用 Visual ETL 作业连接到数据来源

创建新作业时，您可以在 AWS Glue 中编辑 Visual ETL 作业时使用连接以连接到数据。为此，您可以添加使用连接器读入数据的源节点，以及指定写出数据的位置的目标节点。

主题

- 修改数据来源节点的属性
- 使用数据源的数据目录表
- 将连接器用作数据源
- 将 Amazon S3 中的文件用作数据源
- 使用流式处理数据源
- 参考信息

修改数据来源节点的属性

要指定数据源属性，首先在任务图中选择一个数据源节点。然后，在节点详细信息面板的右侧，配置节点属性。

修改数据源节点的属性

1. 转到新任务或已保存任务的可视编辑器。
2. 在任务图中选择一个数据源节点。
3. 在节点详细信息面板中选择 Node properties (节点属性) 选项卡，然后输入以下信息：
   - Name (名称)：（可选）输入要与任务图中节点关联的名称。此名称在此任务的所有节点中应唯一。
   - Node type (节点类型)：节点类型可确定由节点执行的操作。在 Node type (节点类型) 的选项列表中，从标题 Data source (数据源) 下列出的值中选择一个。
4. 配置 Data source properties (数据源属性) 信息。有关详细信息，请参阅以下章节：
   - 使用数据源的数据目录表
   - 将连接器用作数据源
   - 将 Amazon S3 中的文件用作数据源
   - 使用流式处理数据源
5. （可选）配置节点属性和数据源属性后，您可以选择节点详细信息面板中的 Output schema (输出架构) 选项卡，查看数据源的架构。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。如果您尚未在 Job details (任务详细信息) 选项卡上指定 IAM 角色，系统会提示您在此处输入 IAM 角色。
6. （可选）配置节点属性和数据源属性后，您可以选择节点详细信息窗格中的 Data preview (数据预览) 选项卡来预览数据源的数据集。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。使用此功能会产生相关费用，并且一旦您提供 IAM 角色，则会立即开始计费。

使用数据源的数据目录表

对于 Amazon S3 和连接器之外的所有数据源，表必须位于您所选择源类型的 AWS Glue Data Catalog 中。AWS Glue 不会创建数据目录表。

基于数据目录表配置数据源节点

1. 转到新任务或已保存任务的可视编辑器。
2. 在任务图中选择一个数据源节点。
3. 选择 Data source properties (数据源属性) 选项卡，然后输入以下信息：

- S3 source type (S3 源类型)：（仅适用于 Amazon S3 数据源）选择选项 Select a Catalog table (选择目录表) 以使用现有 AWS Glue Data Catalog 表。
- Database (数据库)：在数据目录中选择包含要用于此任务的源表的数据库。您可以使用搜索字段按名称搜索数据库。
- Table (表)：从列表中选择与源数据关联的表。此表必须已位于 AWS Glue Data Catalog 中。您可以使用搜索字段按名称搜索表。
- Partition predicate (分区谓词)：（仅适用于 Amazon S3 数据源）输入基于仅包含分区列的 Spark SQL 的布尔表达式。例如："`(year=='2020' and month=='04')`"。
- Temporary directory (临时目录)：（仅适用于 Amazon Redshift 数据源）输入 Amazon S3 中工作目录位置的路径，在其中您的 ETL 任务可以写入临时中间结果。
- Role associated with the cluster (与集群关联的角色)：（仅适用于 Amazon Redshift 数据源）为要使用的 ETL 任务输入角色，该角色包含 Amazon Redshift 集群权限。有关更多信息，请参阅 the section called “数据源和数据目标权限”。

将连接器用作数据源

如果您为 Node type (节点类型) 选择连接器，请按照 使用自定义连接器编写任务 中的说明操作，完成数据源属性的配置。

将 Amazon S3 中的文件用作数据源

如果您选择 Amazon S3 作为数据源，则可以选择以下任一项：

- 数据目录数据库和表。
- Amazon S3 中的存储桶、文件夹或文件。

如果您将 Amazon S3 存储桶用作数据源，AWS Glue 会从文件中的指定位置检测数据的架构，或者使用您指定的文件作为样本文件。当您使用 Infer schema (推断架构) 按钮时，会检测架构。如果您更改 Amazon S3 位置或样本文件，则必须选择 Infer schema (推断架构)，使用新信息执行架构检测。

配置直接从 Amazon S3 中的文件读取数据的数据源节点

1. 转到新任务或已保存任务的可视编辑器。
2. 在任务图中为 Amazon S3 源选择数据源节点。
3. 选择 Data source properties (数据源属性) 选项卡，然后输入以下信息：

- S3 source type (S3 源类型)：（仅适用于 Amazon S3 数据源）选择选项 S3 location (S3 位置)。
- S3 URL：输入 Amazon S3 存储桶、文件夹或包含任务数据的文件的路径。您可以选择 Browse S3 (浏览 S3)，从您的账户的可用位置中选择路径。
- Recursive (递归)：如果需要 AWS Glue 从 S3 位置在子文件夹中的文件读取数据，请选择该选项。

  如果子文件夹包含分区数据，AWS Glue 不会将文件夹名称中指定的分区信息添加到数据目录。例如，您可考虑 Amazon S3 中的以下文件夹。

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

  如果选择 Recursive (递归)，将 `sales` 文件夹选为 S3 位置，择 AWS Glue 会读取所有子文件夹中的数据，但不会为年份、月份或日创建分区。
- Data format (数据格式)：选择数据的存储格式。您可以选择 JSON、CSV 或 Parquet。您选择的值会告知 AWS Glue 任务如何从源文件读取数据。

> **Note**
>
> 如果您未选择正确的数据格式，AWS Glue 可能会正确推断模式，但任务将无法正确解析源文件中的数据。

  您可以输入其他配置选项，具体取决于您选择的格式。

  - JSON（JavaScript 对象表示法）
    - JsonPath：输入指向用于定义表架构的对象的 JSON 路径。JSON 路径表达式始终引用 JSON 结构，类似于 XPath 表达式与 XML 文档的结合使用方式。JSON 路径中的“根成员对象”始终称为 $，即使它是一个对象或数组。JSON 路径可以用点表示法或括号表示法编写。

      有关 JSON 路径的更多信息，请参阅 GitHub 网站上的 JsonPath。
    - Records in source files can span multiple lines (源文件中的记录可以跨越多行)：如果单个记录可以跨越 CSV 文件中的多行，请选择此选项。
  - CSV（逗号分隔值）

- Delimiter (分隔符)：输入字符以表示行中每个列条目的分隔项，例如 `;` 或者 `,`。
- Escape character (转义字符)：输入用作转义字符的字符。此字符表示紧接转义字符的字符应该按字面意思处理，不应将其解释为分隔符。
- Quote character (引号字符)：输入用于将单独的字符串分组为单个值的字符。例如，如果您的 CSV 文件包含 `"This is a single value"` 之类的值，您将选择 Double quote (") (双引号 ("))。
- Records in source files can span multiple lines (源文件中的记录可以跨越多行)：如果单个记录可以跨越 CSV 文件中的多行，请选择此选项。
- First line of source file contains column headers (源文件的第一行包含列标题)：如果 CSV 文件中的第一行包含列标题而不是数据，请选择此选项。

- Parquet (Apache Parquet 列式存储)

  对于以 Parquet 格式存储的数据，不需要配置其他设置。

- Partition predicate (分区谓词)：要对从数据源读取的数据进行分区，请输入基于 Spark SQL 的布尔表达式，仅包含分区列。例如：`"(year=='2020' and month=='04')"`。
- Advanced options (高级选项)：如果需要 AWS Glue 根据特定文件检测数据的架构，请展开此部分。
  - Schema inference (架构推理)：如果您希望特定的文件，而不是让 AWS Glue 选择文件，则选择选项从 S3 中选择样本文件。
  - Auto-sampled file (自动取样文件)：输入 Amazon S3 中用于推断架构的文件的路径。

  如果要编辑数据源节点并更改选定的样本文件，请选择 Reload schema (重新加载架构)，使用新的样本文件检测架构。

4. 选择 Infer schema (推断架构) 按钮，从 Amazon S3 的源文件中检测架构。如果您更改 Amazon S3 位置或样本文件，则必须再次选择 Infer schema (推断架构)，使用新信息推断架构。

使用流式处理数据源

您可以创建连续运行并使用来自流式处理源的数据的流式处理提取、转换和负载 (ETL) 任务，例如 Amazon Kinesis Data Streams、Apache Kafka 和 Amazon Managed Streaming for Apache Kafka (Amazon MSK)。

配置流式处理数据源的属性

1. 转到新任务或已保存任务的可视化图形编辑器。

2. 在图形中为 Kafka 或 Kinesis Data Streams 选择数据源节点。
3. 选择 Data source properties (数据源属性) 选项卡，然后输入以下信息：

   Kinesis

   - Kinesis source type（Kinesis 源类型）：选择选项 Stream details（流式传输详细信息）可使用直接访问串流源，或选择 Data Catalog table（数据目录表）使用存储在其中的信息。

     如果选择 Stream details（流式传输详细信息），请指定以下附加信息。

     - 数据流的位置：选择流式传输与当前用户关联，还是其他用户关联。
     - Region（区域）：选择存在流式传输的 AWS 区域。此信息用于构建用于访问数据流的 ARN。
     - Stream ARN（流式传输 ARN）：输入 Kinesis 数据流的 Amazon Resource Name (ARN)。如果流式传输位于当前账户内，则可以从下拉列表中选择流式传输名称。您可以使用搜索字段按名称或 ARN 搜索数据流。
     - Data format（数据格式）：从列表中选择数据流使用的格式。

       AWS Glue 会自动从流式处理数据中检测架构。

     如果选择 Data Catalog table（数据目录表），请指定以下附加信息。

     - Database (数据库)：（可选）在 AWS Glue 数据目录中选择数据库，该数据库包含与流式处理数据源关联的表。您可以使用搜索字段按名称搜索数据库。
     - Table (表)：（可选）从列表中选择与源数据关联的表。此表必须已存在于 AWS Glue 数据目录中。您可以使用搜索字段按名称搜索表。
     - Detect schema (检测架构)：选择该选项，允许 AWS Glue 从流式处理数据检测架构，而不是使用数据目录表中的架构信息。如果选择 Stream details（流式传输详细信息）选项，则会自动启用此选项。

   - Starting position（起始位置）：预设情况下，ETL 任务会使用 Earliest（最早）选项，这意味着从流式传输中最早的可用记录开启读取数据。您也可以选择 Latest（最新），表示 ETL 任务应该从流式传输中最新的记录之后开启读取。
   - Window size (窗口大小)：默认情况下，在 100 秒的时段内处理 ETL 任务和写出数据。这可以实现数据的高效处理，并允许对晚于预计时间到达的数据执行聚合。您可以修改此窗口大小以提高及时性或聚合精度。

     AWS Glue 流式处理任务使用检查点而非任务书签来跟踪已读取的数据。

- Connection options（连接选项）：展开此部分以添加键值对，指定其他连接选项。有关您可在此处指定的选项的信息，请参阅AWS Glue 开发人员指南中的 "connectionType": "kinesis"。

Kafka

- Apache Kafka source（Apache Kafka 源）：选择选项 Stream details（流式传输详细信息）可使用直接访问流式处理源，或选择 Data Catalog table（数据目录表）使用存储在其中的信息。

  如果选择 Data Catalog table（数据目录表），请指定以下附加信息。

  - Database (数据库)：（可选）在 AWS Glue 数据目录中选择数据库，该数据库包含与流式处理数据源关联的表。您可以使用搜索字段按名称搜索数据库。
  - Table (表)：（可选）从列表中选择与源数据关联的表。此表必须已存在于 AWS Glue 数据目录中。您可以使用搜索字段按名称搜索表。
  - Detect schema (检测架构)：选择该选项，允许 AWS Glue 从流式处理数据检测架构，而不是将架构信息存储于数据目录表。如果选择 Stream details（流式传输详细信息）选项，则会自动启用此选项。

  如果选择 Stream details（流式传输详细信息），请指定以下附加信息。

  - Connection name（连接名称）：选择包含 Kafka 数据流的访问和身份验证信息的 AWS Glue 连接。您必须使用与 Kafka 流式处理数据源的连接。如果连接不存在，可以使用 AWS Glue 控制台为 Kafka 数据流创建连接。
  - Topic name（主题名称）：输入要从中读取的主题的名称。
  - Data format（数据格式）：选择从 Kafka 事件流读取数据时使用的格式。
- Starting position（起始位置）：预设情况下，ETL 任务会使用 Earliest（最早）选项，这意味着从流式传输中最早的可用记录开启读取数据。您也可以选择 Latest（最新），表示 ETL 任务应该从流式传输中最新的记录之后开启读取。
- Window size (窗口大小)：默认情况下，在 100 秒的时段内处理 ETL 任务和写出数据。这可以实现数据的高效处理，并允许对晚于预计时间到达的数据执行聚合。您可以修改此窗口大小以提高及时性或聚合精度。

  AWS Glue 流式处理任务使用检查点而非任务书签来跟踪已读取的数据。

- Connection options（连接选项）：展开此部分以添加键值对，指定其他连接选项。有关您可在此处指定的选项的信息，请参阅AWS Glue 开发人员指南中的 "connectionType": "kafka"。

> **Note**
>
> 流式处理数据源当前不支持数据预览。

参考信息

最佳实践

- 使用 AWS Glue 构建 ETL 服务管道以增量方式将数据从 Amazon S3 加载到 Amazon Redshift

ETL 编程

- AWS Glue 中的 ETL 的连接类型和选项
- JDBC connectionType 值
- 用于从 Amazon Redshift 移入移出数据的高级选项

使用您自己的 JDBC 驱动程序添加 JDBC 连接

使用 JDBC 连接时，您可以使用自己的 JDBC 驱动程序。当 AWS Glue 爬网程序使用的默认驱动程序无法连接到数据库时，您可以使用自己的 JDBC 驱动程序。例如，如果您想在 Postgres 数据库中使用 SHA-256，而较早的 Postgres 驱动程序不支持此功能，则可以使用自己的 JDBC 驱动程序。

支持的数据来源

支持的数据来源	不支持的数据来源
MySQL	Snowflake
Postgres	
Oracle	

支持的数据来源	不支持的数据来源
Redshift	
SQL Server	
Aurora*	

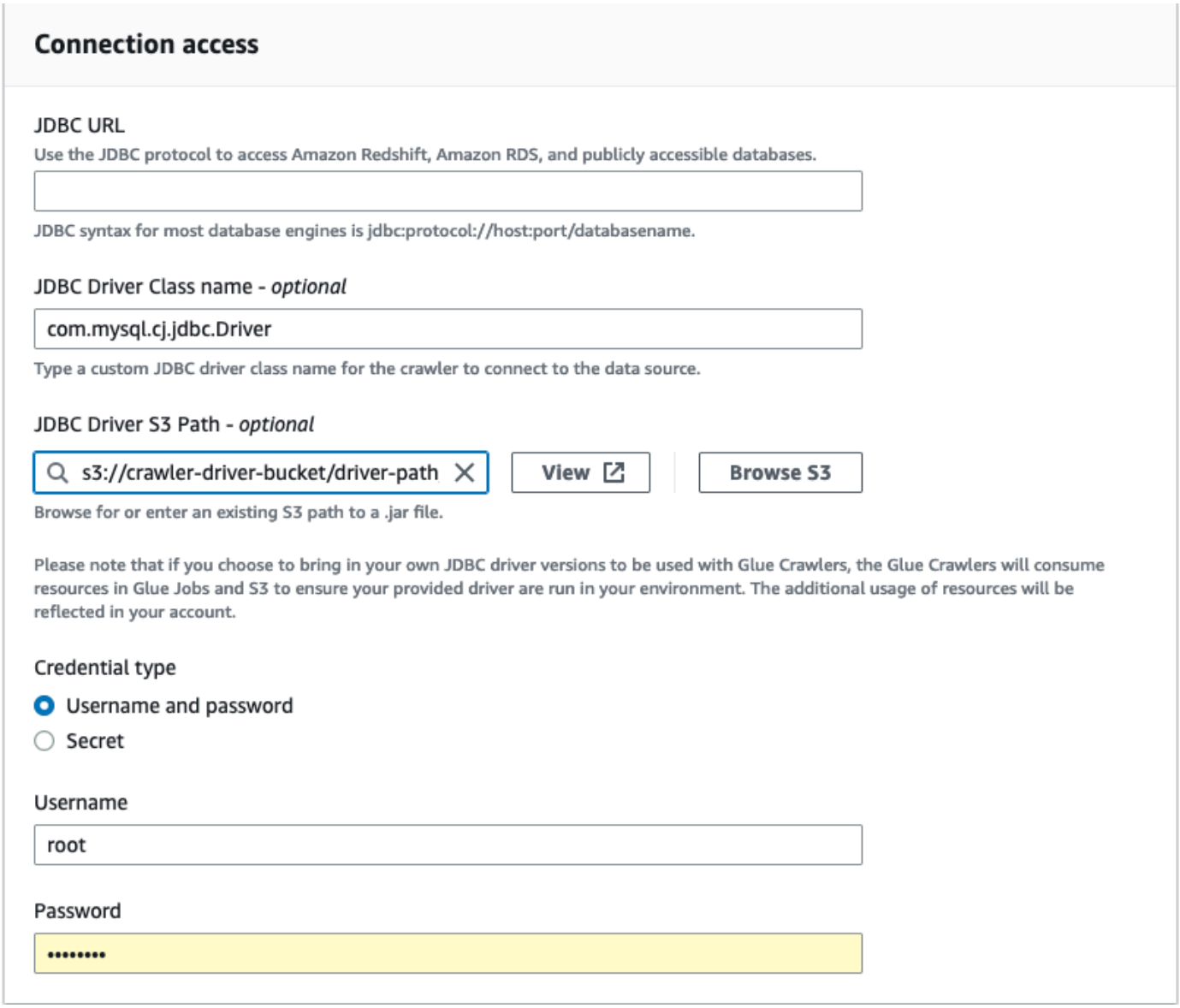
* 如果使用原生 JDBC 驱动程序，则支持。并非所有驱动程序功能都可以利用。

向 JDBC 连接中添加 JDBC 驱动程序

> **Note**
>
> 如果您选择引入自己的 JDBC 驱动程序版本，则 AWS Glue 爬网程序将消耗 AWS Glue 作业和 Amazon S3 存储桶中的资源，以确保您提供的驱动程序在您的环境中运行。额外的资源使用量将反映在您的账户中。AWS Glue 爬网程序和作业的成本属于计费 AWS Glue 类别。此外，提供自己的 JDBC 驱动程序并不意味着爬网程序能够利用该驱动程序的所有功能。

将您自己的 JDBC 驱动程序添加到 JDBC 连接：

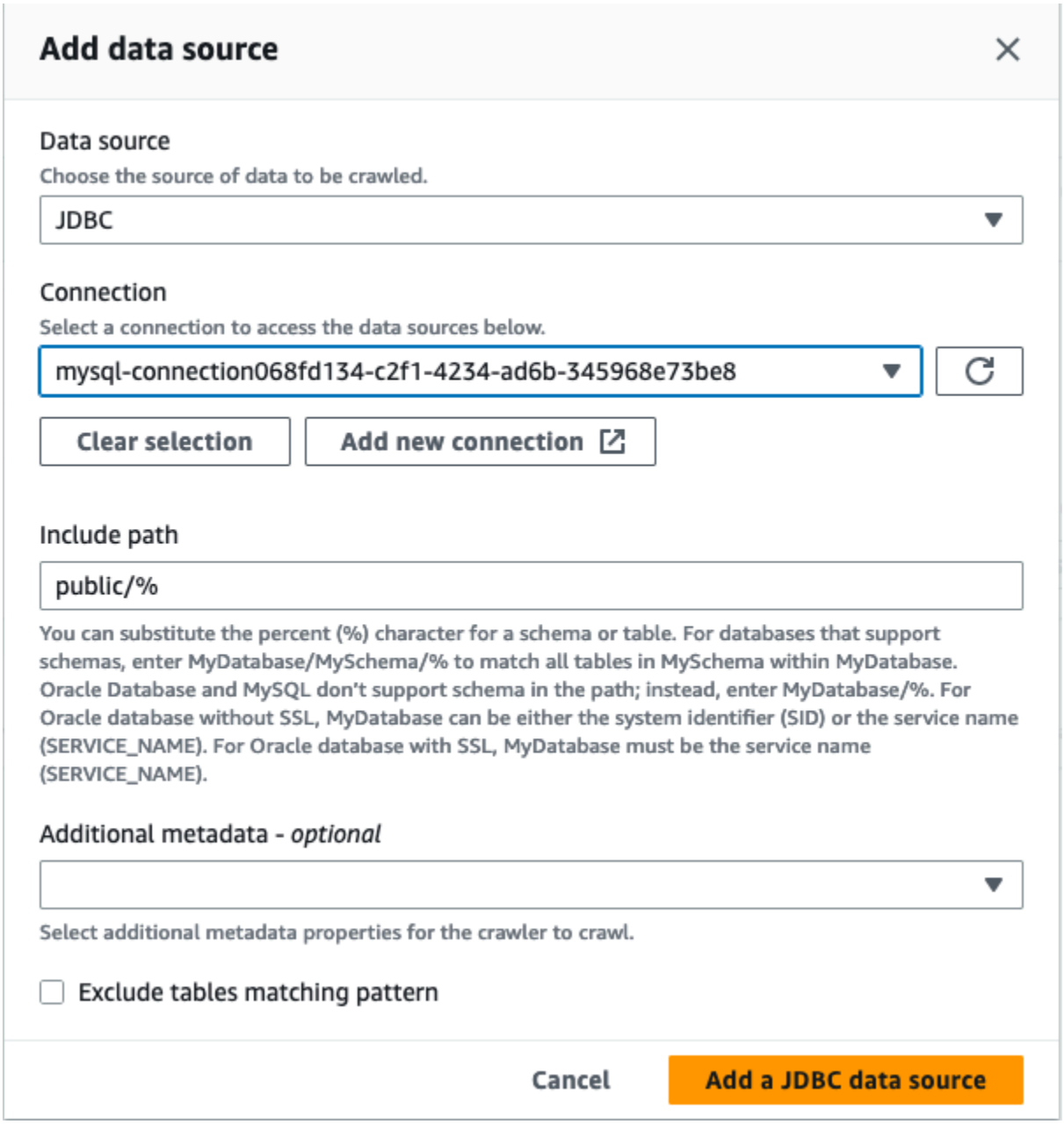
1. 将 JDBC 驱动程序文件添加到 Amazon S3 位置。您可以创建存储桶和/或文件夹，或使用现有存储桶和/或文件夹。
2. 在 AWS Glue 控制台中，选择 Data Catalog 下方左侧菜单中的连接，然后创建新连接。
3. 填写连接属性字段，然后为连接类型选择 JDBC。
4. 在连接访问中，输入 JDBC URL 和 JDBC 驱动程序类名 -（可选）。驱动程序类名必须是 AWS Glue 爬网程序支持的数据来源的名称。



5. 在 JDBC 驱动程序 Amazon S3 路径 - 可选字段中选择 JDBC 驱动程序所在的 Amazon S3 路径。
6. 如果输入用户名和密码或密钥，请填写“凭证类型”字段。完成后，选择创建连接。

> **Note**
>
> 目前不支持测试连接。使用您提供的 JDBC 驱动程序对数据来源进行爬取时，爬网程序会跳过此步骤。

7. 将新创建的连接添加到爬网程序。在 AWS Glue 控制台中，选择 Data Catalog 下方左侧菜单中的爬网程序，然后创建新爬网程序。
8. 在添加爬网程序向导中，在步骤 2 中选择添加数据来源。



9. 选择 JDBC 作为数据来源，然后选择在前面的步骤中创建的连接。完成
10. 要将自己的 JDBC 驱动程序与 AWS Glue 爬虫一起使用，请向该爬虫使用的角色添加以下权限：

    - 授予以下作业操作的权限：`CreateJob`、`DeleteJob`、`GetJob`、`GetJobRun`、`StartJobRun`。
    - 授予 IAM 操作的权限：`iam:PassRole`
    - 授予 Amazon S3 操作的权限：`s3:DeleteObjects`、`s3:GetObject`、`s3:ListBucket`、`s3:PutObject`。
    - 在 IAM policy 中授予服务主体访问存储桶/文件夹的权限。

示例 IAM policy：

```
{
    "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，则必须通过创建接口端点并将其添加到您的路由表中来允许访问 AWS Glue 端点。有关更多信息，请参阅 Creating an interface VPC endpoint for AWS Glue
12. 如果您在数据目录中使用加密，请创建 AWS KMS 接口终端节点并将其添加到您的路由表中。有关更多信息，请参阅为 AWS KMS创建 VPC 端点。

测试 AWS Glue 连接

作为最佳实践，在您在 ETL 作业中使用 AWS Glue 连接之前，最好先使用 AWS Glue 控制台测试该连接。AWS Glue 会使用连接中的参数来确认该连接是否可以访问数据存储，并报告任何错误。有关 AWS Glue 连接的信息，请参阅连接到数据。

测试 AWS Glue 连接

1. 登录 AWS Management Console 并打开 AWS Glue 控制台，网址为 https://console.aws.amazon.com/glue/。

2. 在导航窗格的 Data Catalog 下，选择连接。您也可以在导航窗格中选择 Data Catalog 上方的数据连接。
3. 在连接中选中所需连接旁边的复选框，然后选择操作。在下拉菜单中，选择测试连接。
4. 在 “测试连接” 对话框中，选择一个角色或选择 “创建 IAM 角色”，进入 AWS Identity and Access Management (IAM) 控制台创建新角色。该角色必须拥有相关数据存储的权限。
5. 选择确认。

   开始测试，测试可能需要几分钟才能完成。如果测试失败，请选择故障排除以查看解决问题的步骤。
6. 选择 “日志” 以查看日志 CloudWatch。您必须拥有所需的 IAM 权限才能查看日志。有关更多信息，请参阅 Amazon CloudWatch Logs 用户指南中的AWS 托管（预定义）CloudWatch 日志策略。

将 AWS 调用配置为通过 VPC

特殊作业参数 `disable-proxy-v2` 将允许您将调用通过 VPC 路由到服务，如 Amazon S3、CloudWatch 和 AWS Glue。默认情况下，AWS Glue 将使用本地代理通过 AWS Glue VPC 发送流量，以从 Amazon S3 下载脚本和库；向 CloudWatch 发送请求，以发布日志和指标；以及向 AWS Glue 发送请求，以访问数据目录。即使您的 VPC 没有将正确的路由配置到其他 AWS 服务，如 Amazon S3、CloudWatch 和 AWS Glue，此代理也会允许任务正常运行。AWS Glue 现在为您提供参数，以关闭此行为。有关更多信息，请参阅 AWS Glue 所使用的作业参数。AWS Glue 将继续使用本地代理发布您的 AWS Glue 作业的 CloudWatch 日志。

> **Note**
>
> - 使用 AWS Glue 版本 2.0 及更高版本的 AWS Glue 任务支持此功能。使用此功能时，您需要确保您的 VPC 已通过 NAT 或服务 VPC 终端节点将路由配置到 Simple Storage Service (Amazon S3)。
> - 已弃用的任务参数 `disable-proxy` 仅会将您的调用路由到 Amazon S3，以便通过 VPC 下载脚本和库。建议改用新参数 `disable-proxy-v2`。

示例用法

创建带有 `disable-proxy-v2` 的 AWS Glue 任务：

```
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 中连接到 JDBC 数据存储

通常，您在 Amazon Virtual Private Cloud（Amazon VPC）内部创建资源，以便这些资源不能通过公共 Internet 访问。默认情况下，AWS Glue 无法访问 VPC 中的资源。要让 AWS Glue 能够访问 VPC 中的资源，您必须提供包括 VPC 子网 ID 和安全组 ID 在内的其他 VPC 特定的配置信息。AWS Glue 使用此信息设置弹性网络接口，此接口可让您的函数安全连接到私有 VPC 中的其他资源。

使用 VPC 端点时，请将其添加到您的路由表中。有关更多信息，请参阅 Creating an interface VPC endpoint for AWS Glue 和 先决条件。

在 Data Catalog 中使用加密时，请创建 KMS 接口端点并将其添加到您的路由表中。有关更多信息，请参阅 Creating a VPC endpoint for AWS KMS。

使用弹性网络接口访问 VPC 数据

当 AWS Glue 连接到 VPC 中的 JDBC 数据存储时，AWS Glue 会在您的账户中创建弹性网络接口（使用前缀 `Glue_`）以访问您的 VPC 数据。只要此网络接口连接到 AWS Glue，您就不能删除它。作为创建弹性网络接口的一部分，AWS Glue 将一个或多个安全组关联到它。要使 AWS Glue 创建网络接口，与该资源关联的安全组必须允许具有源规则的入站访问。此规则包含与资源相关联的安全组。这将为具有相同安全组的数据存储提供弹性网络接口访问。

要允许 AWS Glue 与其组件通信，请为所有 TCP 端口指定一个具有自引用入站规则的安全组。通过创建自引用规则，您可以将源限制为 VPC 中的同一安全组，而不将其对所有网络打开。VPC 的默认安全组可能已经为 `ALL Traffic` 设置了自引用入站规则。

您可以在 Amazon VPC 控制台中创建规则。要通过 AWS Management Console 更新规则设置，请导航到 VPC 控制台（https://console.aws.amazon.com/vpc/），然后选择相应的安全组。为 `ALL TCP` 指定入站规则，使其源具有相同的安全组名称。有关组规则的更多信息，请参阅您的 VPC 的安全组。

每个弹性网络接口都会从您指定的子网中的 IP 地址范围分到一个私有 IP 地址。网络接口不会分到任何公有 IP 地址。AWS Glue 需要 Internet 访问（例如，访问没有 VPC 终端节点的 AWS 服务）。您可以在 VPC 中配置网络地址转换（NAT）实例，也可以使用 Amazon VPC NAT 网关。有关更多信息，请

参阅《Amazon VPC 用户指南》中的 NAT 网关。您不能直接使用连接到 VPC 的 Internet 网关作为子网路由表中的路由，因为这需要网络接口具有公有 IP 地址。

VPC 网络属性 `enableDnsHostnames` 和 `enableDnsSupport` 必须设置为 true。有关更多信息，请参阅将 DNS 与您的 VPC 一起使用。

> **Important**
>
> 不要将数据存储置在未接入 Internet 的公有子网或私有子网中。相反，仅通过 NAT 实例或 Amazon VPC NAT 网关将它附加到接入了 Internet 的私有子网。

弹性网络接口属性

要创建弹性网络接口时，您必须提供以下属性：

VPC

包含您的数据存储的 VPC 的名称。

子网

包含您的数据存储的 VPC 中的子网。

安全组

与您的数据存储关联的安全组。AWS Glue 将这些安全组与连接到您的 VPC 的子网的弹性网络接口关联。要允许 AWS Glue 组件进行通信并阻止来自其他网络的访问，则至少必须有一个选定的安全组指定针对所有 TCP 端口的自引用入站规则。

有关使用 Amazon Redshift 管理 VPC 的信息，请参阅在 Amazon Virtual Private Cloud（VPC）中管理集群。

有关使用 Amazon Relational Database Service（Amazon RDS）管理 VPC 的信息，请参阅在 VPC 中使用 Amazon RDS 数据库实例。

使用 MongoDB 或 MongoDB Atlas 连接

为 MongoDB 或 MongoDB Atlas 创建连接后，您可以在 ETL 作业中使用该连接。您可以在 AWS Glue Data Catalog 中创建一个表，并为该表的 `connection` 属性指定 MongoDB Atlas 连接。

AWS Glue 在 MongoDB 连接中存储您的连接 `url` 和凭证。连接 URI 格式如下：

- 对于 MongoDB：mongodb://host:port/database。主机可以是主机名、IP 地址或 UNIX 域套接字。如果连接字符串未指定端口，则使用默认的 MongoDB 端口 27017。
- 对于 MongoDB Atlas：mongodb+srv://server.example.com/database。主机可以是后面对应于 DNS SRV 记录的主机名。SRV 格式不需要端口，将使用默认的 MongoDB 端口 27017。

此外，您可以在作业脚本中指定选项。有关更多信息，请参阅 the section called “MongoDB 连接”。

使用 VPC 终端节点网络爬取 Amazon S3 数据存储

出于安全、审计或控制目的，您可能希望您的 Amazon S3 数据存储或 Amazon S3 支持的数据目录表只能通过 Amazon Virtual Private Cloud 环境（Amazon VPC）访问。本主题介绍如何使用 `Network` 连接类型在 VPC 终端节点中创建和测试与 Amazon S3 数据存储或 Amazon S3 支持的数据目录表的连接。

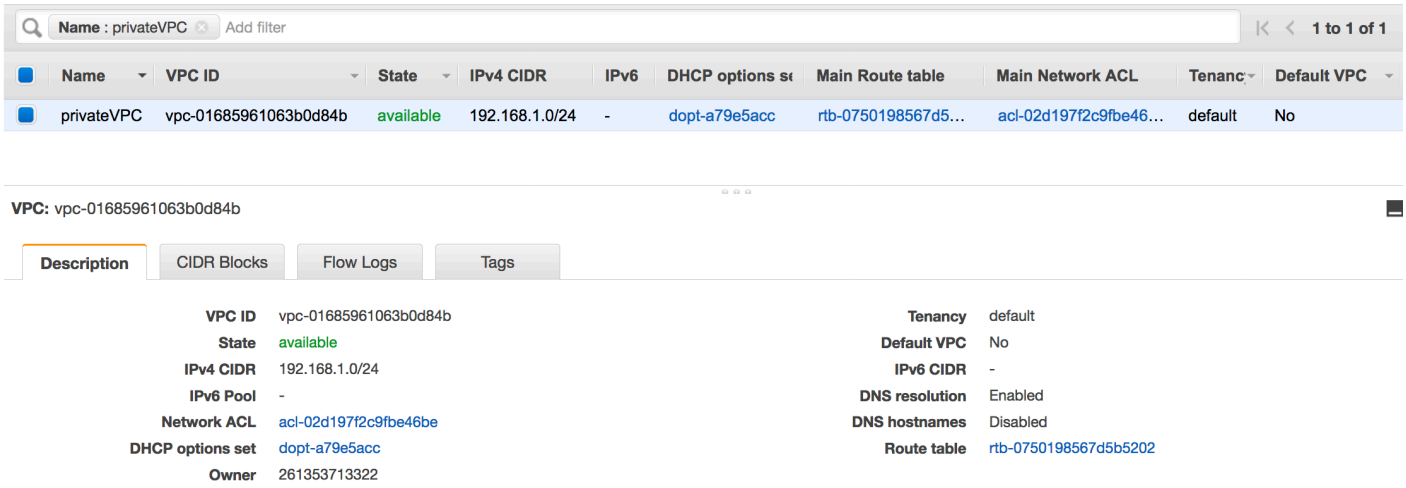
执行以下任务以在数据存储中运行爬网程序：

- the section called “先决条件”
- the section called “创建到 Amazon S3 的连接”
- the section called “测试 Amazon S3 的连接”
- the section called “为 Amazon S3 数据存储创建爬网程序”
- the section called “运行爬网程序”

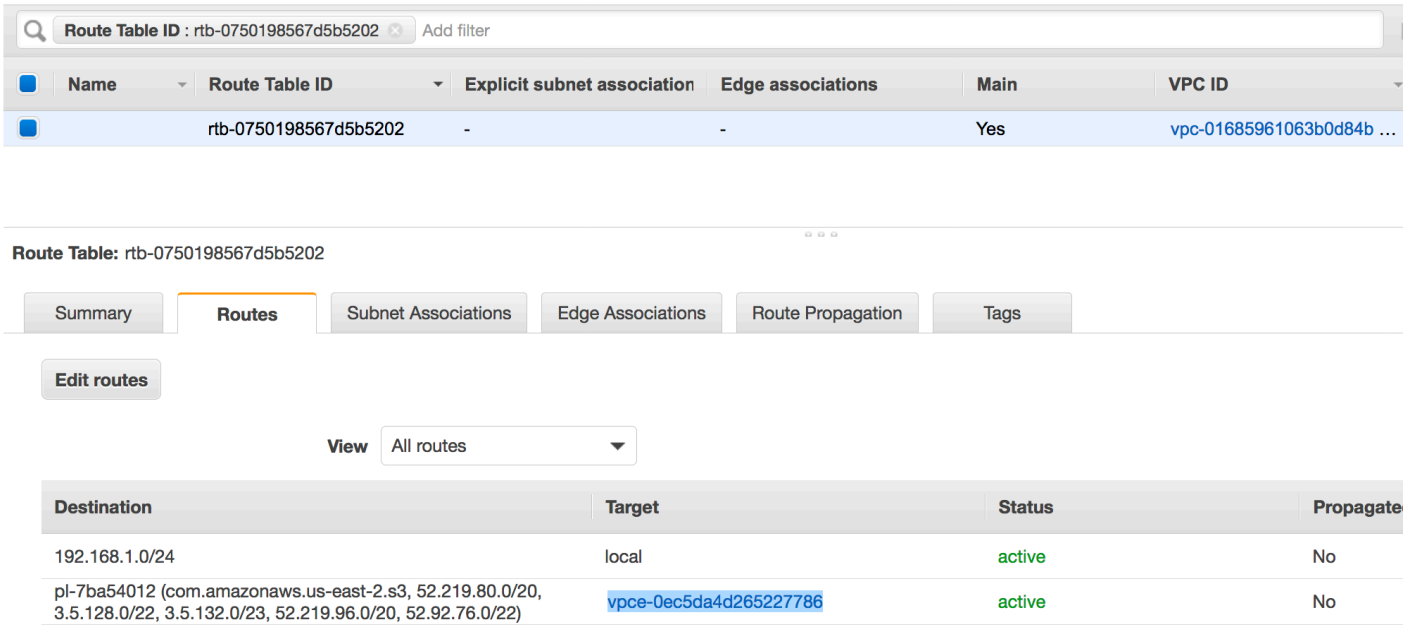
先决条件

检查您是否满足设置 Amazon S3 数据存储或 Amazon S3 支持的数据目录表以通过 Amazon Virtual Private Cloud 环境（Amazon VPC）访问的先决条件。

- 已配置的 VPC。例如：vpc-01685961063b0d84b。有关更多信息，请参阅《Amazon VPC 用户指南》中的 Amazon VPC 入门。
- 附加到 VPC 的 Amazon S3 端点。例如：vpc-01685961063b0d84b。有关更多信息，请参阅《Amazon VPC 用户指南》中的 Amazon S3 端点。



- 指向 VPC 终端节点的路由条目。例如，VPC endpoint(vpce-0ec5da4d265227786) 使用的路由表中的 vpce-0ec5da4d265227786。



- 附加到 VPC 的网络 ACL 允许流量。
- 附加到 VPC 的安全组允许流量。

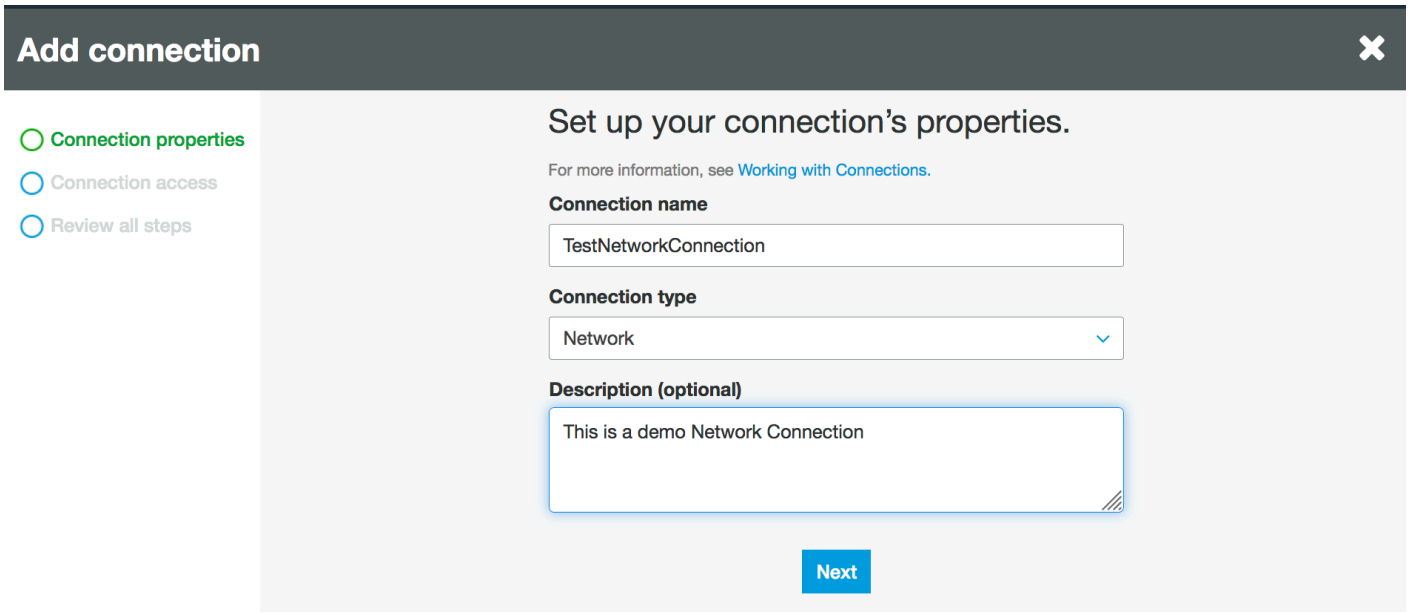
创建到 Amazon S3 的连接

通常，您在 Amazon Virtual Private Cloud（Amazon VPC）内部创建资源，以便这些资源不能通过公共 Internet 访问。默认情况下，AWS Glue 无法访问 VPC 中的资源。要让 AWS Glue 能够访问 VPC 中的资源，您必须提供包括 VPC 子网 ID 和安全组 ID 在内的其他 VPC 特定的配置信息。要创建 Network 连接，您需要指定以下信息：

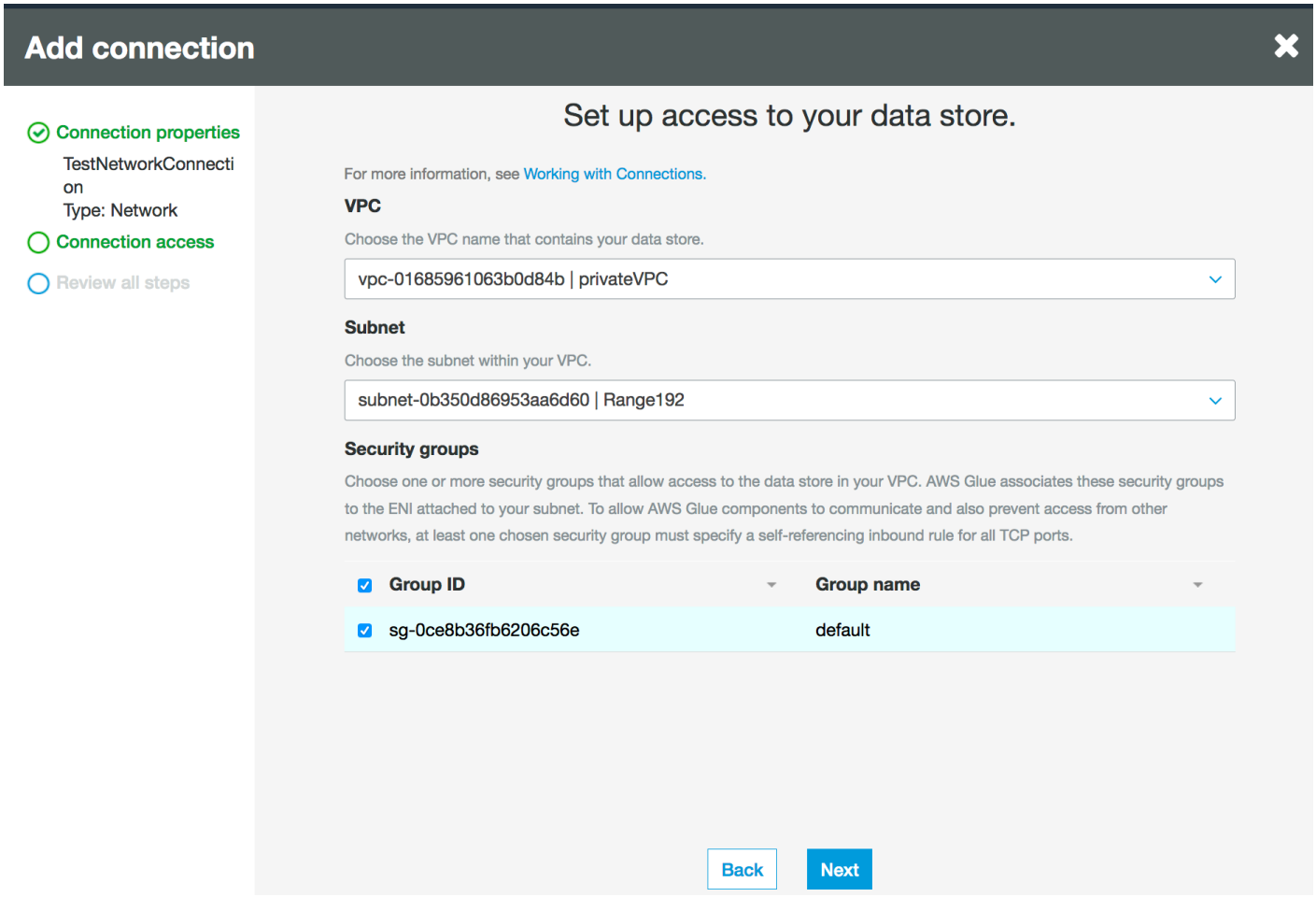
- VPC ID
- VPC 内的子网
- 安全组

设置 Network 连接

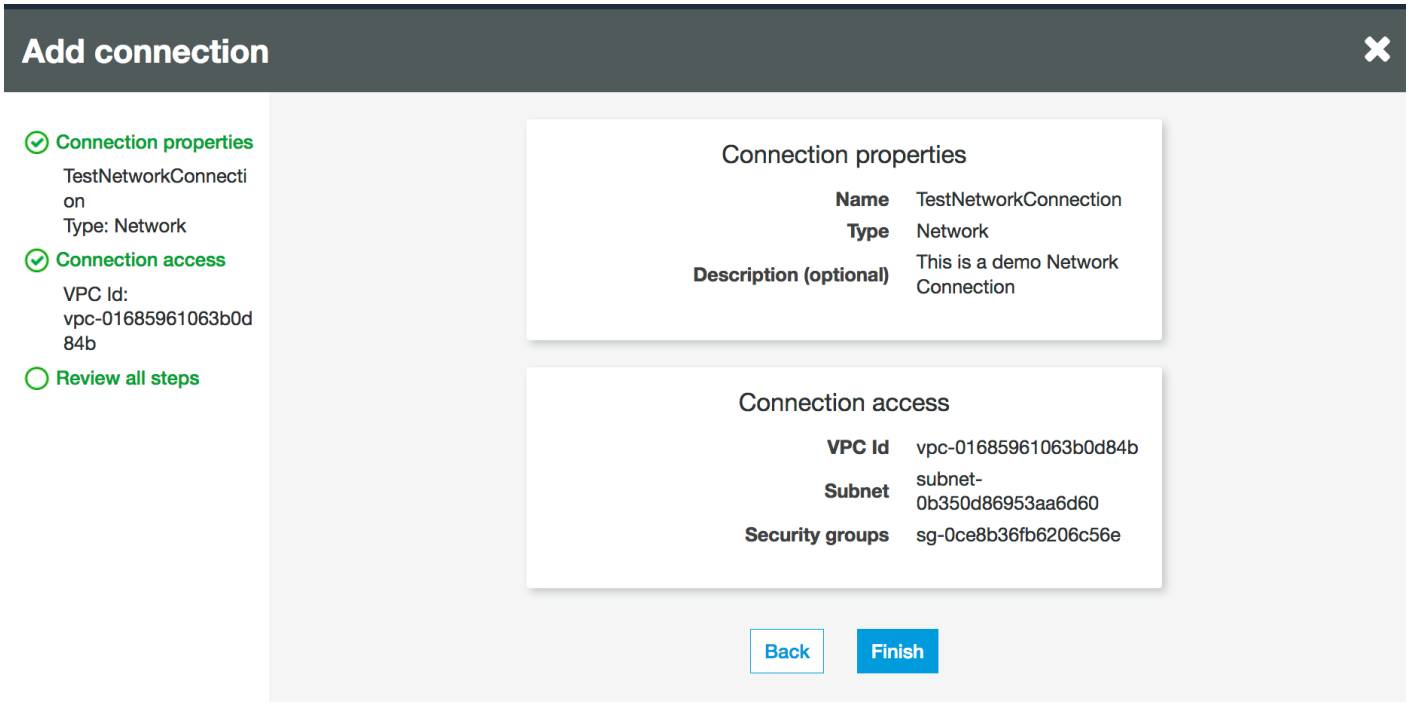
1. 选择 AWS Glue 控制台导航窗格中的 Add connection (添加连接)。
2. 输入连接名称，选择 Network (网络) 作为连接类型。选择下一步。



3. 配置 VPC、子网和安全组信息。
   - VPC：选择包含您的数据存储的 VPC 名称。
   - 子网：选择 VPC 内的子网。
   - 安全组：选择允许访问 VPC 中数据存储的一个或多个安全组。



4. 选择下一步。
5. 验证连接信息并选择 Finish (完成)。



测试 Amazon S3 的连接

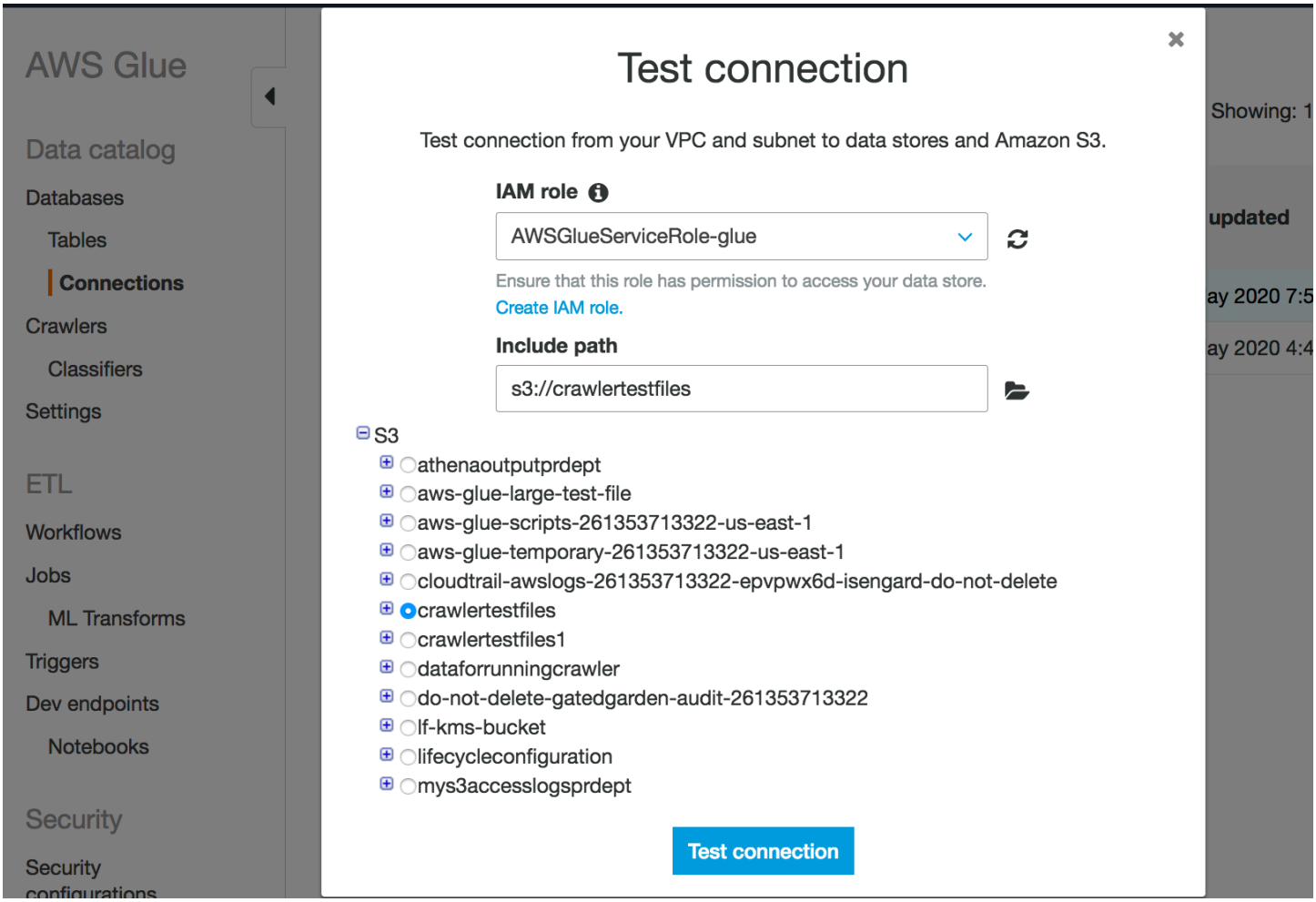
创建 Network 连接后，您可以在 VPC 终端节点中测试 Amazon S3 数据存储的连接。

测试连接时可能会发生以下错误：

- INTERNET CONNECTION ERROR：表示互联网连接问题
- INVALID BUCKET ERROR：表示 Amazon S3 存储桶存在问题
- S3 CONNECTION ERROR：表示未能连接到 Amazon S3
- INVALID CONNECTION TYPE：表示连接类型不具有预期值 NETWORK
- INVALID CONNECTION TEST TYPE：表示网络连接测试类型存在问题
- INVALID TARGET：表示未正确指定 Amazon S3 存储桶

测试 Network 连接：

1. 选择 AWS Glue 控制台中的 Network (网络) 连接。
2. 选择 Test connection (测试连接)。
3. 选择您在上一步中创建的 IAM 角色并指定 Amazon S3 存储桶。
4. 选择 Test connection (测试连接)，开始测试。显示结果可能需要一些时间。



如果收到错误，请检查以下几点：

- 为所选角色提供正确的权限。
- 提供了正确的 Amazon S3 存储桶。
- 安全组和网络 ACL 允许所需的传入和传出流量。
- 您指定的 VPC 已连接到 Amazon S3 VPC 终端节点。

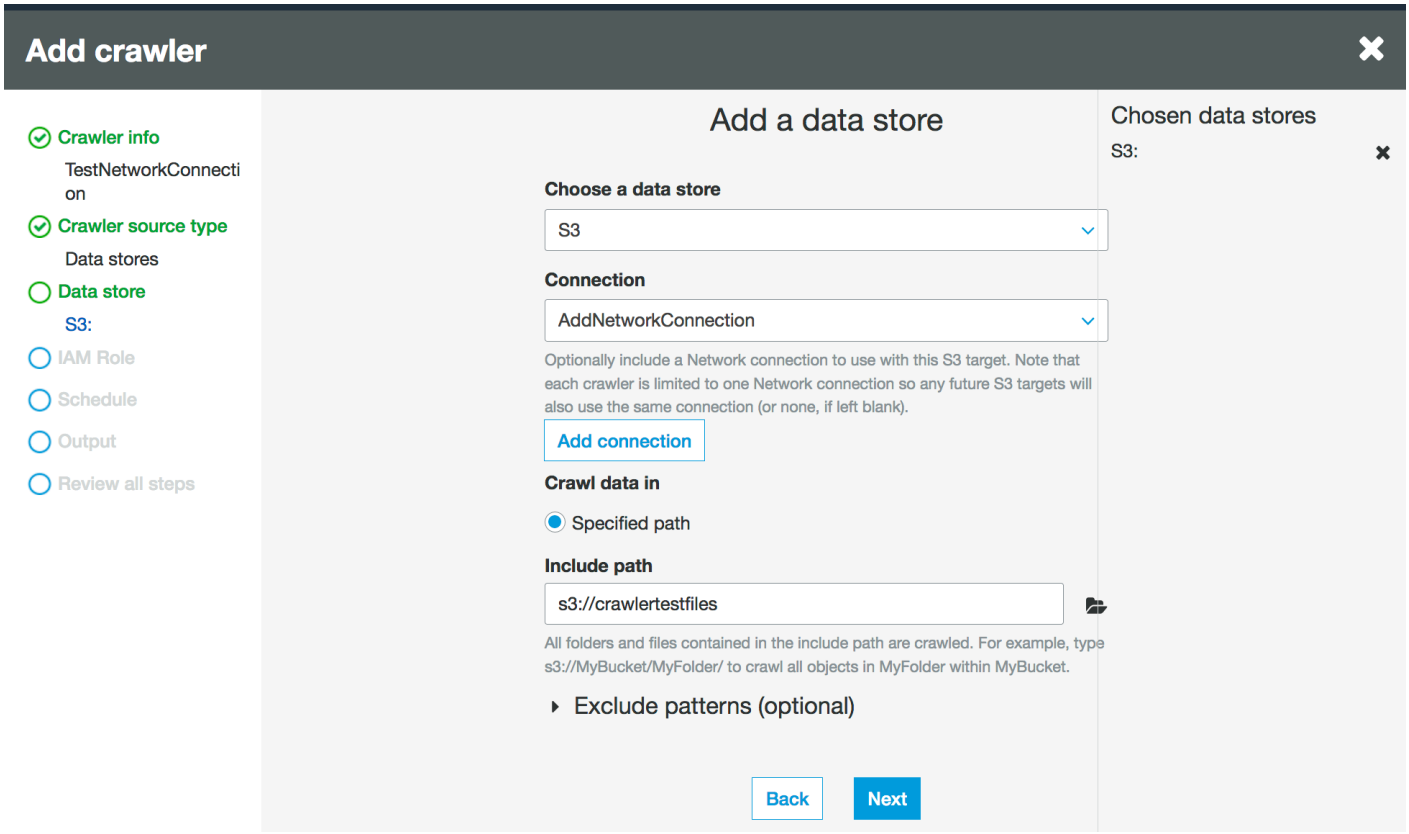
成功测试连接后，您便可创建爬网程序。

为 Amazon S3 数据存储创建爬网程序

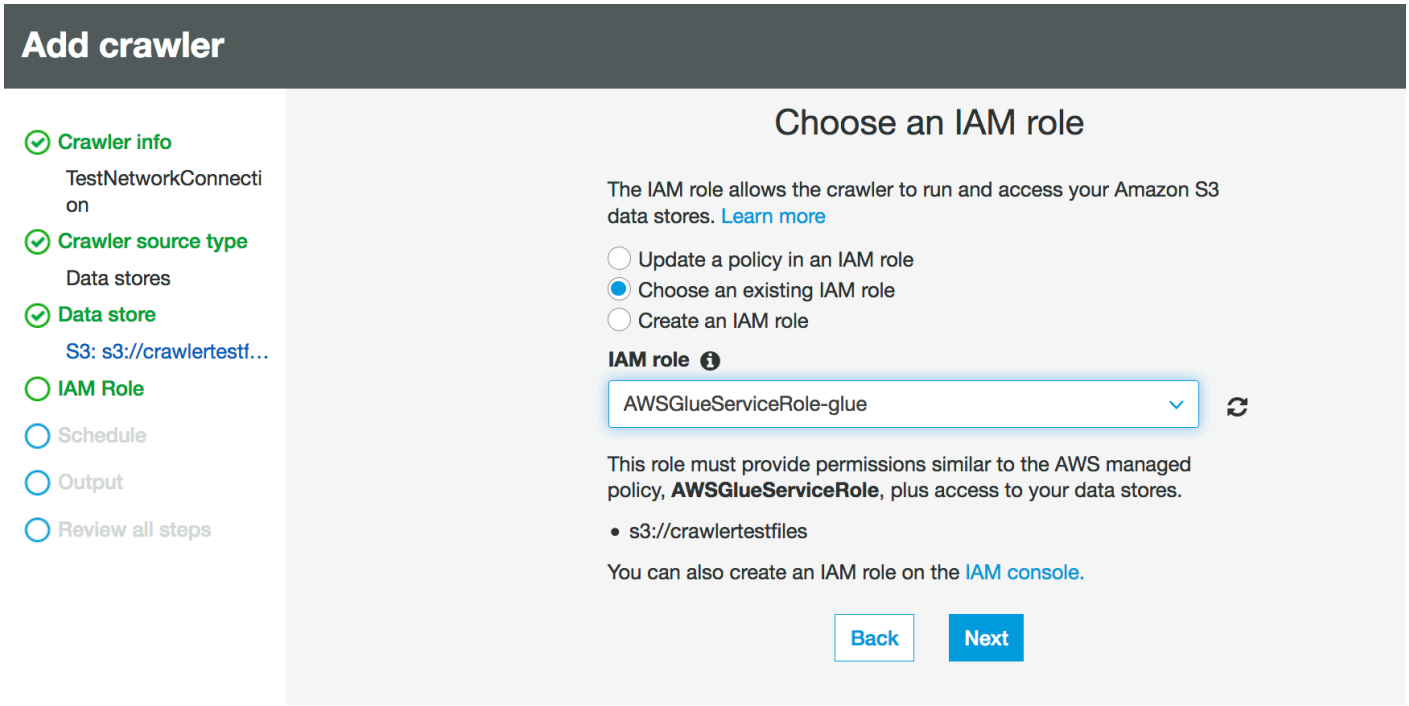
现在，您可以创建一个爬网程序来指定您已创建的 `Network` 连接。有关创建爬网程序的更多详细信息，请参阅配置爬网程序。

1. 首先在 AWS Glue 控制台上的导航窗格中选择 Crawlers (爬网程序)。

2. 选择 添加爬网程序。
3. 指定爬网程序名称，选择 Next (下一步)。
4. 当询问数据源时，选择 S3，并指定 Amazon S3 存储桶前缀和您先前创建的连接。

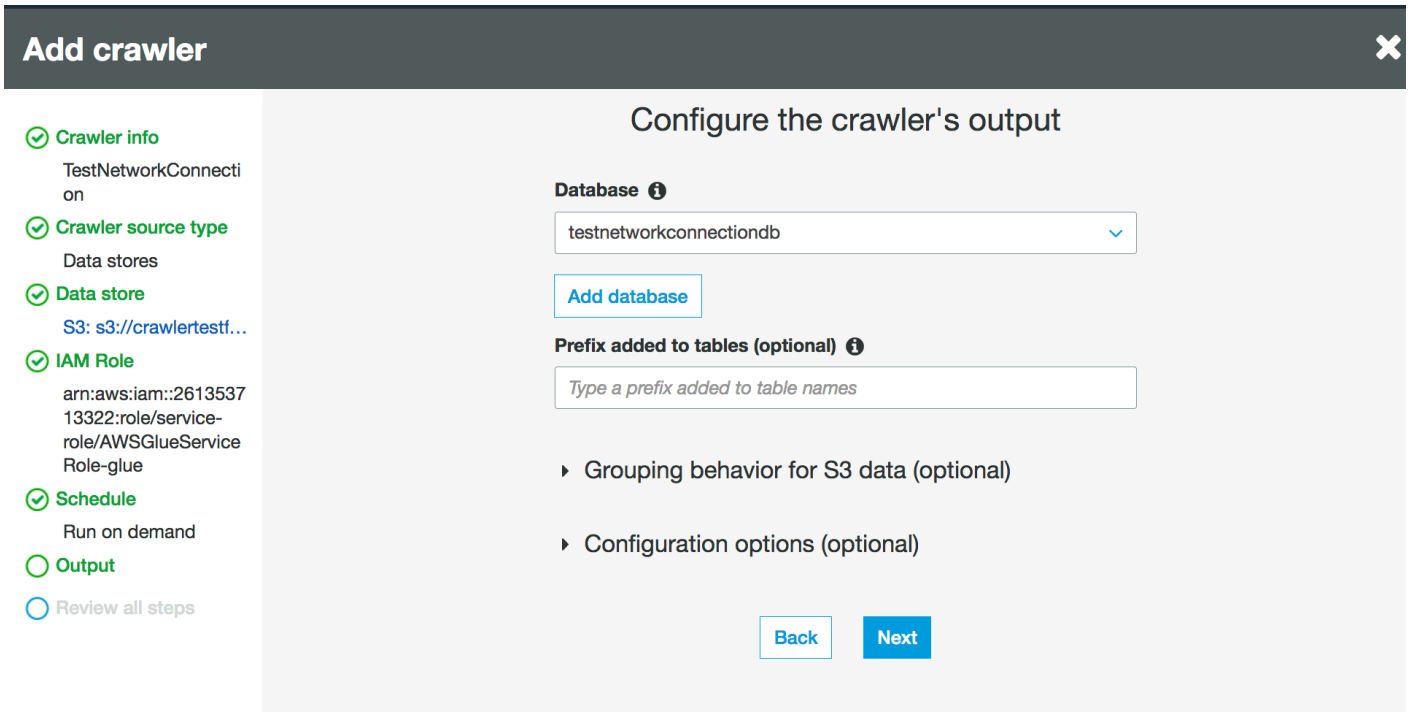


5. 如果需要，请在同一网络连接上添加另一个数据存储。
6. 选择 IAM 角色。IAM 角色必须允许访问 AWS Glue 服务和 Amazon S3 存储桶。有关更多信息，请参阅 the section called “配置爬网程序”。



7. 定义爬网程序的计划。

8. 在数据目录中选择一个现有数据库或创建一个新的数据库条目。

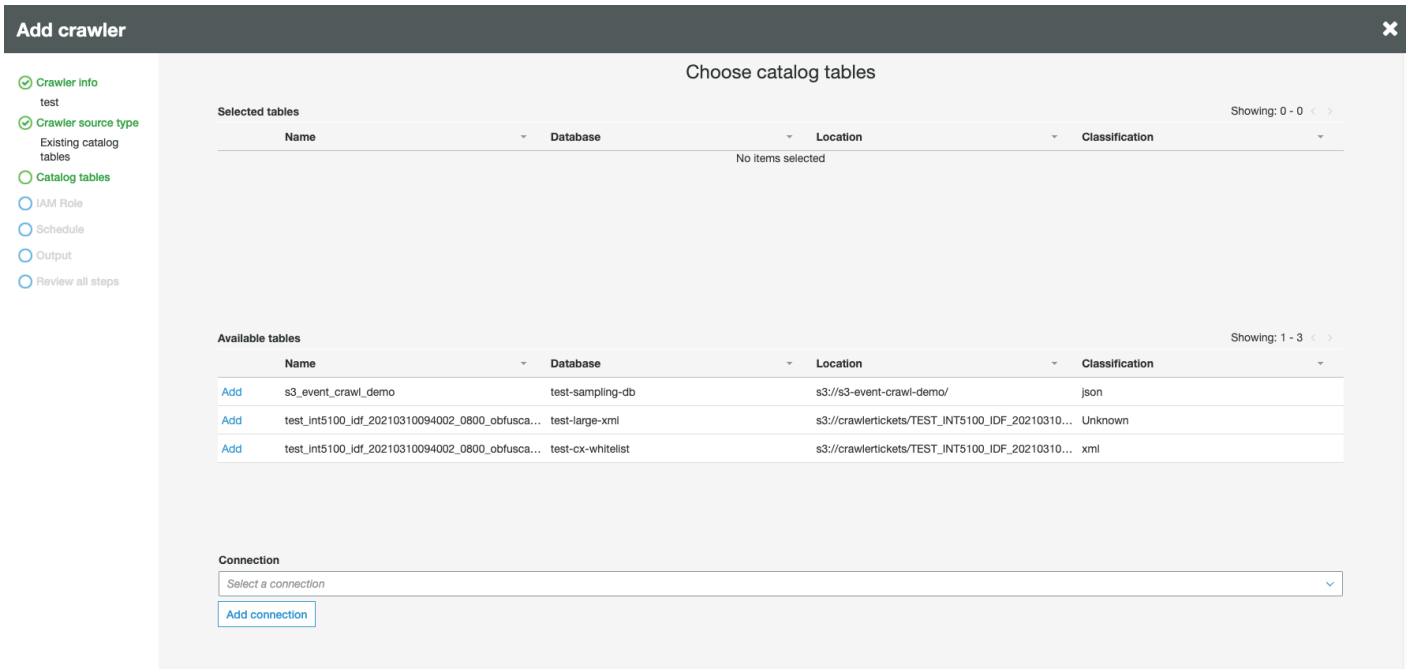


9. 完成剩余的设置。

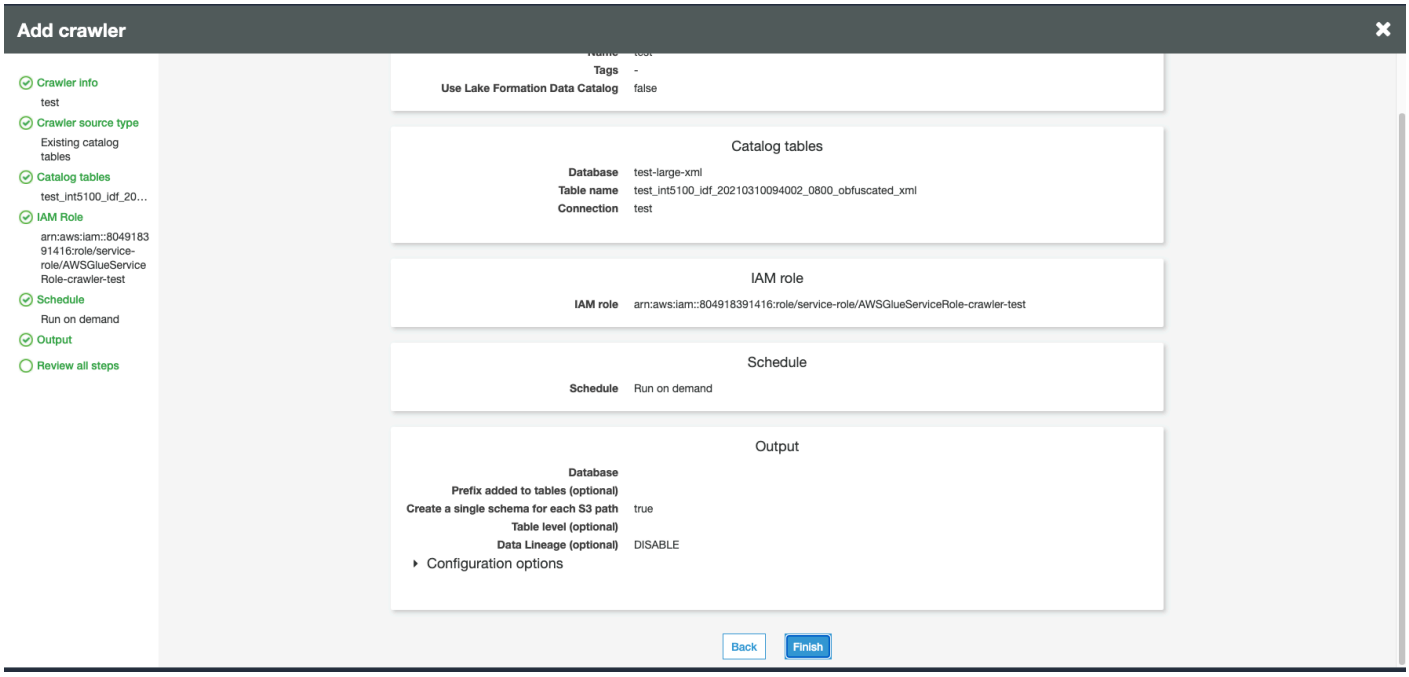
为 Amazon S3 支持的数据目录表创建网络爬取程序

现在，您可以创建指定您已创建 Network 连接的爬取程序和目录源类型。有关创建爬网程序的更多详细信息，请参阅配置爬网程序。

1. 首先在 AWS Glue 控制台上的导航窗格中选择 Crawlers (爬网程序)。
2. 选择 添加爬网程序。
3. 指定爬网程序名称，选择 Next (下一步)。
4. 当询问网络爬取程序源类型时，选择现有目录表，然后指定要从可用表列表中网络爬取的现有目录表。

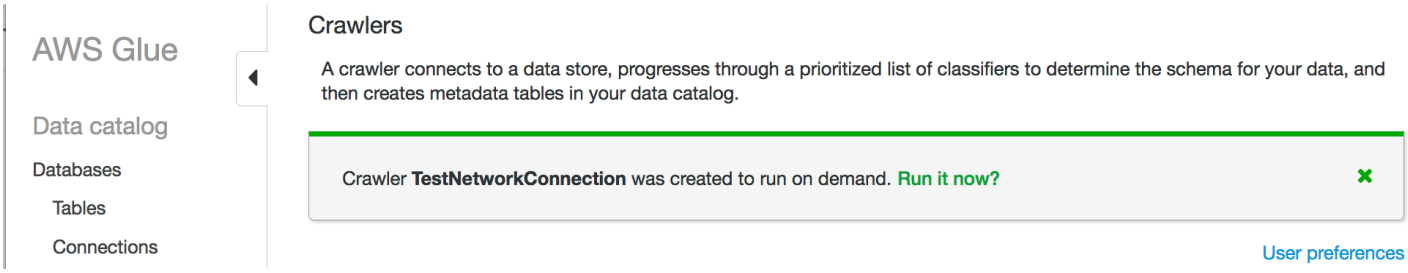


5. 选择 IAM 角色。IAM 角色必须允许访问 AWS Glue 服务和 Amazon S3 存储桶。有关更多信息，请参阅 the section called “配置爬网程序”。
6. 定义爬网程序的计划。
7. 在数据目录中选择一个现有数据库或创建一个新的数据库条目。
8. 完成剩余的设置并查看步骤。



运行爬网程序

运行您的爬网程序。



故障排除

对于与使用 VPC 网关的 Amazon S3 存储桶相关的故障排除，请参阅为什么我无法使用网关 VPC 终端节点连接到 S3 存储桶？

解决 AWS Glue 中的连接问题

当 AWS Glue 爬网程序或任务使用连接属性访问数据存储时，您在尝试连接时可能会遇到错误。AWS Glue 在您指定的 Virtual Private Cloud（VPC）和子网中创建弹性网络接口时，会在子网中使用私有 IP 地址。在 连接中指定的安全组应用于每个弹性网络接口上。请检查安全组是否允许出站访问，以及是否允许连接到数据库集群。

此外，Apache Spark 需要驱动程序和执行器节点之间的双向连接。其中一个安全组需要在所有 TCP 端口上允许入口规则。通过使用自引用安全组将安全组的源限制为自身，可以防止它对世界开放。

以下是您为解决连接问题可以采取的一些典型操作：

- 检查连接的端口地址。
- 检查连接或密钥中的用户名和密码字符串。
- 对于 JDBC 数据存储，验证其是否允许传入连接。
- 验证是否可以在您的 VPC 中访问您的数据存储。
- 如果您使用 AWS Secrets Manager 存储连接凭证，请确保 AWS Glue 的 IAM 角色拥有访问密钥的权限。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的示例：检索密钥值的权限。根据网络设置，您可能还需要创建一个 VPC 端点，以在 VPC 与 Secrets Manager 之间建立私有连接。有关更多信息，请参阅使用 AWS Secrets Manager VPC 端点。

教程：使用 AWS Glue Connector for Elasticsearch

Elasticsearch 是一个流行的开源搜索和分析引擎，其用例包括日志分析、实时应用程序监控和点击流分析等。您可以在 AWS Glue Studio 中配置 AWS Glue Connector for Elasticsearch，从而将 OpenSearch 用作提取、转换、加载（ETL）任务的数据存储。此连接器在 AWS Marketplace 中免费提供。

> **Note**
>
> AWS Marketplace Elasticsearch Spark Connector 已被弃用。请改用 AWS Glue Connector for Elasticsearch 。

在本教程中，我们将展示如何以最少的步骤连接到您的 Amazon OpenSearch Service 节点。

主题

- 步骤 6：运行任务

先决条件

要使用本教程，您必须具备以下内容：

- 对 AWS Glue Studio 的访问权限
- 对 AWS 云中访问 OpenSearch 集群的访问权限
- （可选）对 AWS Secrets Manager 的访问权限。

步骤 1：（可选）为您的 OpenSearch 集群信息创建 AWS 密钥

要安全地存储和使用您的连接凭证，请将您的凭证保存在 AWS Secrets Manager 中。您创建的密钥将在本教程稍后的连接中使用。凭证键值对将作为普通连接选项注入 AWS Glue Connector for Elasticsearch。

有关创建密钥的更多信息，请参阅 使用、创建和管理密钥 AWS Secrets Manager 于 《AWS Secrets Manager 用户指南》。

创建 AWS 密钥

1. 登录 AWS Secrets Manager 控制台。
2. 在服务介绍页面或密钥列表页面上，选择 Store a new secret (存储新密钥)。
3. 在 Store a new secret（存储新密钥）页上，选择 Other type of secret（其他密钥类型）。此选项意味着您必须提供您的密钥的结构和详细信息。
4. 为 OpenSearch 集群用户名添加 Key（密钥）和 Value（值）对。例如：

   `es.net.http.auth.user`：*`username`*（用户名）
5. 选择 + Add row（+ 添加行），然后为密码输入另一个键/值对。例如：

   `es.net.http.auth.pass`：*`password`*
6. 选择 Next（下一步）。
7. 输入密钥名称。例如：my-es-secret。您可以选择性地添加描述。

   记录本教程稍后使用的密钥名称，然后选择 Next（下一步）。
8. 再次选择 Next（下一步），然后选择 Store（存储）创建密钥。

后续步骤

步骤 2：订阅连接器

步骤 2：订阅连接器

AWS Marketplace 中免费提供 AWS Glue Connector for Elasticsearch。

在 AWS Marketplace 中订阅 AWS Glue Connector for Elasticsearch

1. 如果您尚未配置 AWS 账户使用 License Manager，请执行以下操作：

   a. 访问 https://console.aws.amazon.com/license-manager，打开 AWS License Manager 控制台。

   b. 选择 Create customer managed license (创建客户托管式许可证)。

   c. 在 IAM permissions (one-time setup) (IAM 权限 (一次性设置)) 窗口中，选择 I grant AWS License Manager the required permissions (我为 Amazon License Manager 授予所需的权限)，然后选择 Grant permissions (授予权限)。

      如果未看到此窗口，则表示您已配置所需的权限。

2. 请访问 https://console.aws.amazon.com/gluestudio/ 打开 AWS Glue Studio 控制台。
3. 在 AWS Glue Studio 控制台，展开菜单图标（☰），然后在导航窗格中选择 Connectors (连接器)。
4. 在 Connectors (连接器) 页面上，选择 Go to AWS Marketplace (转到 Amazon Web Services Marketplace)。
5. 在 AWS Marketplace 的 Search AWS Glue Studio products（搜索产品）部分中，在搜索字段中输入 AWS Glue Connector for Elasticsearch，然后按 Enter 键。
6. 选择连接器的名称，AWS Glue Connector for Elasticsearch。
7. 在连接器的产品页面上，使用选项卡查看有关连接器的信息。准备好继续使用后，选择 Continue to Subscribe (继续订阅)。
8. 查看软件的使用条款。单击 Accept Terms（接受条款）。
9. 订阅过程完成后，您将看到一条通知：“Thank you for subscribing to this product! You can now configure your software.”（感谢您订阅此产品！现在您可以配置软件。）横幅上方将是按钮 Continue to Configuration（继续配置）。选择继续配置。

10. 在 Configure this software（配置此软件）页面上，选择“Fulfillment”（执行）选项。您可以选择 AWS Glue 1.0/2.0 或 AWS Glue 3.0。然后选择 Continue to Launch（继续启动）。

后续步骤

步骤 3：激活 AWS Glue Studio 并创建连接

步骤 3：激活 AWS Glue Studio 并创建连接

在您选择 Continue to Launch (继续启动) 后，您可以在 AWS Marketplace 中看到 Launch this software (启动此软件) 页面。您在 AWS Glue Studio 中使用链接激活连接器后，创建连接。

要在 AWS Glue Studio 中部署连接器并创建连接

1. 在 AWS Marketplace 控制台的 Launch this software (启动此软件) 页面上，选择 Usage Instructions (使用说明)，然后选择在窗口中显示的链接。

   您的浏览器将重新定向到 AWS Glue Studio 控制台 Create marketplace connection (创建 Marketplace 连接) 页面。
2. 为连接输入名称。例如：my-es-connection。
3. 在 Connection access (连接访问) 部分，为 Connection credential type (连接凭证类型) 选择 User name and password (用户名和密码)。
4. 对于 AWS secret (Amazon 密钥)，输入您的密钥名称。例如：my-es-secret。
5. 在 Network options（网络选项）部分中，输入 VPC 信息，以连接到 OpenSearch 集群。
6. 选择 Create connection and activate connector (创建连接并激活连接器)。

后续步骤

步骤 4：为您的 ETL 任务配置 IAM 角色

步骤 4：为您的 ETL 任务配置 IAM 角色

在创建 AWS Glue ETL 任务时，您可以指定 AWS Identity and Access Management（IAM）角色以供任务使用。角色必须授予对任务使用的所有资源的访问权限，包括 Amazon S3（任何源、目标、脚本、驱动程序文件和临时目录）以及 AWS Glue Data Catalog 对象。

AWS Glue ETL 任务所担任的 IAM 角色还必须具有对在上一部分创建的密钥的访问权限。预设情况下，亚马逊云科技托管式角色 `AWSGlueServiceRole` 无法访问该密钥。要设置密钥的访问控制，请参阅 AWS Secrets Manager 的身份验证和访问控制以及限制对特定密钥的访问。

要为您的 ETL 任务配置 IAM 角色

1. 配置the section called “审核 ETL 任务所需的 IAM 权限”中所述的权限。
2. 配置如the section called “使用连接器所需的权限”中所述的使用 AWS Glue Studio 连接器时所需的其他权限。

后续步骤

步骤 5：创建使用 OpenSearch 连接的任务

步骤 5：创建使用 OpenSearch 连接的任务

为您的 ETL 任务创建角色后，您可以在 AWS Glue Studio 中创建一个任务，改任务使用 Open Spark ElasticSearch 的连接和连接器。

如果您的任务在 Amazon Virtual Private Cloud（Amazon VPC）中运行，请确保 VPC 配置正确。有关更多信息，请参阅 the section called “为 ETL 任务配置 VPC”。

要创建使用 Elasticsearch Spark Connector 的任务

1. 在 AWS Glue Studio 中，选择 Connectors（连接器）。
2. 在 Your connections (您的连接) 列表中，选定您刚才创建的连接并选择 Create job (创建任务)。
3. 在可视任务编辑器中，选择数据源节点。在右侧 Data source properties - Connector (数据源属性 – 连接器) 选项卡上，配置连接器的其他信息。

    a. 选择 Add schema (添加架构)，然后输入数据源中的数据集架构。连接不使用存储在数据目录中的表，这意味着 AWS Glue Studio 不了解数据架构。您必须手动提供此架构信息。有关如何使用架构编辑器的说明，请参阅the section called “编辑自定义转换节点的架构”。

    b. 展开 Connection options (连接选项)。

    c. 选择 Add new option (添加新选项)，然后输入尚未在 AWS 密钥中输入的连接器所需的信息：

    - es.nodes: https://<OpenSearch domain endpoint>
    - es.port: 443
    - path: test

- es.nodes.wan.only.: true

有关这些连接选项的说明，请参阅：https://www.elastic.co/guide/en/elasticsearch/hadoop/current/configuration.html。

4. 将目标节点添加至图表中。

   您的数据目标可以是 Amazon S3，也可以使用 AWS Glue Data Catalog 或连接器的信息将数据写入其他位置。例如，您可以使用数据目录表将数据库写入 Amazon RDS，也可以将连接器用作数据目标来写入 AWS Glue 中不支持的数据存储。

   如果为数据目标选择连接器，则您必须选择为该连接器创建的连接。此外，如果连接器提供程序需要，则您必须添加选项以向连接器提供其他信息。如果您使用包含 AWS 密钥信息的连接，则无需在连接选项中提供用户名和密码身份验证。

5. （可选）添加如the section called “编辑 AWS Glue 托管数据转换节点”中所述的其他数据源和一个或多个转换节点。
6. 从步骤 3 开始，配置如the section called “修改任务属性”中所述的任务属性，然后保存任务。

后续步骤

步骤 6：运行任务

步骤 6：运行任务

保存任务后，您可以运行任务来执行 ETL 操作。

运行您为 AWS Glue Connector for Elasticsearch 创建的任务

1. 使用 AWS Glue Studio 控制台，在可视编辑器页面上，选择 Run (运行)。
2. 在成功横幅中，选择 Run Details (运行详细信息)，也可以选择可视化编辑器的 Runs (运行) 选项卡查看有关任务运行的信息。

通过交互式会话构建 AWS Glue 作业

数据工程师可以使用 AWS Glue 中的交互式会话，从而比以前更快、更轻松地编写 AWS Glue 任务。

主题

AWS Glue 交互式会话概览

借助 AWS Glue 交互式会话，您可以快速构建、测试和运行数据准备和分析应用程序。交互式会话为数据准备提取、转换、加载（ETL）脚本提供了一个编程和可视化界面。交互式会话运行 Apache Spark 分析应用程序，并提供对远程 Spark 运行时环境的按需访问。AWS Glue 透明地管理这些交互式会话的无服务器 Spark。

由于交互式会话非常灵活，因此您可以在自己选择的环境中构建和测试应用程序。您可以通过 AWS Command Line Interface 和 API 创建与操作交互式会话。您可以使用 Jupyter 兼容的笔记本直观地编写和测试笔记本脚本。交互式会话提供了一个开源的 Jupyter 内核，几乎可以在 Jupyter 兼容的任何环境中集成，包括与 PyCharm、IntelliJ 和 VS Code 等 IDE 集成。这可让您在本地环境中编写代码并在交互式会话后端无缝运行它。

使用交互式会话 API，客户可以采用编程方式运行使用 Apache Spark 分析的应用程序，而无需管理 Spark 基础设施。您可以在单个交互式会话中运行一条或多条 Spark 语句。

因此，交互式会话提供了一种更快、更便宜、更灵活的方式来构建和运行数据准备与分析应用程序。要了解如何使用交互式会话，请参阅文档中的本节。AWS Glue 支持的魔术命令

限制

- 交互式会话中不支持任务书签。
- 不支持使用 AWS Command Line Interface 创建笔记本作业。

开始使用 AWS Glue 交互式会话

以下各节介绍如何在本地运行 AWS Glue 交互式会话。

在本地设置交互式会话的先决条件

以下是安装交互式会话的先决条件：

- 支持的 Python 版本为 3.6 - 3.10+。
- 有关 MacOS/Linux 和 Windows 的说明请参阅以下部分。

安装 Jupyter 和 AWS Glue 交互式会话 Jupyter 内核

使用以下命令在本地安装内核。

命令 `install-glue-kernels` 安装适用于 pyspark 和 spark 内核的 jupyter kernelspec，并在正确的目录中安装徽标。

```
pip3 install --upgrade jupyter boto3 aws-glue-sessions
```

```
install-glue-kernels
```

运行 Jupyter

要运行 Jupyter Notebook，请完成以下步骤。

1. 运行以下命令以启动 Jupyter Notebook。

   ```
   jupyter notebook
   ```

2. 选择 New（新建），然后选择其中一个 AWS Glue 内核以开始针对 AWS Glue 进行编码。

配置会话凭证和区域

MacOS/Linux 说明

AWS Glue 交互式会话需要的 IAM 权限与 AWS Glue 任务和开发端点需要的 IAM 权限相同。通过以下两种方式之一指定与交互式会话搭配使用的角色：

1. 使用 `%iam_role` 和 `%region` 魔术命令
2. 在 `~/.aws/config` 中使用其他行

使用魔术命令配置会话角色

在第一个单元格中，键入执行的第一个单元格中的 `%iam_role <YourGlueServiceRole>`。

使用 **`~/.aws/config`** 配置会话角色

AWS Glue交互式会话的服务角色可以在笔记本本身中指定，也可以与 AWS CLI 配置一起存储。如果您有一个通常用于 AWS Glue 任务的角色，则此角色将是该服务角色。如果您没有用于 AWS Glue 作业的角色，请按照本指南为 AWS Glue 配置 IAM 权限来设置一个角色。

要将此角色设置为交互式会话的默认角色：

1. 使用文本编辑器，打开 `~/.aws/config`。
2. 查找用于 AWS Glue 的配置文件。如果您未使用配置文件，请使用 `[Default]` 配置文件。
3. 在配置文件中为您计划使用的角色添加一行，例如 `glue_role_arn=<AWSGlueServiceRole>`。
4. [可选]：如果您的配置文件没有默认区域设置，我建议使用 `region=us-east-1` 添加一个，使用您所需的区域替换 `us-east-1`。
5. 保存配置。

有关更多信息，请参阅 使用 IAM 的交互式会话。

Windows 说明

AWS Glue 交互式会话需要的 IAM 权限与 AWS Glue 任务和开发端点需要的 IAM 权限相同。通过以下两种方式之一指定与交互式会话搭配使用的角色：

1. 使用 `%iam_role` 和 `%region` 魔术命令
2. 在 `~/.aws/config` 中使用其他行

使用魔术命令配置会话角色

在第一个单元格中，键入执行的第一个单元格中的 `%iam_role <YourGlueServiceRole>`。

使用 **`~/.aws/config`** 配置会话角色

AWS Glue交互式会话的服务角色可以在笔记本本身中指定，也可以与 AWS CLI 配置一起存储。如果您有一个通常用于 AWS Glue 任务的角色，则此角色将是该服务角色。如果您没有用于 AWS Glue 任务的角色，请按照本指南为 AWS Glue 设置 IAM 权限来设置一个角色。

要将此角色设置为交互式会话的默认角色：

1. 使用文本编辑器，打开 `~/.aws/config`。
2. 查找用于 AWS Glue 的配置文件。如果您未使用配置文件，请使用 `[Default]` 配置文件。
3. 在配置文件中为您计划使用的角色添加一行，例如 `glue_role_arn=<AWSGlueServiceRole>`。
4. [可选]：如果您的配置文件没有默认区域设置，我建议使用 `region=us-east-1` 添加一个，使用您所需的区域替换 `us-east-1`。
5. 保存配置。

有关更多信息，请参阅 使用 IAM 的交互式会话。

从交互式会话预览版升级

在与版本 0.27 一起发布时，内核已使用新名称进行了升级。要清理内核的预览版本，请从终端运行以下命令或 PowerShell。

> **Note**
>
> 如果您使用需要自定义服务模型的任何其他 AWS Glue 预览版本，则删除内核将会删除自定义服务模型。

```
# 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/
```

与 SageMaker Studio 搭配使用交互式会话

AWS Glue 交互式会话是一个按需的、无服务器的 Apache Spark 运行时系统环境，数据科学家和工程师可以使用它来快速构建、测试和运行数据准备和分析应用程序。您可以通过启动 Amazon SageMaker Studio Classic 笔记本实例来启动 AWS Glue 交互式会话。

有关更多信息，请参阅使用 AWS Glue 交互式会话准备数据。

将交互式会话与 Microsoft Visual Studio Code 配合使用

先决条件

- 安装 AWS Glue 交互式会话并验证其可与 Jupyter Notebook 搭配使用。
- 通过 Jupyter 下载和安装 Visual Studio Code。有关详细信息，请参阅采用 VS 代码的 Jupyter Notebook。

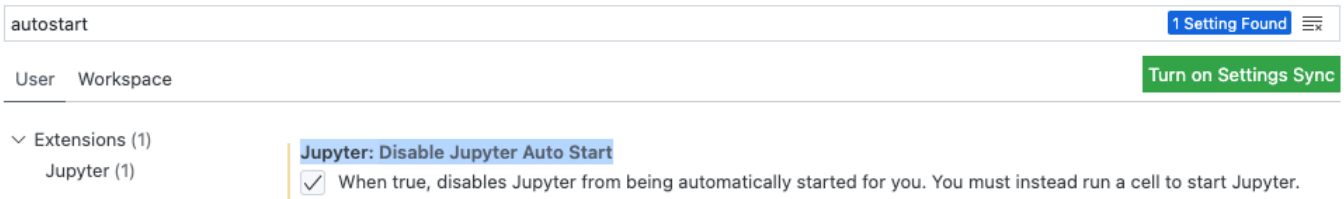
开始使用 VS Code 进行交互式会话

1. 在 VS Code 中禁用 Jupyter AutoStart。

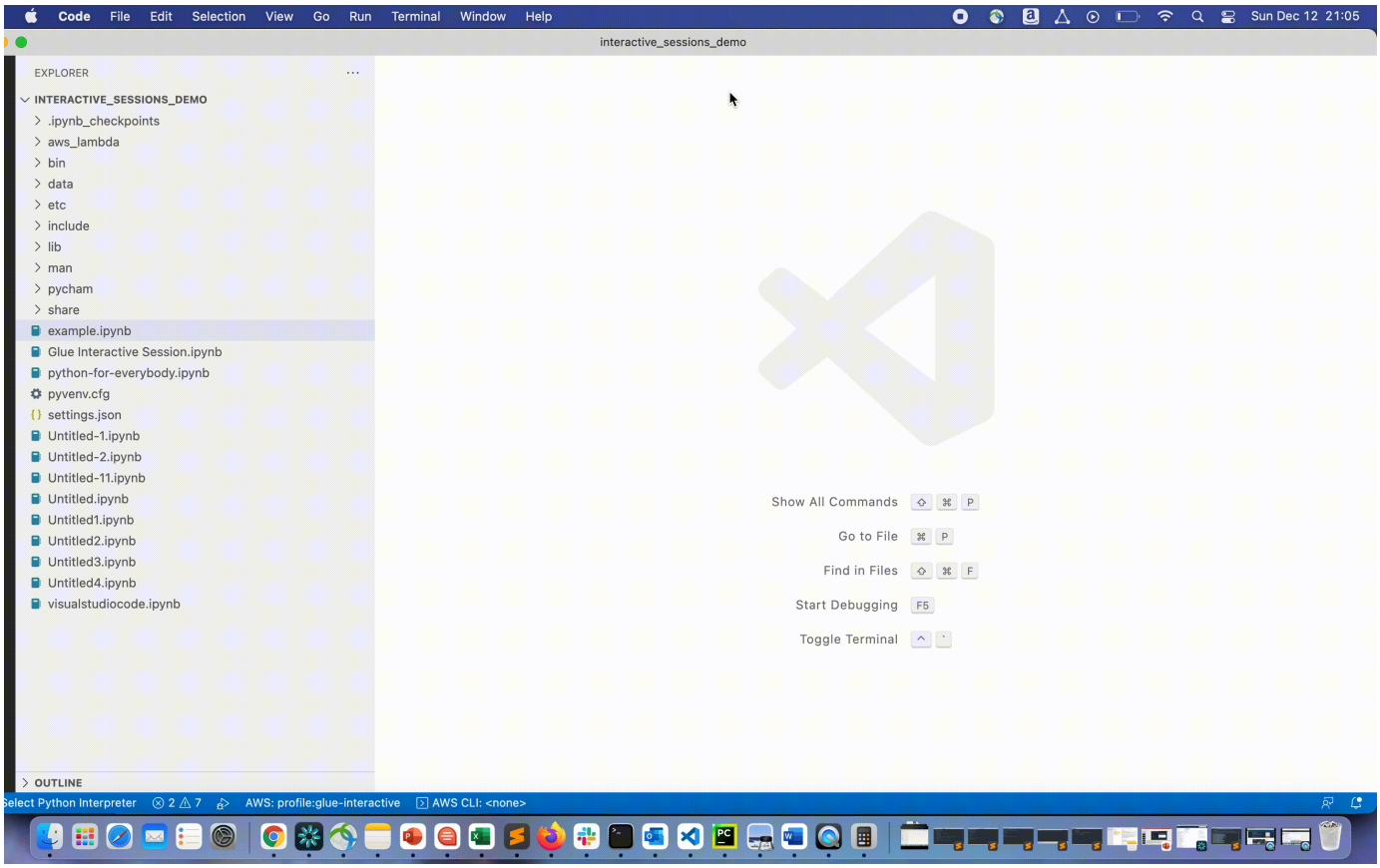
   在 Visual Studio Code 中，Jupyter 内核将自动启动，这将防止您的魔术命令因会话已开启而生效。要在 Windows 上禁用自动启动，请前往文件 > 首选项 > 扩展 > Jupyter > 右键单击 Jupyter，然后选择扩展设置。

   在 MacOS 上，请前往代码 > 设置 > 扩展 > Jupyter > 右键单击 Jupyter，然后选择扩展设置。

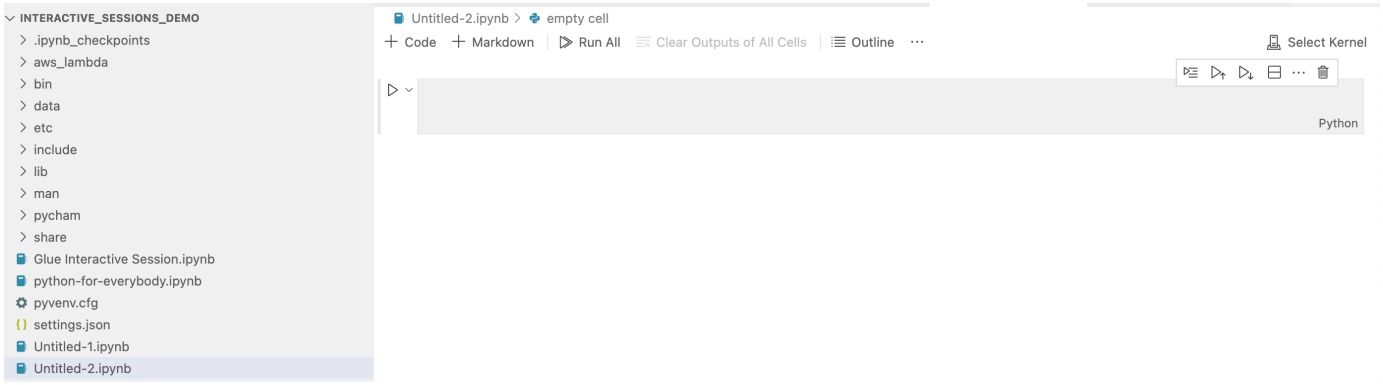
   向下滚动直到看到 Jupyter：禁用 Jupyter 自动启动。选中标有“如果为 true，则禁止为您自动启动 Jupyter”。You must instead run a cell to start Jupyter（您必须运行一个单元格才能开启 Jupyter）。”



2. 转到 File（文件）> New File（新建文件）> Save（保存）以使用您选择的名称将此文件保存为 `.ipynb` 扩展，或者在 select a language（选择语言）下选择 jupyter 并保存文件。



3. 双击文件。系统将显示 Jupyter Shell 并打开笔记本。



4. 在 Windows 上，您首次创建文件时，默认情况下，其未选择内核。单击 Select Kernel（选择内核），此时将显示可用内核的列表。选择 Glue PySpark。

   在 MacOS 上，如果您看不到 Glue PySpark 内核，请尝试以下步骤：

   1. 运行本地 Jupyter 会话以获取 URL。

例如，运行以下命令以启动 Jupyter Notebook。

```
jupyter notebook
```

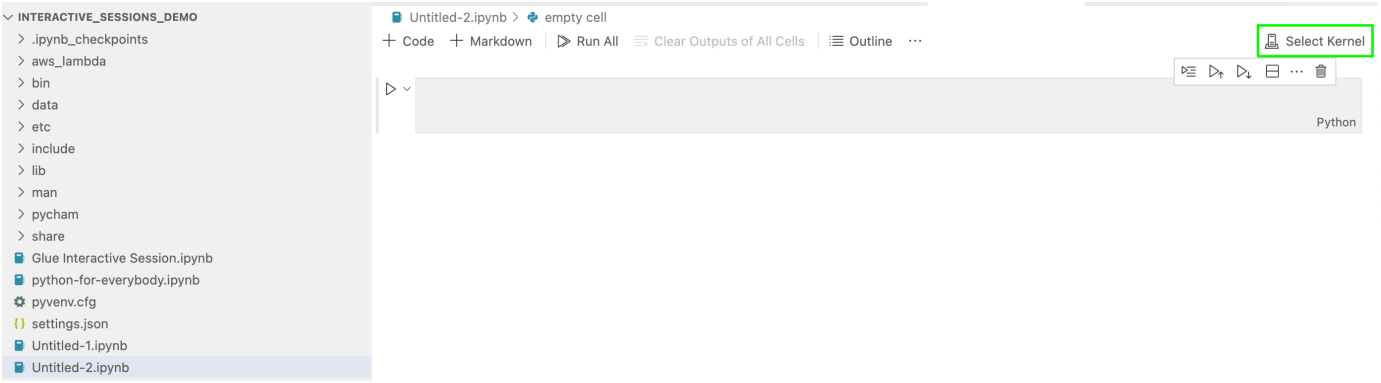
当笔记本首次运行时，您会看到一个如 `http://localhost:8888/?token=3398XXXXXXXXXXXXXXX` 的 URL。

复制 URL。

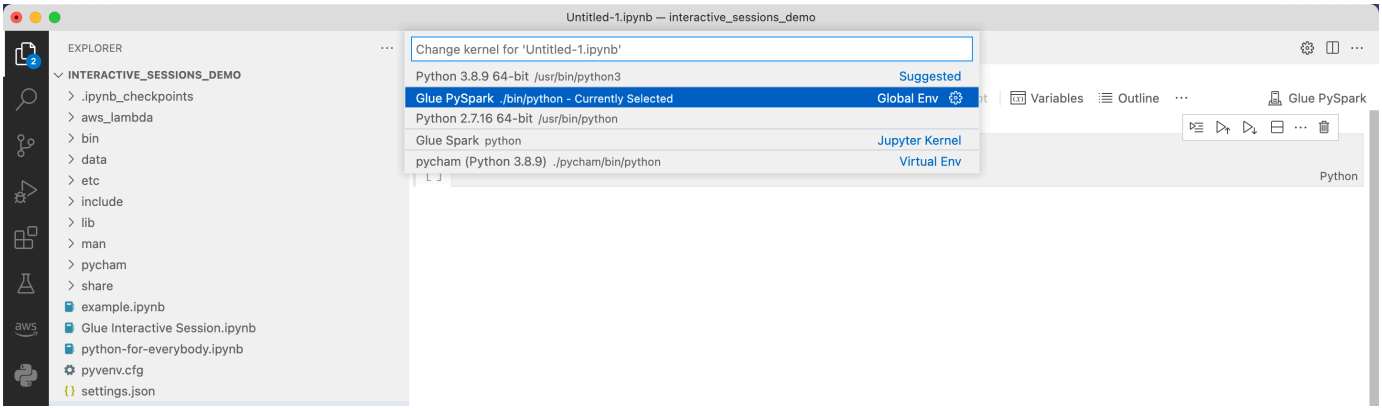
2. 在 VS Code 中，单击当前内核，然后选择其他内核...，然后选择现有的 Jupyter 服务器...。粘贴从上述步骤中复制的 URL。

   如果收到错误消息，请查看 VS Code Jupyter Wiki。

3. 如果成功，这会将内核设置为 Glue PySpark。



选择 Glue PySpark 或 Glue Spark 内核（分别适用于 Python 和 Scala）。



如果您在下拉列表中没有看到 AWS Glue PySpark 和 AWS Glue Spark 内核，请确保您已在上述步骤安装 AWS Glue 内核，或者检查 Visual Studio Code 中的

python.defaultInterpreterPath 设置正确无误。有关更多信息，请参阅 python.defaultInterpreterPath 设置说明。

5. 创建 AWS Glue 交互式会话。继续操作，以您在 Jupyter notebook 中相同的方式创建会话。在第一个单元格的顶部指定任意魔术，并运行代码语句。

为 Jupyter 和 AWS Glue Studio 笔记本配置 AWS Glue 交互式会话

Jupyter 魔术命令简介

Jupyter 魔术命令是可以在单元格开头或作为整个单元格正文运行的命令。魔术命令从用于行魔术命令的 % 和用于单元格魔术命令的 %% 开启。行魔术命令（例如 %region 和 %connections）可以使用单元格中的多个魔术命令运行，也可以使用单元格正文中包括的代码运行，如下例所示。

```
%region us-east-2
%connections my_rds_connection
dy_f = glue_context.create_dynamic_frame.from_catalog(database='rds_tables',
 table_name='sales_table')
```

单元格魔术命令必须使用整个单元格，并且可以使命令跨越多行。%%sql 的例子如下所示。

```
%%sql
select * from rds_tables.sales_table
```

Jupyter 的 AWS Glue 交互式会话支持魔术命令

以下是您可以与 Jupyter notebook 的 AWS Glue 交互式会话搭配使用的魔术命令。

会话魔术命令

名称	Type	描述
%help	不适用	返回所有魔术命令的描述和输入类型列表。
%profile	String	在 AWS 配置中指定要用作凭据提供者的配置文件。

名称	Type	描述
%region	String	指定 AWS 区域; 在其中初始化会话。~/.aws/configure. 的默认值 例如：%region us-west-1
%idle_timeout	Int	执行单元格后，非活动的分钟数，在此之后会话将超时。Spark ETL 会话的默认空闲超时值是默认超时，2880 分钟（48小时）。对于其他会话类型，请查阅该会话类型的文档。 例如：%idle_timeout 3000
%session_id	不适用	返回正在运行的会话的会话 ID。
%session_id_prefix	String	以格式 [session_id_prefix]-[session_id] 定义在所有会话 ID 之前的字符串。如果未提供会话 ID，则将生成随机 UUID。当您在 AWS Glue Studio中运行 Jupyter Notebook 时，不支持此魔术命令。 例如：%session_id_prefix 001
%status		返回当前 AWS Glue 会话的状态，包括其持续时间、配置和执行用户/角色。
%stop_session		停止当前会话。
%list_sessions		按名称和 ID 列出当前正在运行的所有会话。
%session_type	String	将会话类型设置为流媒体、ETL 或 Ray 中的一种。 例如：%session_type Streaming

名称	Type	描述
%glue_version	String	此会话使用的 AWS Glue 版本。 例如：%glue_version 3.0

选择作业类型的魔术命令

名称	Type	描述
%streaming	String	将会话类型更改为 AWS Glue 串流。
%etl	String	将会话类型更改为 AWS Glue ETL。
%glue_ray	String	将 Ray 的会话类型更改 AWS Glue 为。参见 AWS Glue Ray 互动会话支持的魔法。

AWS Glue for Spark 配置魔术命令

%%configure 魔术命令指定一个 json 格式的词典，其中包含会话的所有配置参数。每个参数可以在此处指定，也可以通过单独的魔术命令指定。

名称	Type	描述
%%configure	词典	指定一个 JSON 格式的词典，其中包含会话的所有配置参数。每个参数可以在此处指定，也可以通过单独的魔术命令指定。 有关参数列表和如何使用 %%configure 的示例，请参阅下表：Using %%configure。
%iam_role	String	指定用于执行会话的 IAM 角色 ARN。~/.aws/configure 的默认值。 例如：%iam_role AWSGlueServiceRole

名称	Type	描述
%number_of_workers	Int	任务运行时分配的定义 worker_type 的工件数量。还必须设置 worker_type 。默认 number_of_workers 为 5。 例如：%number_of_workers 2
%additional_python_modules	列出	要包含在集群中的其他 Python 模块的逗号分隔列表（可以来自 PyPI 或 S3）。 示例：%additional_python_modules pandas, numpy 。

名称	Type	描述

%%tags	String	为会话添加标签。在大括号 {} 内指定标签。每个标签名称对都用圆括号 (" ") 括起来，并用逗号 (,) 分隔。
		
		`%%tags`
		`{"billing":"Data-Platform",`
		`"team":"analytics"}`
		
		使用 %status 魔术命令查看与会话相关的标签。
		
		`%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']`

名称	Type	描述

%%assume_role	字典	指定 json 格式的字典或 IAM 角色 ARN 字符串来创建用于跨账户访问的会话。
		
		ARN 示例：
		
		`%%assume_role`
		`{`
		`  'arn:aws:iam::XXXXXXXXXXXX:`
		`role/AWSGlueServiceRole'`
		`}`
		
		凭证示例：
		
		`%%assume_role`
		`{{`
		`  "aws_access_key_id" =`
		`"XXXXXXXXXXXX",`
		`  "aws_secret_access_key" =`
		`"XXXXXXXXXXXX",`
		`  "aws_session_token" =`
		`"XXXXXXXXXXXX"`
		`}}`

%%configure 单元格魔术命令参数

%%configure 魔术命令指定一个 json 格式的词典，其中包含会话的所有配置参数。每个参数可以在此处指定，也可以通过单独的魔术命令指定。有关 %%configure 单元格魔术命令支持的参数的示例，请参见下文。使用为作业指定的运行参数前缀 --。例如：

```
%%configure
{
   "--user-jars-first": "true",
   "--enable-glue-datacatalog": "false"
}
```

有关作业参数的更多信息，请参阅 任务参数。

会话配置

参数	类型	描述

max_retries	Int	在该作业失败时重试的最大次数。
		`%%configure`
		`{`
		`  "max_retries": "0"`
		`}`

max_concurrent_runs	Int	作业允许的最大并发运行数。
		例如：
		`%%configure`
		`{`
		`  "max_concurrent_runs": "3"`
		`}`

会话参数

参数	类型	描述

--enable-spark-ui	布尔值	启用 Spark 用户界面来监控和调试 AWS Glue ETL 作业。
		`%%configure`
		`{`
		`  "--enable-spark-ui": "true"`
		`}`

--spark-event-logs-path	String	指定 Amazon S3 路径。使用 Spark 用户界面监控功能时。 例如：

参数	类型	描述

		%%configure
		{
		"--spark-event-logs-path":
		"s3://path/to/event/logs/"
		}

--script_location	String	指定执行作业的脚本的 S3 路径。
		
		例如：
		
		%%configure
		{
		"script_location": "s3://new-folder-here"
		}

--SECURITY_CONFIGURATION	String	AWS Glue 安全配置的名称
		
		例如：
		
		%%configure
		{
		"--SECURITY_CONFIGURATION": *security-configuration-name* ,
		}

参数	类型	描述

`--job-language`	String	脚本编程语言。接受“scala”或“python”的值。默认值为“python”。
		
		例如：
		
		`%%configure`
		`{`
		`  "--job-language": "scala"`
		`}`

`--class`	String	用作 Scala 脚本之入口点的 Scala 类。默认值为 null。
		
		例如：
		
		`%%configure`
		`{`
		`  "--class": "className"`
		`}`

`--user-jars-first`	布尔值	优先考虑类路径中的客户额外 JAR 文件。默认值为 null。
		
		例如：
		
		`%%configure`
		`{`
		`  "--user-jars-first": "true"`
		`}`

参数	类型	描述

--use-postgres-driver	布尔值	在类路径中设置 Postgres JDBC 驱动程序的优先级，以避免与 JDBC 驱动程序发生冲突。 Amazon Redshift 默认值为 null。
		
		例如：
		
		`%%configure`
		`{`
		`  "--use-postgres-driver": "true"`
		`}`

--extra-files	List(string)	在执行脚本之前，AWS Glue 复制到脚本工作目录中的其他文件（如配置文件）的 Amazon S3 路径。
		
		例如：
		
		`%%configure`
		`{`
		`  "--extra-files": "s3://path/to/additional/files/"`
		`}`

参数	类型	描述

--job-bookmark-option	String	控制作业书签的行为。接受 "、job-bookmark-enable '或job-bookmark-disable' '的值。job-bookmark-pause默认为 “job-bookmark-disable”。
		
		例如：
		
		`%%configure`
		`{`
		`  "--job-bookmark-option": "job-bookmark-enable"`
		`}`

--TempDir	String	指定可用作任务的临时目录的存储桶的 Amazon S3 路径。默认值为 null。
		
		例如：
		
		`%%configure`
		`{`
		`  "--TempDir": "s3://path/to/temp/dir"`
		`}`

参数	类型	描述

--enable-s3-parquet-optimized-committer	布尔值	启用经 EMRFS Amazon S3 优化的提交程序，用于将 Parquet 数据写入 Amazon S3。默认值为“true”。
		
		例如：
		
		`%%configure`
		`{`
		`  "--enable-s3-parquet-optimized-committer": "false"`
		`}`

--enable-rename-algorithm-v2	布尔值	将 EMRFS 重命名算法版本设置为版本 2。默认值为“true”。
		
		例如：
		
		`%%configure`
		`{`
		`  "--enable-rename-algorithm-v2": "true"`
		`}`

--enable-glue-datacatalog	布尔值	支持您将 AWS Glue Data Catalog 用作 Apache Spark Hive 元存储。
		
		例如：
		
		`%%configure`
		`{`
		`  --"enable-glue-datacatalog": "true"`
		`}`

参数	类型	描述

--enable-metrics	布尔值	为作业运行启用作业分析指标的集合。默认值为“false”。
		
		例如：
		
		`%%configure`
		`{`
		`  "--enable-metrics": "true"`
		`}`

--enable-continuous-cloudwatch-log	布尔值	允许对 AWS Glue 作业进行实时的连续日志记录。默认值为“false”。
		
		例如：
		
		`%%configure`
		`{`
		`  "--enable-continuous-cloudw`
		`atch-log": "true"`
		`}`

--enable-continuous-log-filter	布尔值	在创建或编辑为连续日志记录启用的作业时，指定标准筛选器或无筛选器。默认值为“true”。
		
		例如：
		
		`%%configure`
		`{`
		`  "--enable-continuous-log-fi`
		`lter": "true"`
		`}`

参数	类型	描述

`--continuous-log-stream-prefix`	String	为启用持续 Amazon CloudWatch 日志记录的作业指定自定义日志流前缀。默认值为 null。
		例如：
		`%%configure`
		`{`
		`  "--continuous-log-stream-prefix": "prefix"`
		`}`

`--continuous-log-conversionPattern`	String	为已启用连续日志记录的作业指定自定义转换日志模式。默认值为 null。
		例如：
		`%%configure`
		`{`
		`  "--continuous-log-conversionPattern": "pattern"`
		`}`

参数	类型	描述

--conf	String	控制 Spark 配置参数。适用于高级使用案例。在每个参数之前使用 --conf。例如：
		
		%%configure
		{
		"--conf": "spark.hadoop.hive.metastore.glue.catalogid=123456789012 --conf hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory --conf hive.metastore.schema.verification=false"
		}

Spark 作业（ETL 和流传输）魔术命令

名称	Type	描述
%worker_type	String	标准，G.1X 或 G.2X。还必须设置 number_of_workers 。worker_type 默认为 G.1X。
%connections	列出	指定要在会话中使用的连接的逗号分隔列表。 例如： %connections my_rds_connection dy_f = glue_context.create_dynamic_frame.from_catalog(database='rds_tables', table_name='sales_table')

名称	Type	描述
%extra_py_files	列出	Simple Storage Service (Amazon S3) 中其他 Python 文件的逗号分隔列表。
%extra_jars	列出	要包含在集群中的其他 Jar 的逗号分隔列表。
%spark_conf	String	为您的会话指定自定义 Spark 配置。例如，%spark_conf spark.serializer=org.apache.spark.serializer.KryoSerializer 。

适用于 Ray 作业的魔术命令

名称	Type	描述
%min_workers	Int	分配给 Ray 作业的工作线程的最小数量。默认值：1。 例如：%min_workers 2
%object_memory_head	Int	热启动后实例头节点上可用内存的百分比。最小值：0。最大值：100。 例如：%object_memory_head 100
%object_memory_worker	Int	热启动后实例 Worker 节点上可用内存的百分比。最小值：0。最大值：100。 例如：%object_memory_worker 100

操作魔术命令

名称	Type	描述
`%%sql`	String	运行 SQL 代码。首次 `%%sql` 魔术命令之后的所有行将作为 SQL 代码的一部分传递。 例如：`%%sql select * from rds_tables.sales_table`

`%matplot`	Matplotlib figure	使用 matplotlib 库可视化您的数据。
		
		例如：
		
		`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 figure	使用 plotly 库可视化您的数据。
		
		例如：
		
		`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`

命名会话

AWS Glue交互式会话是 AWS 资源，需要一个名称。每个会话的名称应是唯一的，并且可能会受到您的 IAM 管理员的限制。有关更多信息，请参阅 使用 IAM 的交互式会话。Jupyter 内核会自动为您生成唯一的会话名称。但是，可以通过两种方式手动命名会话：

1. 使用位于的 AWS Command Line Interface 配置文件`~.aws/config`。请参阅使用设置 AWS Config AWS Command Line Interface。
2. 使用 `%session_id_prefix` 魔术命令。请参阅 Jupyter 的 AWS Glue 交互式会话支持魔术命令。

会话名称的生成方式如下：

- 提供前缀和 session_id 时：会话名称将为 {前缀}-{UUID}。
- 未提供任何内容时：会话名称将为 {UUID}。

使用前缀会话名称可以让您在 AWS CLI 或控制台中列出会话时识别该会话。

为交互式会话指定 IAM 角色

您必须指定一个 AWS 身份和访问管理 (IAM) 角色才能与在交互式会话中运行的 ETL 代码AWS Glue一起使用。

该角色所需的IAM 权限与运行 AWS Glue 任务所需的 IAM 权限相同。有关为 AWS Glue 任务和交互式会话创建角色的更多信息，请参阅为 AWS Glue 创建 IAM 角色。

可通过以下两种方式指定 IAM 角色：

- 使用位于`~.aws/config`（推荐）的 AWS Command Line Interface 配置文件。更多相关信息，请参阅使用 ~/.aws/config 配置会话。

  > **Note**
  >
  > 使用 `%profile` 魔术命令时，将优先采用该配置文件的 `glue_iam_role` 配置。

- 使用 %iam_role 魔术命令。有关更多信息，请参阅 Jupyter 的 AWS Glue 交互式会话支持魔术命令。

使用命名配置文件配置会话

AWS Glue交互式会话使用与 AWS Command Line Interface 或 boto3 相同的凭据，交互式会话支持并使用命名的配置文件，例如~/.aws/config（Linux 和 macOS）或%USERPROFILE%\.aws\config（Windows） AWS CLI 中的配置文件。有关更多信息，请参阅 Using named profiles。

交互式会话允许在配置文件中指定 AWS Glue 服务角色和会话 ID 前缀，从而利用命名配置文件。要对配置文件角色进行配置，请为命名配置文件的 iam_role 密钥和/或 session_id_prefix 添加一行，如下所示。session_id_prefix 不需要引号。例如，如果要添加 session_id_prefix，请输入 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>
```

如果您有生成凭证的自定义方法，还可以将配置文件配置为使用 credential_process 文件中的 ~/.aws/config 参数。例如：

```
[profile developer]
region=us-east-1
credential_process = "/Users/Dave/generate_my_credentials.sh" --username helen
```

您可以找到有关通过 credential_process 参数获取凭证的更多详情，请参阅此处：使用外部进程获取凭证。

如果在使用的配置文件中未设置区域或 iam_role，则必须在运行的第一个单元格中使用 %region 和 %iam_role 魔术命令进行指定。

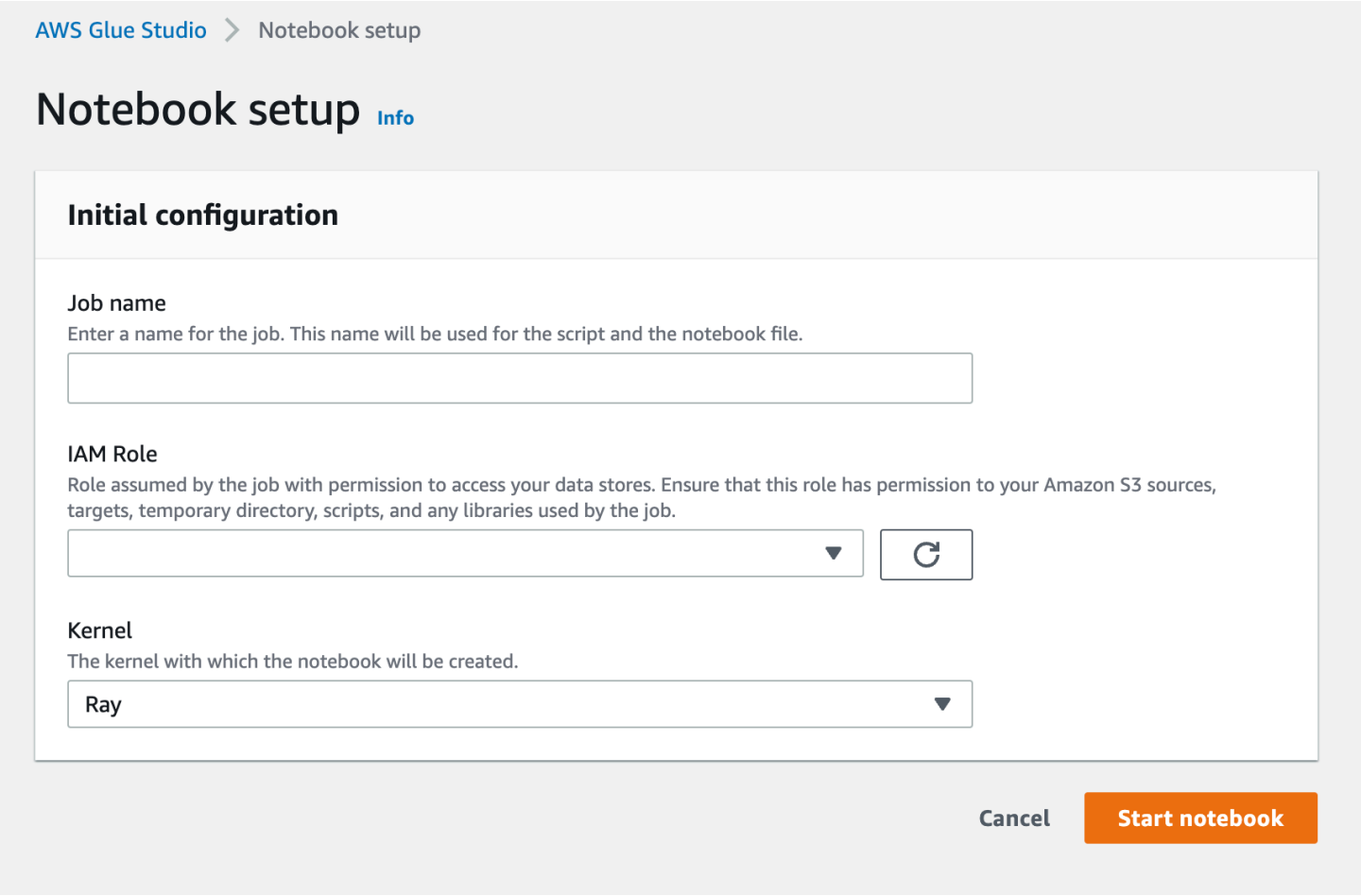
For Ray 互动会话入门（预览版） AWS Glue

> **Warning**
>
> for Ray 互动会话 AWS Glue 的预览已于 2024 年 4 月 30 日结束。您将无法再为 Ray 创建新的交互式会话。 AWS Glue

> **Note**
>
> AWS Glue for Ray 在美国东部（弗吉尼亚北部）、美国东部（俄亥俄州）、美国西部（俄勒冈）、亚太地区（东京）和欧洲（爱尔兰）上市。

AWS Glue Studio 控制台中的 Ray 交互式会话

在 AWS Glue Studio 控制台的 “作业” 页面中，选择现有的 Jupyter 笔记本选项。这将打开 Notebook setup（Notebook 设置）页面，您可以在其中选择 Kernel（内核）。选择 Ray 内核开始 Ray 交互式会话。有关交互式会话及其使用方法的详细信息，请参阅the section called “开始使用 AWS Glue 交互式会话”。



使用 Jupyter 内核的 Ray 交互式会话

要在 AWS Glue Studio 控制台之外使用 Ray 内核，你需要安装我们在 PyPI 上发布的`aws-glue-sessions`软件包。有关使用内核包的更多信息，请参阅 the section called “开始使用 AWS Glue 交互式会话” 文档。

要更新或安装内核，请运行 `pip install --upgrade aws-glue-sessions`。需要 0.37 以上版本才能使用 Ray 内核。

Ray 交互式会话可以访问与 Ray 作业相同的 Ray 库和版本。在预览版中，所有 Ray 互动会话都将使用 Ray 2.4.0。

Ray 交互会话超时默认值

- （会话的）超时默认为：8 小时。
- 默认空闲超时为 1 小时。

AWS Glue Ray 交互式会话支持魔术命令

AWS Glue Jupyter 内核为 Ray 交互式会话提供动力的魔法与 Spark 会话的魔法类似。有关参考，请参阅 the section called “ 为 Jupyter 和 AWS Glue Studio 笔记本配置 AWS Glue 交互式会话 ”。

会话魔术命令

会话魔术命令与 AWS Glue for Ray 预览版之前的魔术命令基本相同。获取有关此预览之外的魔术命令的更多信息，请参阅 the section called “Jupyter 的 AWS Glue 交互式会话支持魔术命令”。我们推出了一种新的魔术命令，可以将会话类型设置为 AWS Glue for Ray。

名称	Type	描述
`%glue_ray`	字符串	将会话类型更改为 AWS Glue for Ray。

AWS Glue config 魔术命令

在交互式会话中配置 AWS Glue 的魔法可能因会话类型而异。目前，我们仅在使用 AWS Glue for Ray 时支持现有魔术命令的这个子集。

> **Warning**
>
> 已知问题：**`additional_python_modules`**
> 在 Ray 交互式会话预览版中，保存笔记本时使用 `additional_python_modules` 魔术命令会导致出现问题。要为 Ray 会话配置 python 模块，请使用 `%%configure` 魔术命令设置 the section called “Ray 作业参数” 中定义的 `pip-install` 参数。

名称	Type	描述
`%%configure`	词典	指定一个 JSON 格式的词典，其中包含会话的所有配置参数。每个参数可以在此处指定，也可以通过单独的魔术命令指定。
`%iam_role`	字符串	指定用于执行会话的 IAM 角色 ARN。~/.aws/configure 的默认值

名称	Type	描述
%number_of_workers	int	任务运行时分配的定义 worker_type 的工件数量。还必须设置 worker_type 。
%worker_type	字符串	在 AWS Glue for Ray 预览版中，唯一支持的工作线程类型是 Z.2X。
%additional_python_modules	列出	要包含在集群中的其他 Python 模块的逗号分隔列表（可以来自 Pypi 或 S3）。

操作魔术命令

AWS Glue Ray 会话不支持任何动作魔法。

使用 IAM 的交互式会话

以下各节介绍 AWS Glue 交互式会话的安全注意事项。

主题

- 交互式会话中使用的 IAM 主体
- 设置客户端主体
- 设置运行时角色
- 将您的会话设为私密会话 TagOnCreate
- IAM policy 注意事项

交互式会话中使用的 IAM 主体

与 AWS Glue 交互式会话配合使用的两个 IAM 主体

- 客户端主体：客户端主体（用户或角色）授权从使用主体的身份式凭证配置的 AWS Glue 客户端执行交互式会话的 API 操作。例如，这可能是通常用于访问 AWS Glue 控制台的 IAM 角色。这也可以是授予使用 IAM 中证书的用户的角色 AWS Command Line Interface，或者交互式会话 Jupyter 内核使用的AWS Glue客户端。
- 运行时角色：运行时角色是客户端主体传递给交互式会话 API 操作的 IAM 角色。AWS Glue 使用此角色在会话中运行语句。例如，此角色可能用于运行 AWS Glue ETL 任务。

有关更多信息，请参阅 设置运行时角色。

设置客户端主体

您必须将身份策略附加到客户端主体，才能调用交互式会话 API。这个角色必须具备执行角色的 `iam:PassRole` 访问权限，您将此角色传递给交互式会话 API（例如 `CreateSession`）。例如，您可以将AWSGlueConsoleFullAccess托管策略附加到一个 IAM 角色，该角色允许您的账户中附加了该策略的用户访问在您的账户中创建的所有会话（例如运行时语句或取消声明）。

如果您想保护您的会话并使其仅对某些 IAM 角色设为私有，例如与创建会话的用户关联的角色，则可以使用名为 Interact AWS Glue ive Session 的基于标签的授权控制 TagOnCreate。有关更多信息，请参阅 将您的会话设为私密会话 TagOnCreate 基于所有者标签的限定范围的托管策略如何将您的会话设为私有。 TagOnCreate有关基于身份的策 的更多信息，请参阅适用于 AWS Glue 的基于身份的策略。

设置运行时角色

您必须将 IAM 角色传递给 CreateSession API 操作AWS Glue才能允许在交互式会话中代入和运行语句。该角色应拥有与运行典型 AWS Glue 任务所需相同的 IAM 权限。例如，您可以使用允许代表您呼叫 AWS 服务的AWSGlueServiceRole策略AWS Glue来创建服务角色。如果您使用 AWS Glue 控制台，它将自动代表您创建服务角色或使用现有的服务角色。此外，您还可以创建自己的 IAM 角色并附加自己的 IAM policy，以授予相似的权限。

如果您想保护您的会话并使其仅对创建会话的用户保密，则可以使用名为 Interact AWS Glue ive Session 的基于标签的授权控件 TagOnCreate。有关更多信息，请参阅 将您的会话设为私密会话 TagOnCreate 基于所有者标签的限定范围的托管策略如何将您的会话设为私有。 TagOnCreate有关基于身份的策略的更多信息，请参阅 Glue 基于身份的策略 AWS。如果您通过 IAM 控制台自己创建执行角色，并且想要使用 TagOnCreate 功能将服务设为私有，请按照以下步骤操作。

1. 创建IAM 角色并将角色类型设置为 `Glue`。
2. 附加此AWS Glue托管策略：AwsGlueSessionUserRestrictedServiceRole
3. 在角色名称前加上策略名称AwsGlueSessionUserRestrictedServiceRole。例如，您可以创建一个名为 AwsGlueSessionUserRestrictedServiceRole-myrole的角色并附加AWS Glue托管策略。AwsGlueSessionUserRestrictedServiceRole
4. 附加如下所示的信任策略，以允许 AWS Glue 代入角色：

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

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

对于交互式会话 Jupyter 内核，您可以在配置文件中指定`iam_role`密钥。 AWS Command Line Interface 更多相关信息，请参阅使用 ~/.aws/config 配置会话。如果您正在使用 AWS Glue 笔记本与交互式会话进行交互，则可以传递运行的第一个单元格中的 `%iam_role` 魔术命令中的执行角色。

将您的会话设为私密会话 TagOnCreate

AWS Glue 交互式会话 (IS) 支持将交互式会话的标记和基于标签的授权 (TBAC) 作为命名资源。除了 TBAC 使用 TagResource 和 UntagResource API 之外，AWS Glue交互式会话还支持仅在使用操作创建会话期间使用给定标签 “标记” 会话的 TagOnCreate CreateSession 功能。这也意味着这些标签将被删除 DeleteSession，也就是说 UntagOnDelete。

TagOnCreate 提供了一种强大的安全机制，可将您的会话设为私密会话，仅供会话创建者使用。例如，您可以将一个 “owner” RequestTag 且值为 $ {aws: userId} 的 IAM 策略附加到客户委托人（例如用户），以便仅在请求中提供与来电者 userID 值匹配的 “所有者” 标签作为用户 ID 标签时才允许创建会话。 CreateSession 此策略允许 AWS Glue 交互式会话仅在会话创建期间创建会话资源并使用 userId 标签标记会话。除此之外，您还可以通过将带有 “owner” ResourceTag 的 IAM 策略附加到您在此期间传入的执行角色来缩小对会话的访问权限（如正在运行的语句），仅限于会话的创建者（又名值为 $ {aws: userId} 的所有者标签）。 CreateSession

为了便于您使用 TagOnCreate 功能将会话设为私有会话创建者，AWS Glue提供了专门的托管策略和服务角色。

如果您想使用 IAM AssumeRole 委托人（即使用通过担任 IAM 角色获得的凭证）创建AWS Glue交互式会话，并且想要将该会话设为对创建者私有，请分别使用与AWSGlueSessionUserRestrictedNotebookPolicy和AWSGlueSessionUserRestrictedNotebookServiceRole关

似的策略。这些策略AWS Glue允许使用 $ {aws:PrincipalTag} 提取所有者标签值。这要求您传递一个值为 $ {aws: userID} 的 userID 标签，如代入角色凭证 SessionTag 中所示。请参阅 ID 会话标签。如果您使用的是带有实例配置文件出售证书的 Amazon EC2 实例，并且您想在 Amazon EC2 实例中创建会话或与会话进行交互，则需要传递一个值为 $ {aws: userID} 的用户 ID 标签， SessionTag 如代入角色凭证。

例如，如果您正在使用 IAM AssumeRole 委托人证书创建会话，并且想要使用 TagOnCreate 功能将您的服务设为私有，请按照以下步骤操作。

1. 从 IAM 控制台自行创建运行时角色。请附上此AWS Glue托管策略，AwsGlueSessionUserRestrictedNotebookServiceRole并在角色名称前加上策略名称AwsGlueSessionUserRestrictedNotebookServiceRole。例如，您可以创建一个名为AwsGlueSessionUserRestrictedNotebookServiceRole-myrole的角色并附加AWS Glue托管策略。AwsGlueSessionUserRestrictedNotebookServiceRole
2. 附加如下所示的信任策略，以允许 AWS Glue 代入以上角色:

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

3. 创建另一个带有前缀的角色AwsGlueSessionUserRestrictedNotebookPolicy并附加AWS Glue托管策略AwsGlueSessionUserRestrictedNotebookPolicy以将会话设为私有。除了托管策略外，请将以下内联策略附加到您在步骤 1 中创建的角色PassRole 以允许 iam:。

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

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

4. 将如下所示的信任策略附加到以上 IAM AWS Glue 以代入角色。

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

> **Note**
>
> 或者，您可以使用单个角色（例如，笔记本角色）并附加上述两个托管策略AwsGlueSessionUserRestrictedNotebookServiceRole和AwsGlueSessionUserRestrictedNotebook
> 此外，将允许角色的 `iam:passrole` 的其他内联策略附加到 AWS Glue。最后，请附加以上信任策略以允许 `sts:AssumeRole` 和 `sts:TagSession`。

AWSGlueSessionUserRestrictedNotebookPolicy

仅当标签键为 “所有者” 和值与委托人（用户或角色）的 AWS 用户 ID 匹配时，才 AWSGlueSessionUserRestrictedNotebookPolicy 提供从笔记本创建AWS Glue交互式会话的权限。有关更多信息，请参阅您可以使用策略变量的位置。此策略已附加到从 AWS Glue Studio 创建 AWS Glue 交互式会话笔记本的主体（用户或角色）。该策略还允许对AWS Glue Studio笔记本有足够的访问权限，以便与交互AWS Glue Studio式会话资源进行交互，这些资源是使用与委托人的 AWS 用户 ID 匹配的 “所有者” 标签值创建的。此策略拒绝在创建会话后从 AWS Glue 会话资源中更改或删除“拥有者”标签的权限。

AWSGlueSessionUserRestrictedNotebookServiceRole

AWSGlueSessionUserRestrictedNotebookServiceRole提供了对AWS Glue Studio笔记本的足够访问权限，可以与AWS Glue交互式会话资源进行交互，这些资源创建的 “所有者” 标签值与笔记本创建者的委托人（用户或角色）的用户 ID 相匹配。 AWS 有关更多信息，请参阅您可以使用策略变量的位置。此服务角色策略附加到作为魔法传递给笔记本或作为执行角色传递给 CreateSession API 的角色。此策略还允许仅当标签键为 “所有者” 和值与委托人的 AWS 用户 ID 匹配时，才允许通过笔记本创建AWS Glue交互式会话。此策略拒绝在创建会话后从 AWS Glue 会话资源中更改或删除“拥有者”标签的权限。该策略还包括写入和读取 Amazon S3 存储桶、写入 CloudWatch 日志、为所AWS Glue使用的 Amazon EC2 资源创建和删除标签的权限。

使用用户策略将会话设为私有

您可以将关联AWSGlueSessionUserRestrictedPolicy到您账户中每个用户的 IAM 角色，以限制他们只能使用值与他们自己的 $ {aws: userId} 匹配的所有者标签创建会话。AWSGlueSessionUserRestrictedNotebookServiceRole您需要AWSGlueSessionUserRestrictedServiceRole分别使用与AWSGlueSessionUserRestrictedNotebookPolicy和类似的AWSGlueSessionUserRestrictedPolicy策略。有关更多信息，请参阅 Using-identity based policies。此策略将缩小会话的访问权限至仅限

于创建者（又称用户的 ${aws:userId}），他们使用表明自身 ${aws:userId} 的拥有者标签来创建会话。如果您按照中的步骤使用 IAM 控制台自己创建了执行角色设置运行时角色，那么除了附加AwsGlueSessionUserRestrictedPolicy托管策略外，还要将以下内联策略附加到您账户中的每个用户，以允许您之前创建iam:PassRole的执行角色使用。

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

AWSGlueSessionUserRestrictedPolicy

仅当AWSGlueSessionUserRestrictedPolicy提供了与其 AWS 用户 ID 相匹配的标签密钥 “所有者” 和值时，才提供使用 CreateSession API 创建AWS Glue交互式会话的权限。此身份策略附加到调用 API 的 CreateSession 用户。该策略还允许与交互AWS Glue式会话资源进行交互，这些资源是使用与其 AWS 用户 ID 匹配的 “所有者” 标签和值创建的。此策略拒绝在创建会话后从 AWS Glue 会话资源中更改或删除“拥有者”标签的权限。

AWSGlueSessionUserRestrictedServiceRole

AWSGlueSessionUserRestrictedServiceRole提供对除会话之外的所有AWS Glue资源的完全访问权限，并允许用户仅创建和使用与用户关联的交互式会话。该策略还包括管理其他 AWS 服务中的 Glue 资源所需的其他权限。AWS Glue该策略还允许为其他 AWS 服务中的AWS Glue资源添加标签。

IAM policy 注意事项

交互式会话是 AWS Glue 中的 IAM 资源。由于它们是 IAM 资源，因此对会话的访问和交互受 IAM policy 的约束。根据附加到客户端主体或管理员配置的执行角色的 IAM policy，客户端主体（用户或角色）将能够创建新会话并与自己的会话和其他会话进行交互。

如果管理员附加了允许访问 AWSGlueServiceRole 该账户中所有AWS Glue资源的 IAM 策略（如 AWSGlueConsoleFullAccess 或），则客户委托人将能够相互协作。例如，如果策略允许，一个用户将能够与其他用户创建的会话进行交互。

如果您想配置根据您的特定需求量身定制的策略，请参阅有关为策略配置资源的 IAM 文档。例如，为了隔离属于某个用户的会话，您可以使用AWS Glue交互会话支持的 TagOnCreate 功能。请参阅 将您的会话设为私密会话 TagOnCreate 。

交互式会话支持根据特定 VPC 条件限制会话创建。请参阅 使用条件键控制设置的策略。

将脚本或笔记本转换为 AWS Glue 任务

您可以通过两种方式将脚本或笔记本转换为 AWS Glue 任务：

- 使用 nbconvert 将 Jupyter `.ipynb` notebook 文档文件转换为 `.py` 文件。有关更多信息，请参阅 nbconvert：将笔记本转换为其他格式。
- 将文件上传到 AWS Glue Studio 笔记本。
  - 在 AWS Glue Studio 控制台中，从导航菜单中选择 Jobs（任务）。
  - 在 Create job（创建任务）部分，选择 Jupyter Notebook。
  - 在 Options（选项）部分，选择 Upload and edit an existing notebook（上传并编辑现有笔记本）。
  - 选择 Choose file（选择文件）以上传 `.ipynb` 文件。

AWS Glue 串流交互式会话

切换串流会话类型

使用 AWS Glue 交互式会话配置魔法命令 `%streaming` 以定义您正在运行的任务并初始化串流交互式会话。

用于交互式开发的采样输入流式传输

我们为帮助增强交互式会话中的交AWS Glue互体验而开发的一种工具是，在下GlueContext方添加了一种获取静态直播快照的新方法 DynamicFrame。 GlueContext允许您检查、交互和实施您的工作流程。

使用 GlueContext 类实例，您将能够找到方法 getSampleStreamingDynamicFrame。此方法要求的参数为:

- dataFrame: Spark 直播 DataFrame
- options：查看以下可用选项

可用选项包括:

- windowSize：这也称为微批处理持续时间。此参数将确定在触发前一批处理后串流查询的等待时间。此参数值必须小于 pollingTimeInMs。
- pollingTimeInMs：该方法运行的总时长。它将触发至少一个微批处理，以从输入流式传输中获取样本记录。
- recordPollingLimit：此参数可帮助您限制要从直播中轮询的记录总数。
- （可选）您也可以使用 writeStreamFunction 将此自定义函数应用于每个记录采样函数。有关 Scala 和 Python 中的示例，请参阅以下内容。

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**
>
> 在采样的 DynFrame 为空时，可能是由以下几个原因造成的：
>
> - 流式传输源设置为“Latest（最新）”，并且在采样周期内没有摄入新数据。
> - 轮询时间不足以处理其摄入的记录。除非整个批处理处理完毕，否则数据不会显示。

在交互式会话中运行串流应用程序

在 AWS Glue 交互式会话中，您可以运行 AWS Glue 串流应用程序，就像您在 AWS Glue 控制台中创建串流应用程序一样。由于交互式会话基于会话，因此在运行时遇到异常不会导致会话停止。目前,我们具有以迭代方式开发批处理函数的额外优势。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 =
 {**})
```

我们在以上例子中包括了方法的无效用法，与退出整个应用程序的常规 AWS Glue 任务不同，用户的编码上下文和定义都完全保留，并且会话仍然在运行。无需引导启动新集群和重新运行所有之前的转换。这使您可以专注于快速迭代批处理函数实施以获得理想的结果。

需要注意的是，交互式会话以阻塞方式评估每个语句，以便会话一次仅能执行一条语句。由于流式传输查询将始终连续且永不结束，具有有效流式传输查询的会话将无法处理任何后续语句，除非这些会话中断。您可以直接从 Jupyter Notebook 发出中断命令，我们的内核将为您处理取消。

以下列正在等待执行的语句序列为例：

```
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
```

在本地开发和测试 AWS Glue 作业脚本

当您开发和测试 AWS Glue for Spark 任务脚本时，有多种可用选项：

- AWS Glue Studio 控制台
  - 可视化编辑器
  - 脚本编辑器
  - AWS Glue Studio 笔记本
- 交互式会话
  - Jupyter notebook
- Docker 映像
  - 本地开发
  - 远程开发
- AWS Glue Studio ETL 库
  - 本地开发

您可以根据您的要求选择以上任何选项。

如果您喜欢无代码或更少代码体验，AWS Glue Studio 可视化编辑器是不错的选择。

如果您更喜欢交互式笔记本体验，AWS Glue Studio 笔记本是一个不错的选择。有关更多信息，请参阅将笔记本与 AWS Glue Studio 和 AWS Glue 结合使用。如果您想使用您自己的本地环境，交互式会话是一个不错的选择。有关更多信息，请参阅将交互式会话与 AWS Glue 结合使用。

如果您更喜欢本地/远程开发体验，Docker 镜像是一个不错的选择。这可以帮助您在任何您喜欢的地方开发和测试 AWS Glue for Spark 任务脚本，而不会产生 AWS Glue 成本。

如果您更喜欢没有 Docker 的本地开发，则在本地安装 AWS Glue ETL 库目录是一个不错的选择。

使用 AWS Glue Studio 开发

AWS Glue Studio 可视化编辑器是一个图形界面，可以方便地在 AWS Glue 中创建、运行和监控提取、转换和加载（ETL）任务。您可以直观地编写数据转换工作流，并在 AWS Glue 的基于 Apache Spark 的无服务器 ETL 引擎上顺畅运行。您可以在任务的每个步骤中检查架构和数据结果。有关更多信息，请参阅《AWS Glue Studio 用户指南》。

使用交互式会话进行开发

交互式会话使您可以在自己选择的环境中构建和测试应用程序。有关更多信息，请参阅将交互式会话与 AWS Glue 结合使用。

使用 Docker 镜像进行开发

> **Note**
>
> 本部分中的说明尚未在 Microsoft Windows 操作系统上进行测试。
> 有关 Windows 平台上的本地开发和测试，请参阅博客 Building an AWS Glue ETL pipeline locally without an AWS account

对于生产就绪型数据平台，AWS Glue 任务的开发过程和 CI/CD 管道是一个关键主题。您可以在 Docker 容器中灵活地开发和测试 AWS Glue 作业。AWS Glue 在 Docker Hub 上托管 Docker 映像，以使用其他实用程序设置开发环境。您可以使用首选 IDE、笔记本或使用 AWS Glue ETL 库的 REPL。本主题介绍如何使用 Docker 映像在 Docker 容器中开发和测试 AWS Glue 版本 4.0 作业。

可以在 Docker Hub 上使用适用于 AWS Glue 的以下 Docker 映像。

- 对于 AWS Glue 版本 4.0：`amazon/aws-glue-libs:glue_libs_4.0.0_image_01`
- 对于 AWS Glue 版本 3.0：`amazon/aws-glue-libs:glue_libs_3.0.0_image_01`
- 对于 `amazon/aws-glue-libs:glue_libs_2.0.0_image_01` 版本 2.0：AWS Glue

这些镜像适用于 x86_64。建议您在此架构上进行测试。但是，有可能在不支持的基础镜像上重新设计本地开发解决方案。

此示例介绍如何在本地计算机上使用 amazon/aws-glue-libs:glue_libs_4.0.0_image_01 和运行容器。此容器镜像已经过 AWS Glue 版本 3.3 Spark 作业测试。此影像包含以下内容：

- Amazon Linux
- AWS Glue ETL 库（aws-glue-libs）
- Apache Spark 3.3.0
- Spark 历史记录服务器
- Jupyter Lab
- Livy
- 其他库依赖项（与其中一个 AWS Glue 任务系统的设置相同）

根据您的要求完成下列部分之一：

- 将容器设置为使用 spark-submit
- 将容器设置为使用 REPL shell（PySpark）
- 将容器设置为使用 Pytest
- 将容器设置为使用 Jupyter Lab
- 将容器设置为使用 Visual Studio 代码

先决条件

在开始之前，请确保已安装 Docker 并且 Docker 守护进程正在运行。有关安装说明，请参阅 Mac 或 Linux 的 Docker 文档。运行 Docker 的计算机托管 AWS Glue 容器。另外，请确保运行 Docker 的主机上的镜像至少有 7GB 的磁盘空间。

有关在本地开发 AWS Glue 代码时的限制的更多信息，请参阅本地开发限制。

配置 AWS

要从容器启用 AWS API 调用，请按照以下步骤设置 AWS 凭证。在以下部分中，我们将使用此 AWS 命名配置文件。

1. 设置 AWS CLI，配置命名配置文件。有关 AWS CLI 配置的更多信息，请参阅 AWS CLI 文档中的配置和凭证文件设置。
2. 在终端中运行以下命令：

```
PROFILE_NAME="<your_profile_name>"
```

您可能还需要设置 AWS_REGION 环境变量以指定要向其发送请求的 AWS 区域。

设置和运行容器

将容器设置为通过 spark-submit 命令运行 PySpark 代码包括以下高级步骤：

1. 从 Docker Hub 拉取镜像。
2. 运行容器。

从 Docker Hub 拉取镜像

请运行以下命令从 Docker Hub 中拉取镜像：

```
docker pull amazon/aws-glue-libs:glue_libs_4.0.0_image_01
```

运行容器

您现在可以使用此镜像运行容器。您可以根据您的要求选择以下任何选项。

spark-submit

您可以通过在容器上运行 `spark-submit` 命令来运行 AWS Glue 任务脚本。

1. 编写脚本并在 `/local_path_to_workspace` 目录下将其另存为 `sample1.py`。本主题中包含示例代码作为附录。

```
$ WORKSPACE_LOCATION=/local_path_to_workspace
$ SCRIPT_FILE_NAME=sample.py
$ mkdir -p ${WORKSPACE_LOCATION}/src
$ vim ${WORKSPACE_LOCATION}/src/${SCRIPT_FILE_NAME}
```

2. 运行以下命令以在容器上执行 `spark-submit` 命令，以提交新的 Spark 应用程序：

```
$ 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. （可选）配置 `spark-submit` 以匹配您的环境。例如，您可以将依赖关系与 `--jars` 配置一起传递。有关更多信息，请参阅 Spark 文档中的使用 spark-submit 启动应用程序。

REPL shell（Pyspark）

您可以运行 REPL（read-eval-print 循环）shell 进行交互式开发。

运行以下命令在容器上执行 PySpark 命令以启动 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

对于单元测试，您可以将 pytest 用于 AWS Glue Spark 任务脚本。

运行以下命令进行准备。

```
$ 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}
```

运行以下命令以在测试套件上执行 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 ===========================================================
```

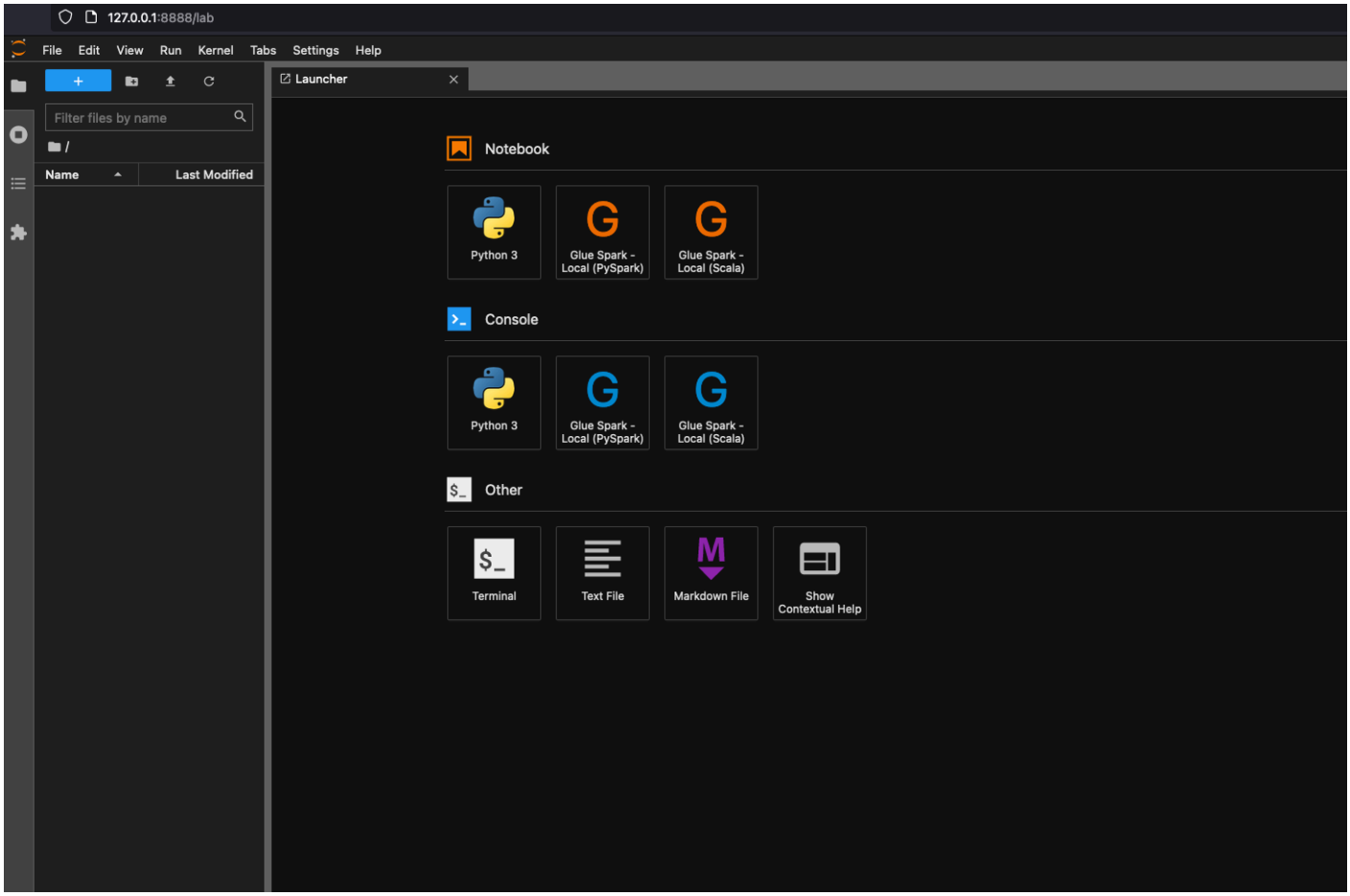
Jupyter Lab

您可以启动 Jupyter 以便在笔记本上进行交互式开发和临时查询。

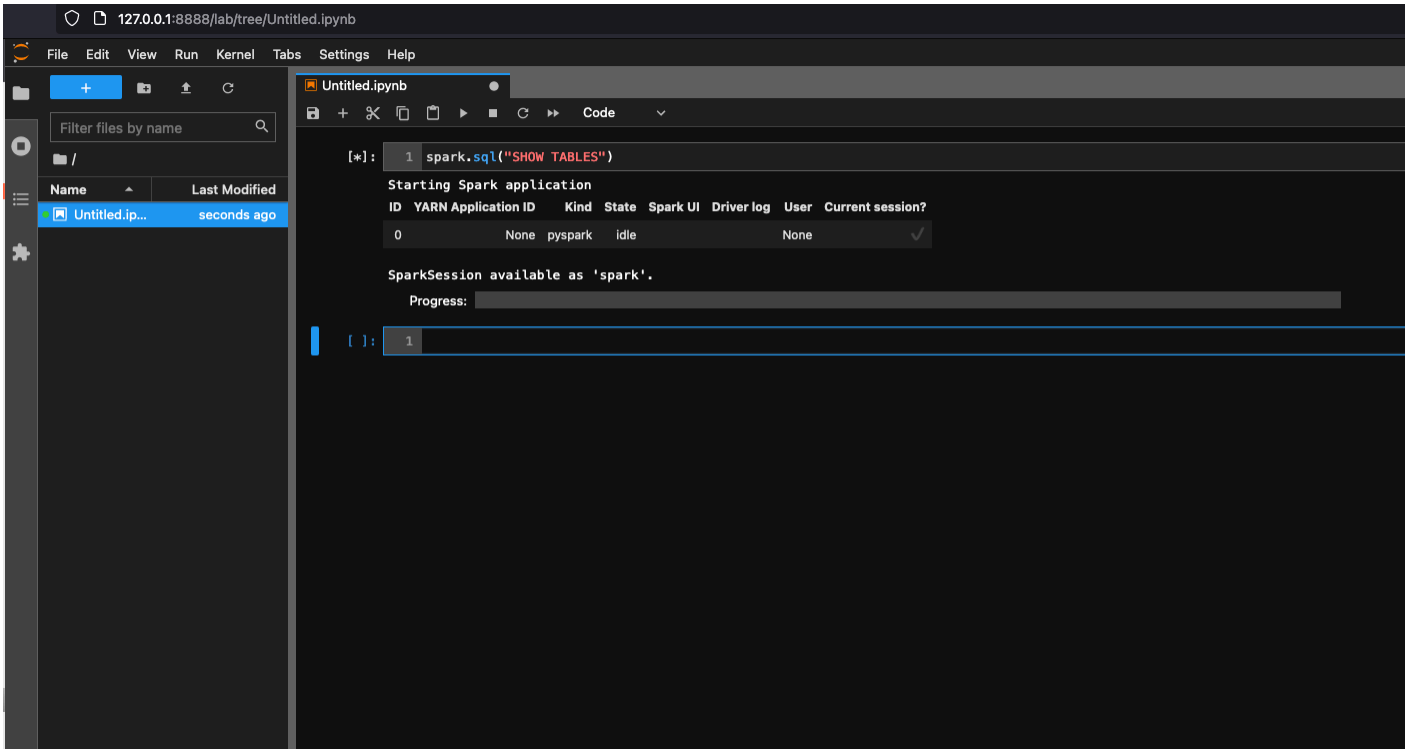
1. 运行以下命令启动 Jupyter Lab：

```
$ 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. 在本地计算机的 Web 浏览器中打开 http://127.0.0.1:8888/lab，以查看 Jupyter 实验室用户界面。



3. 在笔记本下选择 Glue Spark Local (PySpark)。您可以开始在交互式 Jupyter notebook UI 中开发代码。



将容器设置为使用 Visual Studio 代码

先决条件：

1. 安装 Visual Studio 代码。
2. 安装 Python。
3. 安装 Visual Studio Code Remote - 容器
4. 在 Visual Studio 代码中打开工作区文件夹。
5. 选择设置。
6. 请选择 Workspace（工作区）。
7. 请选择 Open Settings (JSON)（打开设置（JSON））。
8. 粘贴以下 JSON 并保存它。

```
{
    "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/",
    ]
```

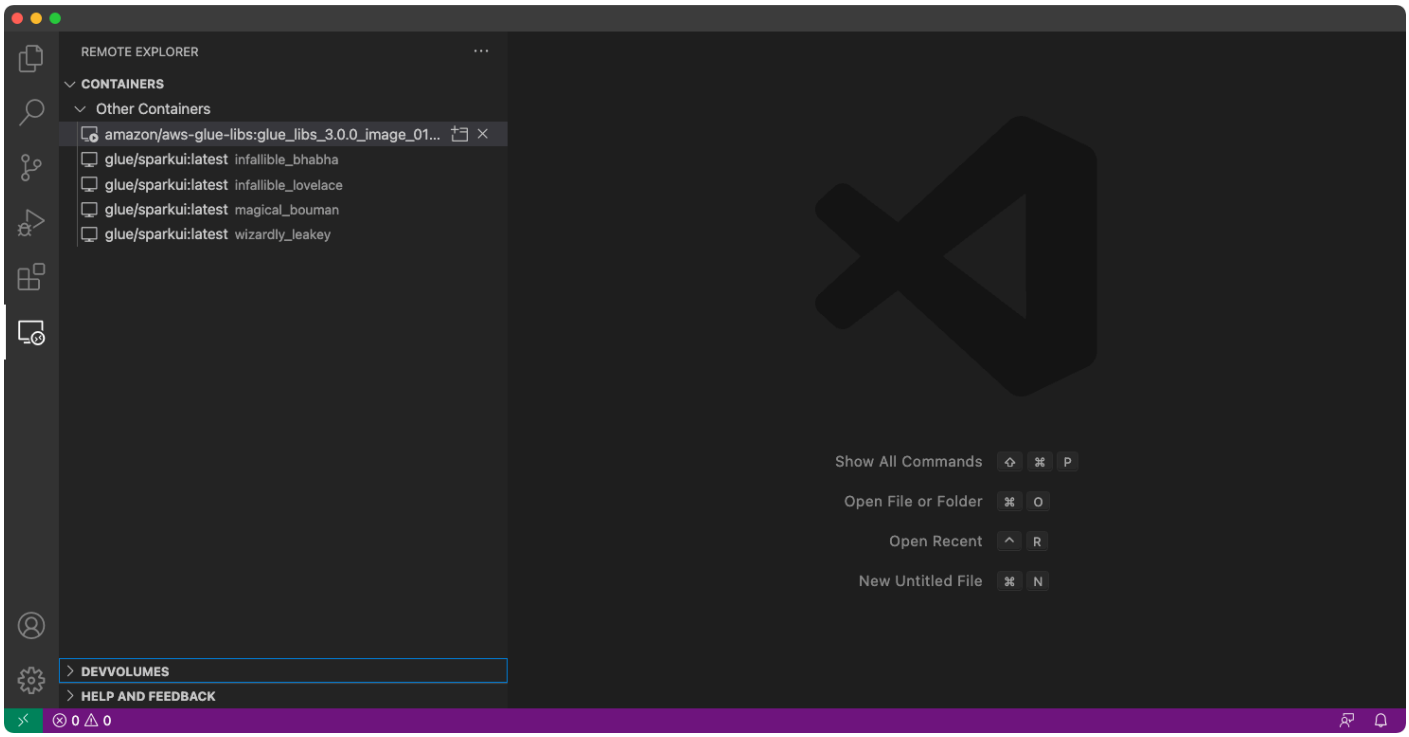
```
}
```

步骤：

1. 运行 Docker 容器。

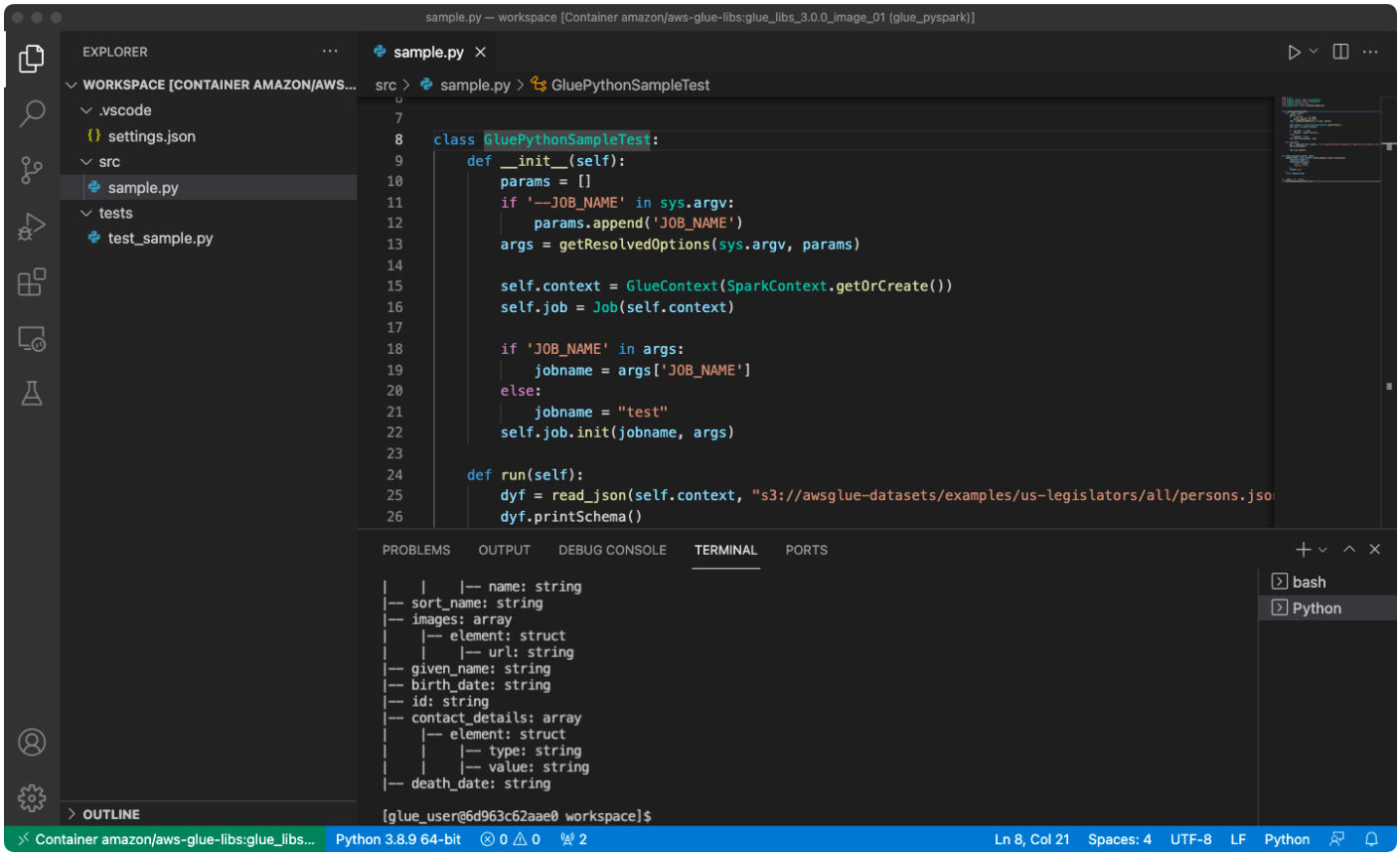
```
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v $WORKSPACE_LOCATION:/
home/glue_user/workspace/ -e AWS_PROFILE=$PROFILE_NAME -e DISABLE_SSL=true
 --rm -p 4040:4040 -p 18080:18080 --name glue_pyspark amazon/aws-glue-
libs:glue_libs_4.0.0_image_01 pyspark
```

2. 启动 Visual Studio 代码。
3. 请选择左侧菜单中的 Remote Explorer，然后选择 amazon/aws-glue-libs:glue_libs_4.0.0_image_01。



4. 右键单击并选择 Attach to Container（附加到容器）。如果显示对话框，请选择 Got it（明白了）。
5. 打开 /home/glue_user/workspace/。
6. 创建 Glue PySpark 脚本，然后选择 Run（运行）。

   您将看到脚本成功运行。



附录：用于测试的 AWS Glue 任务示例代码

本附录提供了脚本作为用于测试目的 AWS Glue 任务示例代码。

sample.py：用于将 AWS Glue ETL 库与 Amazon S3 API 调用结合使用的示例代码

```
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()
```

上面的代码需要 AWS IAM 中的 Amazon S3 权限。您需要授予 IAM 托管策略 `arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess` 或 IAM 自定义策略，以允许您为 Amazon S3 路径调用 `ListBucket` 和 `GetObject`。

test_sample.py：用于 sample.py 单元测试的示例代码。

```
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 库进行开发

AWS Glue ETL 库在公有 Amazon S3 存储桶中可用，并且可以由 Apache Maven 构建系统使用。这使您可以在本地开发和测试 Python 和 Scala 提取、转换和加载 (ETL) 脚本，而无需网络连接。建议使用 Docker 镜像进行本地开发，因为它提供了一个为使用此库而正确配置的环境。

本地开发适用于所有 AWS Glue 版本，包括 AWS Glue 版本 0.9、1.0、2.0 及更高版本。有关可与 AWS Glue 配合使用的 Python 和 Apache Spark 版本的信息，请参阅 Glue version job property。

该库随 Amazon 软件许可证 (https://aws.amazon.com/asl) 一起发布。

本地开发限制

使用 AWS Glue Scala 库进行本地开发时，请记住以下限制。

- 避免使用 AWS Glue 库创建程序集 jar（“fat jar”或“uber jar”），因为这将导致以下功能被禁用：
  - 作业书签
  - AWS Glue Parquet 写入器 (在 AWS Glue 中使用 Parquet 格式)

- FillMissingValues 转换（Scala 或 Python）

这些功能仅在 AWS Glue 作业系统内可用。

- 本地开发不支持 FindMatches 转换。
- 本地开发不支持矢量化的 SIMD CSV 读取器。
- 本地开发不支持用于从 S3 路径加载 JDBC 驱动程序的 customJdbcDriverS3Path 属性。或者，您可以在本地下载 JDBC 驱动程序并从那里加载。
- 本地开发不支持 Glue Data Quality。

使用 Python 在本地开发

完成一些先决步骤，然后使用 AWS Glue 实用工具来测试和提交您的 Python ETL 脚本。

本地 Python 开发的先决条件

完成以下步骤以准备本地 Python 开发：

1. 从 GitHub（https://github.com/awslabs/aws-glue-libs）克隆 AWS Glue Python 存储库。
2. 请执行以下操作之一：
   - 对于 AWS Glue 版本 0.9，请签出分支 `glue-0.9`。
   - 对于 AWS Glue 版本 1.0，请签出分支 `glue-1.0`。AWS Glue 0.9 以上的所有版本均支持 Python 3。
   - 对于 AWS Glue 版本 2.0，请签出分支 `glue-2.0`。
   - 对于 AWS Glue 版本 3.0，请签出分支 `glue-3.0`。
   - 对于 AWS Glue 版本 4.0，请签出 `master` 分支。
3. 从以下位置安装 Apache Maven：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-common/apache-maven-3.6.0-bin.tar.gz。
4. 从下列位置之一安装 Apache Spark 分发：
   - 对于 AWS Glue 版本 0.9：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-0.9/spark-2.2.1-bin-hadoop2.7.tgz
   - 对于 https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-1.0/spark-2.4.3-bin-hadoop2.8.tgz 版本 1.0：AWS Glue
   - 对于 https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-2.0/spark-2.4.3-bin-hadoop2.8.tgz 版本 2.0：AWS Glue

- 对于 AWS Glue 版本 3.0：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-3.0/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3.tgz
- 对于 AWS Glue 版本 4.0：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. 导出 SPARK_HOME 环境变量，将其设置为从 Spark 归档文件中提取的根位置。例如：
   - 对于 AWS Glue 版本 0.9：`export SPARK_HOME=/home/$USER/spark-2.2.1-bin-hadoop2.7`
   - 对于 AWS Glue 版本 1.0 及 2.0：`export SPARK_HOME=/home/$USER/spark-2.4.3-bin-spark-2.4.3-bin-hadoop2.8`
   - 对于 AWS Glue 版本 3.0：`export SPARK_HOME=/home/$USER/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3`
   - 对于 AWS Glue 版本 4.0：`export SPARK_HOME=/home/$USER/spark-3.3.0-amzn-1-bin-3.3.3-amzn-0`

运行您的 Python ETL 脚本

使用可用于本地开发的 AWS Glue jar 文件，您可以在本地运行 AWS Glue Python 程序包。

使用以下实用工具和框架来测试和运行 Python 脚本。下表中列出的命令是从 AWS Glue Python 程序包的根目录运行的。

实用工具	命令	描述
AWS Glue Shell	`./bin/gluepyspark`	在与 AWS Glue ETL 库集成的 shell 中输入并运行 Python 脚本。
AWS Glue 提交	`./bin/gluesparksubmit`	提交完整的 Python 脚本以供执行。
Pytest	`./bin/gluepytest`	编写并运行 Python 代码的单元测试。必须在 PATH 中安装 pytest 模块并使其可用。有关更多信息，请参阅 pytest 文档。

使用 Scala 在本地开发

完成一些先决步骤，然后发出 Maven 命令以在本地运行 Scala ETL 脚本。

本地 Scala 开发的先决条件

完成这些步骤以准备本地 Scala 开发。

步骤 1：安装软件

在此步骤中，您将安装软件并设置所需的环境变量。

1. 从以下位置安装 Apache Maven：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-common/apache-maven-3.6.0-bin.tar.gz。
2. 从下列位置之一安装 Apache Spark 分发：
   - 对于 AWS Glue 版本 0.9：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-0.9/spark-2.2.1-bin-hadoop2.7.tgz
   - 对于 https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-1.0/spark-2.4.3-bin-hadoop2.8.tgz 版本 1.0：AWS Glue
   - 对于 https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-2.0/spark-2.4.3-bin-hadoop2.8.tgz 版本 2.0：AWS Glue
   - 对于 AWS Glue 版本 3.0：https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-3.0/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3.tgz
   - 对于 AWS Glue 版本 4.0：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. 导出 SPARK_HOME 环境变量，将其设置为从 Spark 归档文件中提取的根位置。例如：
   - 对于 AWS Glue 版本 0.9：`export SPARK_HOME=/home/$USER/spark-2.2.1-bin-hadoop2.7`
   - 对于 AWS Glue 版本 1.0 及 2.0：`export SPARK_HOME=/home/$USER/spark-2.4.3-bin-spark-2.4.3-bin-hadoop2.8`
   - 对于 AWS Glue 版本 3.0：`export SPARK_HOME=/home/$USER/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3`
   - 对于 AWS Glue 版本 4.0：`export SPARK_HOME=/home/$USER/spark-3.3.0-amzn-1-bin-3.3.3-amzn-0`

步骤 2：配置您的 Maven 项目

使用以下 pom.xml 文件作为 AWS Glue Scala 应用程序的模板。它包含所需的 `dependencies`、`repositories` 和 `plugins` 元素。将 `Glue version` 字符串替换为以下各项之一：

- `4.0.0` for AWS Glue 版本 4.0
- `3.0.0` 适用于 AWS Glue 版本 3.0
- `1.0.0` 适用于 AWS Glue 版本 1.0 或 2.0
- `0.9.0` 适用于 AWS Glue 版本 0.9

```
  <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>
```

运行您的 Scala ETL 脚本

从 Maven 项目根目录运行以下命令以运行 Scala ETL 脚本。

```
mvn exec:java -Dexec.mainClass="mainClass" -Dexec.args="--JOB-NAME jobName"
```

将 *mainClass* 替换为脚本主类的完全限定类名。将 *jobName* 替换为所需的作业名称。

配置测试环境

有关配置本地测试环境的示例，请参阅以下博客文章：

- Building an AWS Glue ETL pipeline locally without an AWS account
- 使用容器在本地开发 AWS Glue ETL 任务

如果要使用开发终端节点或笔记本电脑测试 ETL 脚本，请参阅使用开发终端节点来开发脚本。

> **Note**
>
> 不支持将开发终端节点与 AWS Glue 版本 2.0 任务一起使用。有关更多信息，请参阅运行 Spark ETL 任务，减少启动时间。

开发终端节点

> **Note**
>
> 从 2023 年 3 月 31 日起，开发端点的控制台体验将被删除。然后将通过 开发终端节点 API 和 AWS Glue CLI 创建、更新和监控开发端点。

出于以下原因，我们强烈建议从开发端点迁移到交互式会话。有关如何从开发端点迁移到交互式会话的所需操作，请参阅从开发端点迁移到交互式会话。

描述	开发终端节点	交互式会话
Glue 版本支持	支持 AWS Glue 0.9 版 和 1.0 版	支持 AWS Glue 2.0 版及更高版本
开发端点在亚太地区（雅加达）（ap-southeast-3 ）、中东（阿联酋）（me-central-1 ）、欧洲（西班牙）（eu-south-2 ）、欧洲（苏黎世）（eu-central-2 ）或后续其他新区域不可用。	交互式会话目前在中东（阿联酋）（me-central-1 ）区域不可用，但后续可能可用。	
Spark 集群的访问方法	支持 SSH、REPL shell、Jupyter notebook、IDE（例如，PyCharm）	支持 AWS Glue Studio notebook、Jupyter notebook、各种 IDE（例如，Visual Studio Code、PyCharm）和 SageMaker 笔记本电脑
第一个查询的时间	需要 10-15 分钟才能设置 Spark 集群	设置临时的 Spark 集群最多可能需要 1 分钟
价格模型	AWS 根据配置端点的时间和 DPU 数量对开发端点收费。开发端点不会超时。每个预置的开发端点的最低计费时段为 10 分钟。此外，AWS 对 Amazon EC2 实例上的 Jupyter notebook 以及使用开发端点配置 SageMaker 笔记本时会收取费用。	AWS 根据会话处于活动状态的时间和 DPU 的数量对交互式会话收费。交互式会话具有可配置的空闲超时。 AWS Glue Studio 笔记本为交互式会话提供了内置界面，并且不收取额外费用。每个互动式会话的最低计费时长为 1 分钟。AWS Glue Studio notebook 为交互式会话提供内置界面，并且不收取额外费用

描述	开发终端节点	交互式会话
控制台体验	只能通过 CLI 和 API 获得	可通过 AWS Glue 控制台、CLI 和 API 获得

从开发端点迁移到交互式会话

使用以下清单确定从开发端点迁移到交互式会话的适当方法。

您的脚本是否依赖于 AWS Glue 0.9 或 1.0 的特定功能（例如，HDFS、YARN 等）？

如果答案是肯定的，请参阅将 AWS Glue 作业迁移到 AWS Glue 版本 3.0，以了解如何从 Glue 0.9 或 1.0 迁移到 Glue 3.0 及更高版本。

使用哪种方法访问开发端点？

如果您使用此方法	然后执行此操作
SageMaker notebook、Jupyter notebook 或 JupyterLab	通过在 Jupyter 上下载 `.ipynb` 文件迁移到 AWS Glue Studio notebook，并通过上传 `.ipynb` 文件创建新的 AWS Glue Studio notebook 作业。或者，您也可以使用 SageMaker Studio 并选择 AWS Glue 内核
Zeppelin notebook	通过复制和粘贴代码或自动使用第三方转换器（如，ze2nb）将 notebook 手动转换为 Jupyter notebook。然后，在 AWS Glue Studio 笔记本或 SageMaker Studio 中使用笔记本。
IDE	请参阅使用 PyCharm 并使用 AWS Glue 交互式会话编辑 AWS Glue 作业，或 Microsoft Visual Studio Code 与交互式会话配合使用。
REPL	在本地安装 `aws-glue-sessions` package，然后运行以下命令： • 对于 Python：`jupyter console --kernal glue_pyspark`

如果您使用此方法	然后执行此操作
	• 对于 Scala：`jupyter console --kernal glue_spark`
SSH	交互式会话中没有相应的选项。或者，您可以使用 Docker 映像。要了解更多信息，请参阅使用 Docker 映像进行开发。

以下几个部分提供有关使用开发终端节点在 AWS Glue 版本 1.0 中开发作业的信息。

主题

使用开发终端节点来开发脚本

> **Note**
>
> 只有 2.0 之前的 AWS Glue 版本支持开发端点。要获得可以创作和测试 ETL 脚本的交互式环境，请使用 AWS GlueStudio 上的笔记本。

AWS Glue 可以创建一个环境（称为开发终端节点），您可以使用该环境以迭代方式开发和测试您的提取、转换和加载 (ETL) 脚本。您可以使用 AWS Glue 控制台或 API 创建、编辑和删除开发终端节点。

管理您的开发环境

在创建开发终端节点时，您可提供配置值来设置开发环境。这些值告知 AWS Glue 如何设置网络，以便您可以安全地访问终端节点，并且终端节点可以访问您的数据存储。

然后，您可以创建一个笔记本以连接到终端节点，并使用您的笔记本编写和测试 ETL 脚本。当您对开发过程的结果感到满意时，可以创建一个运行脚本的 ETL 任务。通过此过程，您可以用交互式方式添加函数并调试脚本。

按照本节中的教程操作，了解如何通过笔记本使用开发终端节点。

主题

开发终端节点工作流

要使用 AWS Glue 开发终端节点，您可以遵循此工作流程：

1. 使用 API 创建开发端点。此终端节点在 Virtual Private Cloud (VPC) 中与您定义的安全组一起启动。
2. API 会对开发端点进行轮询，直到它被预置并做好工作准备。准备就绪之后，使用下列方法之一连接到开发终端节点来创建和测试 AWS Glue 脚本。
   - 在您的账户中创建一个 SageMaker 笔记本。有关如何创建笔记本的更多信息，请参阅 the section called “使用 AWS Glue Studio 笔记本编写代码”。
   - 打开终端窗口以直接连接到开发终端节点。
   - 如果您拥有专业版 JetBrains PyCharm Python IDE，则将它连接到开发终端节点并使用它以交互方式进行开发。如果您在脚本中插入 `pydevd` 语句，则 PyCharm 可以支持远程断点。
3. 当您完成对开发终端节点的调试和测试后，可以将其删除。

AWS Glue 开发终端节点如何与 SageMaker 笔记本配合使用

访问开发终端节点的常见方法之一是使用 SageMaker 笔记本上的 Jupyter。Jupyter notebook 是一个开源的 Web 应用程序，广泛应用于可视化、分析和机器学习等领域。AWS Glue SageMaker 笔记本为您提供 AWS Glue 开发终端节点的 Jupyter notebook 体验。在 AWS Glue SageMaker 笔记本中，Jupyter notebook 环境预先配置了 SparkMagic，这是一个开源 Jupyter 插件，用于将 Spark 任务提交至远程 Spark 集群。Apache Livy 是一种允许通过 REST API 与远程 Spark 集群进行交互的服务。在 AWS Glue SageMaker 笔记本中，SparkMagic 被配置为针对运行在 AWS Glue 开发终端节点上的 Livy 服务器调用 REST API。

以下文本流说明了每个组件的工作原理：

AWS Glue SageMaker 笔记本：(Jupyter → SparkMagic) → (网络) → AWS Glue 开发端点：(Apache Livy → Apache Spark)

在 Jupyter notebook 上运行在每个段落中编写的 Spark 脚本后，Spark 代码将通过 SparkMagic 提交到 Livy 服务器，然后名为“livy-session-N”的 Spark 任务会在 Spark 集群上运行。此任务称为 Livy 会话。Spark 任务将在笔记本会话处于活跃状态时运行。当您从笔记本中关闭 Jupyter 内核或会话超时时，Spark 任务将终止。每个笔记本 (.ipynb) 文件启动一个 Spark 任务。

您可以将单个 AWS Glue 开发终端节点与多个 SageMaker 笔记本实例结合使用。您可以在每个 SageMaker 笔记本实例中创建多个笔记本文件。当您打开每个笔记本文件并运行段落时，将通过 SparkMagic 在 Spark 集群上针对每个笔记本文件启动一个 Livy 会话。每个 Livy 会话对应于一项 Spark 任务。

AWS Glue 开发终端节点和 SageMaker 笔记本的默认行为

Spark 任务基于 Spark 配置运行。有多种方法可以设置 Spark 配置 (例如，Spark 集群配置、SparkMagic 的配置等)。

默认情况下，Spark 会根据 Spark 集群配置将集群资源分配给 Livy 会话。在 AWS Glue 开发终端节点中，集群配置取决于工作线程类型。下表阐述了每个工作线程类型的常见配置。

	Standard	G.1X	G.2X
`spark.driver.memory`	5G	10G	20G
`spark.executor.memory`	5G	10G	20G
`spark.executor.cores`	4	8	16
`spark.dynamicAlloc`	TRUE	TRUE	TRUE

	Standard	G.1X	G.2X
ation.ena bled			

Spark 执行程序的最大数量是通过将 DPU（或 NumberOfWorkers）和工作线程类型结合自动计算得出的。

	Standard	G.1X	G.2X
Spark 执行程序的最大数量	(DPU - 1) * 2 - 1	(NumberOfWorkers - 1)	(NumberOfWorkers - 1)

例如，如果您的开发终端节点有 10 个工作线程，并且工作线程类型为 G.1X，那么您将有 9 个 Spark 执行程序，整个集群将有 90G 的执行程序内存，因为每个执行程序都有 10G 的内存。

无论指定的工作线程类型如何，都将打开 Spark 动态资源分配。如果数据集足够大，Spark 可能会将所有执行程序分配给单个 Livy 会话，因为默认情况下 spark.dynamicAllocation.maxExecutors 未设置。这意味着同一开发端点上的其他 Livy 会话将等待启动新的执行程序。如果数据集很小，Spark 将能够同时将执行程序分配给多个 Livy 会话。

> **Note**
>
> 有关如何在不同使用案例中分配资源，以及如何设置配置以修改行为的详细信息，请参阅 高级配置：在多个用户之间共享开发终端节点。

添加开发终端节点

可使用开发终端节点以迭代方式开发和测试您的提取、转换和加载 (ETL) AWS Glue 中的脚本。只有通过 AWS Command Line Interface 才能使用开发端点。

1. 在命令行窗口中，输入与以下内容类似的命令。

```
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 版本 1.0。此版本同时支持 Python 2 和 Python 3，因此，您可以使用 `arguments` 参数来指明所需的 Python 版本。如果省略 `glue-version` 参数，则假定为 AWS Glue 版本 0.9。有关 AWS Glue 版本的更多信息，请参阅 Glue version job property。

有关其他命令行参数的信息，请参阅《AWS CLI 命令参考》中的创建开发端点。

2. （可选）输入以下命令可检查开发终端节点状态。如果状态变为 `READY`，则开发终端节点已可供使用。

```
aws glue get-dev-endpoint --endpoint-name "endpoint1"
```

访问您的开发终端节点

当您在 Virtual Private Cloud (VPC) 中创建开发终端节点时，AWS Glue 仅返回私有 IP 地址。不会填充公有 IP 地址字段。在创建非 VPC 开发终端节点时，AWS Glue 仅返回一个公有 IP 地址。

如果您的开发终端节点具有 Public address (公有地址)，则确认使用开发终端节点的 SSH 私有密钥是否可访问它，如以下示例中所示。

```
ssh -i dev-endpoint-private-key.pem glue@public-address
```

假设您的开发终端节点具有 Private address (私有地址)，则您的 VPC 子网可以从公共 Internet 路由，并且其安全组允许来自客户端的入站访问。在此情况下，请执行以下步骤以将弹性 IP 地址 附加到开发终端节点，以允许从 Internet 进行访问。

> **Note**
>
> 如果您希望使用弹性 IP 地址，则所使用的子网需要通过路由表关联的 Internet 网关。

通过附加弹性 IP 地址来访问开发终端节点

1. 通过 https://console.aws.amazon.com/glue/ 打开 AWS Glue 控制台。

2. 在导航窗格中，选择 Dev endpoints (开发终端节点)，并导航到开发终端节点详细信息页面。记下 Private address (私有地址)，以供下一步使用。
3. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
4. 在导航窗格中，在 Network & Security (网络与安全性) 下，选择 Network Interfaces (网络接口)。
5. 在 AWS Glue 控制台开发终端节点详细信息页面中搜索与 Private address (私有地址) 对应的 Private DNS (IPv4) (私有 DNS (IPv4)。

   您可能需要修改您的 Amazon EC2 控制台上显示哪些列。记下此地址的 Network interface ID (网络接口 ID) (ENI) (例如 `eni-12345678`) 。
6. 在 Amazon EC2 控制台上的 Network & Security (网络与安全) 下，选择 Elastic IPs (弹性 IP)。
7. 选择 Allocate new address (分配新地址)，然后选择 Allocate (分配) 以分配新的弹性 IP 地址。
8. 在 Elastic IPs (弹性 IP) 页面上，选择新分配的 Elastic IP (弹性 IP)。依次选择 Actions (操作) 和 Associate address (关联地址)。
9. 在 Associate address (关联地址) 页面上，执行以下操作：
   - 对于 Resource type (资源类型)，选择 Network interface (网络接口)。
   - 在 Network interface (网络接口) 框中，输入私有地址的 Network interface ID (网络接口 ID) (ENI)。
   - 选择关联。
10. 确认是否能使用与开发终端节点关联的 SSH 私有密钥来访问新关联的弹性 IP 地址，如以下示例中所示。

    ```
    ssh -i dev-endpoint-private-key.pem glue@elastic-ip
    ```

    有关使用堡垒主机获得对开发终端节点的私有地址的 SSH 访问权限的信息，请参阅 AWS 安全性博客文章 Securely Connect to Linux Instances Running in a Private Amazon VPC。

教程：在 JupyterLab 中设置 Jupyter notebook 以测试和调试 ETL 脚本

在本教程中，您要将 JupyterLab 中运行在本地计算机上的 Jupyter notebook 连接到开发终端节点。执行此操作是为了在部署脚本之前，以交互方式运行、调试和测试 AWS Glue 提取、转换和加载 (ETL) 脚本。本教程使用 Secure Shell (SSH) 端口转发将本地计算机连接到 AWS Glue 开发终端节点。有关更多信息，请参阅 Wikipedia 上的端口转发。

步骤 1：安装 JupyterLab 和 Sparkmagic

您可以使用 conda 或者 pip 安装 JupyterLab。conda 是一个开源程序包管理系统和环境管理系统，在 Windows、macOS 和 Linux 上运行。pip 是 Python 的软件包安装程序。

如果您在 macOS 上安装，则必须先安装 Xcode，然后才能安装 Sparkmagic。

1. 安装 JupyterLab、Sparkmagic 和相关扩展。

```
$ conda install -c conda-forge jupyterlab
$ pip install sparkmagic
$ jupyter nbextension enable --py --sys-prefix widgetsnbextension
$ jupyter labextension install @jupyter-widgets/jupyterlab-manager
```

2. 检查 sparkmagic 的 Location 目录。

```
$ pip show sparkmagic | grep Location
Location: /Users/username/.pyenv/versions/anaconda3-5.3.1/lib/python3.7/site-
packages
```

3. 将您的目录更改为 Location 返回的目录，并安装 Scala 和 PySpark 的内核。

```
$ 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. 下载示例 config 文件。

```
$ curl -o ~/.sparkmagic/config.json https://raw.githubusercontent.com/jupyter-
incubator/sparkmagic/master/sparkmagic/example_config.json
```

   在此配置文件中，您可以配置与 Spark 相关的参数，如 driverMemory 和 executorCores。

步骤 2：启动 JupyterLab

启动 JupyterLab 时，默认的 Web 浏览器会自动打开，显示 URL http://localhost:8888/lab/workspaces/{workspace_name}。

```
$ jupyter lab
```

步骤 3：启动 SSH 端口转发以连接您的开发终端节点

接下来，使用 SSH 本地端口转发将本地端口（此处为 8998）转发到由 AWS Glue（169.254.76.1:8998）定义的远程目标。

1. 打开让您能够访问 SSH 的单独终端窗口。在 Microsoft Windows 中，您可以使用 Git for Windows 提供的 BASH Shell，或者安装 Cygwin。
2. 运行以下 SSH 命令，按如下所示进行修改：

   - 将 *private-key-file-path* 替换为包含与您用于创建开发终端节点的公有密钥对应的私有密钥的 .pem 文件的路径。
   - 如果您正在转发 8998 以外的其他端口，请将 8998 替换为您实际在本地使用的端口号。地址 169.254.76.1:8998 是远程端口，您无法更改。
   - 将 *dev-endpoint-public-dns* 替换为您的开发终端节点的公有 DNS 地址。要查找此地址，请导航到您在 AWS Glue 控制台中的开发终端节点，选择所需名称，并复制在 Endpoint details (终端节点详细信息) 页面中列出的 Public address (公有地址)。

   ```
   ssh -i private-key-file-path -NTL 8998:169.254.76.1:8998 glue@dev-endpoint-public-dns
   ```

   您可能会看到类似如下的警告消息：

   ```
   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)?
   ```

   输入 **yes**，在使用 JupyterLab 时保持终端窗口打开。
3. 检查 SSH 端口转发是否与开发终端节点正确配合使用。

   ```
   $ curl localhost:8998/sessions
   {"from":0,"total":0,"sessions":[]}
   ```

步骤 4：在笔记本段落中运行简单脚本片段

现在，您在 JupyterLab 中的笔记本应该与您的开发终端节点配合使用。在笔记本中输入以下脚本片段并运行它。

1. 检查 Spark 是否成功运行。以下命令指示 Spark 计算 1，然后打印值。

```
spark.sql("select 1").show()
```

2. 检查 AWS Glue Data Catalog 集成是否正常工作。以下命令可列出数据目录中的表。

```
spark.sql("show tables").show()
```

3. 检查使用 AWS Glue 库的简单脚本片段是否有效。

   下面的脚本使用 AWS Glue Data Catalog 中的 `persons_json` 表元数据从您的示例数据中创建 `DynamicFrame`。然后，它会打印出该数据的项目计数和架构。

```
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()
```

脚本的输出如下所示。

```
 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
```

故障排除

- 在 JupyterLab 安装过程中，如果您的计算机位于公司代理或防火墙后面，您可能会遇到由公司 IT 部门管理的自定义安全配置文件造成的 HTTP 和 SSL 错误。

  以下是当 conda 无法连接到其自己的存储库时的一个典型错误示例：

  ```
  CondaHTTPError: HTTP 000 CONNECTION FAILED for url <https://repo.anaconda.com/pkgs/
  main/win-64/current_repodata.json>
  ```

  发生这种情况可能是因为您的公司可能阻止与 Python 和 JavaScript 社区中广泛使用的存储库的连接。有关更多信息，请参阅 JupyterLab 网站上的安装问题。
- 如果您在尝试连接到开发终端节点时遇到连接被拒绝错误，说明您可能正在使用过期的开发终端节点。尝试创建新的开发终端节点并重新连接。

教程：将 SageMaker 笔记本与您的开发终端节点结合使用

在 AWS Glue 中，您可以创建开发终端节点，然后创建 SageMaker 笔记本来帮助开发 ETL 和机器学习脚本。SageMaker 笔记本是一个运行 Jupyter Notebook 应用程序的完全托管的机器学习计算实例。

1. 在 AWS Glue 控制台中，选择 Dev endpoints (开发终端节点) 以导航到开发终端节点列表。
2. 选中要使用的开发终端节点名称旁边的复选框，然后在 Action (操作) 菜单上，选择 Create SageMaker notebook (创建 SageMaker 笔记本)。
3. 填写 Create and configure a notebook (创建和配置笔记本) 页面，如下所示：

   a. 输入笔记本名称。

   b. 在 Attach to development endpoint (附加到开发终端节点) 下，确认开发终端节点。

   c. 选择或创建一个 AWS Identity and Access Management（IAM）角色。

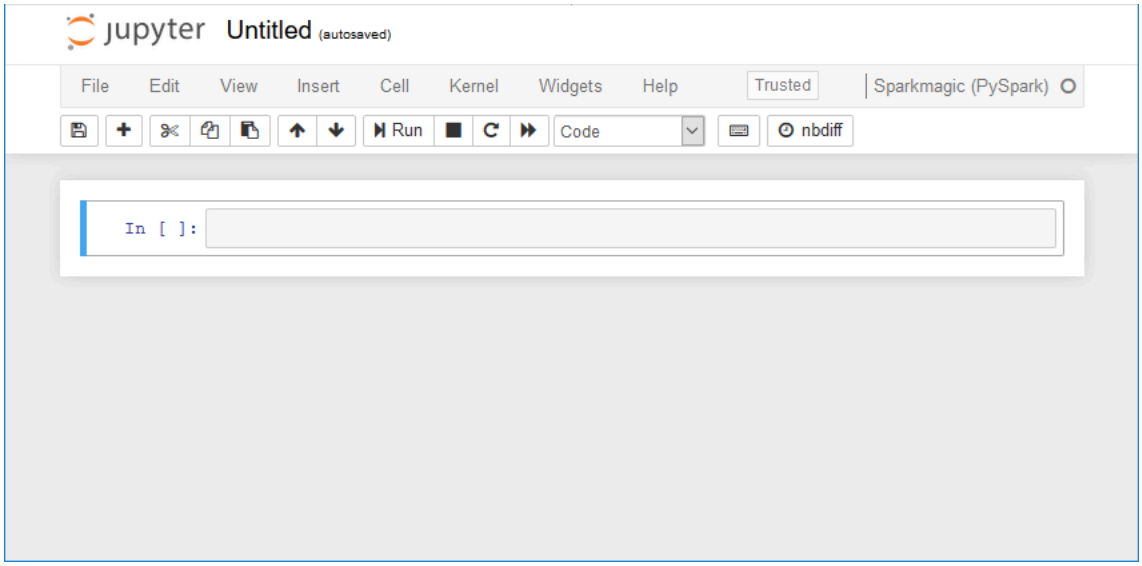
      建议您创建角色。如果您使用现有角色，请确保该角色具有所需的权限。有关更多信息，请参阅 the section called “步骤 6：为 SageMaker 笔记本创建 IAM policy”。

   d. （可选）选择 VPC、子网和一个或多个安全组。

   e. （可选）选择 AWS Key Management Service 加密密钥。

   f. （可选）为笔记本实例添加标签。
4. 选择创建笔记本。在 Notebooks (笔记本) 页面上，选择右上角的刷新图标，然后继续，直至 Status (状态) 显示 `Ready`。
5. 选中新笔记本名称旁边的复选框，然后选择 Open notebook (打开笔记本)。
6. 创建新的笔记本：在 jupyter 页面上，选择 New (新建)，然后选择 Sparkmagic (PySpark)。

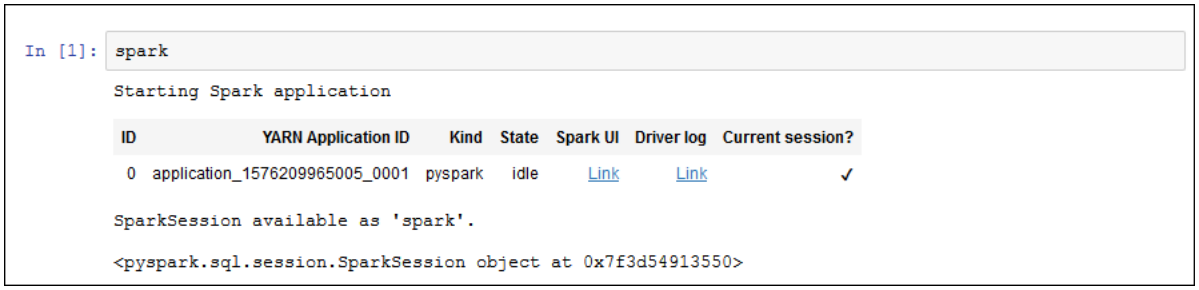
   现在，您的屏幕上显示的内容应类似于：



7. （可选）在页面顶部，选择 Untitled (无标题)，然后为笔记本提供一个名称。
8. 要启动 Spark 应用程序，请在笔记本中输入以下命令，然后在工具栏中选择 Run (运行)。

```
spark
```

在短暂的延迟后，您将会看到以下响应：



9. 创建动态帧并对其运行查询：复制、粘贴并运行以下代码，这将输出 persons_json 表的计数和架构。

```
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()
```

教程：将 REPL shell 与开发终端节点结合使用

在 AWS Glue 中，您可以创建一个开发终端节点，然后调用 REPL（读取 – 评估 – 打印循环）shell，以增量方式运行 PySpark 代码，从而在部署 ETL 脚本前交互式地调试它们。

要在开发端点上使用 REPL，您需要获得 SSH 到该端点的授权。

1. 在您的本地计算机上，打开一个可以运行 SSH 命令的终端窗口，粘贴编辑的 SSH 命令。运行命令。

   假设您已接受使用 Python 3 的 AWS Glue 版本 1.0 作为开发端点，则输出将如下所示：

```
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. 通过键入语句 `print(spark.version)` 测试 REPL shell 是否正常工作。只要显示 Spark 版本，就表明您的 REPL 现在可供使用。
3. 现在，您可以尝试在 shell 中逐行执行以下简单的脚本：

```
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()
```

教程：通过开发终端节点设置 PyCharm Professional

本教程介绍如何将您的本地计算机上运行的 PyCharm Professional Python IDE 连接到开发终端节点，以便您交互运行、调试和测试 AWS Glue ETL（提取、转换和加载）脚本，然后再进行部署。本教程中的说明和屏幕截图基于 PyCharm Professional 版本 2019.3。

要以交互方式连接到开发终端节点，您必须安装 PyCharm Professional。您不能使用免费版进行这项操作。

> **Note**
>
> 本教程使用 Amazon S3 作为数据源。如果要改用 JDBC 数据源，则必须在 Virtual Private Cloud (VPC) 中运行开发终端节点。要使用 SSH 连接到 VPC 中的开发终端节点，则必须创建 SSH 隧道。本教程不包含有关创建 SSH 隧道的说明。有关使用 SSH 连接到 VPC 中的开发终端节点的信息，请参阅 AWS 安全博客中的 Securely Connect to Linux Instances Running in a Private Amazon VPC。

主题

将 PyCharm Professional 连接到开发终端节点

1. 在 PyCharm 中创建新的纯 Python 项目，将其命名为 `legislators`。
2. 在项目中创建一个名为 `get_person_schema.py` 的文件，该文件包含以下内容：

```
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. 请执行以下操作之一：

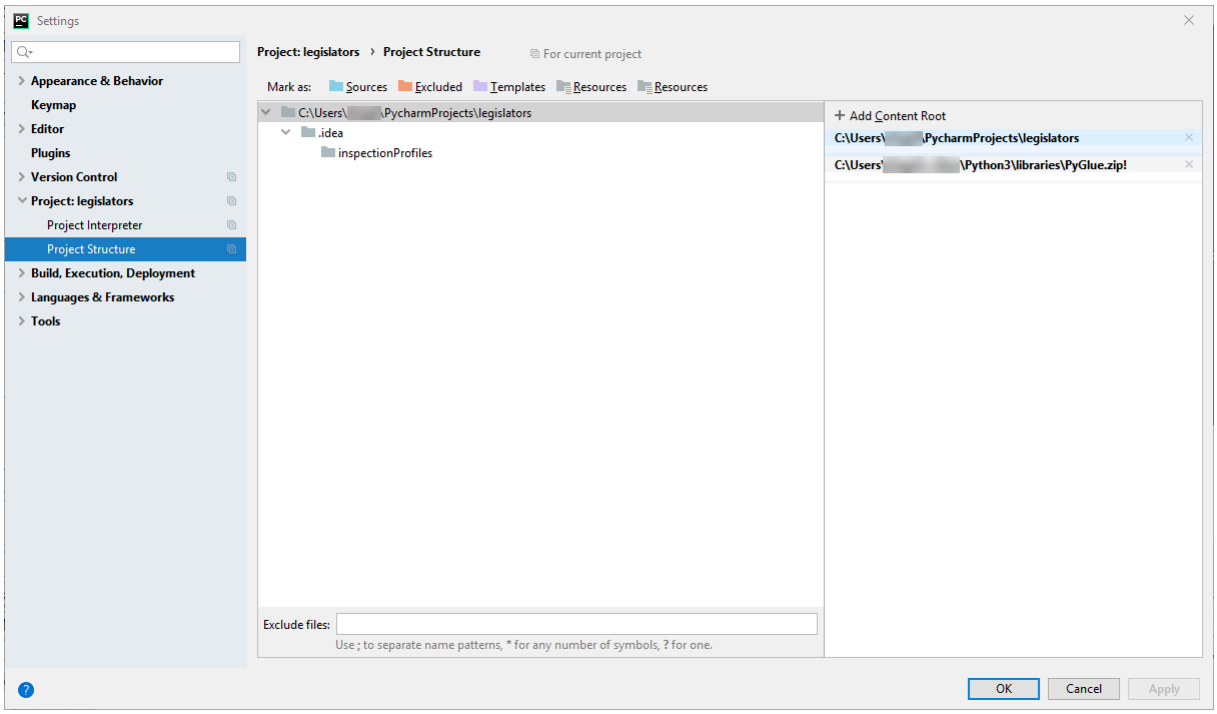
   - 对于 AWS Glue 版本 0.9，将 AWS Glue Python 库文件 `PyGlue.zip` 从 `https://s3.amazonaws.com/aws-glue-jes-prod-us-east-1-assets/etl/python/PyGlue.zip` 下载到您的本地计算机上的方便位置。

- 对于 AWS Glue 版本 1.0 及更高版本，将 AWS Glue Python 库文件 `PyGlue.zip` 从 `https://s3.amazonaws.com/aws-glue-jes-prod-us-east-1-assets/etl-1.0/python/PyGlue.zip` 下载到您的本地计算机上的方便位置。

4. 在 PyCharm 中将 `PyGlue.zip` 添加为您项目的内容根：

   - 在 PyCharm 中，依次选择 File (文件)、Settings (设置)，打开 Settings (设置) 对话框。（也可以按 `Ctrl+Alt+S`。）
   - 展开 `legislators` 项目，然后选择 Project Structure (项目结构)。然后，在右侧窗格中，选择 + Add Content Root (+ 添加内容根)。
   - 导航到您保存 `PyGlue.zip` 的位置，选择该位置，然后选择 Apply (应用)。

   Settings (设置) 屏幕上显示的内容应类似于：



   选择 Apply (应用) 后，保持 Settings (设置) 对话框处于打开状态。

5. 配置部署选项，使用 SFTP 将本地脚本上传到您的开发终端节点 (此功能仅在 PyCharm Professional 中可用)：

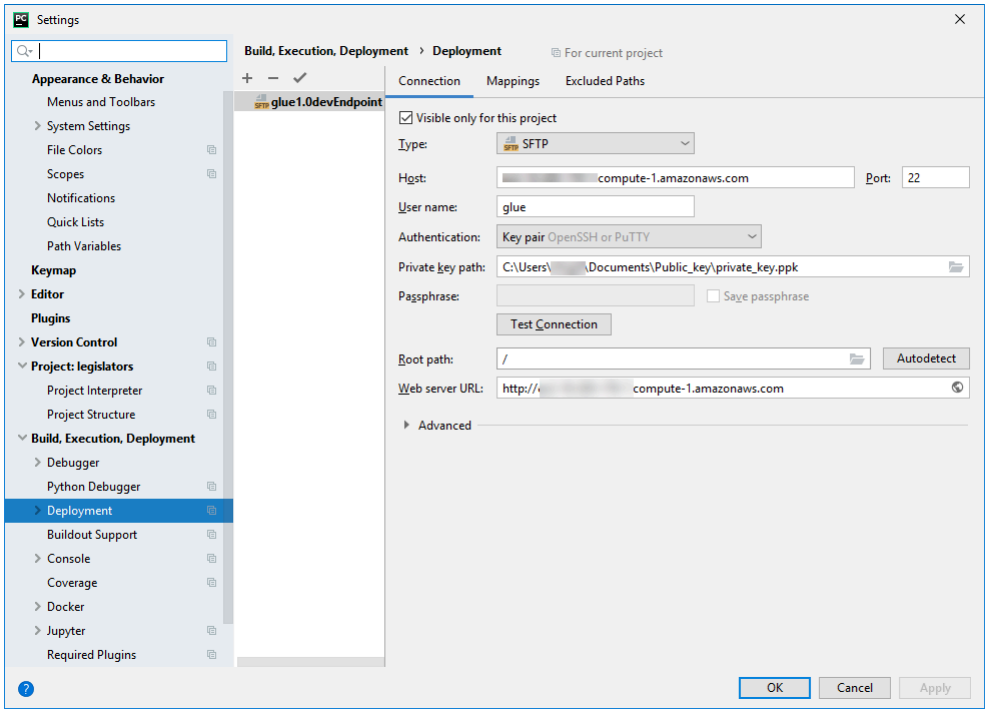
   - 在 Settings (设置) 对话框中，展开 Build, Execution, Deployment (构建、执行、部署) 部分。选择 Deployment (部署) 子部分。
   - 选择中间窗格顶部的 + 图标，添加新服务器。将其 Type (类型) 设置为 SFTP 并为其命名。

- 将 SFTP host (SFTP 主机) 设置为开发终端节点的 Public address (公有地址)，如其详细信息页面上所示。（在 AWS Glue 控制台中选择开发终端节点的名称可显示详细信息页面）。对于正在 VPC 中运行的开发终端节点，请将 SFTP host (SFTP 主机) 设置为主机地址，并将 SSH 隧道的本地端口设置为开发终端节点。
- 将 User name (用户名) 设置为 `glue`。
- 将 Auth type (验证类型) 设置为 Key pair (OpenSSH or Putty) (密钥对 (OpenSSH 或 Putty))。通过浏览到您的开发终端节点私有密钥文件所在的位置，设置 Private key file (私有密钥文件)。请注意，PyCharm 仅支持 DSA、RSA 和 ECDSA OpenSSH 密钥类型，不接受 Putty 的私有格式密钥。您可以使用最新版本的 `ssh-keygen` 生成 PyCharm 接受的密钥对类型，例如使用以下语法：

```
ssh-keygen -t rsa -f <key_file_name> -C "<your_email_address>"
```

- 选择 Test connection (测试连接)，并允许测试连接。如果连接成功，选择 Apply (应用)。

现在，Settings (设置) 屏幕上显示的内容应类似于：

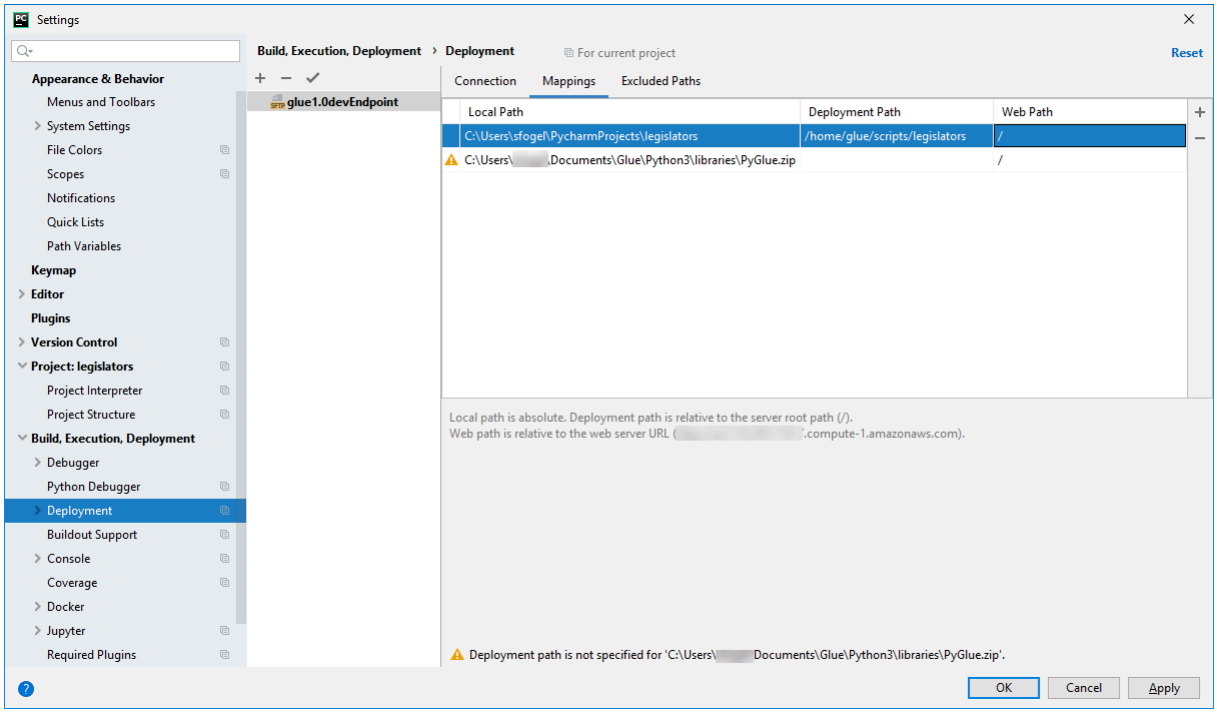


同样，选择 Apply (应用) 后，保持 Settings (设置) 对话框处于打开状态。

6. 将本地目录映射到远程目录进行部署：

- 在右侧窗格中，选择 Deployment (部署) 页面，选择顶部的中间选项卡，即标签为 Mappings (映射) 的选项卡。
- 在 Deployment Path (部署路径) 列中，输入 /home/glue/scripts/ 下的路径，以部署您的项目路径。例如：/home/glue/scripts/legislators。
- 选择 应用。

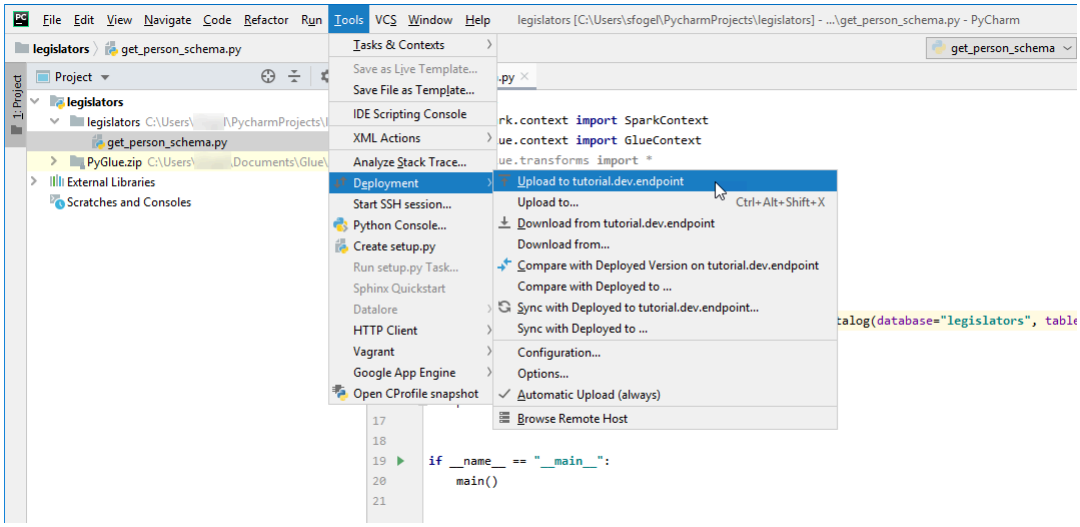
现在，Settings (设置) 屏幕上显示的内容应类似于：



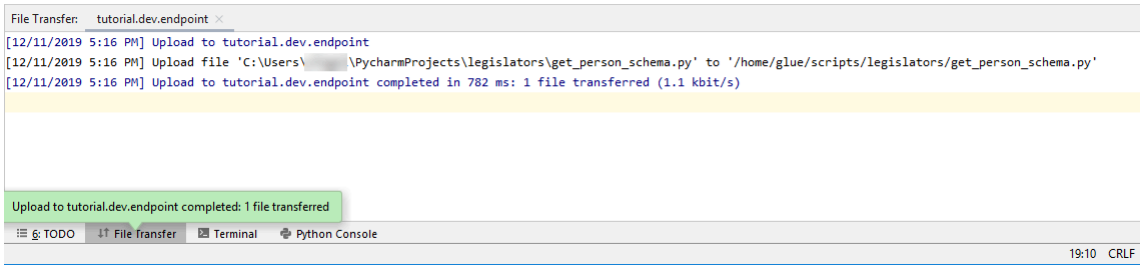
选择 OK (确定) 以关闭 Settings (设置) 对话框。

将脚本部署到您的开发终端节点

1. 选择 Tools (工具)、Deployment (部署)，然后选择要在其下设置开发终端节点的名称，如下图所示：



部署脚本后，屏幕底部显示的内容应类似于：



2. 在菜单栏上，选择 Tools (工具)、Deployment (部署)、Automatic Upload (always) (自动上传(始终))。确保 Automatic Upload (always) (自动上传(始终)) 旁边显示了复选标记。

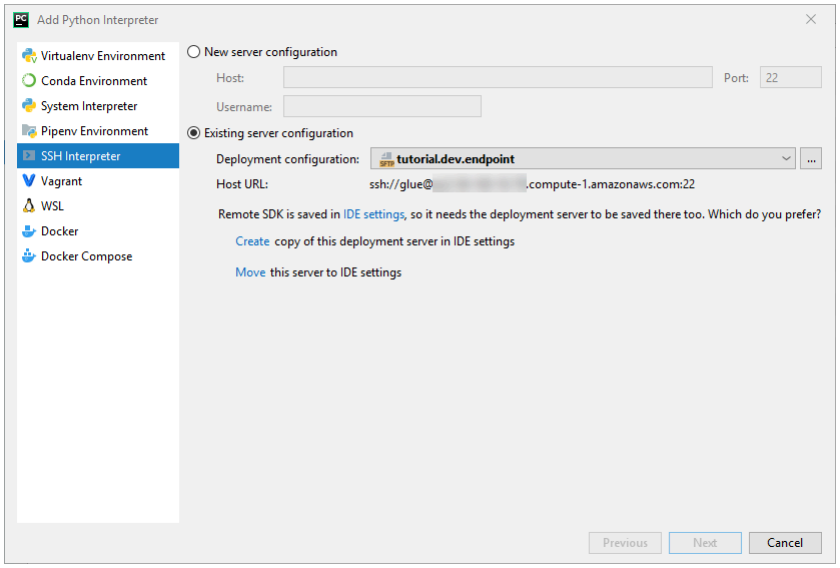
   启用此选项后，PyCharm 会自动将更改后的文件上传到开发终端节点。

配置远程解释器

将 PyCharm 配置为在开发终端节点上使用 Python 解释器。

1. 从 File (文件) 菜单中，选择 Settings (设置)。
2. 展开项目 legislators，然后选择 Project Interpreter (项目解释器)。
3. 选择 Project Interpreter (项目解释器) 列表旁边的齿轮图标，然后选择 Add (添加)。
4. 在 Add Python Interpreter (添加 Python 解释器) 对话框的左侧窗格中，选择 SSH Interpreter (SSH 解释器)。
5. 选择 Existing server configuration (现有服务器配置)，然后在 Deployment configuration (部署配置) 列表中，选择您的配置。

   您的屏幕上显示的内容应类似于以下图像。



6. 选择 Move this server to IDE settings (将此服务器移至 IDE 设置)，然后选择 Next (下一步)。
7. 在 Interpreter (解释器) 字段中，将路径更改为 `/usr/bin/gluepython`（如果您使用的是 Python 2）或 `/usr/bin/gluepython3`（如果您使用的是 Python 3）。然后选择完成。

在开发终端节点上运行您的脚本

要运行脚本，请执行以下操作：

- 在左侧窗格中，右键单击文件名，然后选择 Run '***<filename>***' (运行“<filename>”)。

  在一系列消息之后，最终输出应显示计数和架构。

```
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
```

现在，您已设置为在开发终端节点上远程调试脚本。

高级配置：在多个用户之间共享开发终端节点

本节介绍如何在典型使用案例中将开发终端节点与 SageMaker 笔记本结合使用，以在多个用户之间共享开发终端节点。

单一租户配置

在单一租户使用案例中，为了简化开发人员体验并避免争用资源，建议您让每个开发人员针对他们正在处理的项目使用自己的开发终端节点。这也简化了与工作线程类型和 DPU 计数相关的决策，让它们由开发人员和他们正在处理的项目自行决定。

除非同时运行多个笔记本文件，否则无需处理资源分配。如果您同时在多个笔记本文件中运行代码，将同时启动多个 Livy 会话。要隔离 Spark 集群配置以便同时运行多个 Livy 会话，您可以按照多租户使用案例中引入的步骤进行操作。

例如，如果您的开发终端节点有 10 个工作线程，并且工作线程类型为 `G.1X`，那么您将有 9 个 Spark 执行程序，整个集群将有 90G 的执行程序内存，因为每个执行程序都有 10G 的内存。

无论指定的工作线程类型如何，都将打开 Spark 动态资源分配。如果数据集足
够大，Spark 可能会将所有执行程序分配给单个 Livy 会话，因为默认情况下

`spark.dynamicAllocation.maxExecutors` 未设置。这意味着同一开发端点上的其他 Livy 会话将等待启动新的执行程序。如果数据集很小，Spark 将能够同时将执行程序分配给多个 Livy 会话。

> **Note**
>
> 有关如何在不同使用案例中分配资源，以及如何设置配置以修改行为的详细信息，请参阅 高级配置：在多个用户之间共享开发终端节点。

多租户配置

> **Note**
>
> 请注意，开发终端节点旨在模拟 AWS Glue ETL 环境作为单一租户环境。虽然多租户使用是可能的，但这是一个高级使用案例，建议大多数用户为每个开发终端节点维护单一租户模式。

在多租户使用案例中，您可能需要处理资源分配问题。关键因素是同时使用 Jupyter notebook 的并发用户的数量。如果您的团队在“follow-the-sun”工作流中工作，并且每个时区只有一个 Jupyter 用户，则并发用户的数量只有一个，因此您无需担心资源分配。但是，如果您的笔记本在多个用户之间共享，并且每个用户都以临时的方式提交代码，那么您需要考虑以下几点。

要在多个用户之间分区 Spark 集群资源，您可以使用 SparkMagic 配置。可以使用两种不同的方法来配置 SparkMagic。

（A）使用 %%configure -f 指令

如果要从笔记本中修改每个 Livy 会话的配置，可以在笔记本段落上运行 `%%configure -f` 指令。

例如，如果您要在 5 个执行程序上运行 Spark 应用程序，可以对笔记本段落运行以下命令。

```
%%configure -f
{"numExecutors":5}
```

然后，您将在 Spark UI 上看到只有 5 个执行程序正在运行此任务。

我们建议限制动态资源分配的最大执行程序数量。

```
%%configure -f
```

```
{"conf":{"spark.dynamicAllocation.maxExecutors":"5"}}
```

（B）修改 SparkMagic config 文件

SparkMagic 基于 Livy API 工作。SparkMagic 使用配置（如 driverMemory、driverCores、executorMemory、executorCores、 numExecutors、conf 等）创建 Livy 会话。这些是决定整个 Spark 集群消耗多少资源的关键因素。SparkMagic 允许您提供一个配置文件来指定那些被发送到 Livy 的参数。您可以在此 GitHub 存储库中看到一个示例配置文件。

如果要从笔记本修改所有 Livy 会话的配置，可以修改 /home/ec2-user/.sparkmagic/config.json 以添加 session_config。

要修改 SageMaker 笔记本实例上的配置文件，可以执行以下步骤。

1. 打开 SageMaker 笔记本。
2. 打开终端内核。
3. 运行以下命令：

```
sh-4.2$ cd .sparkmagic
sh-4.2$ ls
config.json logs
sh-4.2$ sudo vim config.json
```

例如，您可以将以下行添加到 /home/ec2-user/.sparkmagic/config.json 并从笔记本重新启动 Jupyter 内核。

```
  "session_configs": {
    "conf": {
      "spark.dynamicAllocation.maxExecutors":"5"
    }
  },
```

指南和最佳实践

为了避免这种资源冲突，可以使用以下基本方法：

- 具有更大的 Spark 集群，方法是增加 NumberOfWorkers（水平扩展）并升级 workerType（垂直扩展）
- 每个用户分配更少的资源（每个 Livy 会话分配更少的资源）

您的方法将取决于您的使用案例。如果您拥有较大的开发终端节点，并且没有大量数据，则资源冲突的可能性将显著降低，因为 Spark 可以根据动态分配策略分配资源。

如上所述，可以基于 DPU（或 NumberOfWorkers）和工作线重型类型的组合自动计算 Spark 执行程序的数量。每个 Spark 应用程序都会启动一个驱动程序和多个执行程序。要进行计算，您将需要 NumberOfWorkers = NumberOfExecutors + 1。下面的矩阵根据并发用户的数量说明了您的开发终端节点需要多少容量。

并发笔记本用户数	要为每个用户分配的 Spark 执行程序数量	您的开发端点的 NumberOfWorkers 总数
3	5	18
10	5	60
50	5	300

如果要为每个用户分配更少的资源，则 spark.dynamicAllocation.maxExecutors（或 numExecutors）将是配置为 Livy 会话参数的最简单参数。如果在 /home/ec2-user/.sparkmagic/config.json 中设置以下配置，那么 SparkMagic 将为每个 Livy 会话分配最多 5 个执行程序。这将有助于隔离每个 Livy 会话的资源。

```
"session_configs": {
    "conf": {
      "spark.dynamicAllocation.maxExecutors":"5"
    }
  },
```

假设有一个具有 18 个工作线程（G.1X）的开发端点，并且同时有 3 个并发笔记本用户。如果您的会话配置具有 spark.dynamicAllocation.maxExecutors=5，那么每个用户可以使用 1 个驱动程序和 5 个执行程序。即使您同时运行多个笔记本段落，也不会发生任何资源冲突。

权衡措施

使用此会话配置 "spark.dynamicAllocation.maxExecutors":"5"，您将能够避免资源冲突错误，并且在存在并发用户访问时不需要等待资源分配。但是，即使有许多可用资源（例如，没有其他并发用户），Spark 也不能为您的 Livy 会话分配 5 个以上的执行程序。

其他说明

停止使用笔记本时，最好停止 Jupyter 内核。这将释放资源，其他笔记本用户可以立即使用这些资源，而无需等待内核过期（自动关闭）。

常见问题

即使遵循指南，您也可能会遇到某些问题。

未找到会话

当您尝试运行笔记本段落时，即使您的 Livy 会话已经终止，也将看到以下消息。要激活 Livy 会话，您需要重启 Jupyter 内核，方法是在 Jupyter 菜单中，选择 Kernel (内核) > Restart (重新启动)，然后再次运行笔记本段落。

```
An error was encountered:
Invalid status code '404' from http://localhost:8998/sessions/13 with error payload:
 "Session '13' not found."
```

YARN 资源不足

当您尝试运行笔记本段落，即使您的 Spark 集群没有足够的资源来启动新的 Livy 会话，您也将看到以下消息。您通常可以通过遵循指南来避免此问题，但是，您可能会遇到此问题。要解决此问题，你可以检查是否有任何不需要的活跃 Livy 会话。如果存在不需要的 Livy 会话，则需要终止这些会话才能释放集群资源。有关详细信息，请参阅下一节。

```
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.
```

监控和调试

本节介绍监控资源和会话的技术。

监控和调试集群资源分配

您可以观看 Spark UI 如何监控每个 Livy 会话分配的资源数量，以及任务上的有效 Spark 配置内容。要激活 Spark UI，请参阅为开发终端节点启用 Apache Spark Web UI。

（可选）如果需要实时查看 Spark UI，可以针对 Spark 集群上运行的 Spark 历史记录服务器配置 SSH 隧道。

```
ssh -i <private-key.pem> -N -L 8157:<development endpoint public address>:18080
 glue@<development endpoint public address>
```

然后您可以在浏览器中打开 http://localhost:8157 以查看 Spark UI。

免费的不需要的 Livy 会话

查看这些过程以关闭笔记本或 Spark 集群中的任何不需要的 Livy 会话。

（a）。从笔记本终止 Livy 会话

您可以关闭 Jupyter notebook 上的内核以终止不需要的 Livy 会话。

（b）。从 Spark 集群终止 Livy 会话

如果仍在运行不需要的 Livy 会话，您可以关闭 Spark 集群上的 Livy 会话。

作为执行此过程的先决条件，您需要为开发终端节点配置 SSH 公钥。

要登录到 Spark 集群，您可以运行以下命令：

```
$ ssh -i <private-key.pem> glue@<development endpoint public address>
```

您可以运行以下命令以查看活跃的 Livy 会话：

```
$ 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
```

然后，您可以使用以下命令关闭 Livy 会话：

```
$ 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
```

管理笔记本

> **Note**
>
> 只有 2.0 之前的 AWS Glue 版本支持开发端点。要获得可以创作和测试 ETL 脚本的交互式环境，请使用 AWS GlueStudio 上的笔记本。

通过笔记本可在开发端点上交互式开发和测试您的 ETL（提取、转换、加载）脚本。AWS Glue 为 SageMaker Jupyter Notebook 提供了一个接口。借助 AWS Glue，您可以创建和管理 SageMaker 笔记本。您还可以从 AWS Glue 控制台中打开 SageMaker 笔记本。

此外，您还可以在支持 SageMaker（但不支持 AWS Glue ETL 任务）的 AWS Glue 开发终端节点上将 Apache Spark 与 SageMaker 结合使用。SageMaker Spark 是用于 SageMaker 的开源 Apache Spark 库。有关更多信息，请参阅将 Apache Spark 与 Amazon SageMaker 结合使用。

> **Important**
>
> 在以下 AWS 区域中，可以使用 AWS Glue 开发终端节点管理 SageMaker 笔记本：

区域	代码
美国东部（俄亥俄）	us-east-2
美国东部（弗吉尼亚州北部）	us-east-1

区域	代码
美国西部（北加利福尼亚）	us-west-1
美国西部（俄勒冈）	us-west-2
亚太地区（东京）	ap-northeast-1
亚太地区（首尔）	ap-northeast-2
亚太地区（孟买）	ap-south-1
亚太地区（新加坡）	ap-southeast-1
亚太地区（悉尼）	ap-southeast-2
加拿大（中部）	ca-central-1
欧洲地区（法兰克福）	eu-central-1
欧洲地区（爱尔兰）	eu-west-1
欧洲地区（伦敦）	eu-west-2

使用 AWS Glue Studio 构建可视化 ETL 作业

AWS Glue 作业将封装连接到源数据的脚本，处理该脚本，然后将其写入数据目标。通常，作业运行提取、转换和加载 (ETL) 脚本。作业可以运行专为 Apache Spark 和 Ray 运行时环境设计的脚本。作业还可以运行通用 Python 脚本（Python shell 作业）。AWS Glue 触发器 可以根据计划或事件或者按需启动作业。您可以监控作业运行以了解运行时指标（例如完成状态、持续时间和开始时间）。

您可以使用 AWS Glue 生成的脚本，也可以提供您自己的脚本。借助源架构和目标位置或架构，AWS Glue Studio 代码生成器可以自动创建 Apache Spark API（PySpark）脚本。您可以将此脚本用作起点，并对其进行编辑以满足您的目标。

AWS Glue 可以用多种数据格式写入输出文件。每种作业类型可能支持不同的输出格式。对于某些数据格式，可以编写常见的压缩格式。

登录到 AWS Glue 控制台

AWS Glue 中的作业包含执行提取、转换和加载 (ETL) 工作的业务逻辑。您可以在 控制台的 ETLAWS Glue 部分中创建作业。

要查看现有任务，请登录 AWS Management Console，然后通过以下网址打开 AWS Glue 控制台：https://console.aws.amazon.com/glue/。然后在 中选择 JobsAWS Glue (作业) 选项卡。Jobs (作业) 列表显示与每个作业关联的脚本的位置、上次修改作业的时间和当前作业书签选项。

创建新任务时或保存任务后，您可以使用 AWS Glue Studio 修改您的 ETL 任务。您可以在可视编辑器中编辑节点或以开发人员模式编辑任务脚本，从而执行此操作。您还可以在可视编辑器中添加和删除节点，以创建更复杂的 ETL 任务。

在 AWS Glue Studio 中创建任务的后续步骤

您使用可视化任务编辑器为任务配置节点。每个节点表示一项操作，例如从源位置读取数据或者为数据应用转换。您添加到任务的每个节点都具有相关属性，提供有关数据位置或转换的信息。

创建和管理任务的后续步骤包括：

- 带有 AWS Glue Studio 的 Visual ETL
- 查看任务脚本

- 修改任务属性
- 保存任务
- 启动任务运行
- 查看最近任务运行的信息
- 访问任务监控控制面板

带有 AWS Glue Studio 的 Visual ETL

您可以使用 AWS Glue Studio 中的简单视觉界面，创建您的 ETL 任务。您使用 Jobs (任务) 页面创建新任务。您还可以使用脚本编辑器或笔记本直接处理 AWS Glue Studio ETL 任务脚本中的代码。

在 Jobs (任务) 页面上，您可以看到您使用 AWS Glue Studio 或 AWS Glue 创建的所有任务。您可以在此页面上查看、管理和运行您的任务。

另请参阅博客教程中的另一个示例，说明了如何使用 AWS Glue Studio 创建 ETL作业。

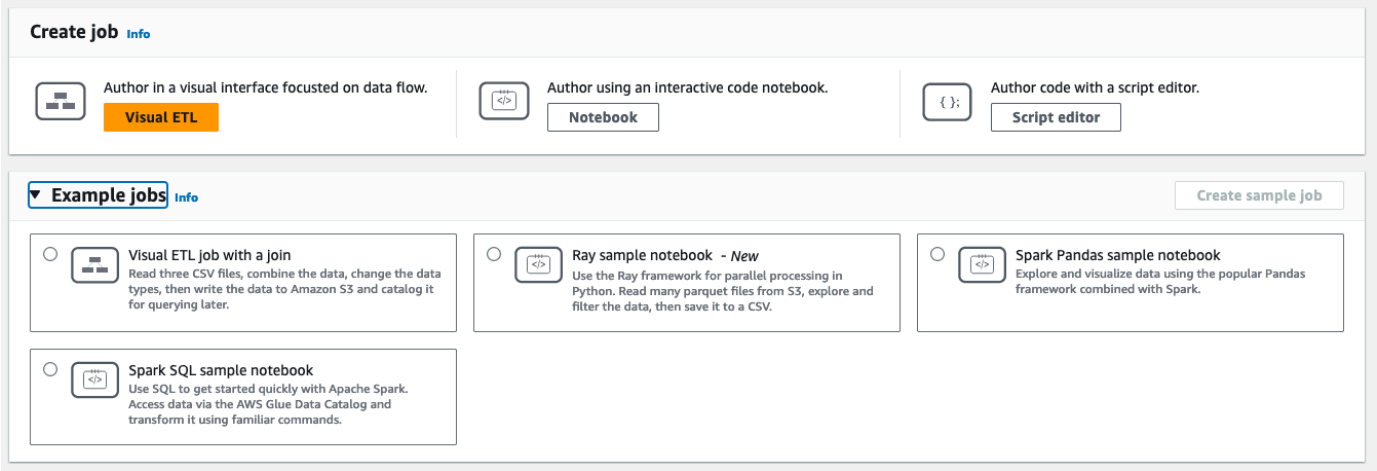
在 AWS Glue Studio 中启动作业

AWS Glue 允许您通过可视化界面、交互式代码笔记本或脚本编辑器创建作业。您可以通过单击任一选项来启动作业，也可以根据示例作业创建新作业。

示例作业使用您选择的工具创建作业。例如，示例作业允许您创建将 CSV 文件联接到目录表中的可视化 ETL 作业，使用 pandas 时在互式代码笔记本中使用 AWS Glue for Ray 或 AWS Glue for Spark 创建作业，或者使用 SparkSQL 在交互式代码笔记本中创建作业。

在 AWS Glue Studio 中从头开始创建作业

1. 登录 AWS Management Console 并打开AWS Glue Studio控制台，网址为 https://console.aws.amazon.com/gluestudio/。
2. 在导航窗格中，选择 ETL 作业。
3. 在创建作业部分中，为您的作业选择一个配置选项。



用于从头开始创建作业的选项：

- Visual ETL - 以数据流为重点的可视化界面中编写
- 使用交互代码笔记本编写 - 基于 Jupyter Notebooks 的笔记本界面中以交互方式编写作业

  选择此选项后，在创建笔记本创作会话之前，必须提供附加信息。有关如何指定此信息的详细信息，请参阅 在 AWS Glue Studio 中开启笔记本。
- 使用脚本编辑器编写代码 – 对于熟悉编程和编写 ETL 脚本的用户，请选择此选项，创建新的 Spark ETL 任务。选择引擎（Python shell、Ray、Spark（Python）或 Spark（Scala））。然后，选择重新开始或上传脚本，从本地文件上传现有脚本。如果您选择使用脚本编辑器，则无法使用可视化任务编辑器来设计或编辑任务。

  Spark 任务会在由 AWS Glue 托管的 Apache Spark 环境中执行。默认情况下，新脚本以 Python 编码。要编写新的 Scala 脚本，请参阅在 AWS Glue Studio 中创建和编辑 Scala 脚本。

在 AWS Glue Studio 中利用示例作业创建作业

您可以选择从示例作业创建作业。在示例作业部分，选择一个示例作业，然后选择创建示例作业。使用其中一个选项创建示例作业提供了一个可供您使用的快速模板。

1. 登录 AWS Management Console 并打开AWS Glue Studio控制台，网址为 https://console.aws.amazon.com/gluestudio/。
2. 在导航窗格中，选择 ETL 作业。
3. 选择一个选项，从示例作业创建作业：

- 用于联接多个源的 Visual ETL 作业 - 读取三个 CSV 文件，合并数据，更改数据类型，然后将数据写入 Amazon S3 并对其进行编目以供日后查询。
- 使用 Pandas 的 Spark 笔记本 - 使用广受欢迎的 Pandas 框架与 Spark 相结合，探索和可视化数据。
- 使用 SQL 的 Spark 笔记本 - 使用 SQL 快速开始使用 Apache Spark。通过 AWS Glue Data Catalog 访问数据，并使用熟悉的命令对其进行转换。

4. 选择创建示例作业。

任务编辑器功能

任务编辑器提供以下功能，用于创建和编辑任务。

- 任务的可视图，每个任务任务都有一个节点：用于读取数据的数据源节点；用于修改数据的转换节点；用于写入数据的数据目标节点。

  您可以查看和配置任务图中每个节点的属性。您还可以查看任务图中每个节点的架构和示例数据。这些功能可帮助您验证任务是否正在以正确的方式修改和转换数据，而无需运行任务。
- Script viewing and editing (脚本查看和编辑) 选项卡，您可以在其中修改为任务生成的代码。
- Job details (任务详细信息) 选项卡，您可以在其中配置各种设置，自定义 AWS Glue ETL 任务的运行环境。
- Runs (运行) 选项卡，您可以在其中查看任务的当前运行和上一次运行，查看任务运行的状态，以及访问任务运行的日志。
- “数据质量”选项卡，您可以在其中将数据质量规则应用于您的作业。
- Schedules (计划) 选项卡，您可以在其中配置任务的开始时间，或设置定期任务运行。
- “版本控制”选项卡，您可以在其中配置 Git 服务以用于您的作业。

在可视任务编辑器中使用架构预览

创建或编辑任务时，您可以使用 Output schema (输出架构) 选项卡查看数据的架构。

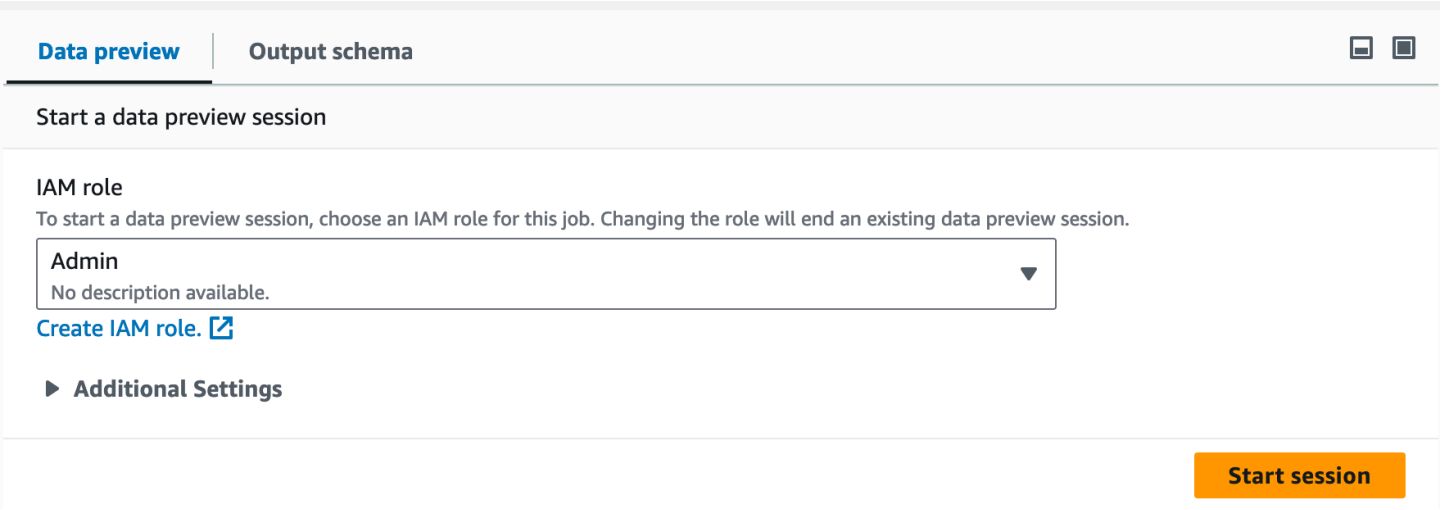
在查看架构之前，任务编辑器需要数据源的访问权限。您可以在编辑器的 Job details (任务详细信息) 选项卡上或者节点的 Output schema (输出架构) 选项卡上指定 IAM 角色。如果 IAM 角色具有数据源的所有必要访问权限，您可以在节点的 Output schema (输出架构) 选项卡上查看架构。

在可视任务编辑器中使用数据预览

数据预览有助于您使用数据样本来创建和测试任务，而无需重复运行作业。借助数据预览，您可以：

- 测试 IAM 角色以确保您有权访问您的数据来源或数据目标。
- 检查转换功能是否在以预期的方式修改数据。例如，如果您使用筛选条件转换，则可以确保筛选条件正在选择合适的数据子集。
- 检查数据。如果数据集包含具有多种类型值的列，则数据预览会显示这些列的元组列表。每个元组都包含数据类型及其值。

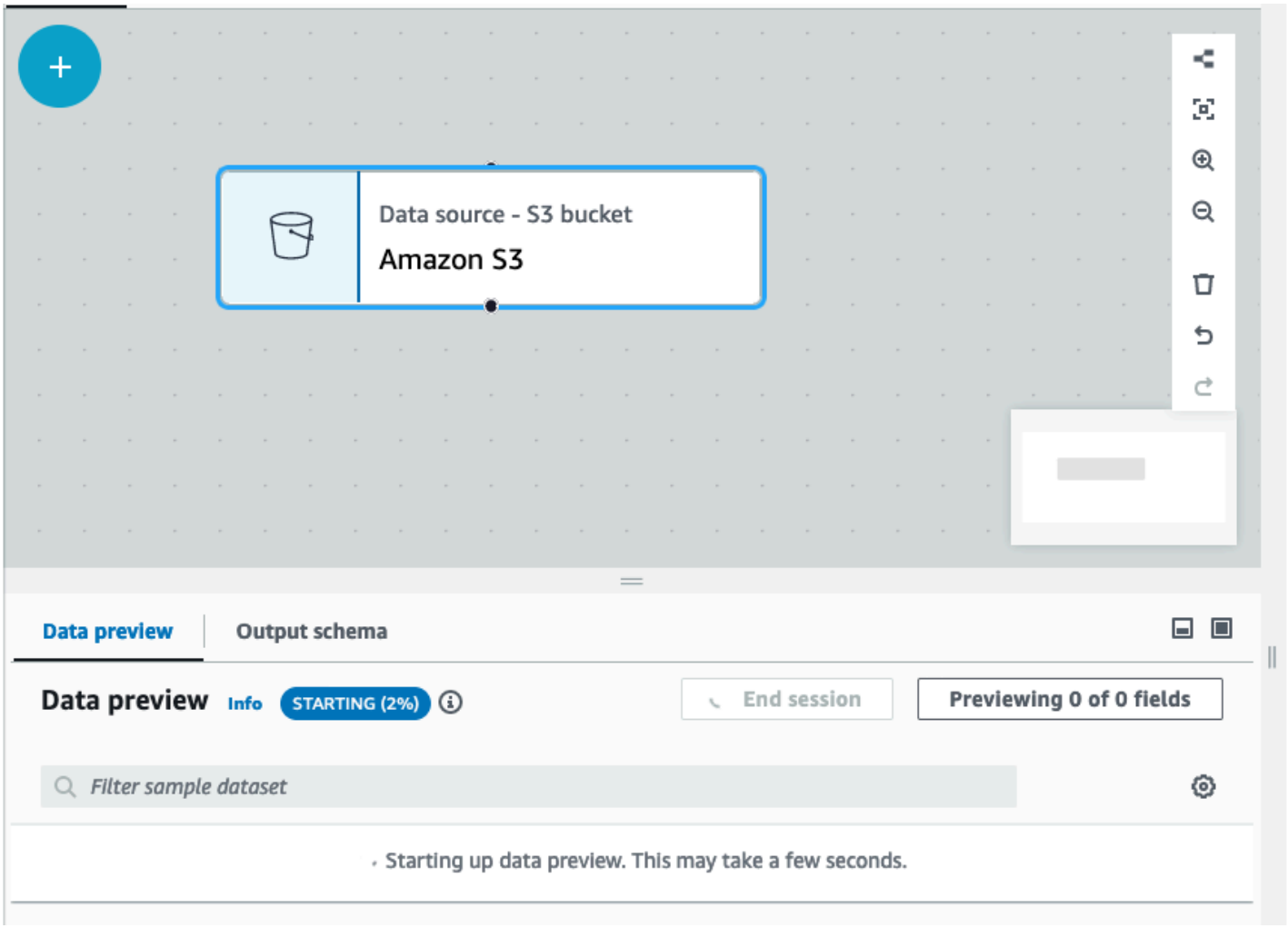
创建或编辑作业时，您可以使用作业画布下方的数据预览选项卡查看数据示例。如果作业中已经配置了该角色或账户中已设置了默认的 IAM 角色，则将自动启动新的数据预览会话。如果之前未配置过任何角色，则可以选择该角色以启动会话。



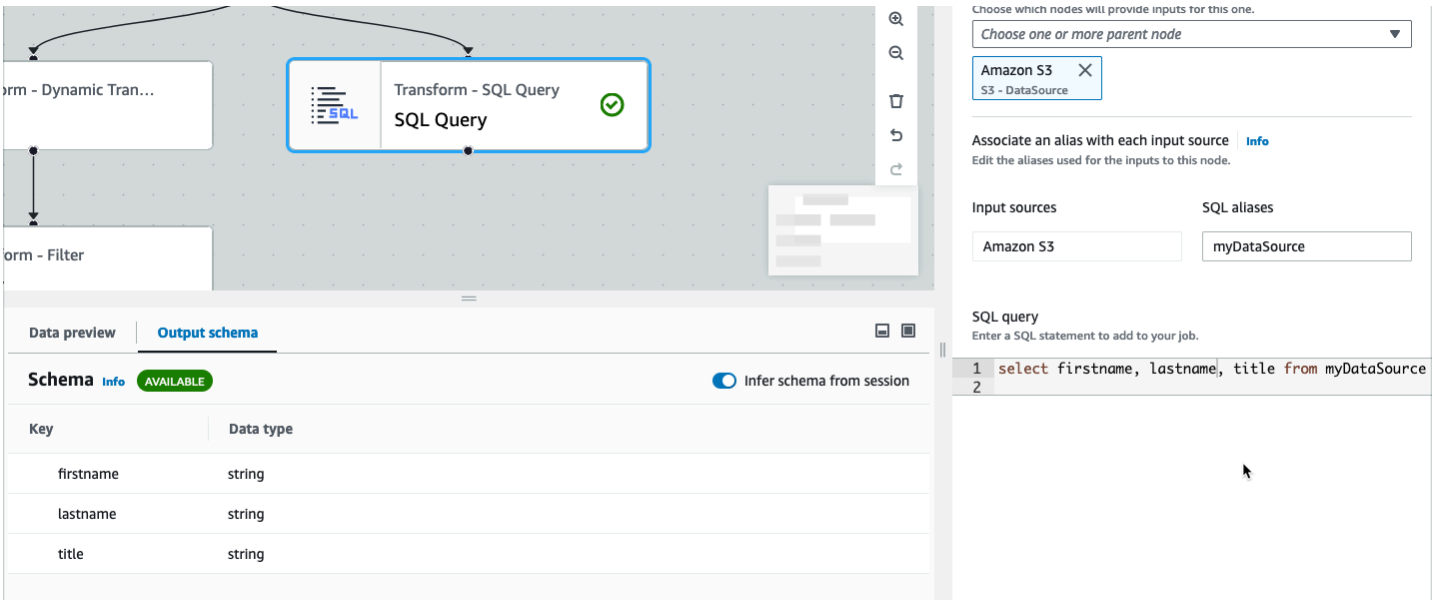
> **Note**
>
> 您为数据预览会话选择的角色也将用于该作业。

您可以单击信息图标来查看会话的状态和进度，以及会话的详细信息。

会话准备就绪后，AWS Glue Studio 将加载所选节点的数据。您可以在加载过程中查看完成百分比。



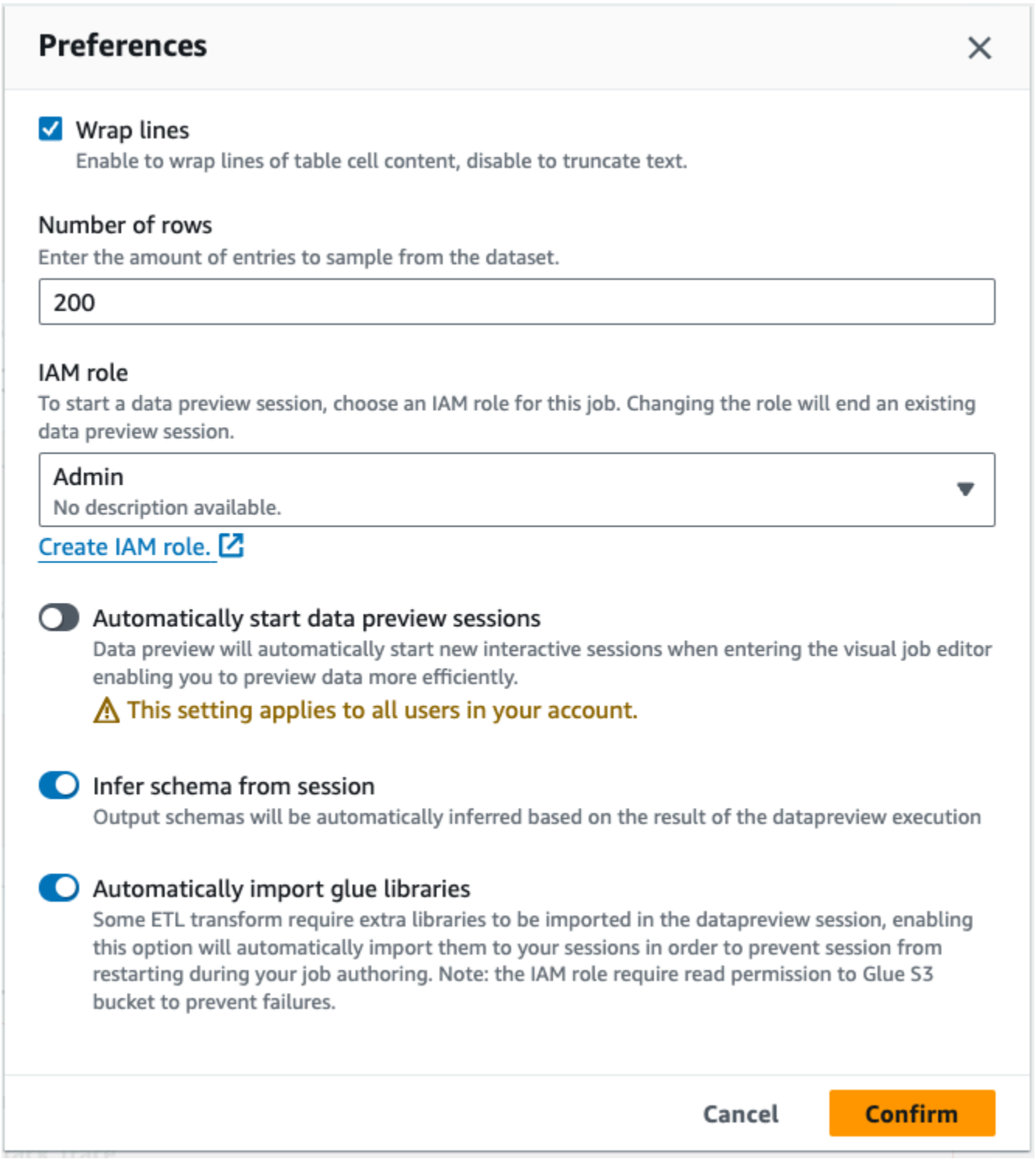
在编写可视化作业过程中，当您在输出 Schema 选项卡中切换从会话推断 Schema 时，AWS Glue Studio 将自动更新所选节点的 Schema。



配置数据预览首选项：

选择设置图标（齿轮符号）以配置数据预览的首选项。这些设置适用于任务图中的所有节点。您可以：

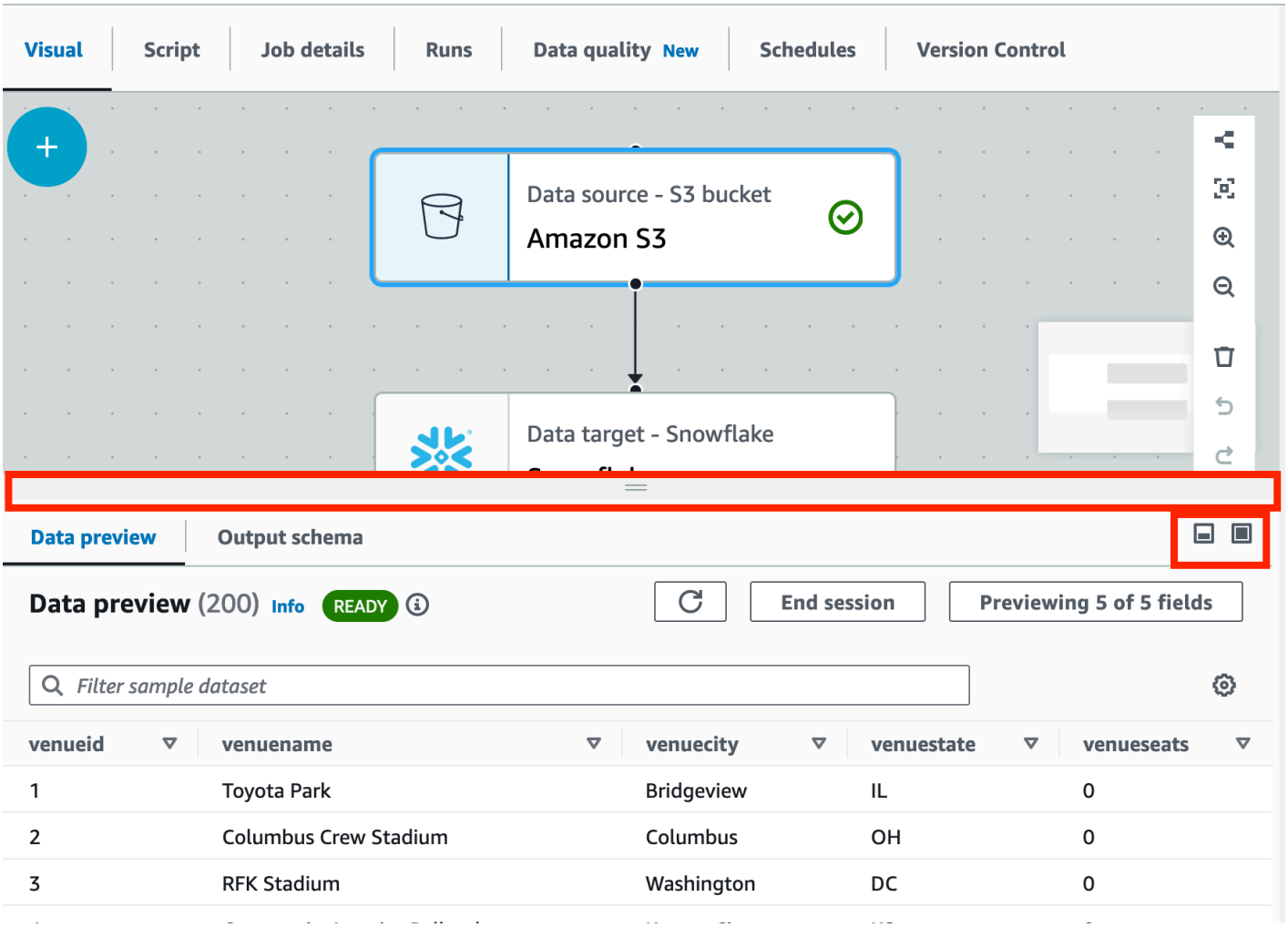
- 选择此选项以将文本换行。此选项将默认启用
- 更改行数（默认为 200）
- 选择一个 IAM 角色或根据需要创建一个 IAM 角色
- 选择此选项以在编写作业时自动启动新会话。这将在编写作业时预置新的交互式会话。此设置适用于账户级别。设置完成后，在编辑任何作业时，它将适用于账户中的所有用户。
- 选择自动推断 Schema。系统将自动推断所选节点的输出 Schema
- 选择此选项以自动导入 AWS Glue 库。此功能非常实用，因为可以在添加需要重新启动会话的新转换时防止数据预览重新启动新会话



其他功能包括：

- 选择 Previewing x of y fields (预览 y 个字段中的 x 个字段) 按钮，选择要预览的列（字段）。您使用默认设置预览数据时，任务编辑器会显示数据集的前 5 列。您可以更改此选项以显示全部或不显示（不推荐）。

- 水平和垂直滚动浏览数据预览窗口。
- 使用最大化按钮展开“数据预览”选项卡展以叠加任务图，以便更好地查看数据和数据结构。同样，使用最小化按钮最小化“数据预览”选项卡。您也可以抓住手柄窗格并向上拖动以展开数据预览选项卡。



- 使用结束会话停止数据预览。停止会话后，您可以选择新的 IAM 角色，并设置其他设置（例如开启或关闭设置）以自动启动新会话、推断架构或导入 AWS Glue 库，然后重新启动会话。

使用数据预览时的限制

使用数据预览时，您可能会遇到以下限制。

- 您首次选择 Data preview (数据预览) 选项卡时，您必须选择 IAM 角色。此角色必须有权访问创建数据预览所需的数据和其他资源。
- 提供 IAM 角色后，需要一段时间才能查看数据。对于数据少于 1GB 的数据集，可能最多需要一分钟。如果您拥有较大的数据集，则应使用分区来缩短加载时间。直接从 Amazon S3 加载数据可以实现更出色的性能。

- 如果您拥有非常大的数据集，并且查询用于数据预览的数据需要 15 分钟以上，则请求将超时。数据预览有 30 分钟的空闲超时。要缓解此情况，请减小数据集大小以使用数据预览。
- 默认情况下，您可以查看“数据预览”选项卡中的前 50 列。如果列没有数据值，您将收到一条消息，指明没有要显示的数据。您可以增加采样的行数，也可以选择不同的列来查看数据值。
- 数据预览当前不支持流式处理数据源或使用自定义连接器的数据源。
- 一个节点上的错误会影响整个任务。如果任何一个节点在数据预览中出现错误，则该错误将在所有节点上显示，直到您更正。
- 如果更改任务的数据源，您可能需要更新该数据源的子节点以匹配新架构。例如，如果您有用于修改列的 ApplyMapping 节点，并且该列不存在于替换数据源中，您需要更新 ApplyMapture 转换节点。
- 如果您查看 SQL 查询转换节点的 Data preview (数据预览) 选项卡，并且 SQL 查询使用不正确的字段名称，则 Data preview (数据预览) 选项卡将显示错误。

脚本代码生成

使用可视化编辑器创建任务时，将自动为您生成 ETL 代码。AWS Glue Studio 会创建功能完整的任务脚本，并将其保存在 Amazon S3 位置。

AWS Glue Studio 生成的代码有两种形式：原始版本，即经典版本，以及更新的精简版本。预设情况下，会使用新的代码生成器创建任务脚本。选择 Generate classic script（生成经典脚本）切换按钮，就可以使用 Script（脚本）选项卡上的经典代码生成器生成任务脚本。

生成的新版本代码中的一些差异包括：

- 大型的注释数据块将不再添加到脚本中
- 代码中的输出结构将使用您在可视化编辑器中指定的节点名称。在类脚本中，输出结构将简单地命名为 `DataSource0`、`DataSource1`、`Transform0`、`Transform1`、`DataSink0`、`DataSink1` 等。
- 长命令将拆分为多行，以免滚动页面才能查看整条命令。

AWS Glue Studio 中的新功能需要新版本的代码生成，并且不适用于经典代码脚本。尝试运行这些任务时，系统会提示您更新。

编辑 AWS Glue 托管数据转换节点

AWS Glue Studio 提供了两种类型的转换：

- AWS Glue-原生转换 — 适用于所有用户并由 AWS Glue 管理。

- 自定义视觉转换 — 允许您上传自己的转换以在 AWS Glue Studio 中使用

AWS Glue 托管数据转换节点

AWS Glue Studio 提供一组内置转换，可用于处理数据。您的数据从任务图中的一个节点传递到名为 DynamicFrame 的数据结构（这是 Apache Spark SQL DataFrame 的扩展）中的另一个节点。

在作业的预填充图中，数据来源节点和数据目标节点之间是更改架构转换节点。您可以将此转换节点配置为修改数据，也可以使用其他转换。

以下内置转换适用于 AWS Glue Studio：

- ChangeSchema：将数据来源中的数据属性键映射到数据目标中的数据属性键。您可以重命名键、修改键的数据类型以及选择要从数据集中删除的键。
- SelectFields：选择要保留的数据属性键。
- DropFields：选择要删除的数据属性键。
- RenameField：重命名单个数据属性键。
- Spigot：将数据样本写入 Amazon S3 存储桶。
- Join：使用指定数据属性键上的比较短语将两个数据集联接到一个数据集。您可以使用内部、外部、左、右、左半和左反联接。
- 联合：合并多个数据来源中具有相同架构的行。
- SplitFields：将数据属性键拆分为两个 DynamicFrames。输出是 DynamicFrames 集合：一个具有选定的数据属性键，另一个具有剩余的数据属性键。
- SelectFromCollection：请从 DynamicFrames 集合中选择一个 DynamicFrame。输出是选定的 DynamicFrame。
- FillMissingValues：查找数据集中缺少值的记录，并添加包含由输入决定的建议值的新字段
- Filter：根据筛选条件将数据集拆分为两个。
- 删除 Null 字段：如果列中的所有值都为“null”，则从数据集中移除这些列。
- 删除重复项：通过选择匹配整行或指定键，从数据来源中删除行。
- SQL：在文本输入字段中输入 SparkSQL 代码以使用 SQL 查询转换数据。输出为单个 DynamicFrame。
- 聚合：对所选字段和行执行计算（例如平均值、总和、最小值、最大值），并使用新计算的值创建新字段。
- 扁平化：将结构内的字段提取到顶级字段中。

- UUID：为每行添加一个带有通用唯一标识符的列。
- 标识符：为每行添加一个带有数字标识符的列。
- 到时间戳：将列转换为时间戳类型。
- 格式化时间戳：将时间戳列转换为格式化字符串。
- 条件路由器转换：对传入数据应用多个条件。传入数据的每一行都通过一组筛选条件进行评估，然后处理到相应的组中。
- 串联列转换：使用带有可选间隔符的其他列的值来生成新的字符串列。
- 拆分字符串转换：使用正则表达式将字符串分解为令牌数组，以定义拆分的完成方式。
- 数组转列转换：将数组类型的列的部分或全部元素提取到新列中。
- 添加当前时间戳转换：用数据处理时间标记行。这对于审计目的或跟踪数据管道中的延迟非常有用。
- 将行转置为列转换：通过旋转选定列上的唯一值来聚合数字列，这些列会变成新列。如果选择了多列，则将这些值串联起来命名新列。
- 反转置列为行转换：将列转换为新列的值，为每个唯一值生成一行。
- 自动平衡处理转换：更好地在工作线程之间重新分配数据。当数据不平衡或数据来自源时不允许对其进行足够的并行处理时，这很有用。
- 派生列转换：根据数学公式或 SQL 表达式定义一个新列，您可以在其中使用数据中的其他列以及常量和文字。
- 查找转换：当键与数据中定义的查找列匹配时，从定义的目录表中添加列。
- 分解数组或映射到行转换：将嵌套结构中的值提取到更易于操作的单个行中。
- 记录匹配转换：调用现有的记录匹配机器学习数据分类转换。
- 移除空行转换：从数据集中移除所有列均为空（null）或空（empty）的行。
- 解析 JSON 列转换：解析包含 JSON 数据的字符串列并将其转换为结构或数组列，具体取决于 JSON 是对象还是数组。
- 提取 JSON 路径转换：从 JSON 字符串列中提取新列。
- 从正则表达式中提取字符串片段：使用正则表达式提取字符串片段并从中创建新列，如果使用正则表达式组则创建多列。
- Custom transform：在文本输入字段中输入代码以使用自定义转换。输出是 `DynamicFrames` 的集合。

在 AWS Glue Studio 中使用数据准备配方

AWS Glue Studio 允许您在可视化工作流程中使用 AWS Glue DataBrew 配方。这允许客户的 AWS Glue DataBrew 配方与其他 AWS Glue Studio 节点一起在 AWS Glue 作业中运行。

在 DataBrew 中，配方是一组数据转换步骤。DataBrew 配方规定了如何转换已读取的数据，但没有描述在何处如何读取数据，以及如何在何处写入数据。这是在 AWS Glue Studio 的“源节点”和“目标节点”中配置。有关配方的更多信息，请参阅 Creating and using AWS Glue DataBrew recipes。

数据准备配方节点可从“资源”面板中找到。您可以将数据准备配方节点连接到可视化工作流程中的另一个节点，无论它是数据来源节点还是其他转换节点。选择 AWS Glue DataBrew 配方和版本后，配方中应用的步骤将显示在节点属性选项卡中。

先决条件

- 您已经在 AWS Glue DataBrew 中创建了一个 AWS Glue DataBrew 配方。
- 如以下部分所述，您拥有所需的 IAM 权限。

AWS Glue DataBrew 的 IAM 权限

本主题提供的信息可帮助您了解 IAM 管理员了解可以在数据准备配方转换的 AWS Identity and Access Management（IAM）policy 中使用的操作和资源。

有关 AWS Glue 中的其他安全信息，请参阅 Access Management。

下表列出了用户执行特定操作以使用数据准备配方转换所需的权限。

数据准备配方转换操作

操作	描述
`databrew:ListRecipes`	授予检索 AWS Glue DataBrew 配方的权限。
`databrew:ListRecipeVersions`	授予检索 AWS Glue DataBrew 配方版本的权限。
`databrew:DescribeRecipe`	授予检索 AWS Glue DataBrew 配方描述的权限。

您用于访问此功能的角色应具有允许多个 AWS Glue DataBrew 的策略。您可以通过使用包含必要操作的 AWSGlueConsoleFullAccess 策略或在角色中添加以下内联策略来实现此目的：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "databrew:ListRecipes",
                "databrew:ListRecipeVersions",
                "databrew:DescribeRecipe"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

要使用数据准备配方转换，必须将 IAM:PassRole 操作添加到权限策略中。

其他必需的权限

操作	描述
iam:PassRole	向 IAM 授予权限，允许用户传递已批准的角色。

如果没有这些权限，则会发生以下错误：

```
"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

- 并非所有 AWS Glue DataBrew 配方都受 AWS Glue 支持。有些配方将无法在 AWS Glue Studio 中运行。
  - 不支持带有 UNION 和 JOIN 转换的配方，但是，AWS Glue Studio 已经有“联接”和“联合”转换节点，可以在数据准备配方节点之前或之后使用。
- 从 AWS Glue 4.0 版开始的作业支持数据准备配方节点。在作业中添加数据准备配方节点后，将自动选择此版本。
- 数据准备配方节点需要 Python。当向作业中添加数据准备配方节点时，会自动设置此值。
- 使用数据预览时，您需要在作业中添加数据准备配方节点后重新启动数据预览会话。

如何在 AWS Glue Studio 中使用 AWS Glue DataBrew 配方

要在 AWS Glue Studio 中使用 AWS Glue DataBrew 配方，请先在 AWS Glue DataBrew 中创建配方。如果您已有要使用的配方，则可跳过这一步。

要在 AWS Glue DataBrew 中创建 AWS Glue DataBrew 配方，请执行以下操作：

1. 在 AWS Glue DataBrew 中编写配方。有关更多信息，请参阅 Getting started with AWS Glue DataBrew。
2. 保存您的配方。
3. 发布您的配方。这将把您的配方发布为版本 1.0。

要在 AWS Glue Studio 中使用数据准备配方节点，请执行以下操作：

您可以在可视化 ETL 作业中使用多个数据准备配方节点。为此，请按照以下步骤添加一个数据准备配方节点，然后向该作业添加另一个数据准备配方节点。例如，工作流程可能遵循以下模式：

- 数据来源 1 > 配方 1 > 输出 1
- 数据来源 2 > 配方 2 > 输出 2
- 输出 1，输出 2 > 联接

1. 使用数据来源在 AWS Glue Studio 中启动 AWS Glue 作业。
2. 将数据准备配方节点添加到您的数据来源。
3. 在搜索字段中键入配方名称，按名称筛选配方。

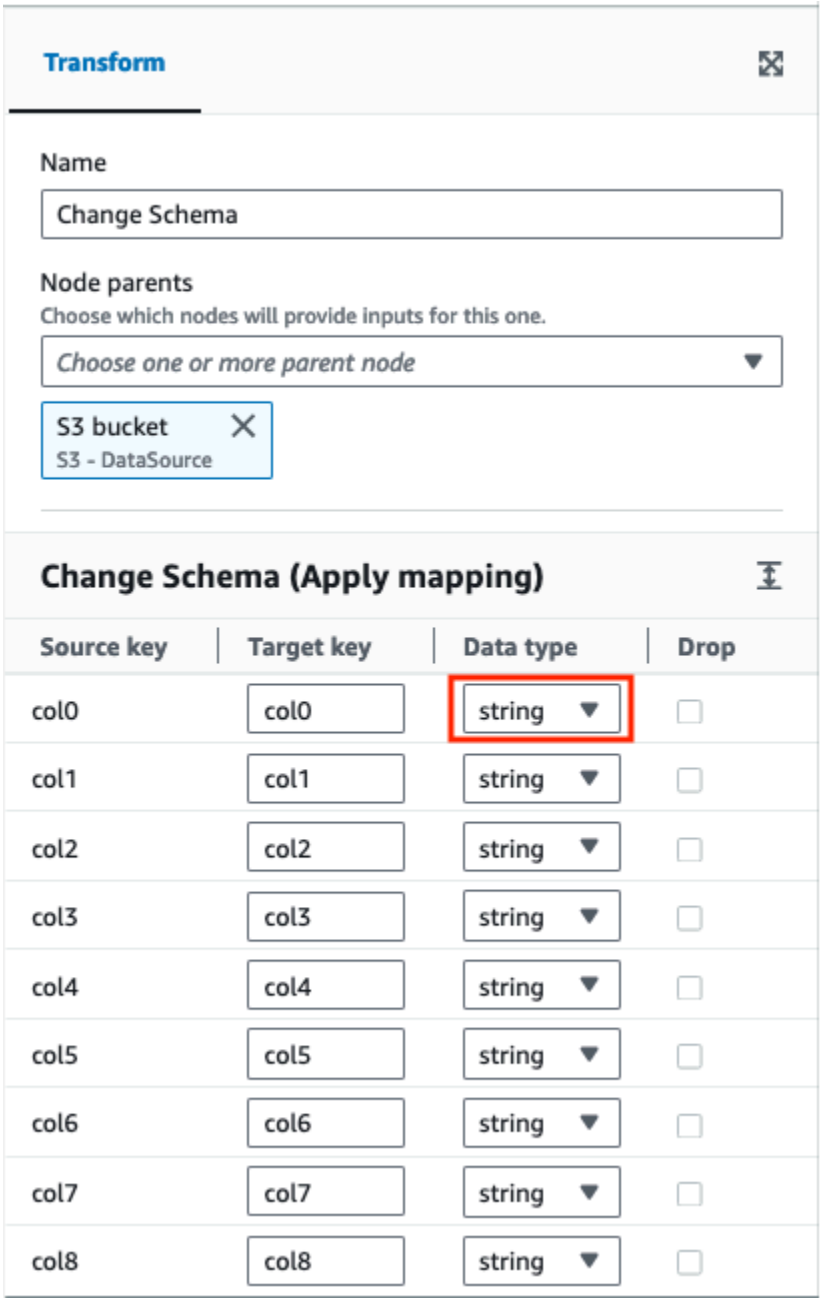
4. 选择已发布的版本。只有已发布的版本可用。
5. 根据需要添加其他转换节点，然后添加数据目标节点以保存作业输出，从而完成作业的编写。
6. 在作业详细信息选项卡中进行必要的配置更改，例如根据需要命名作业和调整分配的容量，然后保存作业。
7. 从操作下拉菜单中选择运行来运行作业。

如果数据来源为 Amazon S3 且数据格式为 CSV，则要更改架构，请执行以下操作：

如果 CSV 文件中的所有列最初都是作为 AWS Glue Studio 中的字符串数据类型加载的，则需要确保该列的数据类型与 AWS Glue DataBrew 配方中的其余步骤兼容。

AWS Glue DataBrew 配方仅规定如何转换已读取的数据。不描述在何处如何读取数据。

1. 在多步骤配方节点之前添加一个更改架构节点。
2. 单击更改架构节点，然后根据需要在列的转换中选择新的数据类型，将架构更改为与 AWS Glue DataBrew 中的列数据类型相同。



要在数据来源为无标头时更改架构，请执行以下操作：

AWS Glue DataBrew 配方仅规定如何转换已读取的数据。不描述在何处如何读取数据。

在 AWS Glue Studio 中加载无标题数据集时，默认标题名称与在 AWS Glue DataBrew 中加载的标题名称不同。

1. 在 ETL 作业中，在数据准备配方节点之前添加一个更改架构节点。

2. 选择更改架构节点，将列名更改为与 AWS Glue DataBrew 配方中使用的相同名称。

使用更改架构重新映射数据属性键

更改架构转换将源数据属性键重新映射到目标数据所需的配置。在“更改架构”转换节点中，您可以：

- 更改多个数据属性键的名称。
- 如果支持新数据类型并且两种数据类型之间存在转换路径，则更改数据属性键的数据类型。
- 指示要删除的数据属性键，以选择数据属性键的子集。

您还可以根据需要向作业图中添加其他更改架构节点，例如，修改其他数据源或遵循联接转换。

使用带有十进制数据类型的更改架构

对十进制数据类型使用更改架构转换时，更改架构转换会将精度修改为默认值 (10,2)。要修改此设置并为您的用例设置精度，您可以使用 SQL Query 转换并以特定的精度转换列。

例如，如果您有一个名为 “” 且类型为 “DecimalDecimalCol” 的输入列，并且想要将其重新映射到名为 “OutputDecimalCol” 的输出列，其特定精度为 (18,6)，则需要：

1. 在更改架构转换之后添加后续的 SQL 查询转换。
2. 在 SQL 查询转换中，使用 SQL 查询将重新映射的列转换为所需的精度。SQL 查询将如下所示：

```
SELECT col1, col2, CAST(DecimalCol AS DECIMAL(18,6)) AS OutputDecimalCol
FROM __THIS__
```

在上面的 SQL 查询中：

- `col1`和`col2`是数据中的其他列，您希望在不做任何修改的情况下通过这些列。
- `DecimalCol` 是输入数据中的原始列名。
- `CAST (DecimalCol AS DECIMAL (18,6)) `将` DecimalCol `转换为十进制类型，精度为 18 位和 6 位小数。
- `AS OutputDecimalCol `将转换后的列重命名为``。OutputDecimalCol

通过使用 SQL Query 转换，您可以覆盖更改架构转换设置的默认精度，并将十进制列显式转换为所需的精度。这种方法允许您利用更改架构转换来重命名和重构数据，同时通过后续的 SQL 查询转换来处理十进制列的精度要求。

在作业中添加更改架构转换

> **Note**
>
> 更改架构转换不区分大小写。

将更改架构转换节点添加到作业图

1. （可选）打开资源面板，然后选择更改架构将新转换添加到作业图（如果需要）。
2. 在节点属性面板中，输入作业图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 选择节点属性面板中的转换选项卡。
4. 修改输入架构：
   - 要重命名数据属性键，请在 Target key (目标键) 字段中输入键的新名称。
   - 要更改数据属性键的数据类型，请从 Data type (数据类型) 列表中为键选择新数据类型。
   - 要从目标架构中删除数据属性键，请选中该键对应的 Drop (删除) 复选框。
5. （可选）配置转换节点属性后，您可以选择节点详细信息面板中的 Output schema (输出架构) 选项卡，查看数据的修改架构。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。如果您尚未在 Job details (任务详细信息) 选项卡上指定 IAM 角色，系统会提示您在此处输入 IAM 角色。
6. （可选）配置节点属性和转换属性后，您可以选择节点详细信息窗格中的 Data preview (数据预览) 选项卡来预览已修改的数据集。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。使用此功能会产生相关费用，并且一旦您提供 IAM 角色，则会立即开始计费。

使用删除重复项

删除重复项转换为您提供两个选项，从而从数据来源中移除行。您可以选择删除完全相同的重复行，也可以选择要匹配的字段，并根据所选字段仅删除这些行。

例如，在此数据集中，您有重复的行，其中一些行中的所有值与另一行中的所有值完全相同，而行中的某些值相同或不同。

行	名称	Email	Age	省/自治区/直辖市	备注
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	这是一个重复的行，并且与所有值完全匹配，如行 #4
7	Rose	rose@gmail	23	NJ	这是一个重复的行，并且与所有值完全匹配，如行 #5

如果您选择匹配整行，则第 6 行和第 7 行将从数据集中删除。现在的数据集为：

行	名称	Email	Age	省/自治区/直辖市
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

如果您选择指定键，则可以选择删除与“姓名”和“电子邮件”匹配的行。这使您可以更精细地控制数据集的“重复行”。通过指定“姓名”和“电子邮件”，数据集现在为：

行	名称	Email	Age	省/自治区/直辖市
1	Joy	joy@gmail	33	NY
2	Tim	tim@gmail	45	OH
3	Rose	rose@gmail	23	NJ

请记住以下事项：

- 为了将行识别为重复行，值区分大小写。行中的所有值都必须具有相同的大小写 — 这适用于您选择的任一选项（匹配整行或指定键）。
- 所有值都以字符串形式读入。
- 删除重复项转换使用 Spark dropDuplicates 命令。
- 使用删除重复项转换时，保留第一行并删除其他行。
- 删除重复项转换不会更改数据帧的架构。如果您选择指定键，则所有字段都将保留在生成的数据帧中。

使用 SelectFields 删除大多数数据属性键

您可以使用 SelectFields 转换从数据集创建数据属性键的子集。您可以指明要保留的数据属性键，其余的属性键将从数据集中删除。

> **Note**
>
> SelectFields 转换区分大小写。如果您需要以不区分大小写的方式选择字段，则使用 ApplyMapping。

将 SelectFields 转换节点添加到任务图

1. （可选）打开资源面板，然后选择 SelectFields 将新转换添加到作业图（如果需要）。

2. 在 Node properties (节点属性) 选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 选择节点详细信息窗格中的 Transform (转换) 选项卡。
4. 在标题 SelectFields 下面，选择要保留的数据集中的数据属性键。所有未选择的数据属性键都将从数据集中删除。

   您还可以选中列标题 Field (字段) 旁边的复选框，自动选择数据集中的所有数据属性键。然后，您可以取消选择单个数据属性键，将其从数据集中删除。
5. （可选）配置转换节点属性后，您可以选择节点详细信息面板中的 Output schema (输出架构) 选项卡，查看数据的修改架构。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。如果您尚未在 Job details (任务详细信息) 选项卡上指定 IAM 角色，系统会提示您在此处输入 IAM 角色。
6. （可选）配置节点属性和转换属性后，您可以选择节点详细信息窗格中的 Data preview (数据预览) 选项卡来预览已修改的数据集。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。使用此功能会产生相关费用，并且一旦您提供 IAM 角色，则会立即开始计费。

使用 DropFields 保留大多数数据属性键

您可以使用 DropFields 转换从数据集创建数据属性键的子集。您可以指明要从数据集中删除的数据属性键，其余的键均保留。

> **Note**
>
> DropFields 转换区分大小写。如果您需要以不区分大小写的方式选择字段，则使用更改架构。

将 DropFields 转换节点添加到任务图

1. （可选）打开资源面板，然后选择 DropFields 将新转换添加到作业图（如果需要）。
2. 在 Node properties (节点属性) 选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 选择节点详细信息窗格中的 Transform (转换) 选项卡。
4. 在标题 DropFields 下面，选择要从数据源中删除的数据属性键。

   您还可以选中列标题 Field (字段) 旁边的复选框，自动选择数据集中的所有数据属性键。然后，您可以取消选择单个数据属性键，以便将它们保留在数据集。

5. （可选）配置转换节点属性后，您可以选择节点详细信息面板中的 Output schema (输出架构) 选项卡，查看数据的修改架构。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。如果您尚未在 Job details (任务详细信息) 选项卡上指定 IAM 角色，系统会提示您在此处输入 IAM 角色。
6. （可选）配置节点属性和转换属性后，您可以选择节点详细信息窗格中的 Data preview (数据预览) 选项卡来预览已修改的数据集。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。使用此功能会产生相关费用，并且一旦您提供 IAM 角色，则会立即开始计费。

重命名数据集中的字段

您可以使用 RenameField 转换来更改数据集中单个属性键的名称。

> **Note**
>
> RenameField 转换区分大小写。如果您需要不区分大小写的转换，则使用 ApplyMapping。

> **Tip**
>
> 如果您使用更改架构转换，则可以使用单个转换重命名数据集中的多个数据属性键。

将 RenameField 转换节点添加到任务图

1. （可选）打开资源面板，然后选择 RenameField 将新转换添加到作业图（如果需要）。
2. 在 Node properties (节点属性) 选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 选择 Transform (转换) 选项卡。
4. 在标题 Transform (数据字段) 下面，从源数据中选择属性键，然后在 New field name (新字段名称) 字段中输入新名称。
5. （可选）配置转换节点属性后，您可以选择节点详细信息面板中的 Output schema (输出架构) 选项卡，查看数据的修改架构。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。如果您尚未在 Job details (任务详细信息) 选项卡上指定 IAM 角色，系统会提示您在此处输入 IAM 角色。

6. （可选）配置节点属性和转换属性后，您可以选择节点详细信息窗格中的 Data preview (数据预览) 选项卡来预览已修改的数据集。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。使用此功能会产生相关费用，并且一旦您提供 IAM 角色，则会立即开始计费。

使用 Spigot 对数据集进行采样

要测试任务执行的转换，您可能需要获取数据样本，以检查转换是否按预期工作。Spigot 转换将数据集中的记录子集写入 Amazon S3 存储桶中的 JSON 文件。数据采样方法可以是文件开头的指定记录数，或用于选取记录的概率因子。

将 Spigot 转换节点添加到任务图

1. （可选）打开资源面板，然后选择 Spigot 将新转换添加到作业图（如果需要）。
2. 在 Node properties (节点属性) 选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 选择节点详细信息窗格中的 Transform (转换) 选项卡。
4. 输入 Amazon S3 路径或选择 Browse S3 (浏览 S3)，在 Amazon S3 中选择位置。在此位置，任务将数据写入包含数据样本的 JSON 文件。
5. 输入采样方法的信息。您可以为指定 Number of records (记录数) 的值，从数据集的开头开始写入，指定 Probability threshold (概率阈值)（以十进制值输入，最大值为 1），挑选任何指定记录。

   例如，要从数据集中写入前 50 条记录，您可以将 Number of records (记录数) 设置为 50，将 Probability threshold (概率阈值) 设置为 1（100%）。

联接数据集

Join 转换允许您将两个数据集合成一个数据集。您可以在每个数据集的架构中指定键名称以进行比较。输出 `DynamicFrame` 包含键符合联接条件的行。每个数据集中满足连接条件的行将合并为输出 `DynamicFrame` 中的单一行，其中包含在任一数据集中找到的所有列。

将 Join 转换节点添加到任务图

1. 如果只有一个可用的数据源，则必须向任务图中添加新的数据源节点。
2. 为联接选择源节点。打开资源面板，然后选择联接将新转换添加到作业图。
3. 在 Node properties (节点属性) 选项卡上，输入任务图中节点的名称。

4. 在标题 Node parents (父节点) 下的 Node properties (节点属性) 选项卡中，添加一个父节点，以便有两个数据集为联接提供输入。父节点可以是数据源节点，也可以是转换节点。

   > **Note**
   >
   > 一个联接只能有两个父节点。

5. 选择 Transform (转换) 选项卡。

   如果您看到一条消息，指出有存在冲突的键名称，您可以执行以下操作：

   - 请选择 Resolve it (解决)，自动将 ApplyMapping 转换节点添加到您的任务图。ApplyMapping 节点会为数据集中与其他数据集中的键同名的键添加前缀。例如，如果使用默认值 **`right`**，则右侧数据集中与左侧数据集中的键同名的键都重命名为 `(right)`*`key name`*。
   - 在任务图中的前面手动添加转换节点，以删除或重命名冲突的键。

6. 在 Join type (联接类型) 列表中选择联接类型。

   - Inner join (内部联接)：对于基于联接条件的每个匹配项，返回包含来自两个数据集的列的行。不会返回不满足联接条件的行。
   - Left join (左侧联接)：左侧数据集中的所有行，以及右侧数据集中满足连接条件的行。
   - Right join (右侧联接)：右侧数据集中的所有行，以及左侧数据集中满足连接条件的行。
   - Outer join (外部联接)：两个数据集中的所有行。
   - Left semi join (左半联接)：左侧数据集中基于连接条件在右侧数据集中具有匹配项的所有行。
   - Left anti join (左反联接)：左侧数据集中基于连接条件在右侧数据集中没有匹配项的所有行。

7. 在标题 Join conditions (联接条件) 下的 Transform (转换) 选项卡中，选择 Add condition (添加条件)。从每个数据集中选择要比较的属性键。比较运算符左侧的属性键称为左侧数据集，右侧的属性键称为右侧数据集。

   对于更复杂的联接条件，您可以多次选择 Add condition (添加条件)，
   添加其他匹配键。如果您意外添加了条件，您可以选择删除图标
   （ 🗑 ）
   将其删除。

8. （可选）配置转换节点属性后，您可以选择节点详细信息面板中的 Output schema (输出架构) 选项卡，查看数据的修改架构。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。如果您尚未在 Job details (任务详细信息) 选项卡上指定 IAM 角色，系统会提示您在此处输入 IAM 角色。

9. （可选）配置节点属性和转换属性后，您可以选择节点详细信息窗格中的 Data preview (数据预览) 选项卡来预览已修改的数据集。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。使用此功能会产生相关费用，并且一旦您提供 IAM 角色，则会立即开始计费。

有关连接输出架构的示例，请考虑使用以下属性键在两个数据集之间进行联接：

```
Left: {id, dept, hire_date, salary, employment_status}
Right: {id, first_name, last_name, hire_date, title}
```

使用 = 比较运算符将联接配置为匹配 `id` 和 `hire_date` 键。

因为两个数据集都包含 `id` 和 `hire_date` 键，所以您选择 Resolve it (解决) 可自动将前缀 **`right`** 添加到正确数据集中的键。

输出架构中的键如下：

```
{id, dept, hire_date, salary, employment_status,
(right)id, first_name, last_name, (right)hire_date, title}
```

使用联合合并行

如果要联合来自多个具有相同架构的数据来源的行，则可以使用联合转换节点。

联合转换有两种类型：

1. ALL — 应用 ALL 时，生成的联合不会删除重复行。
2. DISTINCT — 应用 DISTINCT 时，生成的联合会删除重复的行。

联合与联接

您可以使用联合来合并行。您可以使用联接来合并列。

在 Visual ETL 画布中使用联合转换

1. 添加多个数据来源以执行联合转换。要添加数据来源，请打开资源面板，然后从“来源”选项卡中选择数据来源。在使用联合转换之前，必须确保联合中涉及的所有数据来源都具有相同的架构和结构。

2. 如果您至少有两个数据来源要使用联合转换进行组合，请将其添加到画布中来创建联合转换。打开画布上的资源面板并搜索“联合”。您也可以在资源面板中选择“转换”选项卡，向下滚动直到找到联合转换，然后选择联合。
3. 在作业画布上选择联合节点。在节点属性窗口中，选择要连接到联合转换的父节点。
4. AWS Glue 检查兼容性以确保联合转换可以应用于所有数据来源。如果数据来源的架构相同，则允许该操作。如果数据来源没有相同的架构，则会显示一条无效的错误消息：“此联合的输入架构不一样。请考虑使用 ApplyMapping 来匹配架构。” 要修复此问题，请选择使用 ApplyMapping。
5. 选择联合类型。
    1. 全部 - 默认情况下，选择“全部联合”类型；如果数据组合中有重复行，这将导致重复行。
    2. 不同 - 如果要从生成的数据组合中删除重复的行，请选择“不同”。

使用 SplitFields 将一个数据集拆分为两个

SplitFields 转换允许您选择输入数据集中的某些数据属性键，并将其放入一个数据集，将未选定的键放入单独的数据集。此转换的输出是 `DynamicFrames` 集合。

> **Note**
>
> 您必须使用 SelectFromCollection 转换将 `DynamicFrames` 集合转换为单个 `DynamicFrame`，然后将输出发送到目标位置。

SplitFields 转换区分大小写。如果您需要不区分大小写的属性键名称，则将 ApplyMapping 转换添加为父节点。

将 SplitFields 转换节点添加到任务图

1. （可选）打开资源面板，然后选择 SplitFields 将新转换添加到作业图（如果需要）。
2. 在 Node properties (节点属性) 选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 选择 Transform (转换) 选项卡。
4. 选择要放入第一个数据集的属性键。未选择的键将置于第二个数据集。
5. （可选）配置转换节点属性后，您可以选择节点详细信息面板中的 Output schema (输出架构) 选项卡，查看数据的修改架构。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供

IAM 角色以访问数据。如果您尚未在 Job details (任务详细信息) 选项卡上指定 IAM 角色，系统会提示您在此处输入 IAM 角色。

6. （可选）配置节点属性和转换属性后，您可以选择节点详细信息窗格中的 Data preview (数据预览) 选项卡来预览已修改的数据集。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。使用此功能会产生相关费用，并且一旦您提供 IAM 角色，则会立即开始计费。
7. 配置 SelectFromCollection 转换节点，处理生成的数据集。

SelectFromCollection 转换概览

某些变换具有多个数据集（而不是单个数据集）作为输出，例如 SplitFields。SelectFromCollection 转换从数据集集合（一组 DynamicFrames）中选择一个数据集（DynamicFrame）。转换输出是选定的 DynamicFrame。

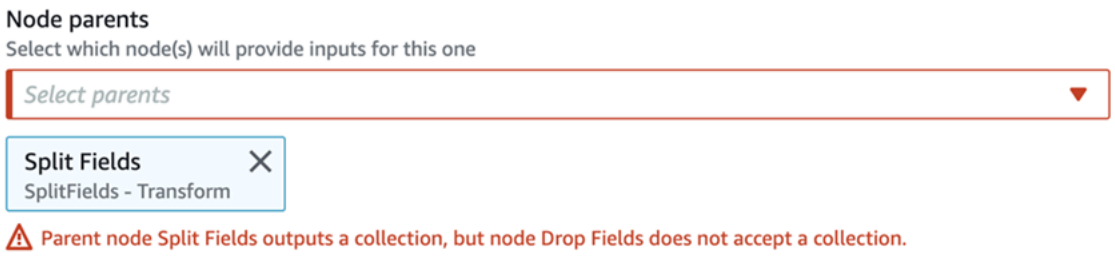
使用如下用于创建 DynamicFrames 集合的转换后，您必须使用此转换：

- 自定义代码转换
- SplitFields

如果您在这些转换之后没有将 SelectFromCollection 转换节点到您的任务图，您将收到任务错误。

此转换的父节点必须是返回 DynamicFrames 集合的节点。如果您为返回单个 DynamicFrame 的转换节点（例如 Join 转换）选择父节点，您的任务将返回错误。

同样，如果您在工作图中使用 SelectFromCollection 节点作为转换的父节点，而该转换需要单个 DynamicFrame 作为输入，那么您的工作将返回错误。



使用 SelectFromCollection 选择要保留的数据集

使用 SelectFromCollection 转换将 DynamicFrames 集合转换为单个 DynamicFrame。

将 SelectFromCollection 转换节点添加到任务图

1. （可选）打开资源面板，然后选择 SelectFromCollection 将新转换添加到作业图（如果需要）。
2. 在 Node properties (节点属性) 选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 选择 Transform (转换) 选项卡。
4. 在标题 Frame index (帧索引) 下面，选择您要从 `DynamicFrames` 集合中选择的 `DynamicFrame` 对应的数组索引编号。

   例如，如果此转换的父节点是 SplitFields 转换，则在该节点的 Output schema (输出架构) 选项卡上，您可以看到每个 `DynamicFrame` 的架构。如果您要保留与 Output 2 (输出 2) 的架构关联的 `DynamicFrame`，您需要选择 **1** 作为 Frame index (帧索引) 的值，这是列表中的第二个值。

   输出中仅包含 `DynamicFrame`。
5. （可选）配置转换节点属性后，您可以选择节点详细信息面板中的 Output schema (输出架构) 选项卡，查看数据的修改架构。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。如果您尚未在 Job details (任务详细信息) 选项卡上指定 IAM 角色，系统会提示您在此处输入 IAM 角色。
6. （可选）配置节点属性和转换属性后，您可以选择节点详细信息窗格中的 Data preview (数据预览) 选项卡来预览已修改的数据集。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。使用此功能会产生相关费用，并且一旦您提供 IAM 角色，则会立即开始计费。

查找并填充数据集中的缺失值

您可以在数据集中使用 FillMissingValues 转换，查找数据集中缺少值的记录，并添加包含由输入决定的值的新字段。输入数据集用于训练机器学习（ML）模型，该模型确定缺失值。如果您使用增量数据集，则每个增量集都会用作 ML 模型的训练数据，因此结果可能不是如此准确。

在任务图中使用 FillMissingValues 转换节点

1. （可选）打开资源面板，然后选择 FillMissingValues 将新转换添加到作业图（如果需要）。
2. 在 Node properties (节点属性) 选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 选择 Transform (转换) 选项卡。
4. 对于 Data field (数据字段)，从要分析缺失值的源数据中选择列或字段名称。

5. （可选）在 New field name (新字段名称) 字段中，输入每条记录所添加字段的名称，该字段将保存所分析字段的估计替换值。如果分析的字段没有缺失值，则分析字段中的值将复制到新字段。

   如果没有为新字段指定名称，默认名称是已分析列的名称，已附加 `_filled`。例如，如果您为 Data field (数据字段) 输入 **`Age`**，没有为 New field name (新字段名称) 指定值，则名为 **`Age_filled`** 的新字段会添加到每个记录。
6. （可选）配置转换节点属性后，您可以选择节点详细信息面板中的 Output schema (输出架构) 选项卡，查看数据的修改架构。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。如果您尚未在 Job details (任务详细信息) 选项卡上指定 IAM 角色，系统会提示您在此处输入 IAM 角色。
7. （可选）配置节点属性和转换属性后，您可以选择节点详细信息窗格中的 Data preview (数据预览) 选项卡来预览已修改的数据集。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。使用此功能会产生相关费用，并且一旦您提供 IAM 角色，则会立即开始计费。

筛选数据集中的键

使用 Filter 转换基于正则表达式从输入数据集中筛选记录，从而创建新记录。不满足筛选条件的行将从输出中删除。

- 对于字符串数据类型，您可以筛选键值与指定字符串匹配的行。
- 对于数值数据类型，您可以通过使用比较运算符 <、>、=、!=、<= 和 >=，将键值与指定值进行比较来筛选行。

如果您指定多个筛选条件，则默认使用 AND 运算符整合结果，但您可以改为选择 OR。

Filter 转换区分大小写。如果您需要不区分大小写的属性键名称，则将 ApplyMapping 转换添加为父节点。

将 Filter 转换节点添加到任务图

1. （可选）打开资源面板，然后选择筛选器将新转换添加到作业图（如果需要）。
2. 在 Node properties (节点属性) 选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 选择 Transform (转换) 选项卡。
4. 选择 Global AND 或者 Global OR。这将确定多个筛选条件的组合方式。所有条件都使用 AND 或者 OR 运算符。如果您只有一个筛选条件，则可以选择其中的任意一个。

5. 选择 Filter condition (筛选条件) 部分中的 Add condition (添加条件) 添加筛选条件。

   在 Key (键) 字段中，从数据集中选择属性键名称。在 Operation (运算符) 字段中，选择比较运算符。在 Value (值) 字段中，输入比较值。下面是一些筛选条件示例。

   - `year >= 2018`
   - `State matches 'CA*'`

   筛选字符串值时，请确保比较值使用与任务属性（Python 或 Scala）中选择的脚本语言相匹配的正则表达式格式。

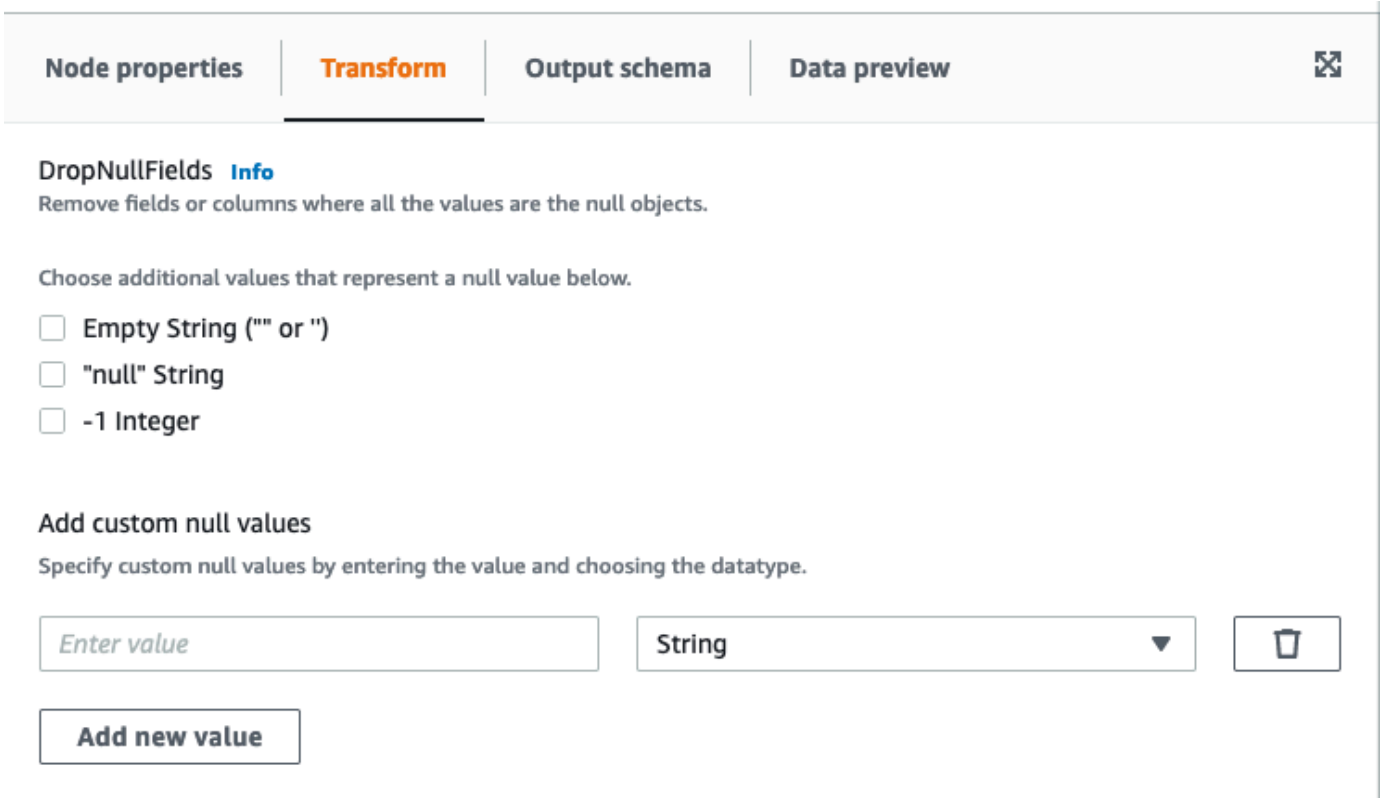
6. 根据需要添加额外的筛选条件。
7. （可选）配置转换节点属性后，您可以选择节点详细信息面板中的 Output schema (输出架构) 选项卡，查看数据的修改架构。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。如果您尚未在 Job details (任务详细信息) 选项卡上指定 IAM 角色，系统会提示您在此处输入 IAM 角色。
8. （可选）配置节点属性和转换属性后，您可以选择节点详细信息窗格中的 Data preview (数据预览) 选项卡来预览已修改的数据集。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。使用此功能会产生相关费用，并且一旦您提供 IAM 角色，则会立即开始计费。

使用 DropNullFields 删除空值字段

如果字段中的所有值均为“null”（空），请使用 DropNullFields 转换从数据集中删除字段。预设情况下，AWS Glue Studio 将识别空对象，但某些值（如空字符串、“null”（空）字符串、-1 整数或其他占位符（如零））不会自动识别为空。

要使用 DropNullFields

1. 将 DropNullFields 节点添加到任务图中。
2. 在 Node properties（节点属性）选项卡上，选择其他表示空值的值。您可以选择不选择任何值，也可以选择所有值：



- 空字符串(""或'')- 将删除包含空字符串的字段
- “null string”（空字符串）- 将删除包含带有“null”（空）一词的字符串的字段
- -1 整数 - 将删除包含 -1（负一）整数的字段

3. 如果需要，还可以指定自定义空值。这些空值可能仅适用于您的数据集。要添加自定义空值，请选择 Add new value（添加新值）。
4. 输入自定义空值。例如，可以是零，也可以是用于表示数据集中空值的任何值。
5. 在下拉字段中选择数据类型。数据类型可以是字符串或整数。

> **Note**
>
> 自定义空值及其数据类型必须完全匹配，才能将字段识别为空值并删除字段。部分匹配（只有自定义空值匹配，但数据类型不匹配）不会导致删除字段。

使用 SQL 查询转换数据

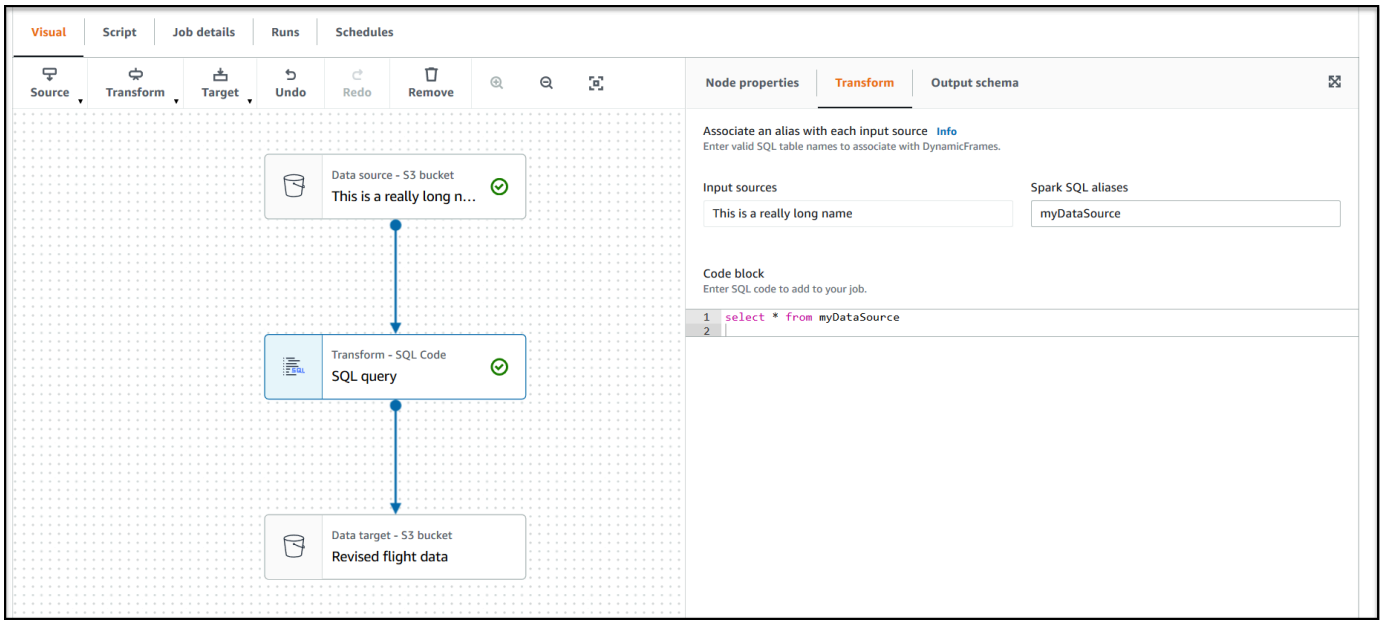
您可以使用 SQL 转换以 SQL 查询形式编写您自己的转换。

SQL 转换节点可以有多个数据集作为输入，但只能生成单个数据集作为输出。包含文本字段，您可以在其中输入 Apache SparkSQL 查询。您可以为用作输入的每个数据集分配别名，以帮助简单地执行 SQL 查询。有关 SQL 语法的更多信息，请参阅 Spark SQL 文档。

> **Note**
>
> 如果您对 VPC 中的数据源使用 Spark SQL 转换，请将 AWS Glue VPC 终端节点添加到包含数据源的 VPC。有关配置开发终端节点的更多信息，请参阅《AWS Glue 开发人员指南》中的添加开发终端节点、针对开发终端节点设置您的环境和访问您的开发终端节点。

在任务图中使用 SQL 转换节点

1. （可选）在需要时将转换节点添加到任务图。选择 Spark SQL 作为节点类型。
2. 在 Node properties (节点属性) 选项卡上，输入任务图中节点的名称。如果尚未选择父节点，或者您希望为 SQL 转换选择多个输入，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。根据需要添加额外的父节点。
3. 选择节点详细信息窗格中的 Transform (转换) 选项卡。
4. SQL 查询的源数据集由您在每个节点的 Name (名称) 字段中指定的名称标识。如果您不想使用这些名称，或者这些名称不适合 SQL 查询，则可以将名称与每个数据集关联。控制台提供默认别名，例如 `MyDataSource`。



例如，如果 SQL 转换节点的父节点名为 `Rename Org PK field`，您可以将名称 `org_table` 与此数据集一起使用。然后，您可以在 SQL 查询中使用此别名来代替节点名称。

5. 在标题 Code block (代码数据块) 下的文本输入字段中，粘贴或输入 SQL 查询。文本字段显示 SQL 语法突出显示和关键字建议。
6. 在选中 SQL 转换节点的情况下，选择 Output schema (输出架构) 选项卡，然后选择 Edit (编辑)。提供描述 SQL 查询输出字段的列和数据类型。

   使用页面的 Output schema (输出架构) 部分中的以下操作指定架构：

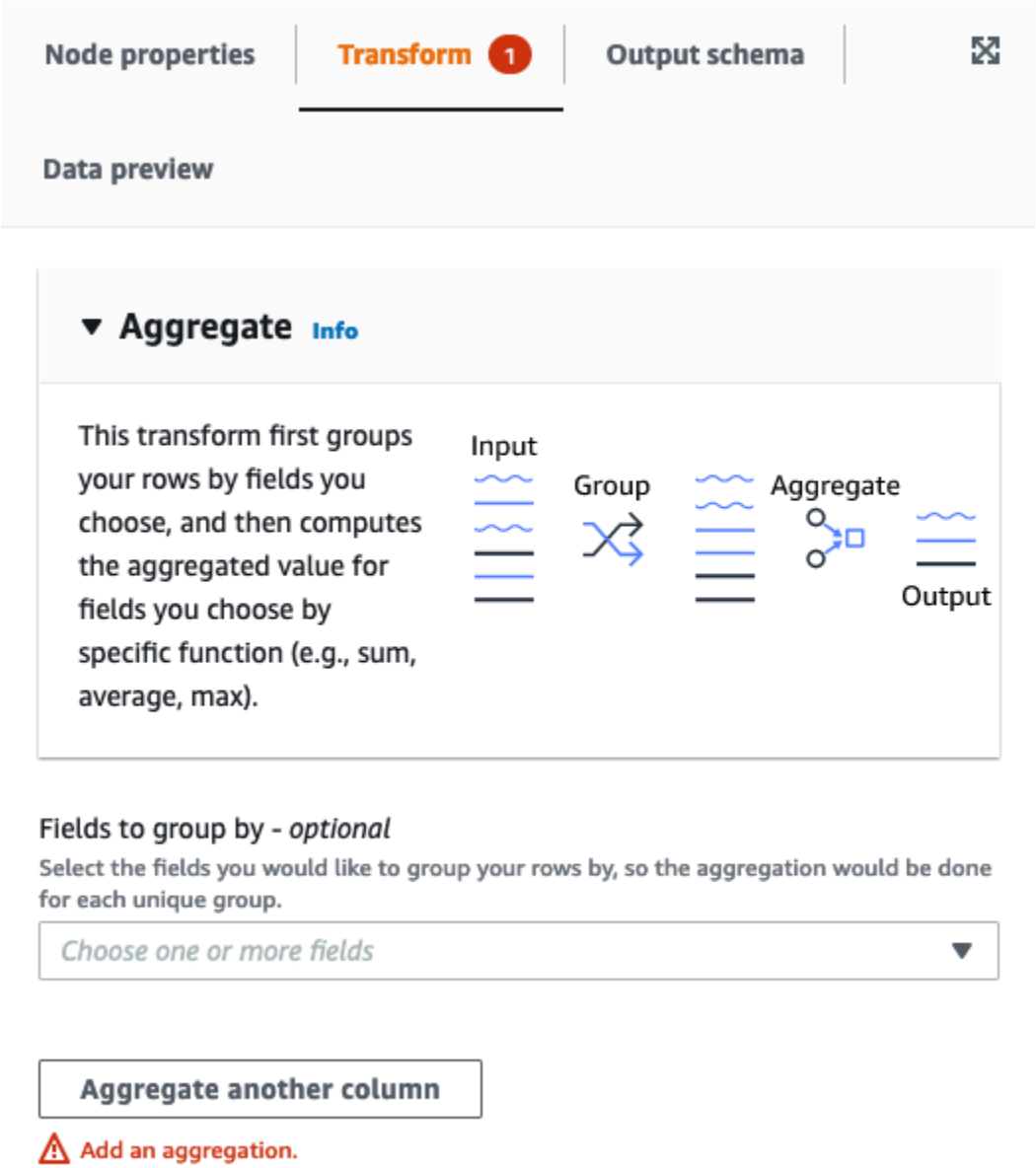
   - 要重命名列，请将光标放在列的 Key (键) 文本框（也称为 field (字段) 或者 property key (属性键)），然后输入新名称。
   - 要更改列的数据类型，请从下拉列表中选择该列的新数据类型。
   - 要为架构添加新的顶级列，请选择 Overflow (溢出)（ … ）按钮，然后选择 Add root key (添加根键)。新列将添加到架构顶部。
   - 要从架构中删除列，请选择键名称最右侧的删除图标（ 🗑 ）。
7. 指定输出架构后，选择 Apply (应用) 保存您的更改并退出架构编辑器。如果不想保存更改，请选择 Cancel (取消) 以编辑架构编辑器。
8. （可选）配置节点属性和转换属性后，您可以选择节点详细信息窗格中的 Data preview (数据预览) 选项卡来预览已修改的数据集。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。使用此功能会产生相关费用，并且一旦您提供 IAM 角色，则会立即开始计费。

使用聚合对选定字段执行汇总计算

要使用 Aggregate 转换

1. 将 Aggregate 节点添加到任务图中。
2. 在 Node properties（节点属性）选项卡上，通过选择下拉字段，选择要组合在一起的字段（可选）。您可以一次选择多个字段，也可以在搜索栏中键入来搜索字段名称。

   选择字段后，将显示名称和数据类型。要删除字段，请在字段上选择“X”。



3. 选择 Aggregate another column（聚合另一列）。至少需要选择一个字段。

4. 在 Field to aggregate（要聚合的字段）下拉列表中选择一个字段。
5. 选择要应用于所选字段的聚合函数：

- avg - 计算平均值
- countDistinct - 计算唯一非空值的数量
- count - 计算非空值的数量
- first - 返回满足“group by”（分组依据）条件的第一个值
- last - 返回满足“group by”（分组依据）条件的最后一个值
- kurtosis - 计算频率分布曲线峰值的锐度
- max - 返回满足“group by”（分组依据）条件的最高值
- min - 返回满足“group by”（分组依据）条件的最低值
- skewness - 衡量正态分布概率分布的不对称性
- stddev_pop - 计算总体标准差并返回总体方差的平方根
- sum - 组中所有值的总和。
- sumDistinct - 组中不同值的总和。
- var_samp - 组的样本方差（忽略空值）
- var_pop - 组的总体方差（忽略空值）

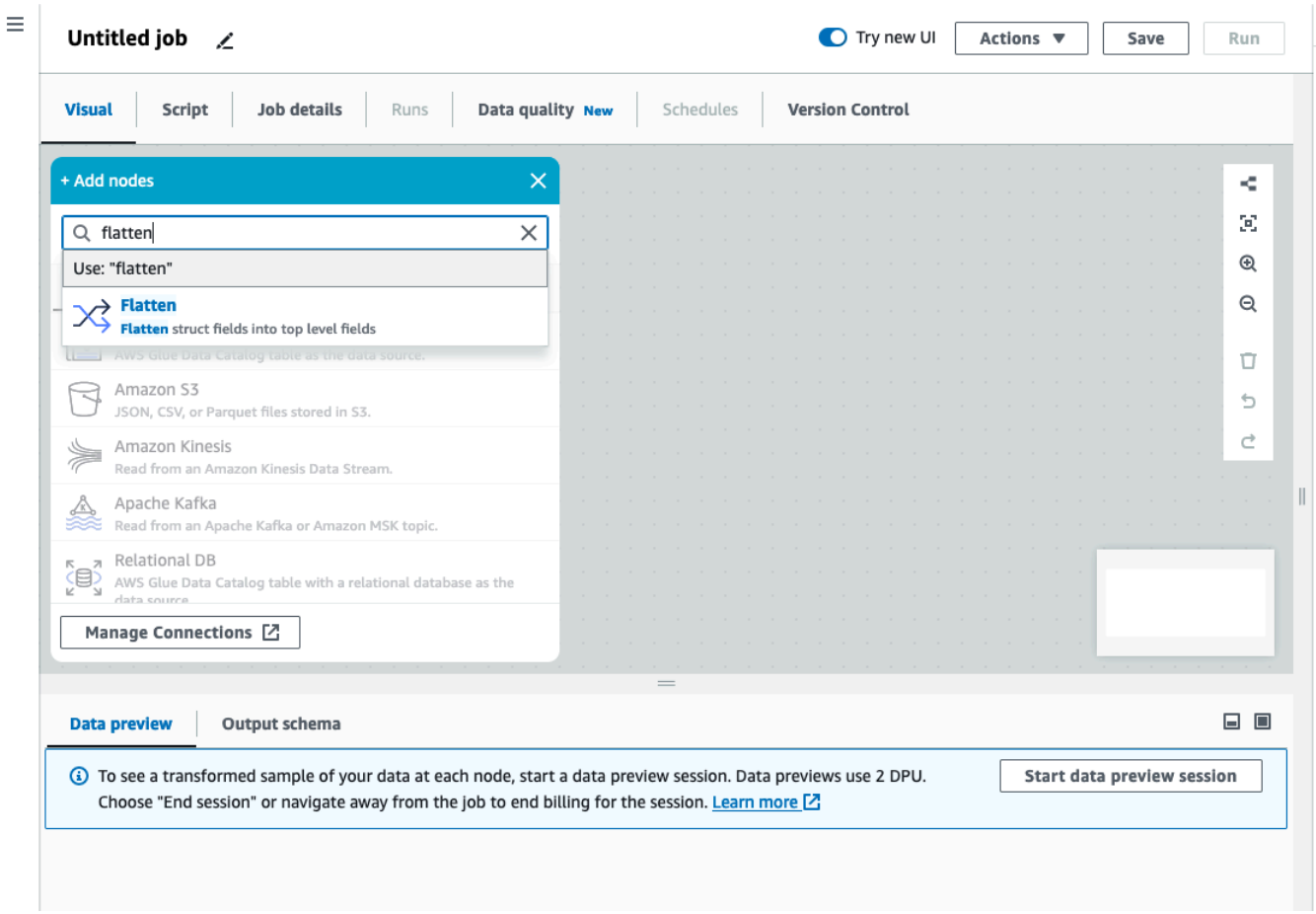
扁平化嵌套结构

扁平化数据中嵌套结构的字段，使它们成为顶级字段。使用以结构字段名称为前缀的字段名来命名新字段，用点分隔。

例如，如果数据中有一个名为“phone_numbers”的 Struct 类型的字段，
该字段中有一个名为“home_phone”的“Struct”字段，其中包含两个字
段：“country_code”和“number”。扁平化后，这两个字段将成为顶级字段，分别命名
为“phone_numbers.home_phone.country_code”和“phone_numbers.home_phone.number”。

将扁平化转换节点添加到任务图

1. 打开资源面板，选择转换选项卡，然后选择展平将新转换添加到作业图。您也可以通过输入“展平”，然后单击“展平”节点来使用搜索栏。添加节点时选择的节点将是其父节点。



2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. （可选）在转换选项卡上，可以限制要扁平化的嵌套级别的最大值。例如，将该值设置为 1 意味着只会扁平化顶级结构。将最大值设置为 2 将扁平化顶级结构及其下一级结构。

添加 UUID 列

添加 UUID（通用唯一标识）列时，将为每行分配一个唯一的字符串（36 个字符）。

将 UUID 转换节点添加到任务图

1. 打开资源面板，然后选择 UUID 将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. （可选）在转换选项卡上，您可以自定义新列的名称。默认情况下，新列将被命名为 "uuid"。

添加标识符列

为数据集中的每一行分配一个数字标识符。

在任务图中添加标识符转换节点

1. 打开资源面板，然后选择标识符将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. （可选）在转换选项卡上，您可以自定义新列的名称。默认情况下，将被命名为“id”。
4. （可选）如果作业以增量方式处理和存储数据，则要避免在两次作业运行之间重复使用相同的 id。

   在转换选项卡上，勾选复选框选项唯一。标识符中会包含作业时间戳，使标识符在多次运行之间是唯一的。为了允许较大的数字，列（而不是 Long 数据类型）将为十进制。

将列转换为时间戳类型。

您可以使用转换到时间戳将数值列或字符串列的数据类型更改为时间戳，这样就可以将列存储为该数据类型，或将列应用于需要时间戳的其他转换。

将 To timestamp 转换节点添加到任务图

1. 打开资源面板，然后选择转换为时间戳将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 在转换选项卡上，输入要转换的列的名称。
4. 在转换选项卡上，定义如何通过选择类型解析选定的列。

   如果该值是一个数字，可以选择相应的选项用秒（Unix/Python 时间戳）、毫秒或微秒表示。

   如果值是格式化字符串，请选择 "iso" 类型，该字符串需要符合 ISO 格式的变体之一，例如：“2022-11-02T14:40:59.915Z”。

   如果还不知道类型，或者不同的行使用不同的类型，则可以选择“自动检测”，系统会以较小的性能成本做出最佳猜测。

5. （可选）在转换选项卡上，您可以创建一个新列并通过输入新列的名称来保留原始列，而不是转换所选列。

将时间戳列转换为格式化字符串。

根据模式将时间戳列格式化为字符串。您可以使用格式化时间戳以所需格式的字符串形式获取日期和时间。您可以使用 Spark 日期语法以及大多数 Python 日期代码来定义格式。

例如，如果您希望日期字符串格式类似“2023-01-01 00:00”，则可以使用 Spark 语法（如“yyyy-MM-dd HH:mm”），或等效的 Python 日期代码（如“%Y-%m-%d %H:%M”）来定义这样的格式。

在任务图中添加格式化时间戳转换节点

1. 打开资源面板，然后选择格式化时间戳将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 在转换选项卡上，输入要转换的列的名称。
4. 在转换选项卡上，输入要使用的时间戳格式模式，使用 Spark 日期语法或 Python 日期代码表示。
5. （可选）在转换选项卡上，您可以创建一个新列并通过输入新列的名称来保留原始列，而不是转换所选列。

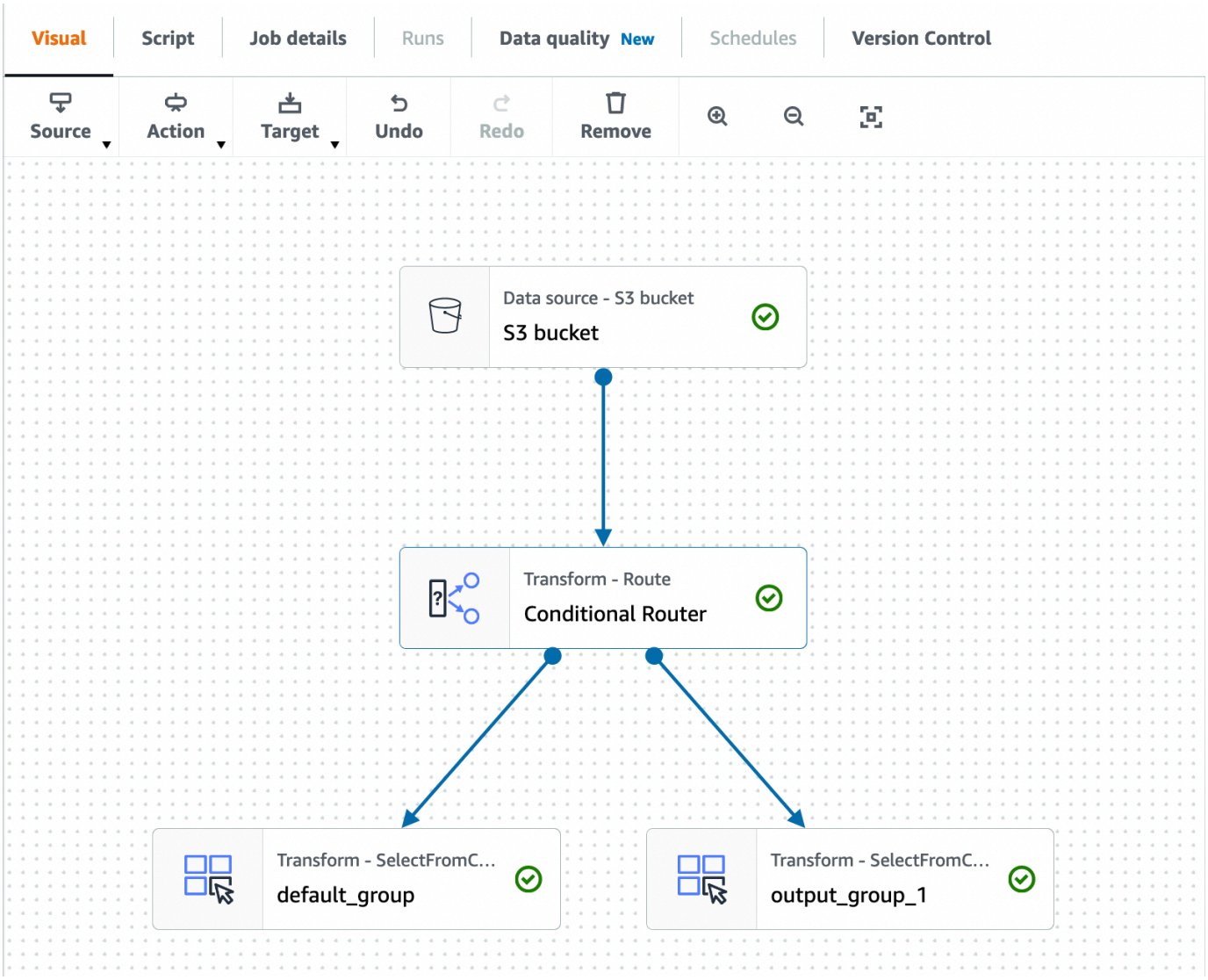
创建条件路由器转换

条件路由器转换允许您对传入数据应用多个条件。传入数据的每一行都通过一组筛选条件进行评估，然后处理到相应的组中。如果某行满足多个组筛选条件，则转换会将该行传递给多个组。如果某行不满足任何条件，则可以将其删除或路由到默认输出组。

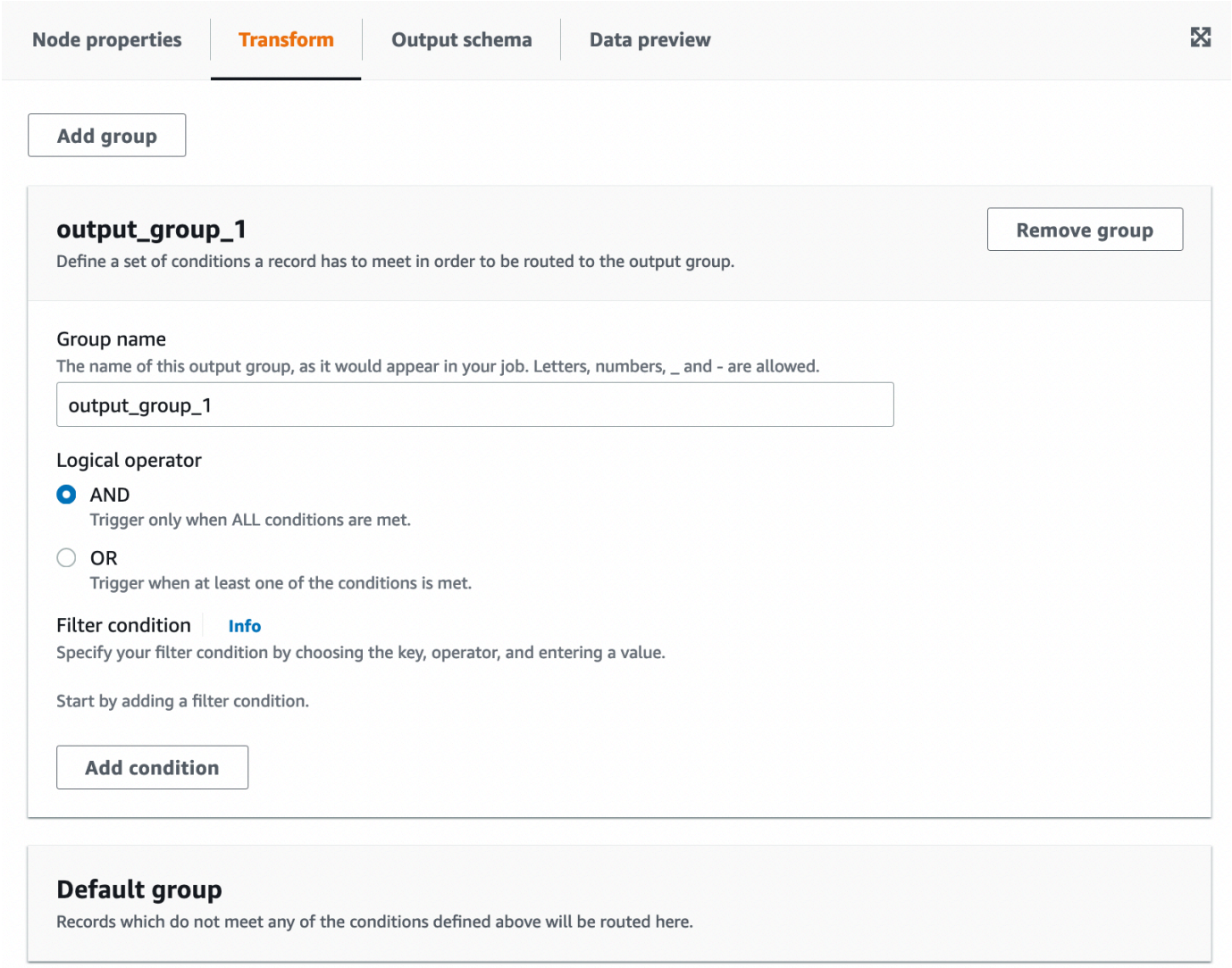
此转换类似于筛选器转换，但对想要在多个条件下测试相同输入数据的用户很有用。

要添加条件路由器转换，请执行以下操作：

1. 选择要执行条件路由器转换的节点。可以是源节点或其他转换。
2. 选择操作，然后使用搜索栏查找并选择“条件路由器”。添加条件路由器转换以及两个输出节点。一个输出节点为“默认组”，包含的记录不满足其他输出节点中定义的任何条件。无法编辑默认组。

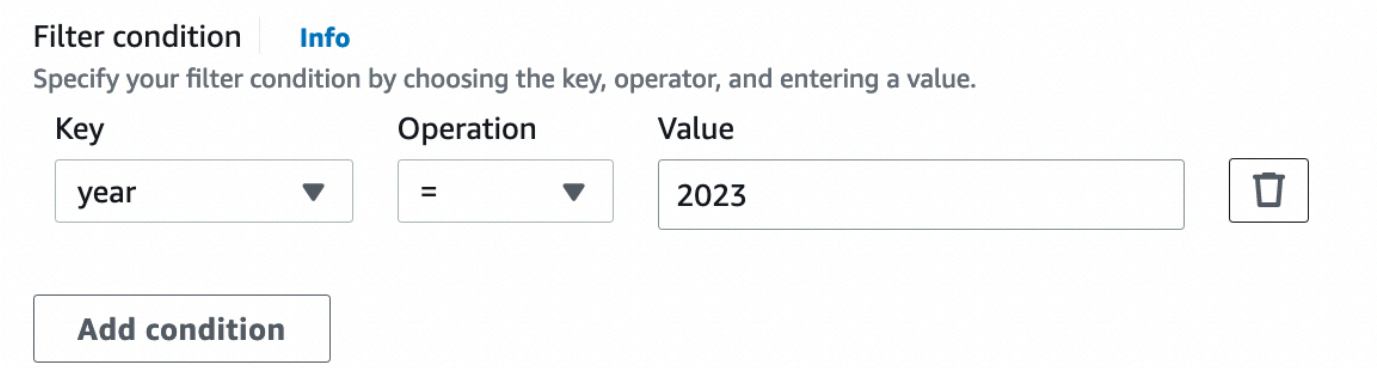


您可以通过选择添加组来添加其他输出组。您可以为每个输出组命名并添加筛选条件和逻辑运算符。



3. 输入新名称以重命名输出组。AWS Glue Studio 会自动为您命名群组（例如，'output_group_1'）。
4. 选择逻辑运算符（AND、OR），然后通过指定键、运算和值来添加筛选条件。逻辑运算符允许您实施多个筛选条件，并对指定的每个筛选条件执行逻辑运算符。

   指定键时，您可以从架构中的可用键进行选择。然后根据选择的键类型选择可用的操作。例如，如果键类型为“字符串”，则可供选择的操作是“匹配”。



5. 在值字段中输入值。选择添加条件以添加其他筛选条件。要移除筛选条件，请选择垃圾桶图标。

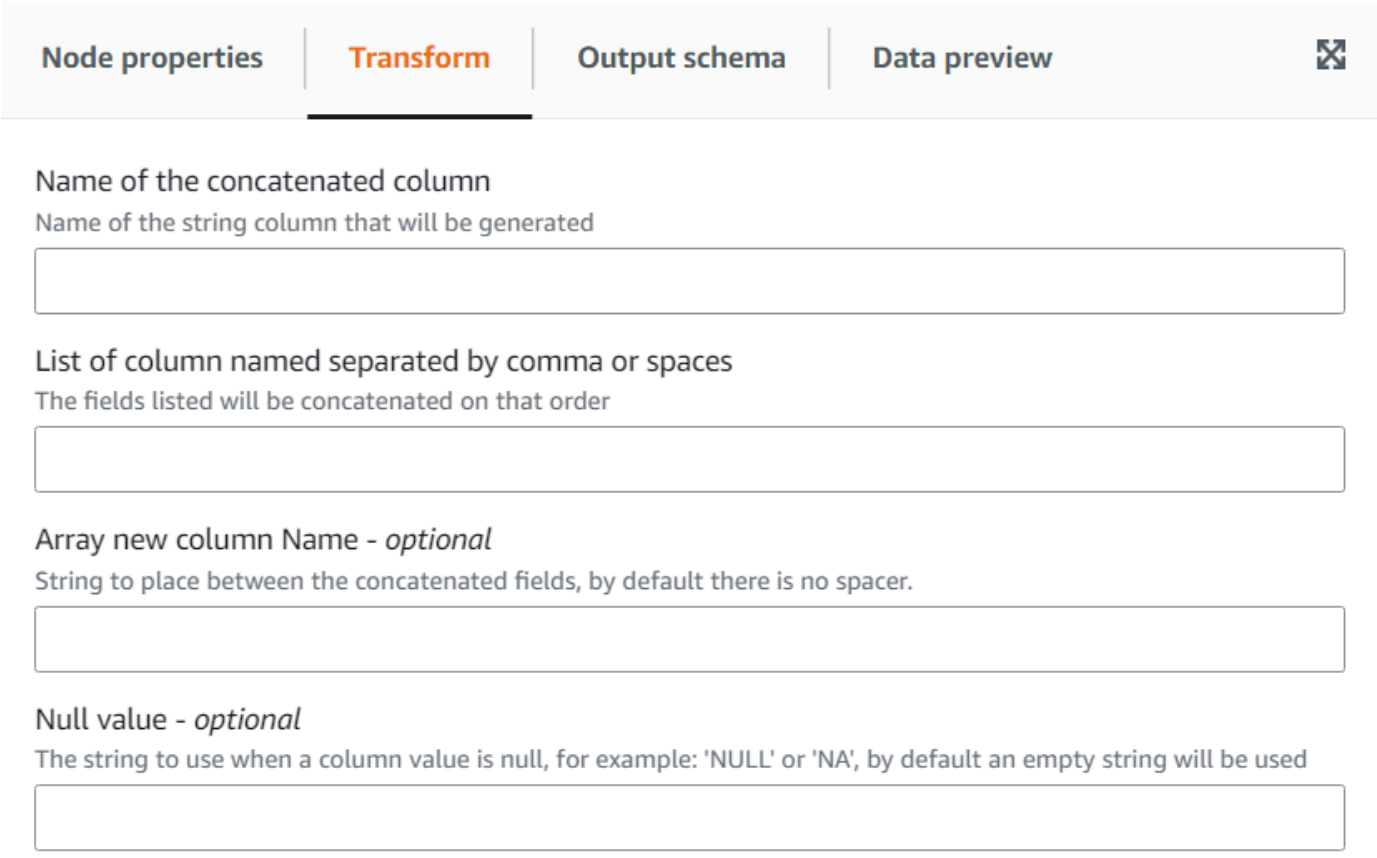
使用“串联列”转换来附加列

串联转换允许使用带有可选间隔符的其他列的值来生成新的字符串列。例如，如果我们将串联的列“日期”定义为“年”、“月”和“日”（按此顺序）的串联，并以“-”作为间隔符，则会得到：

day	month	year	date
01	01	2020	2020-01-01
02	01	2020	2020-01-02
03	01	2020	2020-01-03
04	01	2020	2020-01-04

要添加串联转换，请执行以下操作：

1. 打开资源面板。然后选择“串联列”，将新的转换添加到作业图。添加节点时选择的节点将是其父节点。
2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 在转换选项卡上，输入用于保存串联字符串的列的名称以及要串联的列。您检查下拉列表中各列的顺序将是使用的顺序。



4. 间隔符 — 可选 — 输入要在串联的字段之间放置的字符串。默认无间隔符。
5. 空值 — 可选 — 输入列值为空时要使用的字符串。默认情况下，如果列的值为“NULL”或“NA”，则使用空字符串。

使用拆分字符串转换来分解字符串列

拆分字符串转换允许您使用正则表达式将字符串分解为令牌数组，以定义拆分的完成方式。然后，您可以将该列保留为数组类型，或者在此之后应用数组转列转换，将数组值提取到顶级字段中，前提是每个令牌都有我们事先知道的含义。此外，如果令牌的顺序无关紧要（例如，一组类别），则可以使用分解转换为每个值生成单独的行。

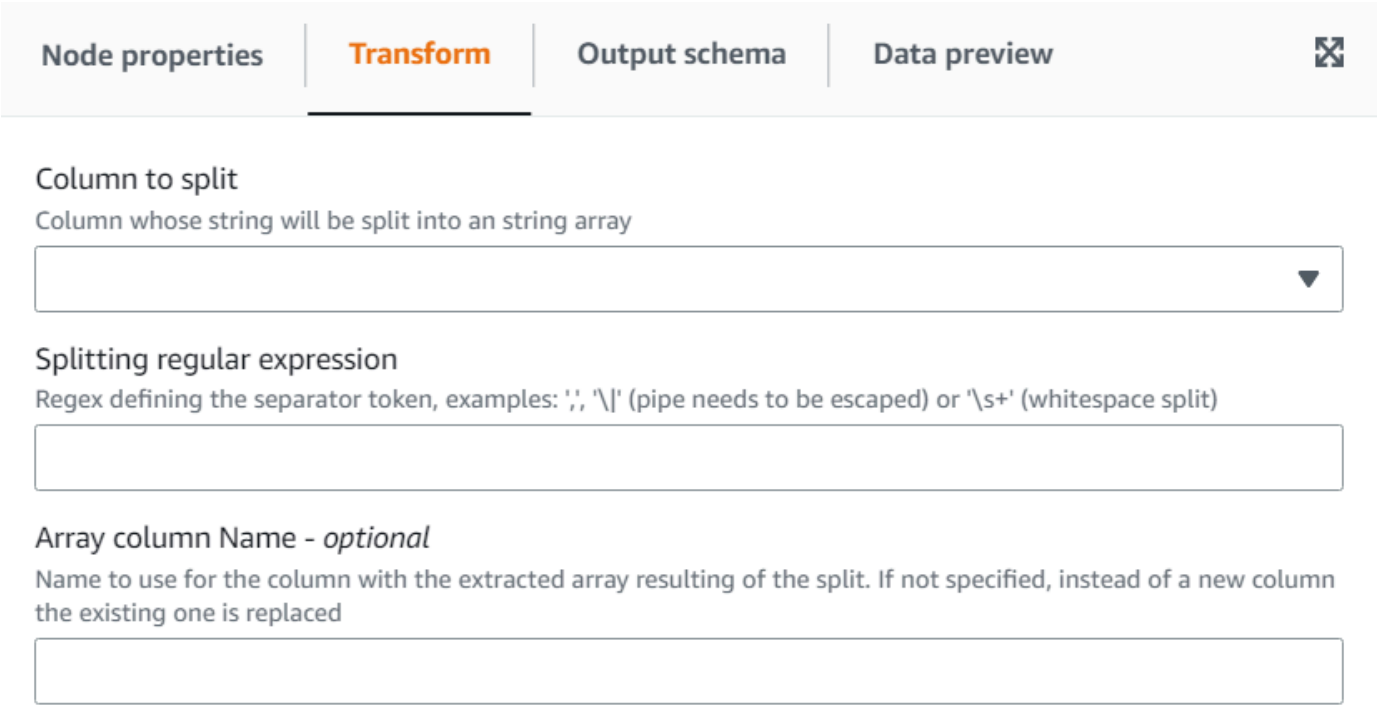
例如，您可以使用逗号作为模式拆分“类别”列，以添加一列“categories_arr”。

product_id	categories	categories_arr
1	运动、冬季	[运动、冬季]
2	花园、工具	[花园、工具]

product_id	categories	categories_arr
3	电子游戏	[电子游戏]
4	游戏、棋盘游戏、社交	[游戏、棋盘游戏、社交]

要添加拆分字符串转换，请执行以下操作：

1. 打开资源面板，然后选择“拆分字符串”将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 在转换选项卡上，选择要拆分的列，然后输入用于拆分字符串的模式。在大多数情况下，您可以只输入字符，除非它作为正则表达式具有特殊含义并且需要转义。需要转义的字符有：`\.[]{}()<>*+-=!?^$|` 在字符前面加一个反斜杠。例如，如果您想用点（'.'）分隔，则需要输入 `\.`。然而，逗号没有特殊含义，可以按原样指定：`,`。



4. （可选）如果要保留原始字符串列，则可以为新的数组列输入名称，这样既保留原始字符串列，又保留新的令牌化数组列。

使用“数组转列”转换将数组中的元素提取到顶级列中

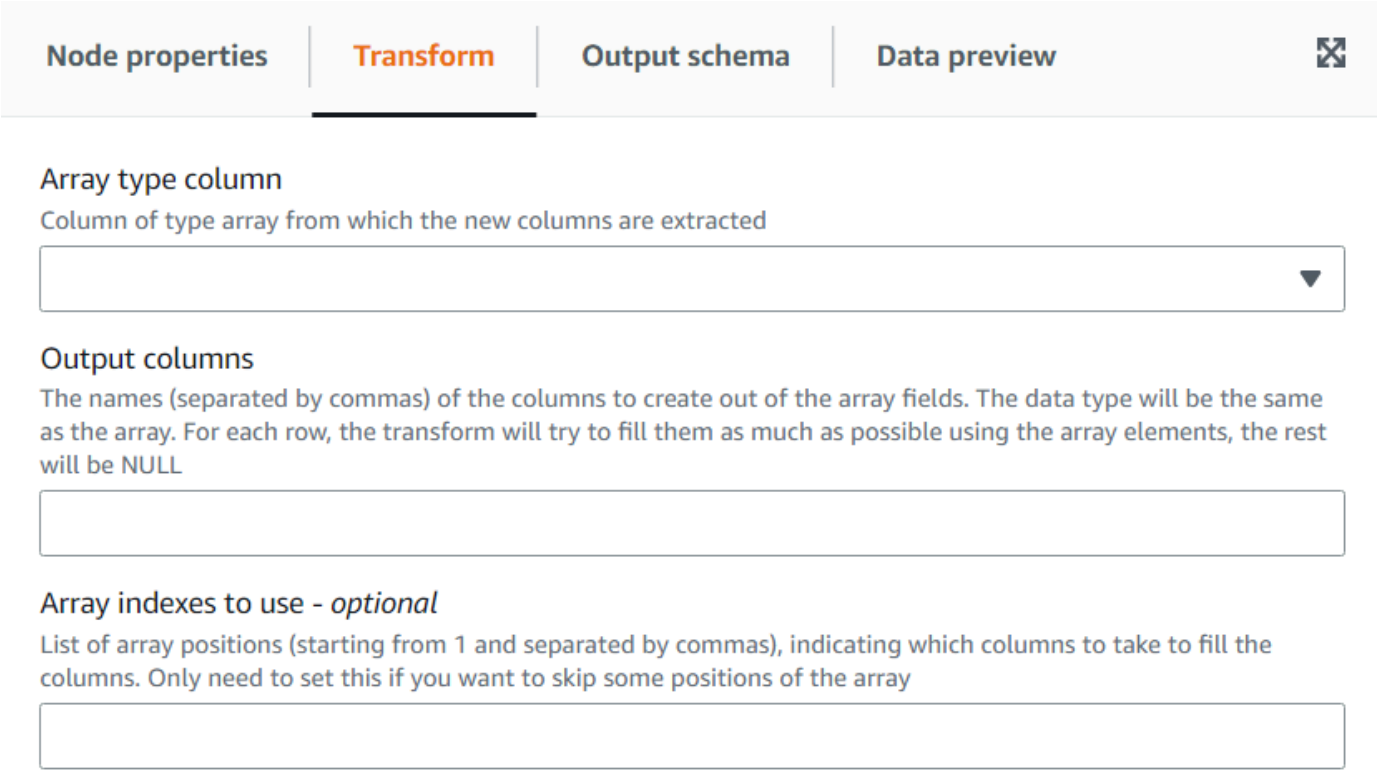
“数组转列”转换允许将数组类型的列的部分或全部元素提取到新列中。如果数组有足够的值可供提取，则转换将尽可能多地填充新列，也可以选择提取指定位置的元素。

例如，如果您有一个数组列“子网”，这是在 ip v4 子网上应用“拆分字符串”转换的结果，则可以将第一个和第四个位置提取到新列“first_octect”和“forth_octect”中。在本例中，转换的输出将是（请注意，最后两行的数组比预期的要短）：

子网	first_octect	fourth_octect
[54, 240, 197, 238]	54	238
[192, 168, 0, 1]	192	1
[192, 168]	192	
[]		

要添加“数组转列”转换，请执行以下操作：

1. 打开资源面板，然后选择数组转列将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 在转换选项卡上，选择要提取的数组列，然后为提取的令牌输入新列的列表。



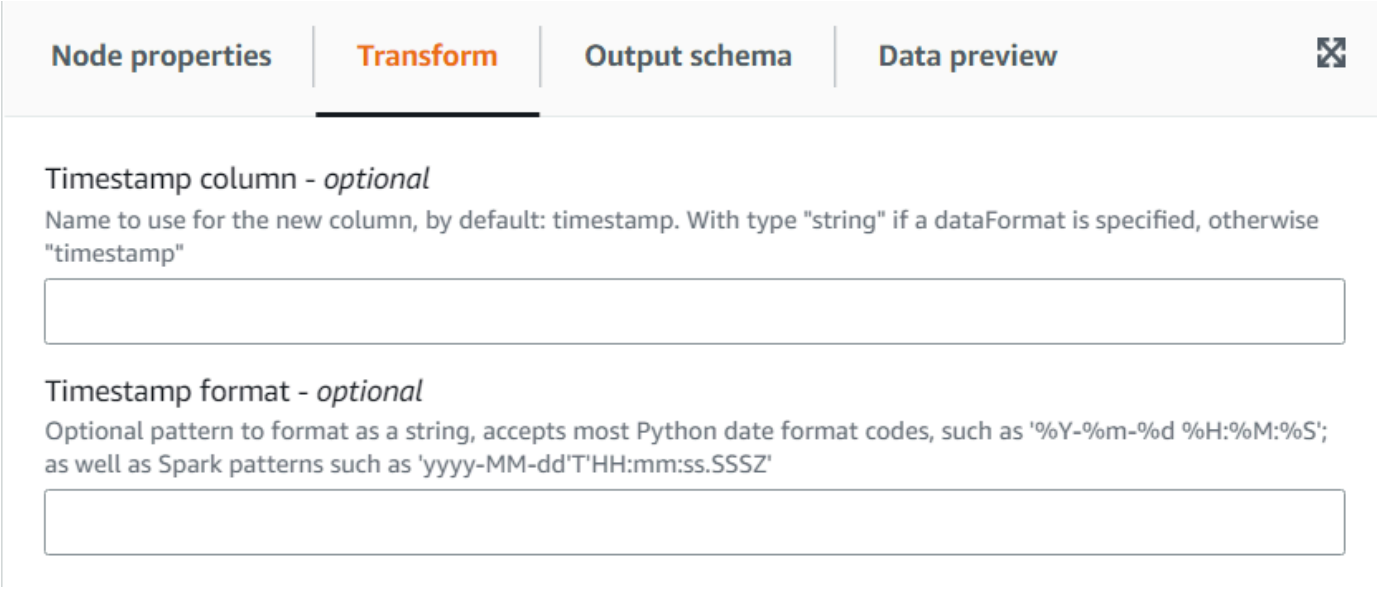
4. （可选）如果您不想使用数组令牌来分配给列，则可以指定要采用的索引，这些索引将按指定的相同顺序分配给列的列表。例如，如果输出列是“column1, column2, column3”，索引 为“4, 1, 3”，则数组的第四个元素将转到第 1 列，第一个元素进入第 2 列，第三个元素进入第 3 列（如果数组短于索引号，则将设置为 NULL 值）。

使用“添加当前时间戳”转换

添加当前时间戳转换允许您用数据处理时间标记行。这对于审计目的或跟踪数据管道中的延迟非常有用。您可以将此新列添加为时间戳数据类型或格式化字符串。

要添加“添加当前时间戳”转换，请执行以下操作：

1. 打开资源面板，然后选择添加当前时间戳将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。



3. （可选）在转换选项卡上，为新列输入自定义名称，如果您希望该列为格式化的日期字符串，则输入格式。

使用“将行转置为列”转换

将行转置为列转换允许您通过旋转选定列上的唯一值来聚合数字列，这些列会变成新列（如果选择了多个列，则将这些值串联起来以命名新列）。这样，行就可以合并，同时拥有更多的列，每个唯一值都有部分聚合。例如，如果您有按月和国家/地区划分的销售额数据集（为了便于说明而排序）：

年	月	country	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

年	月	country	amount
2020	Feb	us	12
2020	Jan	us	90

如果您将 amount 和 country 列转置为聚合列，则会根据原始 country 列创建新的列。在下面的表中，将会创建新的 de、uk 和 us 列，而不是 country 列。

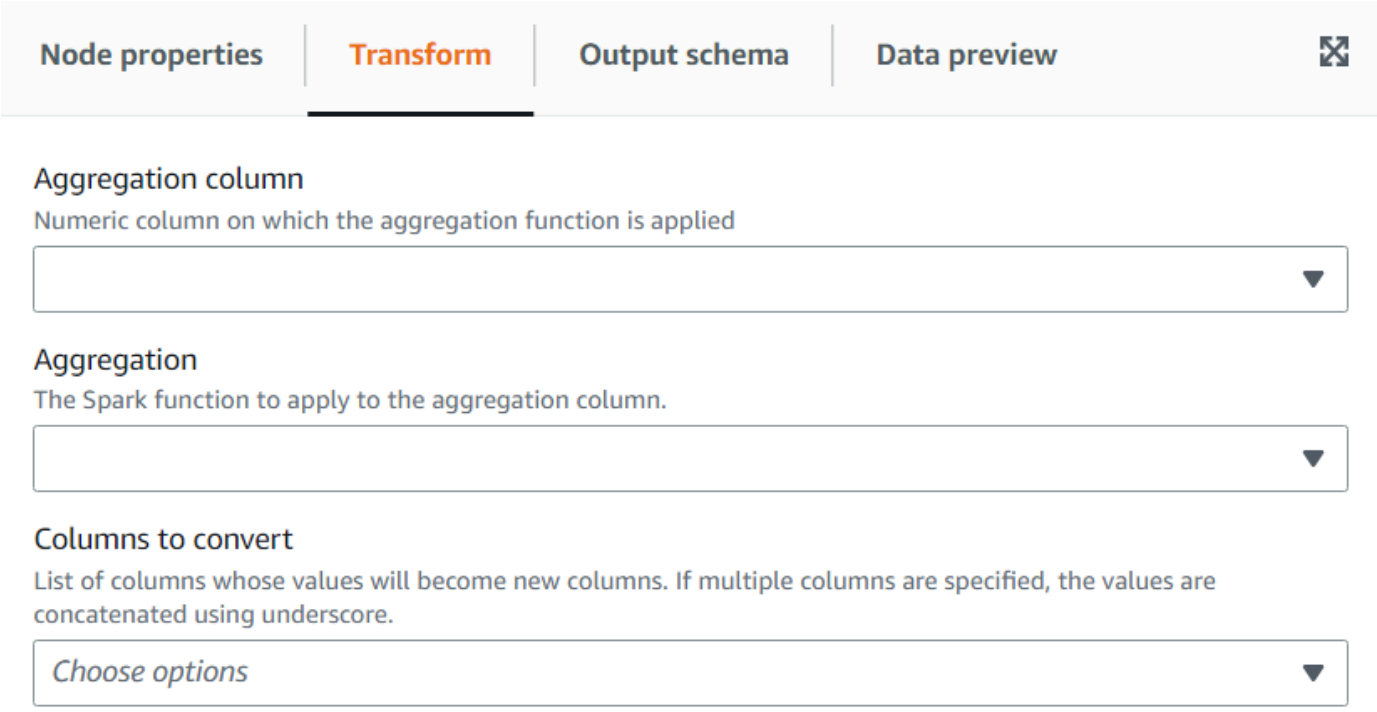
年	月	de	uk	us
2020	Jan	42	32	64
2020	Jan	11	67	18
2021	Jan			90

相反，如果您想同时对月份和县进行转置，则会为这些列的值的每种组合获得一列：

year	Jan_de	Jan_uk	Jan_us	Feb_de	Feb_uk	Feb_us
2020	42	32	64	11	67	18
2021			90			

要添加“将行转置为列”转换，请执行以下操作：

1. 打开资源面板，然后选择将行转置为列将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 在转换选项卡上，选择要聚合以生成新列值的数值列、要应用的聚合函数以及要将其唯一值转换为新列的列。



使用“反转置列为行”转换

反转置转换将列转换为新列的值，为每个唯一值生成一行。它与转置相反，但请注意，它并不等效，因为它无法将聚合的具有相同值的行分隔开来，也无法将组合拆分为原始列（您可以稍后使用拆分转换来做到这一点）。例如，如果您具有以下表：

年	月	de	uk	us
2020	Jan	42	32	64
2020	Feb	11	67	18
2021	Jan			90

您可以将“de”、“uk” 和“us”列反转置为值为“金额”的“国家/地区”列中，然后得到以下内容（出于说明起见，此处排序）：

年	月	country	amount
2020	Jan	uk	32

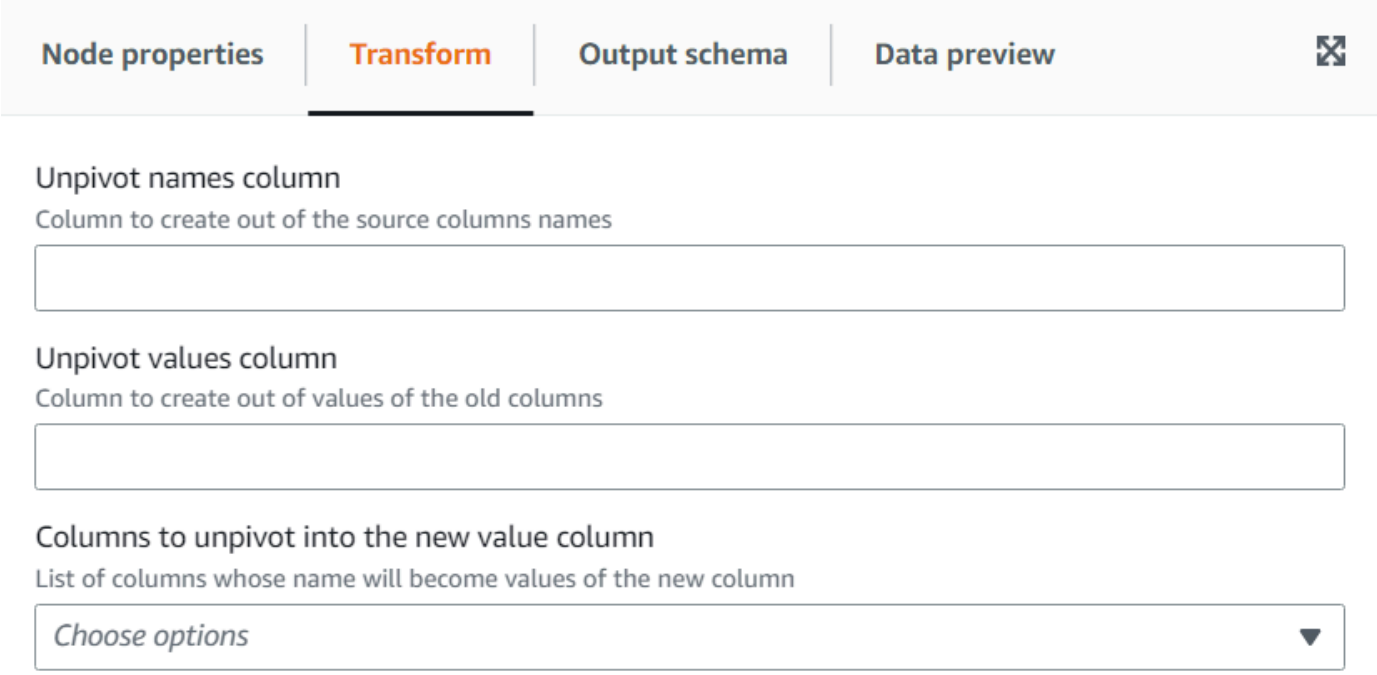
年	月	country	amount
2020	Jan	de	42
2020	Jan	us	64
2020	Feb	uk	67
2020	Feb	de	11
2020	Feb	us	18
2021	Jan	us	90

请注意，默认情况下不会生成具有空值的列（“de”和“2021 年 1 月的 uk”）。您可以启用该选项以获得：

年	月	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	

要添加“反转置列为行”转换，请执行以下操作：

1. 打开资源面板，然后选择反转置列为行将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 在转换选项卡上，输入要创建的新列，以保存选择反转置的列的名称和值。

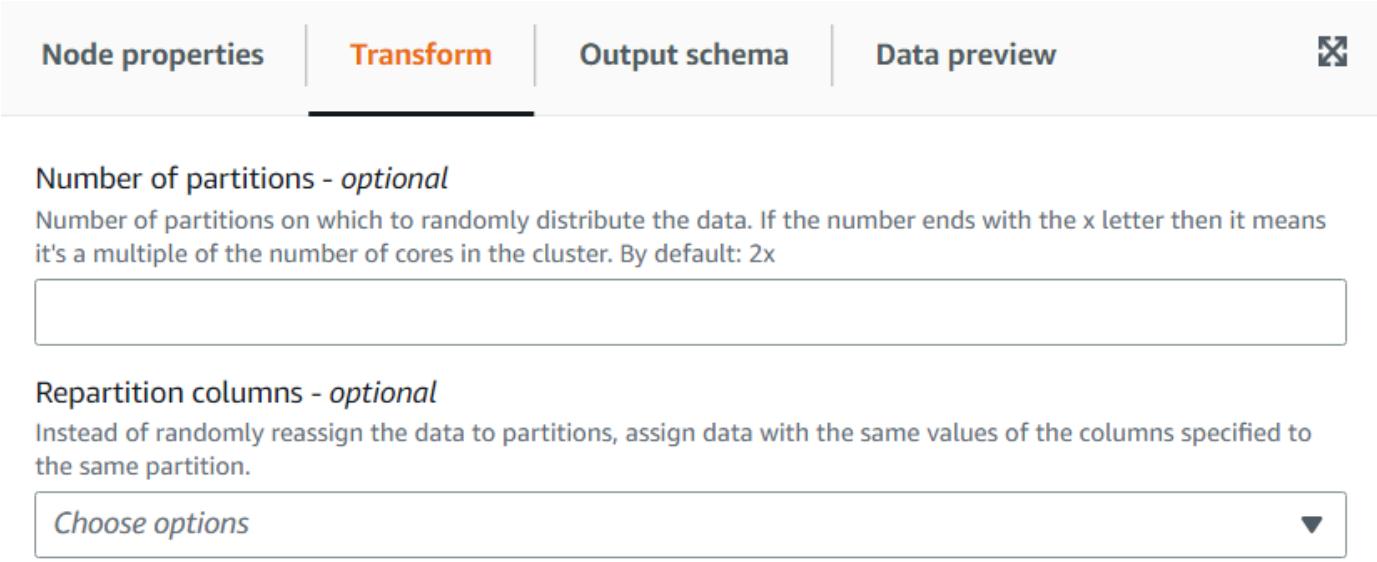


使用自动平衡处理转换来优化运行时

自动平衡处理转换在工作线程之间重新分配数据，以提高性能。当数据不平衡或数据来自源时不允许对其进行足够的并行处理时，这很有用。在源代码为 GZIP 或 JDBC 的情况下，这种情况很常见。数据的再分发具有适度的性能成本，因此，如果数据已经很好地平衡，优化可能并不总是能弥补这种努力。在下面，转换使用 Apache Spark 重新分区在最适合集群容量的多个分区之间随机重新分配数据。对于高级用户，可以手动输入多个分区。此外，它还可用于通过根据指定列重新组织数据来优化分区表的写入。这会使输出文件更加合并。

1. 打开资源面板，然后选择自动平衡处理将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。

3. （可选）在转换选项卡上，可以输入多个分区。通常，建议您让系统决定该值，但是如果您需要控制该值，则可以调整乘数或输入特定值。如果要保存按列分区的数据，则可以选择与重新分区列相同的列。这样，它将最大限度地减少每个分区上的文件数量，并避免每个分区有多个文件，这将阻碍查询该数据的工具的性能。



使用派生列转换合并其他列

派生列转换允许您根据数学公式或 SQL 表达式定义一个新列，您可以在其中使用数据中的其他列以及常量和文字。例如，要从 "success" 和 "count" 列中导出 "percentage" 列，可以输入 SQL 表达式："success * 100 / count || '%'"。

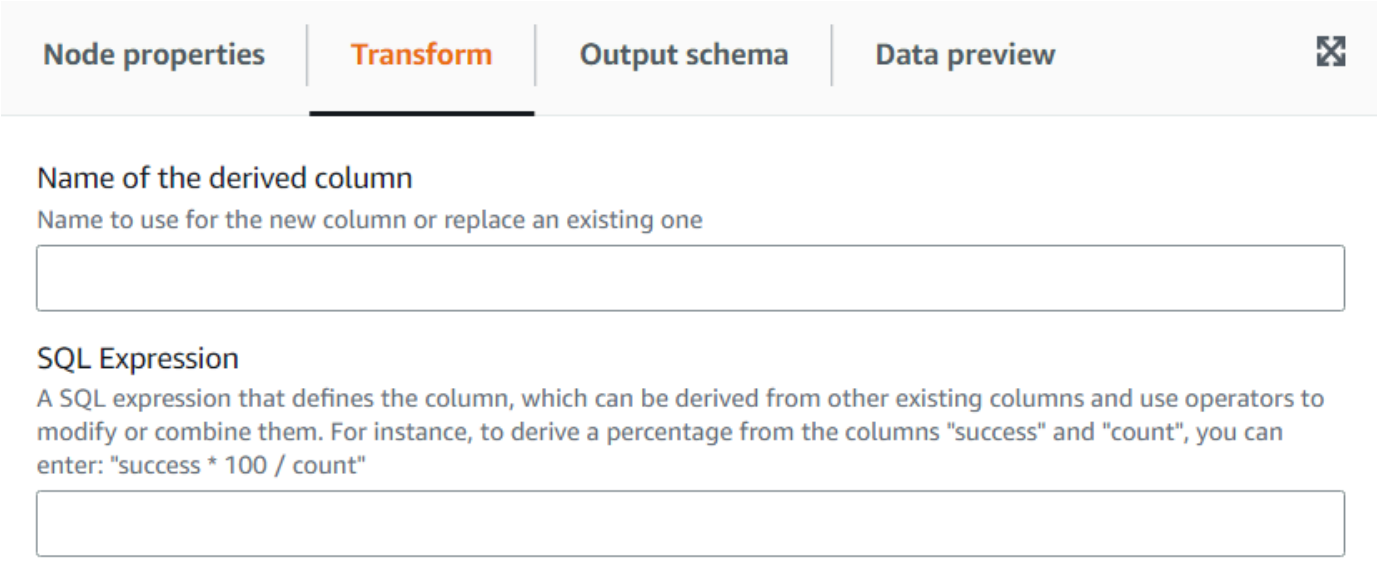
示例结果：

success	count	percentage
14	100	14%
6	20	3%
3	40	7.5%

要添加派生列转换，请执行以下操作：

1. 打开资源面板，然后选择派生列将新转换添加到作业图。添加节点时选择的节点将是其父节点。

2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 在转换选项卡上，输入列名称及其内容表达式。



使用查找转换从目录表中添加匹配数据

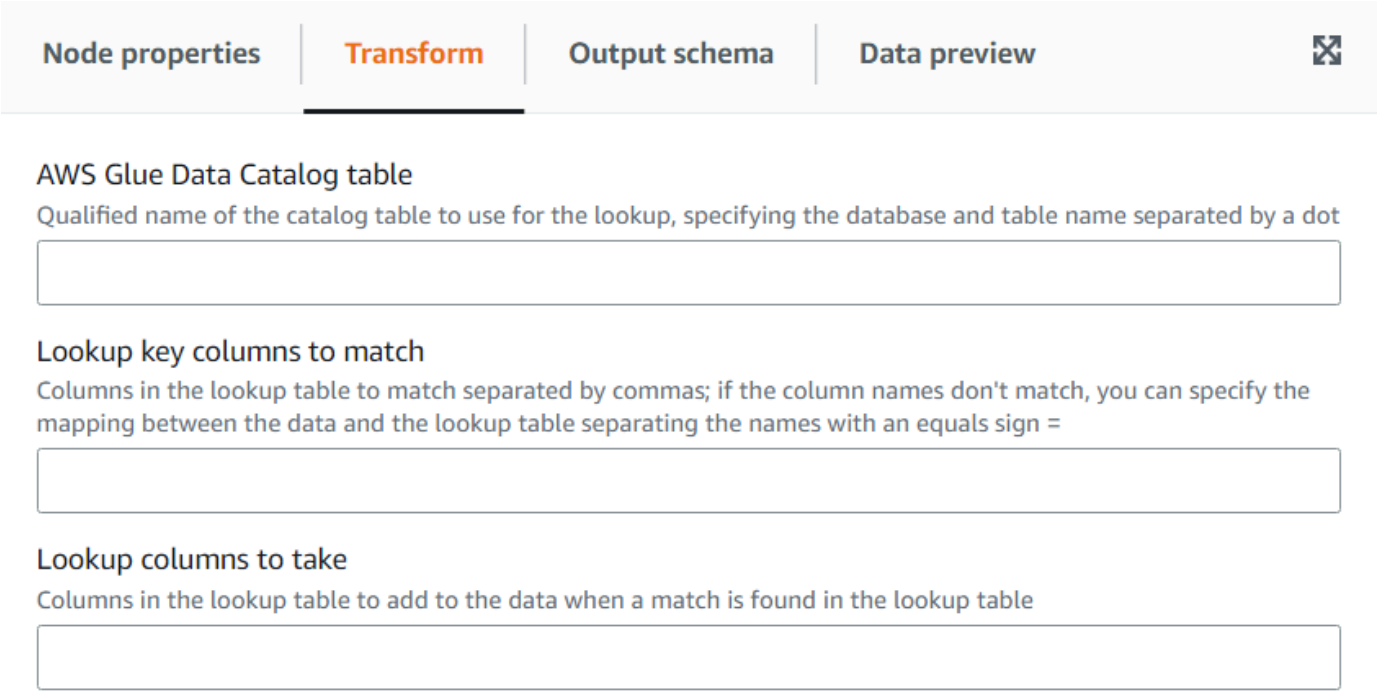
当键与数据中定义的查找列匹配时，查找转换允许您从定义的目录表中添加列。这等效于使用作为条件匹配列在数据和查找表之间进行左外联接。

要添加查找转换，请执行以下操作：

1. 打开资源面板，然后选择查找将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 在转换选项卡上，输入用于执行查找的完全限定的目录表名称。例如，如果您的数据库是“mydb”，您的表是“mytable”，那么输入“mydb.mytable”。然后输入在查找表中查找匹配项的条件（如果查询键为符合式）。输入以逗号分隔的键列列表。如果一个或多个键列的名称不同，则需要定义匹配映射。

   例如，如果数据列是“user_id”和“region”，并且在用户表中，相应的列名为“id”和“region”，则在要匹配的列字段中输入：”user_id=id, region“。您可以执行 region=region，但不需要，因为它们是一样的。
4. 最后，输入要从查找表中匹配的行中提取的列，以将其合并到数据中。如果未找到匹配项，则这些列将设置为 NULL。

> **Note**
>
> 在查找转换之下，为了提高效率，它使用左联接。如果查找表具有复合键，请确保将要匹配的列设置为匹配所有键列，这样只能出现一个匹配项。否则，将匹配多个查找行，这将导致为每个匹配项添加额外的行。



使用“分解数组或映射到行”转换

分解转换允许您将嵌套结构中的值提取到更易于操作的单个行中。对于数组，转换将为数组的每个值生成一行，复制该行中其他列的值。对于映射，转换将为每个条目生成一行，键和值为列，再加上该行中的任何其他列。

例如，如果我们有这个数据集，它有一个包含多个值的“类别”数组列。

product_id	类别
1	[运动、冬季]
2	[花园、工具]

product_id	类别
3	[电子游戏]
4	[游戏、棋盘游戏、社交]
5	[]

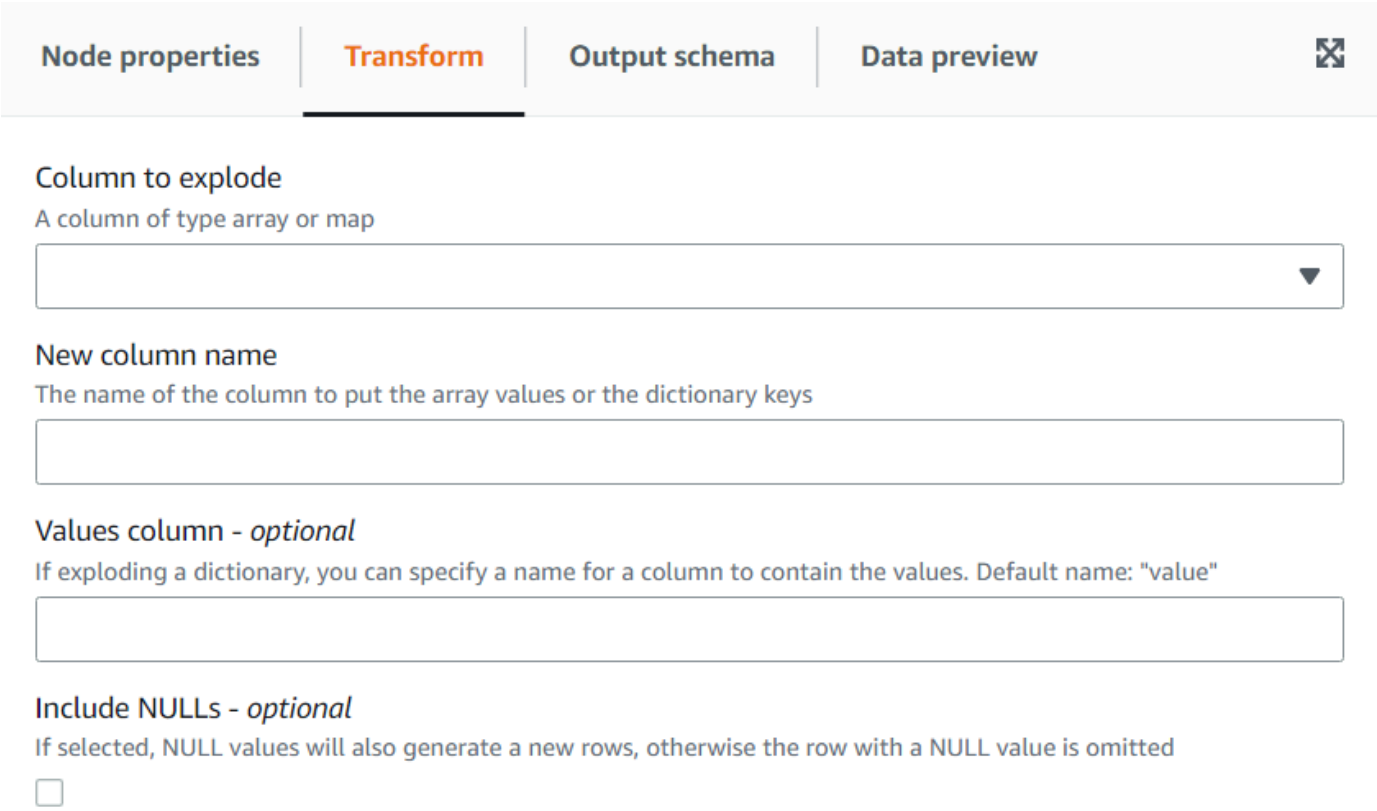
如果您将“类别”列分解为具有相同名称的列，则将覆盖该列。您可以选择要包含 NULL 以获得以下内容（出于说明目的排序）：

product_id	类别
1	运动
1	冬天
2	花园
2	工具
3	电子游戏
4	游戏
4	棋盘游戏
4	社交
5	

要添加“分解数组”或“映射成行”转换，请执行以下操作：

1. 打开资源面板，然后选择分解数组或映射到行将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. （可选）在节点属性选项卡上，输入任务图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。

3. 在转换选项卡上，选择要分解的列（它必须是数组或地图类型）。然后，如果要分解地图，则为数组中的项目输入列的名称，或者为键和值输入列的名称。
4. （可选）在转换选项卡上，默认情况下，如果要分解的列为 NULL 或结构为空，则在分解的数据集中将省略该列。如果要保留该行（将新列设置为 NULL），请选中“包含 NULL”。

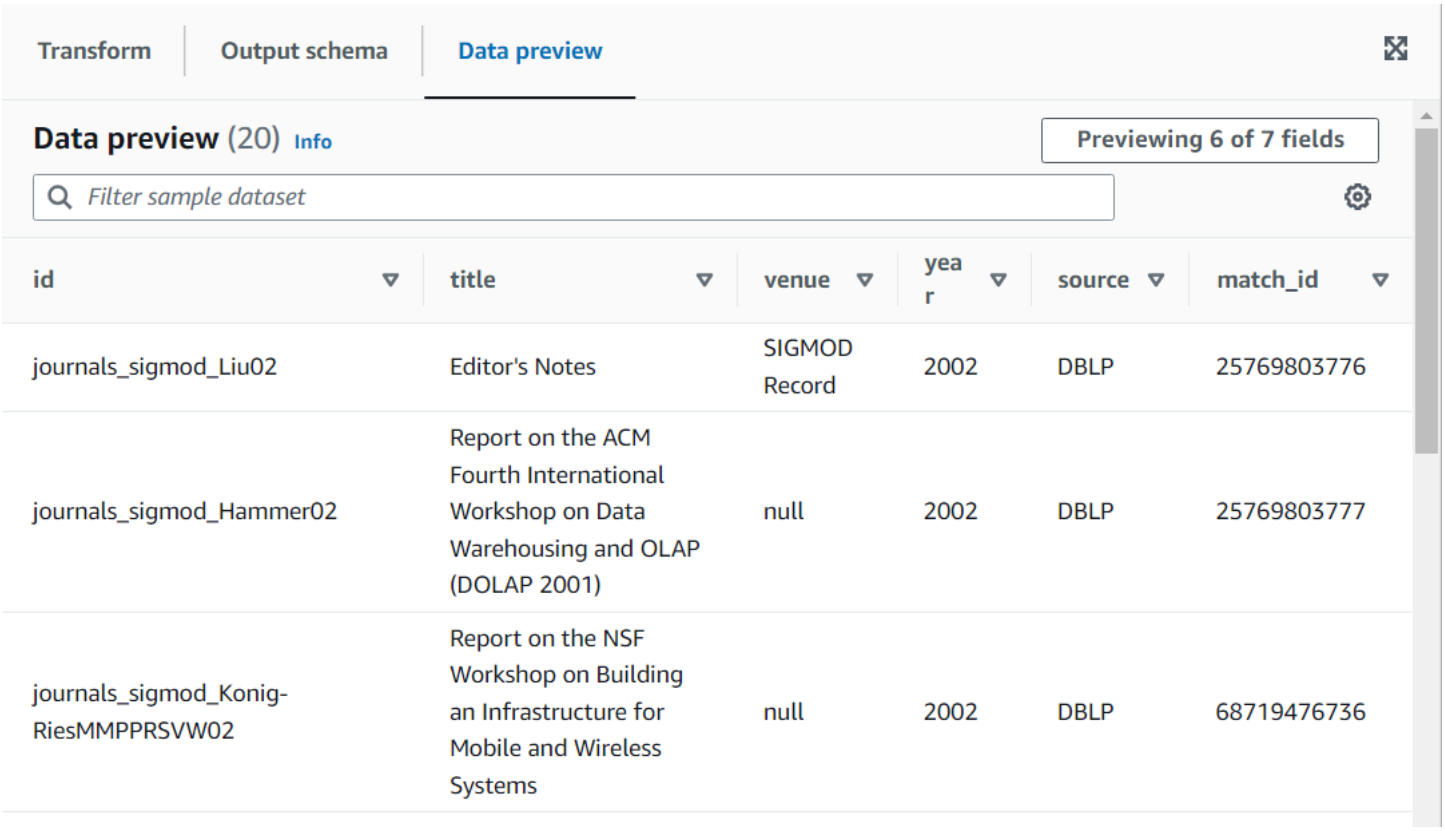


使用“记录匹配”转换调用现有的数据分类转换

此转换调用现有的记录匹配机器学习数据分类转换。

转换根据标签对照训练过的模型评估当前数据。添加了“match_id”列，将每行分配给一组根据算法训练被视为等效的项目。有关更多信息，请参阅 Record matching with Lake Formation FindMatches。

> **Note**
>
> AWS Glue 使用的可视化作业版本必须与 AWS Glue 用于创建“记录匹配”转换的版本相匹配。



将“记录匹配”转换节点添加到作业图

1. 打开资源面板，然后选择记录匹配将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. 在节点属性面板上，输入作业图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 在转换选项卡上，输入从机器学习转换页面获取的 ID：

4. （可选）在转换选项卡上，可以选中添加置信度分数的选项。该模型将以额外计算为代价，将每个匹配的置信度分数作为附加列进行估计。

移除空行

此转换从数据集中移除所有列均为空的行。此外，您可以扩展此标准以包括空字段，从而保留至少有一列非空的行。

将移除空行转换节点添加到作业图

1. 打开资源面板，然后选择移除空行将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. 在节点属性面板上，输入作业图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. （可选）在转换选项卡上，如果想要求行不为 null 而且不为空，请选中扩展选项，这样，就此转换而言，空字符串、数组或映射将被视为 null 字符串、数组或映射。

解析包含 JSON 数据的字符串列

这种转换解析包含 JSON 数据的字符串列并将其转换为结构或数组列，具体取决于 JSON 是对象还是数组。或者，您可以同时保留已解析的列和原始列。

可以提供或推断 JSON 架构（对于 JSON 对象），并可选择采样。

将解析 JSON 列转换节点添加到作业图

1. 打开资源面板，然后选择解析 JSON 列将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. 在节点属性面板上，输入作业图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 在转换选项卡上，选择包含 JSON 字符串的列。
4. （可选）在转换选项卡上，使用 SQL 语法输入 JSON 数据所遵循的架构，例如：如果是对象，则为 "field1 STRING, field2 INT"；或者如果是数组，则为 "ARRAY<STRING>"。

   如果是数组，则需要架构；但对于对象，如果未指定架构，则将使用数据推断出架构。为了减少推断架构的影响（尤其是在大型数据集上），您可以通过输入用于推断架构的样本比率来避免读取整个数据两次。如果该值小于 1，则使用相应的随机样本比率来推断架构。如果数据可靠且对象在行间一致，则可以使用较小的比率（例如 0.1）来提高性能。
5. （可选）在转换选项卡上，如果要同时保留原始字符串列和已解析的列，则可以输入新的列名。

提取 JSON 路径

此转换提取从 JSON 字符串列中提取新列。当您只需要几个数据元素并且不想将整个 JSON 内容导入表架构时，此转换非常有用。

将提取 JSON 路径转换节点添加到作业图

1. 打开资源面板，然后选择提取 JSON 路径将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. 在节点属性面板上，输入作业图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 在转换选项卡上，选择包含 JSON 字符串的列。输入以逗号分隔的多个 JSON 路径表达式，每个表达式都引用如何从 JSON 数组或对象中提取值。例如，如果 JSON 列包含一个属性为“prop_1”和“prop2”的对象，则可以提取两个对象，指定它们的名称“prop_1，prop_2”。

   如果 JSON 字段包含特殊字符，例如，要从此 JSON `{"a. a": 1}` 中提取属性，则可以使用路径 `$['a. a']`。唯一的例外是逗号，因为它是为分隔路径而保留的。然后为每个路径输入相应的列名，用逗号分隔。
4. （可选）在转换选项卡上，您可以勾选提取后删除 JSON 列，当提取所需部分后不需要其余的 JSON 数据时，这是有意义的。

使用正则表达式提取字符串片段

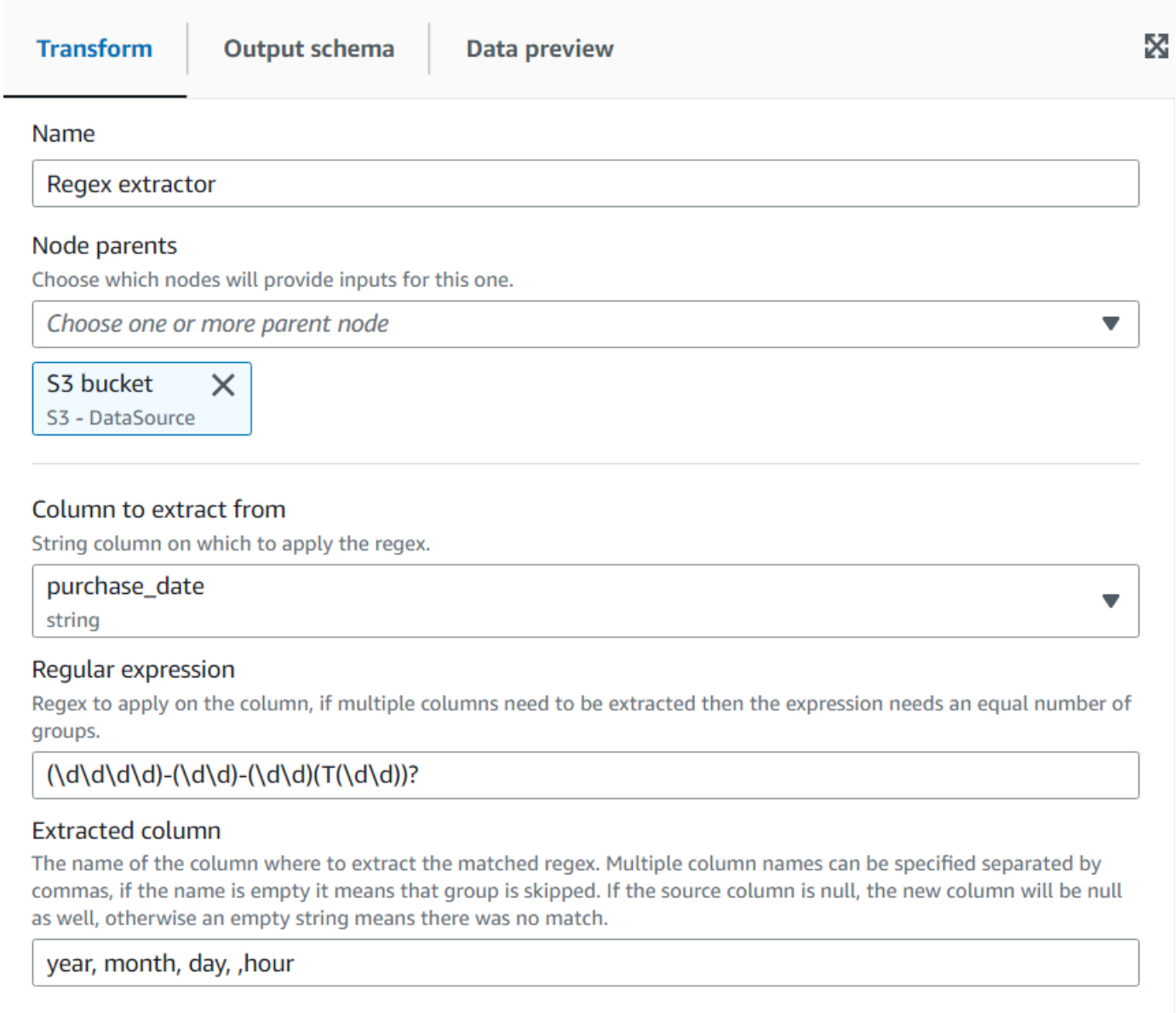
此转换使用正则表达式提取字符串片段并从中创建新列，如果使用正则表达式组则创建多列。

将正则表达式提取器转换节点添加到作业图

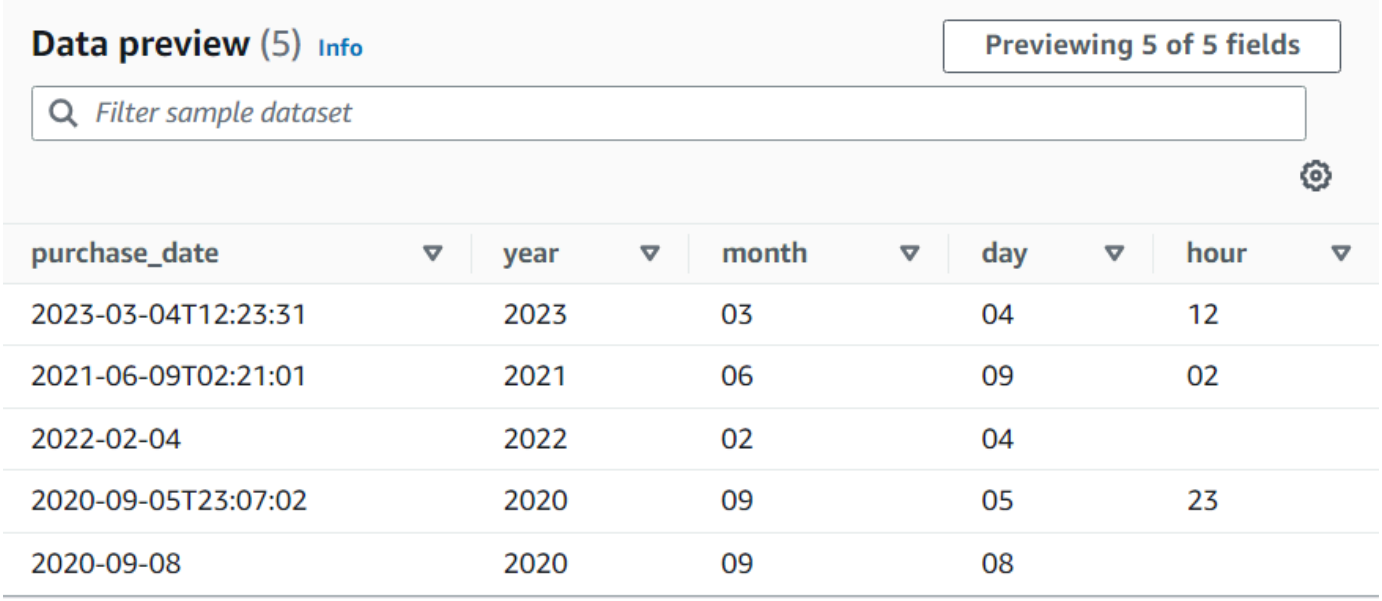
1. 打开资源面板，然后选择正则表达式提取器将新转换添加到作业图。添加节点时选择的节点将是其父节点。
2. 在节点属性面板上，输入作业图中节点的名称。如果尚未选择父节点，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。
3. 在转换选项卡上，输入正则表达式和需要应用正则表达式的列。然后输入用于存储匹配字符串的新列的名称。仅当源列为空时，新列才会为空；如果正则表达式不匹配，则该列将为空。

   如果正则表达式使用组，则会有一个用逗号分隔的相应列名，但是您可以通过将列名留空来跳过组。

例如，如果您有一列“purchase_date”，其中包含同时使用长和短 ISO 日期格式的字符串，则需要提取年、月、日和小时（如果有）。请注意：小时组是可选的，否则在不可用的行中，所有提取的组都将是空字符串（因为正则表达式不匹配）。在这种情况下，我们不希望该组将时间设为可选，而是将内部时间设为可选；因此我们将名称留空并且不会被提取（该组将包含 T 字符）。



最终呈现数据预览：



purchase_date	year	month	day	hour
2023-03-04T12:23:31	2023	03	04	12
2021-06-09T02:21:01	2021	06	09	02
2022-02-04	2022	02	04	
2020-09-05T23:07:02	2020	09	05	23
2020-09-08	2020	09	08	

创建自定义转换

如果需要对数据执行更复杂的转换，或者希望将数据属性键添加到数据集，则可以为您的任务图添加 Custom code (自定义代码) 转换。自定义代码节点允许您输入执行转换的脚本。

使用自定义代码时，必须使用架构编辑器来指示通过自定义代码对输出所做的更改。编辑架构时，您可以执行以下操作：

- 添加或删除数据属性键
- 更改数据属性键的数据类型
- 更改数据属性键的名称。
- 重构嵌套属性键

您必须使用 SelectFromCollection 转换，从自定义转换节点的结果中选择单个 `DynamicFrame`，然后将输出发送到目标位置。

使用以下任务，将自定义转换节点添加到任务图。

将自定义代码转换节点添加到任务图

将自定义转换节点添加到任务图

1. （可选）打开资源面板，然后选择自定义转换将自定义转换添加到作业图。

2. 在 Node properties (节点属性) 选项卡上，输入任务图中节点的名称。如果尚未选择父节点，或者您希望为自定义转换选择多个输入，请从 Node parents (父节点) 列表中选择一个节点，用作转换的输入源。

输入自定义转换节点的代码

您可以将代码输入或复制到输入字段。任务使用此代码执行数据转换。您可以以 Python 或 Scala 提供代码片段。该代码应采用一个或多个 DynamicFrames 作为输入，并返回 DynamicFrames。

输入自定义转换节点的脚本

1. 在任务图中选择了自定义变换节点后，选择 Transform (转换) 选项卡。
2. 在标题 Code block (代码数据块) 下的文本输入字段中，粘贴或输入转换节点。您使用的代码必须与 Job details (任务详细信息) 选项卡上为任务指定的语言匹配。

   当引用代码中的输入节点时，AWS Glue Studio 会命名根据创建顺序返回的 DynamicFrames 任务图节点。在代码中使用以下命名方法之一：

   - 经典代码生成 – 使用功能名称引用任务图中的节点。
     - 数据源节点：DataSource0、DataSource1、DataSource2 等。
     - 转换节点：Transform0、Transform1、Transform2 等。
   - 新代码生成 – 使用节点的 Node properties（节点属性）选项卡上指定的名称，并附加“_node1”、“_node2”等。例如，S3bucket_node1、ApplyMapping_node2、S3bucket_node2、MyCustomNodeName_node

   有关新代码生成器的更多信息，请参阅 脚本代码生成。

以下示例显示了要在代码框中输入的代码的格式：

Python

下面的示例采用收到的第一个 DynamicFrame，将其转换为 DataFrame 以应用本地筛选方法（仅保留拥有超过 1000 票的记录），然后在返回前将其转换回 DynamicFrame。

```
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

下面的示例采用收到的第一个 DynamicFrame，将其转换为 DataFrame 以应用本地筛选方法（仅保留拥有超过 1000 票的记录），然后在返回前将其转换回 DynamicFrame。

```
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)
  }
}
```

编辑自定义转换节点的架构

使用自定义转换节点时，AWS Glue Studio 无法自动推断由转换创建的输出架构。您可以使用架构编辑器描述由自定义转换代码实现的架构更改。

自定义代码节点可以有任意数量的父节点，每个节点都提供 DynamicFrame 作为自定义代码的输入。自定义代码节点会返回 DynamicFrames。用作输入的每个 DynamicFrame 都具有关联架构。您必须添加架构，描述自定义代码节点返回的每个 DynamicFrame。

> **Note**
>
> 当您在自定义转换上设置自己的架构时，AWS Glue Studio 不会沿用上一个节点中的架构。要更新架构，请选择“Custom transform node”（自定义转换节点），然后选择“Data preview”（数据预览）选项卡。生成预览后，选择“Use Preview Schema”（使用预览架构）。该架构随后将会替换为使用预览数据的架构。

编辑自定义转换节点的架构

1. 在任务图中选择自定义转换节点后，在节点详细信息面板中，选择 Output schema (输出架构) 选项卡。
2. 选择 Edit (编辑) 以更改架构。

如果您有嵌套的数据属性键（如数组或对象），则可以选择 Expand-Rows (展开行) 图标（ ），展开子数据属性键的列表。选择此图标后，它将更改为 Collapse-Rows (折叠行) 图标（ ），您可以选择折叠子属性键列表。

3. 使用页面右侧部分中的以下操作修改架构：

   - 要重命名属性键，请将光标放在 Key (键) 文本框，然后输入新名称。
   - 要更改属性键的数据类型，请使用列表为属性键选择新数据类型。
   - 要为架构添加新的顶级属性，请选择 Cancel (取消) 按钮左侧的 Overflow (溢出) ( ) 图标，然后选择 Add root key (添加根键)。
   - 要将子属性键添加到架构，请选择与父键关联的 Add-Key (添加键) 图标 。输入子键的名称，然后选择数据类型。
   - 要从架构中删除属性键，请选择键名称最右侧的 Remove (删除) 图标（ ）。

4. 如果您的自定义转换代码使用多个 `DynamicFrames`，您可以添加额外的输出架构。

   - 要添加新的空架构，请选择 Overflow (溢出) ( ) 图标，然后选择 Add output schema (添加输出架构)。
   - 要将现有架构复制到新的输出架构，请确保要复制的架构在架构选择器中显示。选择 Overflow (溢出) ( ) 图标，然后选择 Duplicate (重复)。

   要删除输出架构，请确保要复制的架构在架构选择器中显示。选择 Overflow (溢出) ( ) 图标，然后选择 Delete (删除)。

5. 将新的根键添加到新架构或编辑重复的键。

6. 修改输出架构时，请选择 Apply (应用) 保存您的更改并退出架构编辑器。

   如果您不想保存更改，请选择 Cancel (取消) 按钮。

配置自定义转换输出

自定义代码转换会返回 DynamicFrames，即使结果集中只有一个 DynamicFrame。

处理自定义转换节点的输出

1. 添加 SelectFromCollection 转换节点，该节点将自定义转换节点作为其父节点。更新此转换以指示要使用的数据集。请参阅使用 SelectFromCollection 选择要保留的数据集了解更多信息。
2. 如果要使用由自定义转换节点生成的其他 DynamicFrames，则将其他 SelectFromCollection 转换添加任务图。

   考虑以下情况：添加自定义转换节点以将航班数据集拆分为多个数据集，但在每个输出架构中复制某些标识属性键，如航班日期或航班号。为每个输出架构添加 SelectFromCollection 转换节点，以自定义转换节点为父节点。
3. （可选）然后，您可以使用每个 SelectFromCollection 转换节点作为任务中其他节点的输入，或作为数据目标节点的父节点。

AWS Glue 自定义视觉转换

可通过自定义视觉转换创建转换并使其可用于 AWS Glue Studio 作业。自定义视觉转换可以让可能不熟悉编码的 ETL 开发人员能够使用 AWS Glue Studio 界面搜索和使用不断增长的转换库。

您可以创建自定义视觉转换，然后将其上传到 Amazon S3，以便可通过 AWS Glue Studio 中的可视化编辑器使用来处理这些作业。

主题

开始使用自定义视觉转换

要创建自定义视觉转换，请执行以下步骤。

- 第 1 步。创建 JSON 配置文件
- 第 2 步。实施转换逻辑
- 第 3 步。验证自定义视觉转换
- 第 4 步。根据需要更新自定义视觉转换
- 第 5 步。在 AWS Glue Studio 中使用自定义视觉转换

首先设置 Amazon S3 存储桶，然后继续执行 Step 1（步骤 1）。Create a JSON config file（创建 JSON 配置文件）。

先决条件

客户提供的转换保存在客户 AWS 账户中。该账户是这些转换的所有者，因此拥有查看（搜索和使用）、编辑或删除这些转换的所有权限。

要在 AWS Glue Studio 中使用自定义转换，您需要在该 AWS 账户中创建两个文件并将其上传到 Amazon S3 资产存储桶：

- Python 文件 — 包含转换函数
- JSON 文件 — 描述了转换。这也称作定义转换所需的配置文件。

要将文件配对在一起，请对两者使用相同的名称。例如：

- myTransform.json
- myTransform.py

或者，您可以通过提供包含该图标的 SVG 文件来为自定义视觉转换指定一个自定义图标。要将文件配对在一起，请对图标使用相同的名称：

- myTransform.svg

AWS Glue Studio 将使用它们各自的文件名自动匹配它们。对于任何现有模块，文件名不能相同。

转换文件名的建议命名惯例

AWS Glue Studio 会将您的文件作为模块（例如，`import myTransform`）导入作业脚本。因此，您的文件名必须遵循为 python 变量名（标识符）设置的相同命名规则。具体而言，它们必须以字母或下划线开头，且完全由字母、数字和/或下划线组成。

> **Note**
>
> 确保您的转换文件名与现有的已加载 python 模块（例如，`sys, array, copy` 等）不冲突，以避免意外的运行时问题。

设置 Amazon S3 存储桶

您创建的转换存储在 Amazon S3 中，归您的 AWS 账户所有。您只需将文件（json 和 py）上传到当前存储所有作业脚本的 Amazon S3 资产文件夹（例如，`s3://aws-glue-assets-<accountid>-<region>/transforms`），即可创建新的自定义视觉转换。如果使用自定义图标，也请将其上传。默认情况下，AWS Glue Studio 将读取同一 S3 存储桶中的 /transforms 文件夹中的所有 .json 文件。

第 1 步。创建 JSON 配置文件

需要一个 JSON 配置文件来定义和描述您的自定义视觉转换。配置文件的架构如下所示。

JSON 文件结构

字段

- `name: string` — （必需）用于标识转换的转换系统名称。遵循为 python 变量名（标识符）设置的相同命名规则。具体而言，它们必须以字母或下划线开头，且完全由字母、数字和/或下划线组成。
- `displayName: string` — （可选）显示在 AWS Glue Studio 可视化作业编辑器中的转换名称。如果未指定 `displayName`，则将 `name` 用作 AWS Glue Studio 中的转换名称。
- `description: string` — （可选）转换描述显示在 AWS Glue Studio 中且可搜索。
- `functionName: string` — （必需）Python 函数名称用于标识要在 Python 脚本中调用的函数。
- `path: string` — （可选）Python 源文件的完整 Amazon S3 路径。如果未指定，则 AWS Glue 会通过文件名匹配将 .json 和.py 文件配对在一起。例如，JSON 文件的名称 `myTransform.json` 将与位于同一 Amazon S3 位置的 Python 文件 `myTransform.py` 配对。
- `parameters: Array of TransformParameter object` — （可选）在 AWS Glue Studio 可视化编辑器中配置参数时要显示的参数列表。

TransformParameter 字段

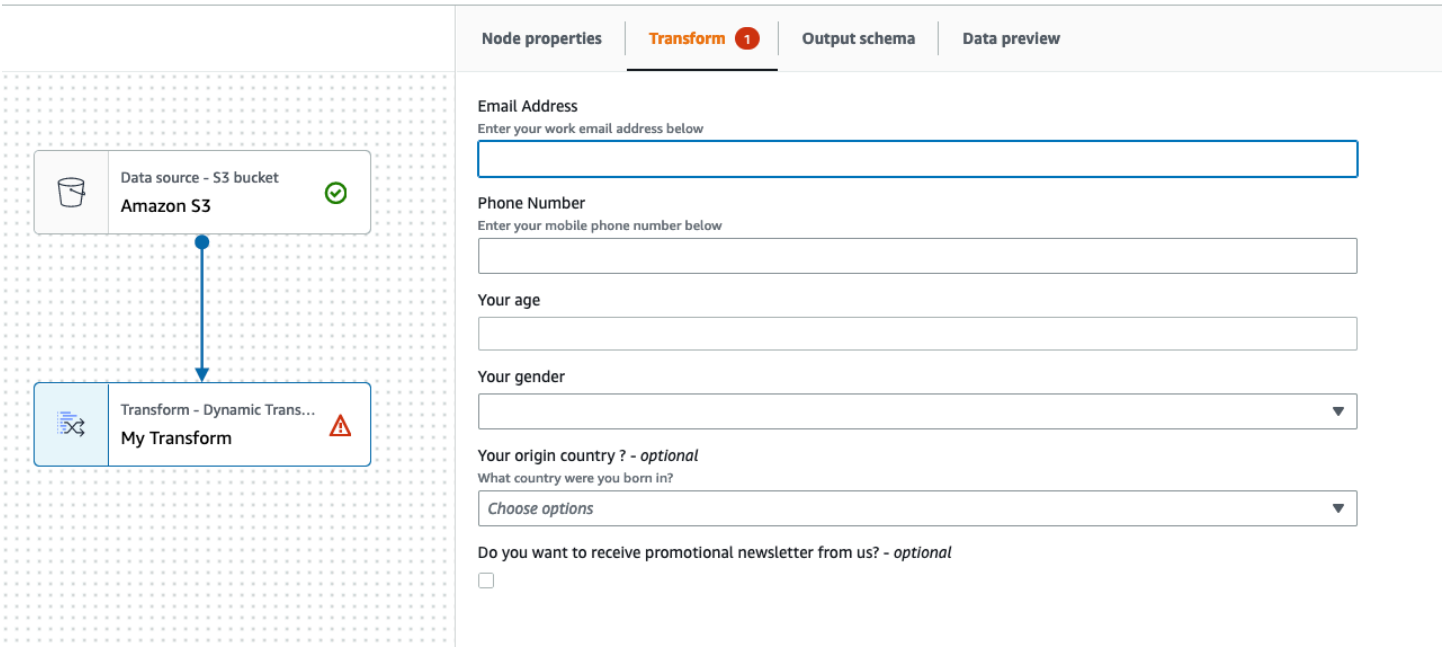
- `name: string` — （必需）要在作业脚本中作为命名参数传递给 python 函数的参数名称。遵循为 python 变量名（标识符）设置的相同命名规则。具体而言，它们必须以字母或下划线开头，且完全由字母、数字和/或下划线组成。
- `displayName: string` — （可选）显示在 AWS Glue Studio 可视化作业编辑器中的转换名称。如果未指定 `displayName`，则将 `name` 用作 AWS Glue Studio 中的转换名称。
- `type: string` — （必需）接受常见 Python 数据类型的参数类型。有效值：'str' | 'int' | 'float' | 'list' | 'bool'。
- `isOptional: boolean` — （可选）确定该参数是否为可选参数。默认情况下，所有参数都是必需的。
- `description: string` — （可选）AWS Glue Studio 中会显示描述，以帮助用户配置转换参数。
- `validationType: string` — （可选）定义验证此参数的方式。目前，仅支持正则表达式。默认情况下，验证类型设置为 `RegularExpression`。
- `validationRule: string` — （可选）用于在 `validationType` 设置为 `RegularExpression` 时在提交前验证表单输入的正则表达式。正则表达式语法必须符合 RegExp Ecmascript 规范。
- `validationMessage: string` — （可选）验证失败时显示的消息。
- `listOptions: An array of TransformParameterListOption object` 或 `string` 或字符串值“column”— （可选）显示在“选择”或“多选”UI 控件中的选项。接受以逗号分隔的值列表或类型为 `TransformParameterListOption` 的强类型 JSON 对象。它还可以通过指定字符串值“column”来动态填充父节点架构中的列列表。
- `listType: string` — （可选）定义类型 = 'list' 时的选项类型。有效值：'str' | 'int' | 'float' | 'list' | 'bool'。接受常见 Python 数据类型的参数类型。

转换参数列表选项字段

- `value: string | int | float | bool` — （必需）选项值。
- `label: string` — （可选）显示在选择下拉列表中的选项标签。

AWS Glue Studio 中的转换参数

默认情况下，除非在 .json 文件中标记为 `isOptional`，否则参数为必填。在 AWS Glue Studio 中，参数显示在 Transform（转换）选项卡中。该示例显示了用户定义的参数，例如电子邮件地址、电话号码、您的年龄、您的性别和您的原籍国。

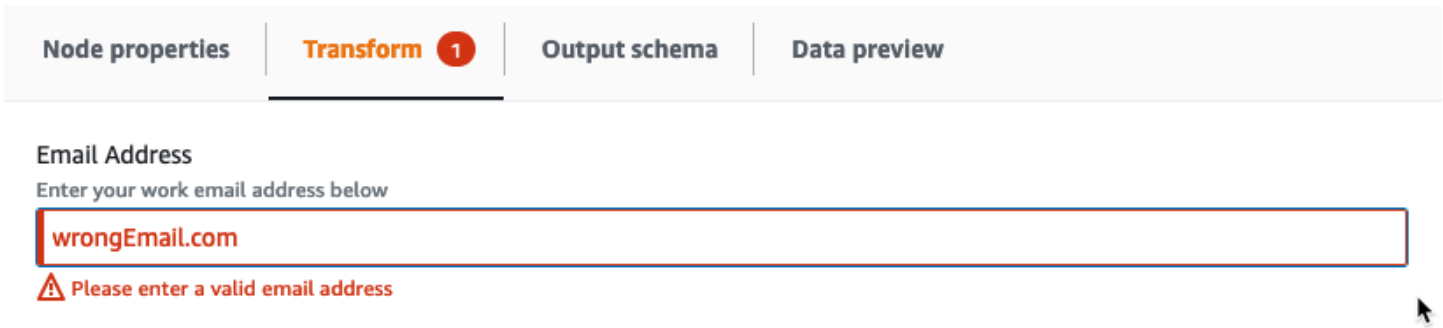


通过在 json 文件中使用正则表达式指定 `validationRule` 参数并在 `validationMessage` 中指定验证消息，在 AWS Glue Studio 中强制执行某些验证。

```
    "validationRule": "^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})$",
    "validationMessage": "Please enter a valid US number"
```

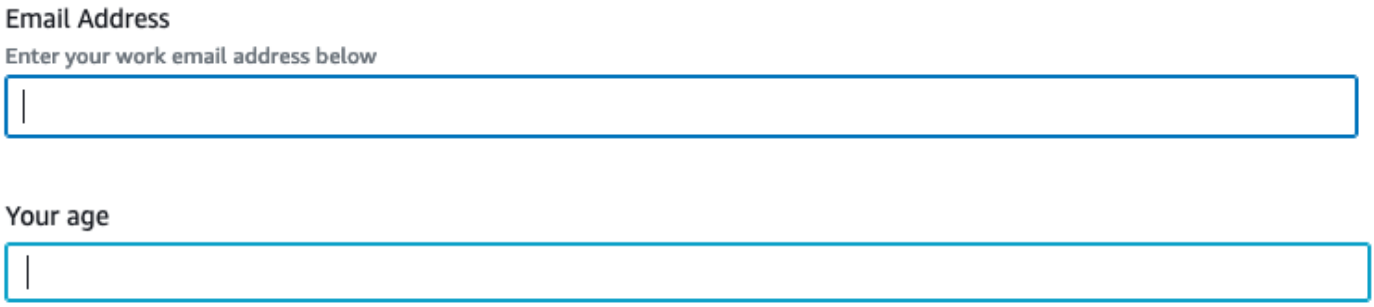
> **Note**
>
> 由于验证是在浏览器中进行的，因此您的正则表达式语法必须符合 RegExp Ecmascript 规范。这些正则表达式不支持 Python 语法。

添加验证可以防止用户保存用户输入不正确的作业。AWS Glue Studio 将显示此示例中所示的验证消息：

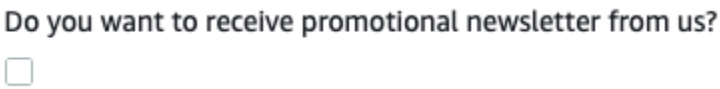


参数会根据参数配置显示在 AWS Glue Studio 中。

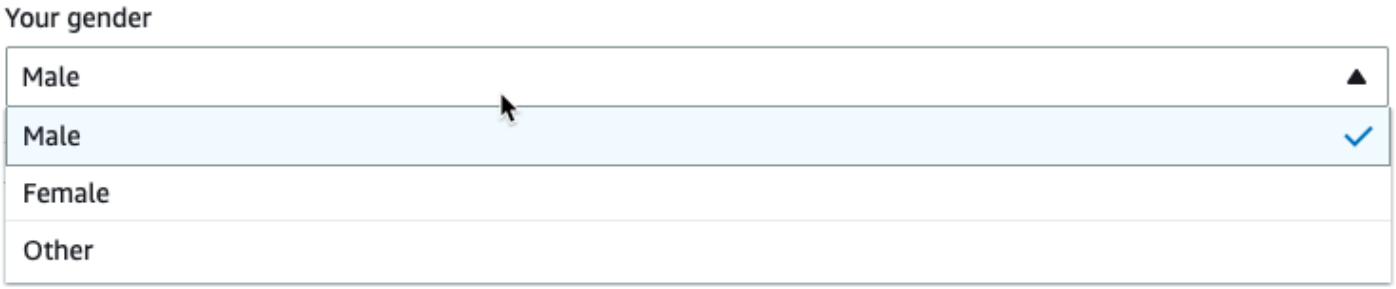
- 当 `type` 为以下任一项时：`str`、`int` 或 `float`，将显示一个文本输入字段。例如，屏幕截图显示了“Email Address”和“Your age”参数的输入字段。



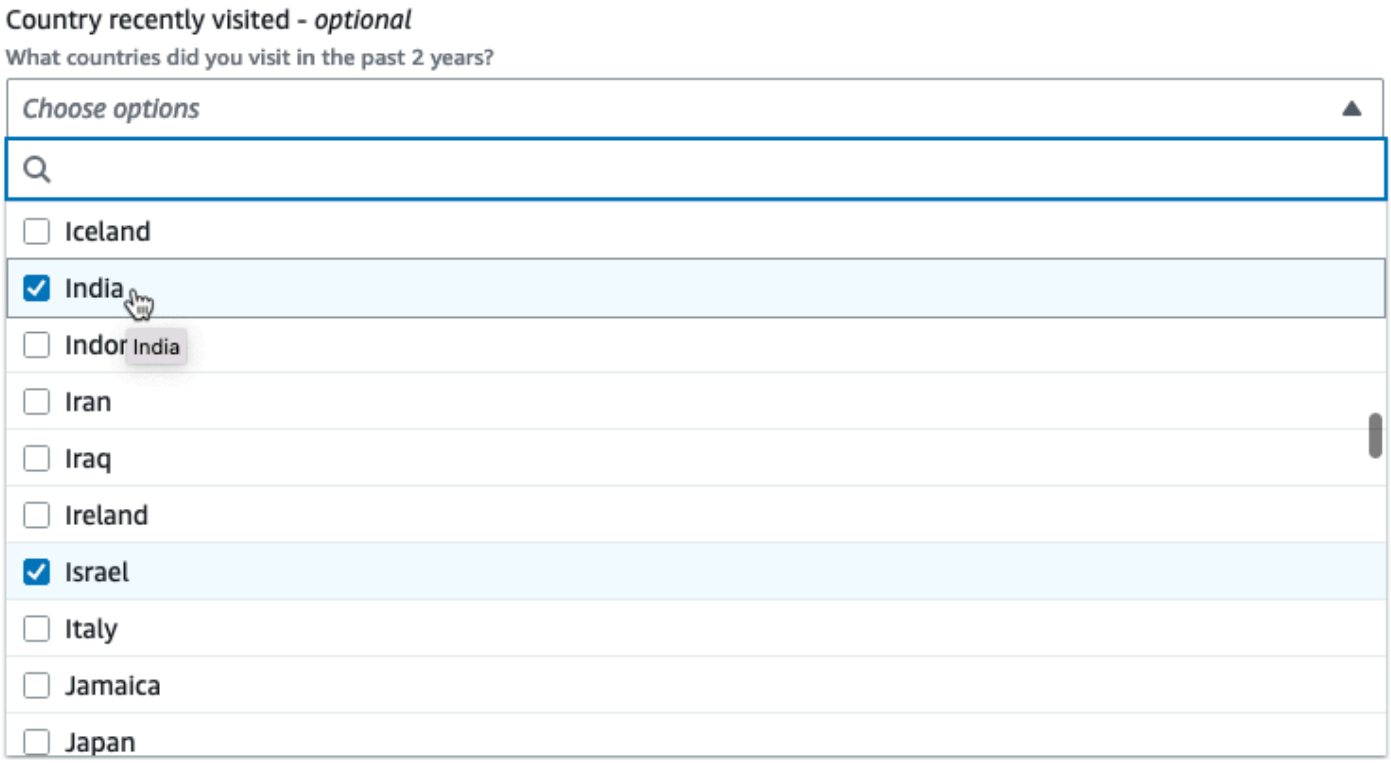
- 当 `type` 为 `bool` 时，将显示一个复选框。



- 当 `type` 为 `str` 且提供了 `listOptions` 时，将显示一个单选列表。



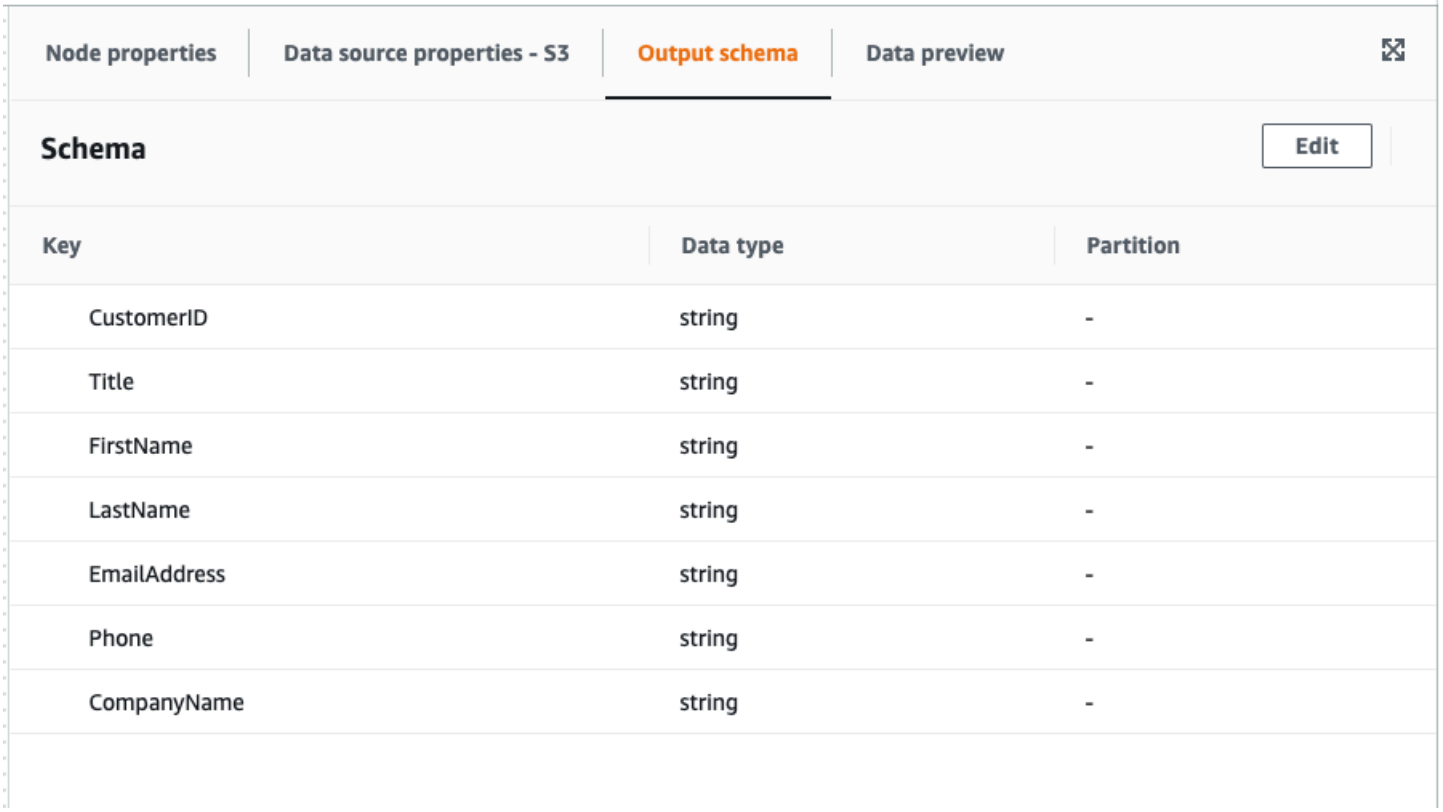
- 当 `type` 为 `list` 且提供了 `listOptions` 和 `listType` 时，将显示一个多选列表。



将列选择器显示为参数

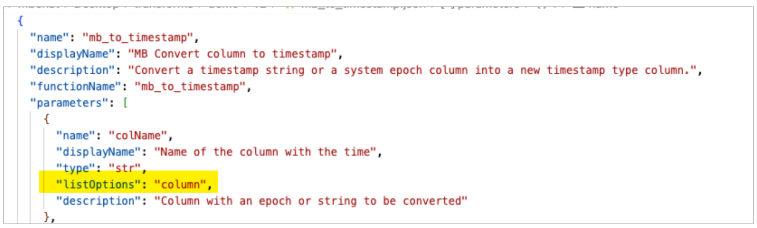
如果配置要求用户从架构中选择一列，则可以显示列选择器，这样用户就无需键入列名。通过将 `listOptions` 字段设置为“column”，AWS Glue Studio 可以根据父节点输出架构动态显示列选择器。AWS Glue Studio 可以显示单列或多列选择器。

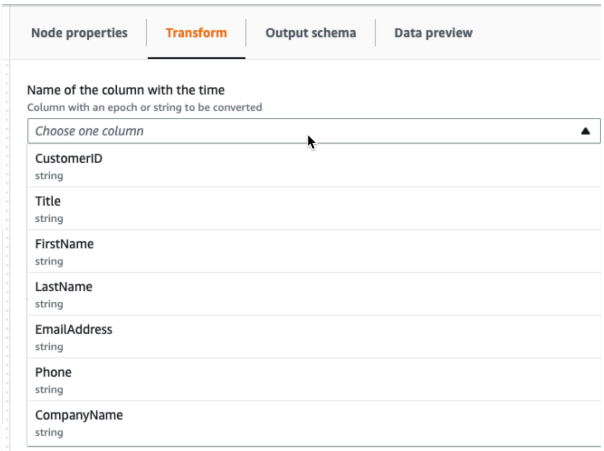
以下示例使用架构：



要将自定义视觉转换参数定义为显示单列，请执行以下操作：

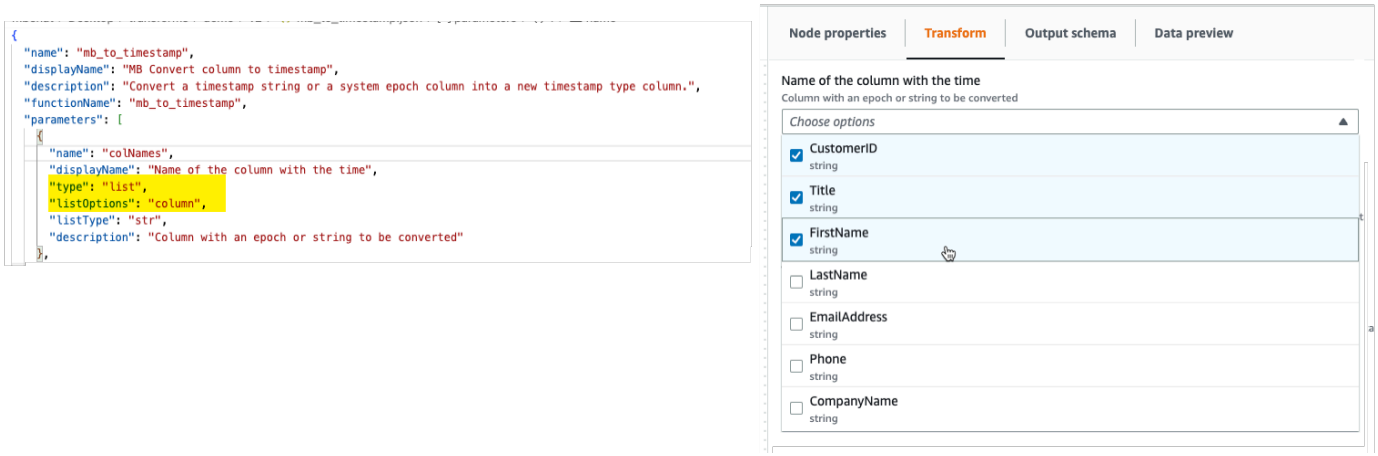
1. 在您的 JSON 文件中，将 `parameters` 对象的 `listOptions` 值设置为“column”。这允许用户从 AWS Glue Studio 中的选择列表中选择一列。





2. 您也可以将参数定义为以下值，从而允许选择多列：

   - `listOptions: "column"`
   - `type: "list"`



第 2 步。实施转换逻辑

> **Note**
>
> 自定义视觉转换仅支持 Python 脚本。不支持 Scala。

要添加实现 .json 配置文件定义的函数的代码，建议将 Python 文件放在与 .json 文件相同的位置，名称相同但扩展名为“.py”。AWS Glue Studio 自动将 .json 和 .py 文件配对，因此您无需在配置文件中指定 Python 文件的路径。

在 Python 文件中，添加已声明的函数，配置命名参数，并将其注册到 `DynamicFrame` 中使用。以下是 Python 文件的示例：

```
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
```

建议使用 AWS Glue 笔记本作为开发和测试 python 代码的最快方法。请参阅开始在 AWS Glue Studio 中使用笔记本。

为了说明如何实现转换逻辑，以下示例中的自定义视觉转换是一种转换，用于过滤传入数据，仅保留与美国特定州相关的数据。.json 文件包含作为 `custom_filter_state` 的 `functionName` 的参数和两个参数（类型为 “str”的“state”和“colName”）。

示例配置 .json 文件是：

```
{
"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 中实现配套脚本

1. 启动 AWS Glue 笔记本并运行为启动会话提供的初始单元。运行初始单元会创建所需的基本组件。
2. 创建一个执行示例中描述的过滤的函数并将其注册到 `DynamicFrame`。复制下面的代码并粘贴到 AWS Glue 笔记本的单元中。

```
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. 创建或加载示例数据，以测试同一个单元或新单元中的代码。如果您在新单元中添加示例数据，请不要忘记运行该单元。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. 测试使用不同的参数验证“custom_filter_state”：

```
[14]: dynf1.custom_filter_state("state", "NY").show()
      {"count": 2, "state": "NY"}
```

5. 运行多个测试后，使用 .py 扩展名保存代码，并使用镜像 .json 文件名的名称命名 .py 文件。.py 和 .json 文件应位于同一个转换文件夹中。

   复制以下代码并将其粘贴到文件中，并使用 .py 文件扩展名对其进行重命名。

```
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，打开可视化作业，然后从可用 Transforms（转换）列表中选择该转换，将其添加到该作业中。

   要在 Python 脚本代码中重用此转换，请在作业中的“引用文件路径”下将 Amazon S3 路径添加到 .py 文件，然后在脚本中，将 python 文件的名称（不带扩展名）添加到文件顶部，将其导入。例如：`import` <文件名（不带扩展名）>

第 3 步。对 AWS Glue Studio 中的自定义视觉转换进行验证和故障排除

AWS Glue Studio 在将自定义视觉转换加载到 AWS Glue Studio 之前验证 JSON 配置文件。验证包括：

- 是否存在必填字段
- JSON 格式验证
- 参数不正确或无效
- .py 和.json 文件同时存在于同一 Amazon S3 路径中
- 匹配 .py 和.json 的文件名

如果验证成功，则转换将在可视化编辑器的可用 Actions（操作）列表中列出。如果提供了自定义图标，则该图标应在操作旁边可见。

如果验证失败，则 AWS Glue Studio 不加载自定义视觉转换。

第 4 步。根据需要更新自定义视觉转换

创建并使用后，只要转换遵循相应的 json 定义，就可以更新转换脚本：

- 分配给 DynamicFrame 时使用的名称与 json `functionName` 非常匹配。
- 函数参数必须在 json 文件中定义，如 第 1 步。创建 JSON 配置文件 中所述。
- Python 文件的 Amazon S3 路径无法更改，因为任务直接依赖于该路径。

> **Note**
>
> 如果需要进行任何更新，请确保脚本和 .json 文件持续更新，并且使用新的转换再次正确保存所有可视作业。如果更新后未保存可视作业，则不会应用和验证更新。如果 Python 脚本文件被重命名或未放在 .json 文件旁边，则需要在 .json 文件中指定完整路径。

自定义图标

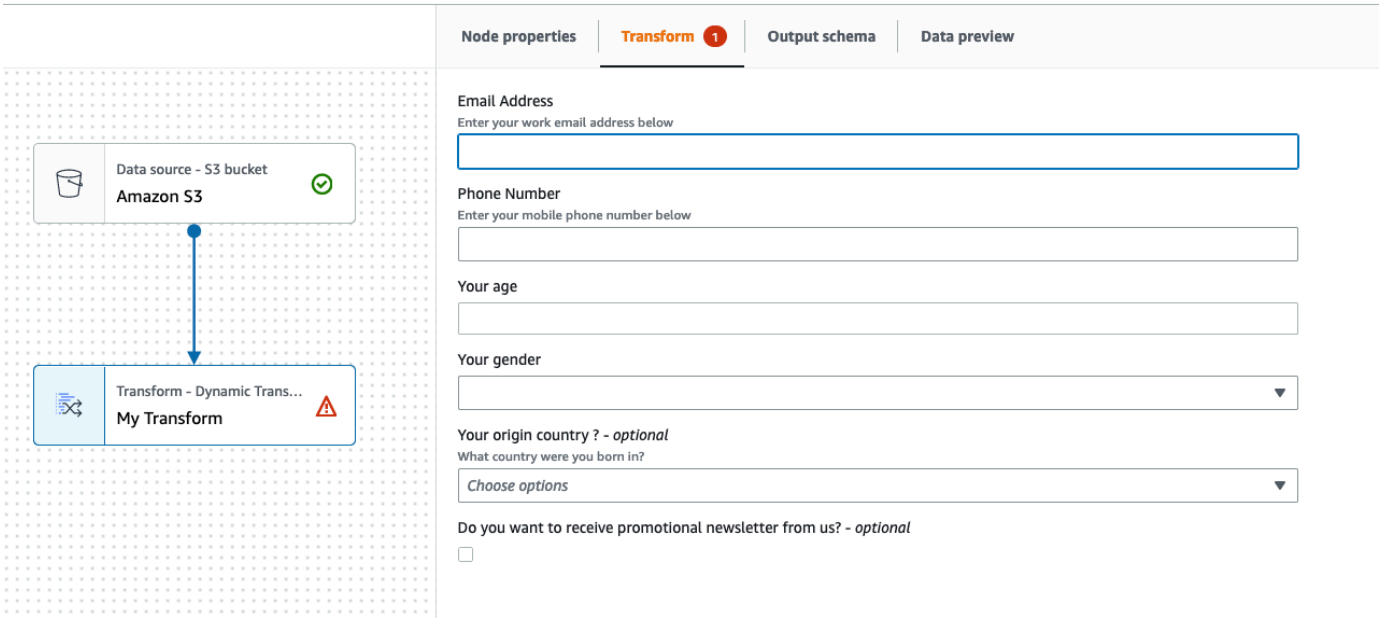
如果您确定操作的默认图标无法直观地将其区分为工作流程的一部分，则可以提供自定义图标，如 the section called “ 开始使用自定义视觉转换 ” 中所述。您可以通过更新 Amazon S3 中托管的相应 SVG 来更新图标。

为了获得最佳效果，请按照 Cloudscape Design System 的指导方针，将图像设计为以 32x32px 的分辨率查看。有关 Cloudscape 准则的更多信息，请参阅 Cloudscape 文档

第 5 步。在 AWS Glue Studio 中使用自定义视觉转换

要在 AWS Glue Studio 中使用自定义视觉变换，请上传配置和源文件，然后从 Action（操作）菜单中选择转换。任何需要值或输入的参数都可以在 Transform（转换）选项卡中找到。

1. 将这两个文件（Python 源文件和 JSON 配置文件）上传到存储作业脚本的 Amazon S3 资产文件夹。默认情况下，AWS Glue 将读取同一 Amazon S3 存储桶中的 /transforms 文件夹中的所有 JSON 文件。
2. 从 Action（操作）菜单中，选择自定义视觉转换。它使用您在 .json 配置文件中指定的转换 `displayName` 或名称命名。
3. 输入在配置文件中配置的任何参数的值。



用法示例

下面是 .json 配置文件中所有可能参数的示例。

```
{
  "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
    }
  ]
}
```

自定义可视化脚本的示例

以下示例执行等效转换。但是，第二个示例（SparkSQL）是最干净、最高效的，其次是 Pandas UDF，最后是第一个示例中的低级映射。以下示例是将两列相加的简单转换的完整示例：

```
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
```

以下示例是利用 SparkSQL API 的等效转换。

```
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
```

以下示例使用相同的转换，但使用的是 pandas UDF，这比使用普通 UDF 更有效。有关编写 pandas UDF 的更多信息，请参阅：Apache Spark SQL 文档。

```
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
```

视频

以下视频介绍了可视化自定义转换，并演示了如何使用它们。

将数据湖框架与 AWS Glue Studio 配合使用

概述

开源数据湖框架简化了对存储在 Amazon S3 上的数据湖中的文件的增量数据处理。AWS Glue 3.0 及更高版本支持以下开源数据湖存储框架：

- Apache Hudi
- Linux Foundation Delta Lake
- Apache Iceberg

截至 AWS Glue 4.0，AWS Glue 为这些框架提供原生支持，因此您可以以交易一致的方式读取和写入存储在 Amazon S3 中的数据。无需安装单独的连接器或完成额外的配置步骤即可在 AWS Glue 作业中使用这些框架。

通过 Spark 脚本编辑器任务，数据湖框架可用作 AWS Glue Studio 中的来源或目标。有关使用 Apache Hudi、Apache Iceberg 和 Delta Lake 的更多信息，请参阅：在 AWS Glue ETL 任务中使用数据湖框架。

从 AWS Glue 流式处理源创建开放表格式

AWS Glue 流式处理 ETL 作业会持续消耗来自流式处理源的数据，清理和转换动态数据，并在几秒钟内使其可用于分析。

AWS 提供的多种服务都可以满足您的需求。AWS Database Migration Service 等数据库复制服务可以将数据从您的源系统复制到 Amazon S3，后者常用于托管数据湖的存储层。尽管在支持在线源应用程序的关系数据库管理系统（RDBMS）上应用更新非常简单直接，但很难在数据湖上执行这种 CDC 流程。开源数据管理框架可简化增量数据处理和数据管道的开发，能够很好地解决这一问题。

有关更多信息，请参阅：

- 使用 AWS Glue 流式处理功能创建基于 Apache HUDI 的近实时事务处理数据湖
- 构建与 GDPR 一致的实时 Apache Iceberg 数据湖

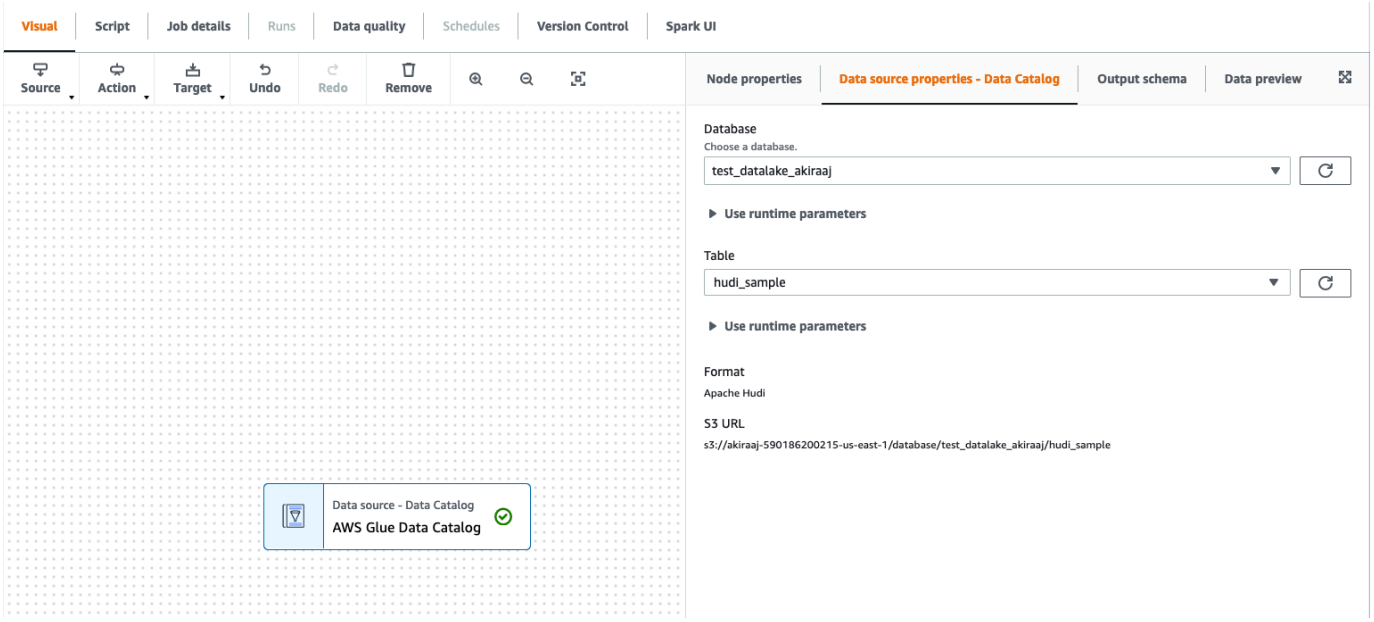
在 AWS Glue Studio 中使用 Hudi 框架

创建或编辑作业时，AWS Glue Studio 会根据您使用的 AWS Glue 版本自动为您添加相应的 Hudi 库。有关更多信息，请参阅在 AWS Glue 中使用 Hudi 框架。

在 Data Catalog 数据来源中使用 Apache Hudi 框架

要为作业添加 Hudi 数据来源格式，请执行以下操作：

1. 从“来源”菜单中选择“AWS Glue Studio Data Catalog”。
2. 在数据来源属性选项卡中，选择数据库和表。
3. AWS Glue Studio 将格式类型显示为 Apache Hudi 和 Amazon S3 URL。



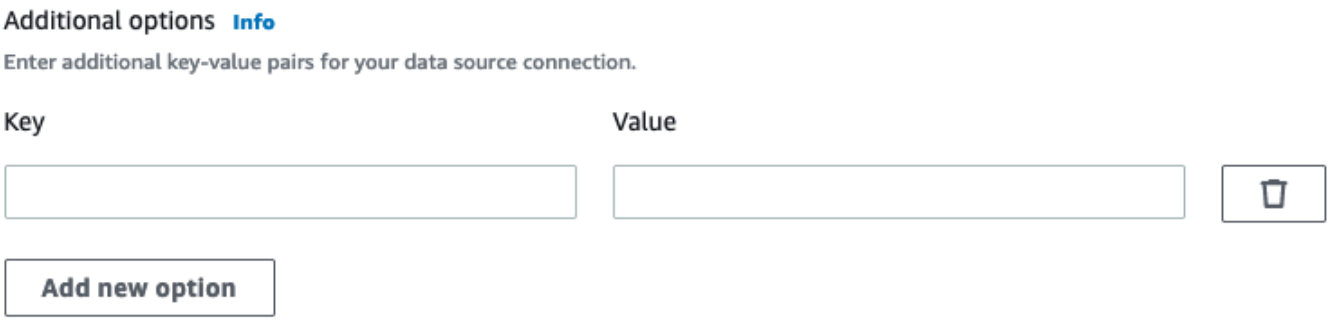
在 Amazon S3 数据来源中使用 Hudi 框架

1. 从“来源”菜单中选择 Amazon S3。
2. 如果您选择 Data Catalog 表作为 Amazon S3 来源类型，请选择数据库和表。
3. AWS Glue Studio 显示格式为 Apache Hudi 和 Amazon S3 URL。
4. 如果您选择 Amazon S3 位置作为 Amazon S3 来源类型，请通过单击浏览 Amazon S3 选择 Amazon S3 URL。
5. 在数据格式中，选择“Apache Hudi”。

> **Note**
>
> 如果 AWS Glue Studio 无法从您选择的 Amazon S3 文件夹或文件推断出架构，请选择其他选项来选择新的文件夹或文件。
> 在其他选项中，从架构推断下的以下选项中进行选择：

- 让 AWS Glue Studio 自动选择示例文件：AWS Glue Studio 将在 Amazon S3 位置选择一个示例文件，以便推断出架构。在自动取样文件字段中，您可以查看自动选择的文件。
- 从 Amazon S3 中选择示例文件：单击浏览 Amazon S3，选择要使用的 Amazon S3 文件。

6. 单击推断架构。然后可以通过单击输出架构选项卡来查看输出架构。
7. 选择其他选项以输入键值对。



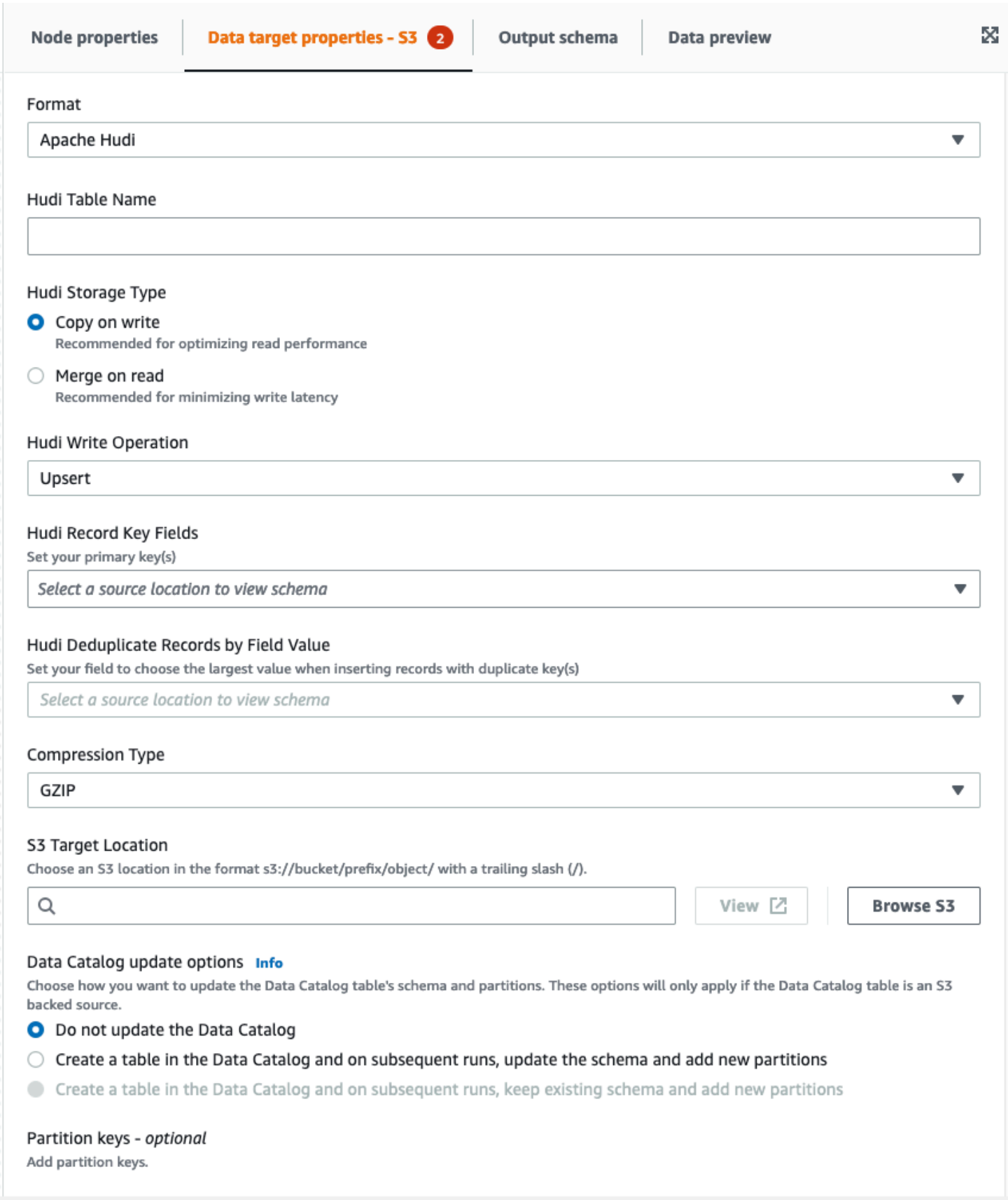
在数据目标中使用 Apache Hudi 框架

在 Data Catalog 数据目标中使用 Apache Hudi 框架

1. 从目标菜单中选择“AWS Glue Studio Data Catalog”。
2. 在数据来源属性选项卡中，选择数据库和表。
3. AWS Glue Studio 将格式类型显示为 Apache Hudi 和 Amazon S3 URL。

在 Amazon S3 数据目标中使用 Apache Hudi 框架

输入值或从可用选项中进行选择以配置 Apache Hudi 格式。有关 Apache Hudi 的更多信息，请参阅 Apache Hudi 文档。


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 表名：Hudi 表的名字。
- Hudi 存储类型：从两个选项中选择：
  - 写入时复制：可优化读取性能的推荐选项。这是默认 Hudi 存储类型。每次更新都会在写入时创建文件的新版本。
  - 读取时合并：可将写入延迟降至最低的推荐选项。更新记录到基于行的增量文件中，并根据需要进行压缩以创建新版本的列式文件。
- Hudi 写入操作：从以下选项中进行选择：
  - 更新插入：默认操作，操作时首先通过查找索引将输入记录标记为插入或更新。在您更新现有数据时推荐选择。
  - 插入：插入记录，但不检查现有记录，并可能导致重复。
  - 批量插入：插入记录，建议用于大量数据。
- Hudi 记录键字段：使用搜索栏搜索并选择主记录键。Hudi 中的记录由主键标识，主键由记录键和记录所属的分区路径组成。
- Hudi 预合并字段：实际写入之前在预合并中使用的字段。如果两个记录具有相同的键值，AWS Glue Studio 会为预合并字段选择值最大的记录。设置增量值（例如 updated_at）所属的字段。
- 压缩类型：选择一种压缩类型选项：未压缩、GZIP、LZO 或 Snappy。
- Amazon S3 目标位置：通过单击浏览 S3 来选择 Amazon S3 目标位置。
- Data Catalog 更新选项：从以下选项中进行选择：
  - Do not update the Data Catalog (请勿更新数据目录)：（默认）如果您不希望任务更新数据目录（即使架构更改或添加了新分区），请选择此选项。
  - 在 Data Catalog 中创建表并在后续运行时更新架构并添加新分区：如果选择此选项，作业将在第一次作业运行时在 Data Catalog 中创建表。在后续任务运行时，如果架构发生更改或添加了新分区，任务将更新数据目录表。

    您还必须从数据目录中选择数据库并输入表名。
  - Create a table in the Data Catalog and on subsequent runs, keep existing schema and add new partitions (在数据目录中创建表，并在后续运行时保持现有架构并添加新分区)：如果选择此选项，任务将在第一次任务运行时创建数据目录中的表。在后续任务运行时，任务只更新数据目录表以添加新分区。

    您还必须从数据目录中选择数据库并输入表名。
- Partition keys (分区键)：选择要在输出中用作分区键的列。要添加更多分区键，请选择 Add a partition key (添加分区键)。
- 其他选项：根据需要输入键值对。

通过 AWS Glue Studio 生成代码

保存作业时，如果检测到 Hudi 来源或目标，则会将以下作业参数添加到作业中：

- `--datalake-formats`：在可视化作业中检测到的数据湖格式的不同列表（可以直接选择“格式”，或间接选择由数据湖支持的目录表)。
- `--conf` ：根据 `--datalake-formats` 的值生成。例如，如果 `--datalake-formats` 的值为 'hudi'，则 AWS Glue 为此参数生成的值为 `spark.serializer=org.apache.spark.serializer.KryoSerializer –conf spark.sql.hive.convertMetastoreParquet=false`。

覆盖 AWS Glue 提供的库

要使用 AWS Glue 不支持的 Hudi 版本，可以指定自己的 Hudi 库 JAR 文件。要使用自己的 JAR 文件，请执行以下操作：

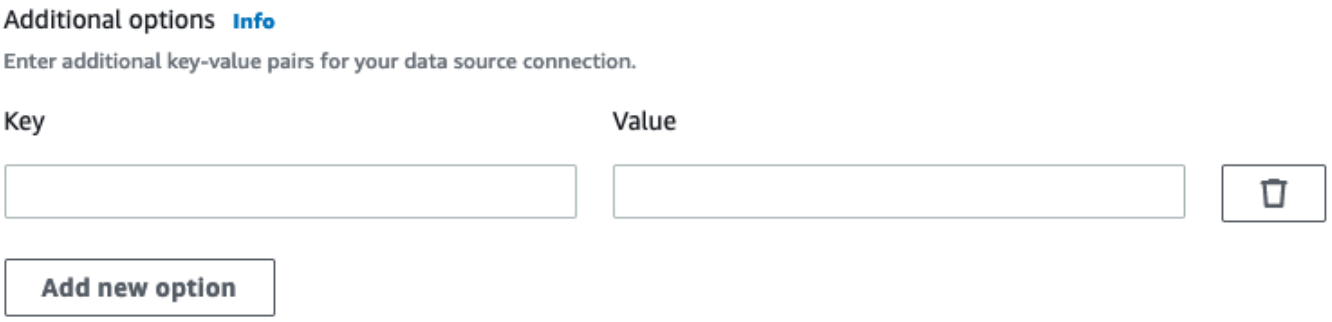
- 使用 `--extra-jars` 作业参数。例如，`'--extra-jars': 's3pathtojarfile.jar'`。有关更多信息，请参阅 AWS Glue 作业参数。
- 请勿包含 `hudi` 作为 `--datalake-formats` 作业参数的值。输入空白字符串作为值可确保 AWS Glue 不会自动为您提供任何数据湖库。有关更多信息，请参阅在 AWS Glue 中使用 Hudi 框架。

在 AWS Glue Studio 中使用 Delta Lake 框架

在数据来源中使用 Delta Lake 框架

在 Amazon S3 数据来源中使用 Delta Lake 框架

1. 从“来源”菜单中选择 Amazon S3。
2. 如果您选择 Data Catalog 表作为 Amazon S3 来源类型，请选择数据库和表。
3. AWS Glue Studio 显示格式为 Delta Lake 和 Amazon S3 URL。
4. 选择其他选项以输入键值对。例如，键值对可能为：键：timestampAsOf 和值：2023-02-24 14:16:18。



5. 如果您选择 Amazon S3 位置作为 Amazon S3 来源类型，请通过单击浏览 Amazon S3 选择 Amazon S3 URL。
6. 在数据格式中，选择 Delta Lake。

> **Note**
>
> 如果 AWS Glue Studio 无法从您选择的 Amazon S3 文件夹或文件推断出架构，请选择其他选项来选择新的文件夹或文件。
> 在其他选项中，从架构推断下的以下选项中进行选择：
>
> - 让 AWS Glue Studio 自动选择示例文件：AWS Glue Studio 将在 Amazon S3 位置选择一个示例文件，以便推断出架构。在自动取样文件字段中，您可以查看自动选择的文件。
> - 从 Amazon S3 中选择示例文件：单击浏览 Amazon S3，选择要使用的 Amazon S3 文件。

7. 单击推断架构。然后可以通过单击输出架构选项卡来查看输出架构。

在 Data Catalog 数据来源中使用 Delta Lake 框架

1. 从来源菜单中选择“AWS Glue Studio Data Catalog”。
2. 在数据来源属性选项卡中，选择数据库和表。
3. AWS Glue Studio 将格式类型显示为 Delta Lake 和 Amazon S3 URL。

> **Note**
>
> 如果尚未将 Delta Lake 来源注册为 AWS Glue 数据目录表，则有两种选择：

1. 为 Delta Lake 数据创建 AWS Glue 爬网程序。有关更多信息，请参阅如何为 Delta Lake 数据存储指定配置选项。
2. 使用 Amazon S3 数据来源选择 Delta Lake 数据来源。请参阅 在 Amazon S3 数据来源中使用 Delta Lake 框架 。

在数据目标中使用 Delta Lake 格式

在 Data Catalog 数据目标中使用 Delta Lake 格式

1. 从目标菜单中选择“AWS Glue Studio Data Catalog”。
2. 在数据来源属性选项卡中，选择数据库和表。
3. AWS Glue Studio 将格式类型显示为 Delta Lake 和 Amazon S3 URL。

在 Amazon S3 数据来源中使用 Delta Lake 格式

输入值或从可用选项中进行选择以配置 Delta Lake 格式。

- 压缩类型：选择一种压缩类型选项：未压缩或 Snappy。
- Amazon S3 目标位置：通过单击浏览 S3 来选择 Amazon S3 目标位置。
- Data Catalog 更新选项：在 Glue Studio 可视化编辑器中，此格式不支持更新 Data Catalog。
  - Do not update the Data Catalog (请勿更新数据目录)：（默认）如果您不希望任务更新数据目录（即使架构更改或添加了新分区），请选择此选项。
  - 要在 AWS Glue 作业执行后更新 Data Catalog，请运行或计划 AWS Glue 爬网程序。有关更多信息，请参阅如何为 Delta Lake 数据存储指定配置选项。
- 分区键：选择要在输出中用作分区键的列。要添加更多分区键，请选择 Add a partition key (添加分区键)。
- 可选择其他选项以输入键值对。例如，键值对可能为：键：timestampAsOf 和值：2023-02-24 14:16:18。

在 AWS Glue Studio 中使用 Apache Iceberg 框架

在数据目标中使用 Apache Iceberg 框架

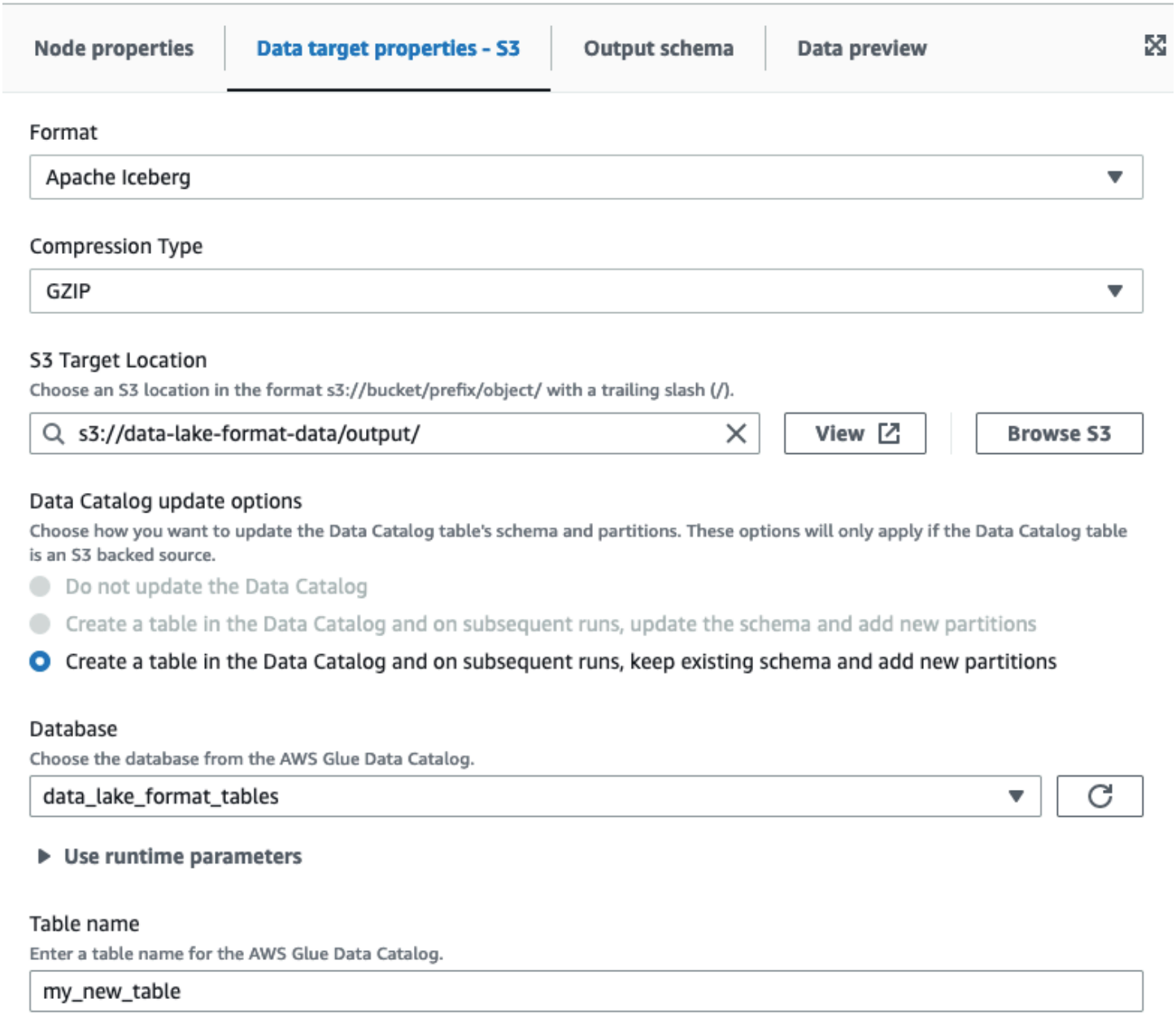
在 Data Catalog 数据目标中使用 Apache Iceberg 框架

1. 从目标菜单中选择“AWS Glue Studio Data Catalog”。
2. 在数据来源属性选项卡中，选择数据库和表。
3. AWS Glue Studio 将格式类型显示为 Apache Iceberg 和 Amazon S3 URL。

在 Amazon S3 数据目标中使用 Apache Iceberg 框架

输入值或从可用选项中进行选择以配置 Apache Iceberg 格式。

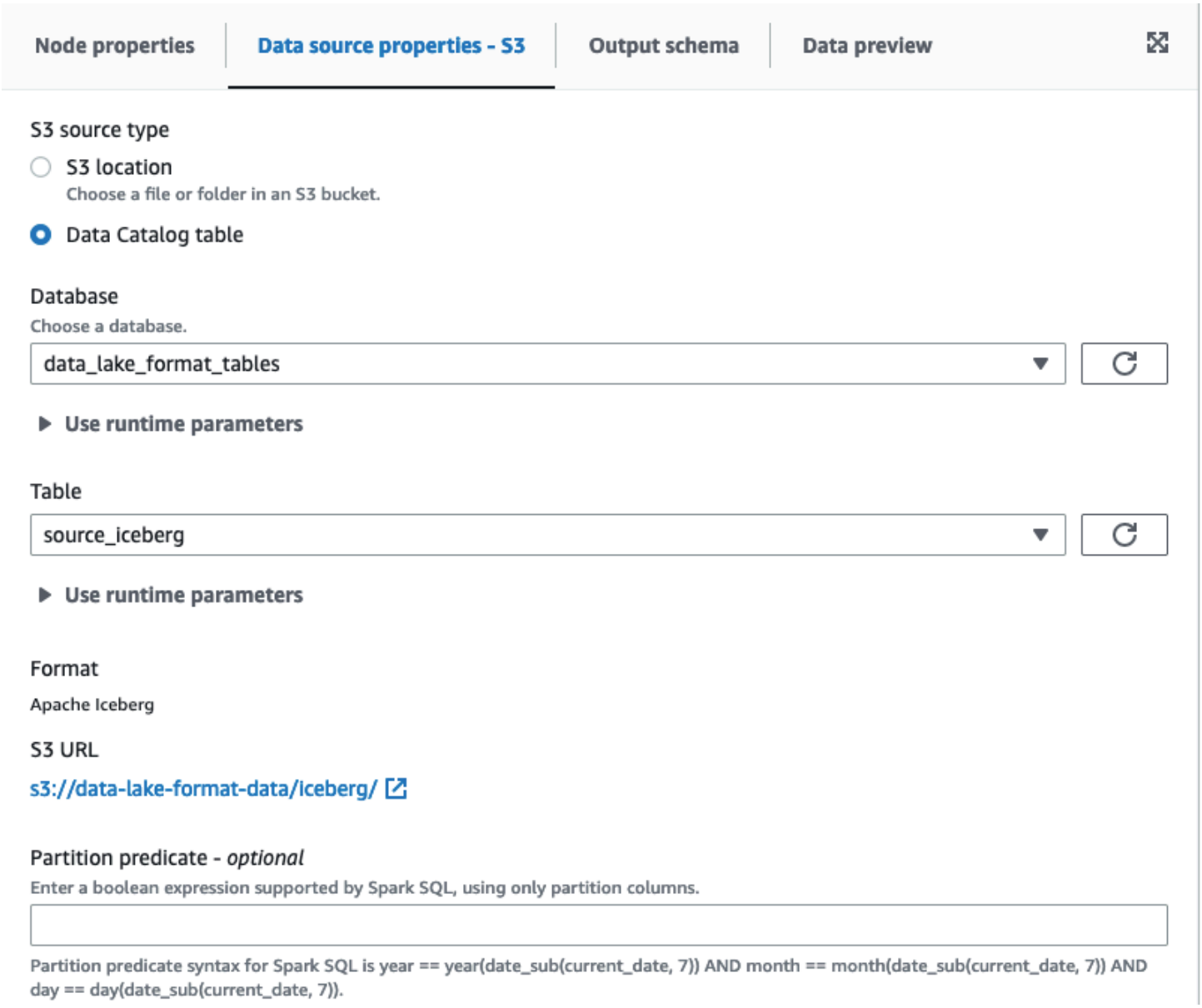
- 格式 — 从下拉菜单中选择 Apache Iceberg。
- Amazon S3 目标位置 — 通过单击浏览 S3 来选择 Amazon S3 目标位置。
- Data Catalog 更新选项 — 在 Data Catalog 中创建表，并在后续运行时选择保留现有架构并添加新分区才能继续。使用 AWS Glue 编写新的 Iceberg 表需要将 Data Catalog 配置为 Iceberg 表的目录。要更新已在 Data Catalog 中注册的现有 Iceberg 表，请选择 Data Catalog 作为目标。
  - 数据库 — 从 Data Catalog 中选择数据库。
  - 表名称 — 输入表名的值。Apache Iceberg 表名必须全部为小写。如果需要，请使用下划线，因为不允许使用空格。例如“data_lake_format_tables”。



在 Amazon S3 数据来源中使用 Apache Iceberg 框架

在 Data Catalog 数据来源中使用 Apache Iceberg 框架

1. 从来源菜单中选择“AWS Glue Studio Data Catalog”。
2. 在数据来源属性选项卡中，选择数据库和表。
3. AWS Glue Studio 将格式类型显示为 Apache Iceberg 和 Amazon S3 URL。



在 Amazon S3 数据来源中使用 Apache Iceberg 框架

Apache Iceberg 不能作为 AWS Glue Studio 中 Amazon S3 源节点的数据选项提供。

配置数据目标节点

数据目标是任务写入转换后数据的位置。

数据目标选项概览

您的数据目标（也称为数据接收器）可以是：

- S3 – 任务将数据写入您选择的 Amazon S3 位置的文件中，并以您指定的格式写入。

如果您为数据目标配置分区列，则任务会根据分区键将数据集写入 Amazon S3 的目录。

- AWS Glue Data Catalog – 任务使用与数据目录中的表关联的信息将输出数据写入目标位置。

  您可以手动创建表，也可以使用爬网程序创建表。您还可以使用 AWS CloudFormation 模板在数据目录中创建表。
- 连接器 – 连接器是一段代码，便于您在数据存储和 AWS Glue 之间通信。任务使用连接器和关联的连接将输出数据写入目标位置。您可以订阅 AWS Marketplace 中提供的连接器，或者您还可以创建自己的自定义连接器。有关更多信息，请参阅 将连接器添加到 AWS Glue Studio

您可以选择在任务写入 Amazon S3 数据目标时更新数据目录。当架构或分区发生更改时，不需要爬网程序更新数据目录，此选项可以更轻松地使表保持最新状态。此选项可选择性地将新表添加到数据目录、更新表分区以及直接从任务更新表的方案，简化将数据用于分析的过程。

编辑数据目标节点

数据目标是任务写入转换后数据的位置。

在任务图中添加或配置数据目标节点

1. （可选）如果需要添加目标节点，请在可视化编辑器顶部工具栏中选择 Target (目标)，然后选择 S3 或者 Glue Data Catalog。
   - 如果选择 S3，则任务将数据集写入您指定的 Amazon S3 位置中的一个或多个文件。
   - 如果选择 AWS Glue Data Catalog，则任务将写入从数据目录中选择的表所描述的位置。
2. 在任务图中选择一个数据目标节点。选择节点时，节点详细信息面板将在页面右侧显示。
3. 选择 Node properties (节点属性) 选项卡，然后输入以下信息：
   - Name (名称)：输入要与任务图中节点关联的名称。
   - Node type (节点类型)：应该已选择一个值，但您可以根据需要对其进行更改。
   - Node parents (父节点)：父节点是任务图中提供要写入目标位置的输出数据的节点。对于预填充的任务图，目标节点应该已经选择父节点。如果没有显示父节点，则从列表中选择父节点。

     目标节点具有单个父节点。
4. 配置 Data target properties (数据目标属性) 信息。有关详细信息，请参阅以下章节：
   - 将 Amazon S3 用于数据目标
   - 使用数据目标的数据目录表

- 将连接器用作数据目标

5. （可选）配置数据目标节点属性后，您可以选择节点详细信息面板中的 Output schema (输出架构) 选项卡，查看数据的输出架构。当您首次为任务中的任何节点选择此选项卡时，系统会提示您提供 IAM 角色以访问数据。如果您尚未在 Job details (任务详细信息) 选项卡上指定 IAM 角色，系统会提示您在此处输入 IAM 角色。

将 Amazon S3 用于数据目标

对于 Amazon S3 和连接器之外的所有数据源，表必须位于您所选择源类型的 AWS Glue Data Catalog 中。AWS Glue Studio 不会创建数据目录表。

配置写入 Amazon S3 的数据目标节点

1. 转到新任务或已保存任务的可视编辑器。
2. 在任务图中选择一个数据源节点。
3. 选择 Data source properties (数据源属性) 选项卡，然后输入以下信息：

   - Format (格式)：从列表中选择格式。数据结果的可用格式类型包括：
     - JSON：JavaScript 对象表示法。
     - CSV：逗号分隔的值。
     - Avro：Apache Avro JSON 二进制。
     - Parquet：Apache Parquet 列式存储
     - Glue Parquet：自定义 Parquet 编写器类型，已针对 `DynamicFrames` 作为数据格式优化。它不需要数据预先计算的架构，而是动态计算和修改架构。
     - ORC：Apache 优化的行列式（ORC）格式。

     要了解有关这些格式选项的更多信息，请参阅《AWS Glue 开发人员指南》中的 AWS Glue 中 ETL 输入和输出的格式选项。
   - Compression Type (压缩类型)：您可以选择使用 `gzip` 或者 `bzip2` 格式压缩日期。默认值为无压缩，或 None (无)。
   - S3 Target Location (S3 目标位置)：Amazon S3 存储桶和数据输出的位置。您可以选择 Browse S3 (浏览 S3) 按钮，查看您有权访问的 Amazon S3 存储桶，然后选择一个作为目标目的地。
   - 数据目录更新选项
     - Do not update the Data Catalog (请勿更新数据目录)：（默认）如果您不希望任务更新数据目录（即使架构更改或添加了新分区），请选择此选项。

- Create a table in the Data Catalog and on subsequent runs, update the schema and add new partitions (在数据目录中创建表并在后续运行时，更新架构并添加新分区)：如果选择此选项，任务将在第一次任务运行时创建数据目录中的表。在后续任务运行时，如果架构发生更改或添加了新分区，任务将更新数据目录表。

  您还必须从数据目录中选择数据库并输入表名。

- Create a table in the Data Catalog and on subsequent runs, keep existing schema and add new partitions (在数据目录中创建表，并在后续运行时保持现有架构并添加新分区)：如果选择此选项，任务将在第一次任务运行时创建数据目录中的表。在后续任务运行时，任务只更新数据目录表以添加新分区。

  您还必须从数据目录中选择数据库并输入表名。

- Partition keys (分区键)：选择要在输出中用作分区键的列。要添加更多分区键，请选择 Add a partition key (添加分区键)。

使用数据目标的数据目录表

对于 Amazon S3 和连接器之外的所有数据源，表必须位于您所选择目标类型的 AWS Glue Data Catalog 中。AWS Glue Studio 不会创建数据目录表。

为使用数据目录表的目标配置数据属性

1. 转到新任务或已保存任务的可视编辑器。
2. 在任务图中选择一个数据目标节点。
3. 选择 Data target properties (数据目标属性) 选项卡，然后输入以下信息：

   - Database (数据库)：从列表中选择包含用作目标的表的数据库。此数据库必须已存在于数据目录中。
   - Table (表)：从列表中选择定义输出数据架构的表。此表必须已存在于数据目录中。

     数据目录中的表包含列的名称、数据类型定义、分区信息以及有关目标数据集的其他元数据。您的任务写入数据目录中此表描述的位置。

     有关在数据目录中创建表的更多信息，请参阅 AWS Glue 开发人员指南中的在数据目录中定义表。
   - 数据目录更新选项

- Do not change table definition (请勿更改表定义)：（默认）如果不希望任务更新数据目录（即使架构更改或添加了新分区），请选择此选项。
- Update schema and add new partitions (更新架构并添加新分区)：如果选择此选项，任务将更新数据目录表（如果架构更改或添加新分区）。
- Keep existing schema and add new partitions (保留现有架构并添加新分区)：如果选择此选项，任务将仅为了添加新分区而更新数据目录表。
- Partition keys (分区键)：选择要在输出中用作分区键的列。要添加更多分区键，请选择 Add a partition key (添加分区键)。

将连接器用作数据目标

如果您为 Node type (节点类型) 选择连接器，请按照 使用自定义连接器编写任务 中的说明操作，完成数据目标属性的配置。

编辑或上载任务脚本

使用 AWS Glue Studio 可视化编辑器编辑任务脚本或上载您自己的脚本。

仅当任务使用 AWS Glue Studio 创建时，您才能使用可视化编辑器编辑任务节点。如果任务是使用 AWS Glue 控制台、通过 API 命令或命令行界面（CLI）创建，您可以使用 AWS Glue Studio 中的脚本编辑器编辑任务脚本、参数和计划。您还可以将任务转换为仅脚本模式，编辑 AWS Glue Studio 中所创建任务的脚本。

编辑任务脚本或上载您自己的脚本

1. 如果创建新任务，请在 Jobs (任务) 页面上，选择 Spark script editor (Spark 脚本编辑器) 选项创建 Spark 任务，或选择 Python Shell script editor (Python Shell 脚本编辑器) 创建 Python Shell 任务。您可以编写新脚本，也可以上载现有脚本。如果选择 Spark script editor (Spark 脚本编辑器)，您可以编写或上载 Scala 脚本或 Python 脚本。如果选择 Python Shell script editor (Python Shell 脚本编辑器)，则只能编写或上载 Python 脚本。

   选择用于创建新任务的选项后，在显示的 Options (选项) 部分，您可以使用启动程序脚本（使用 boilerplate 代码创建新脚本）启动，也可以上载本地文件以用作任务脚本。

   如果选择 Spark script editor (Spark 脚本编辑器)，您可以上载 Python 或 Scala 脚本文件。Scala 脚本必须具有文件扩展名 `.scala`。您必须将 Python 脚本识别为 Python 类型的文件。如果选择 Python Shell script editor (Python Shell 脚本编辑器)，则只能上载 Python 脚本文件。

   完成选择后，选择 Create (创建) 创建任务并打开可视化编辑器。

2. 转到新任务或已保存任务的可视任务编辑器，然后选择 Script (脚本) 选项卡。
3. 如果未使用脚本编辑器选项创建新任务，并且从未编辑过现有任务的脚本，则 Script (脚本) 选项卡显示标题 Script (Locked) (脚本（已锁定）)。这意味着脚本编辑器处于只读模式。选择 Edit script (编辑脚本) 解锁脚本以进行编辑。

   要使脚本可编辑，AWS Glue Studio 将您的任务从可视化任务转换为仅脚本任务。如果解锁脚本进行编辑，则在保存此任务后，无法再使用可视化编辑器。

   在确认窗口中，选择 Confirm (确认) 以继续浏览，或选择 Cancel (取消) 以保持任务可用于可视化编辑。

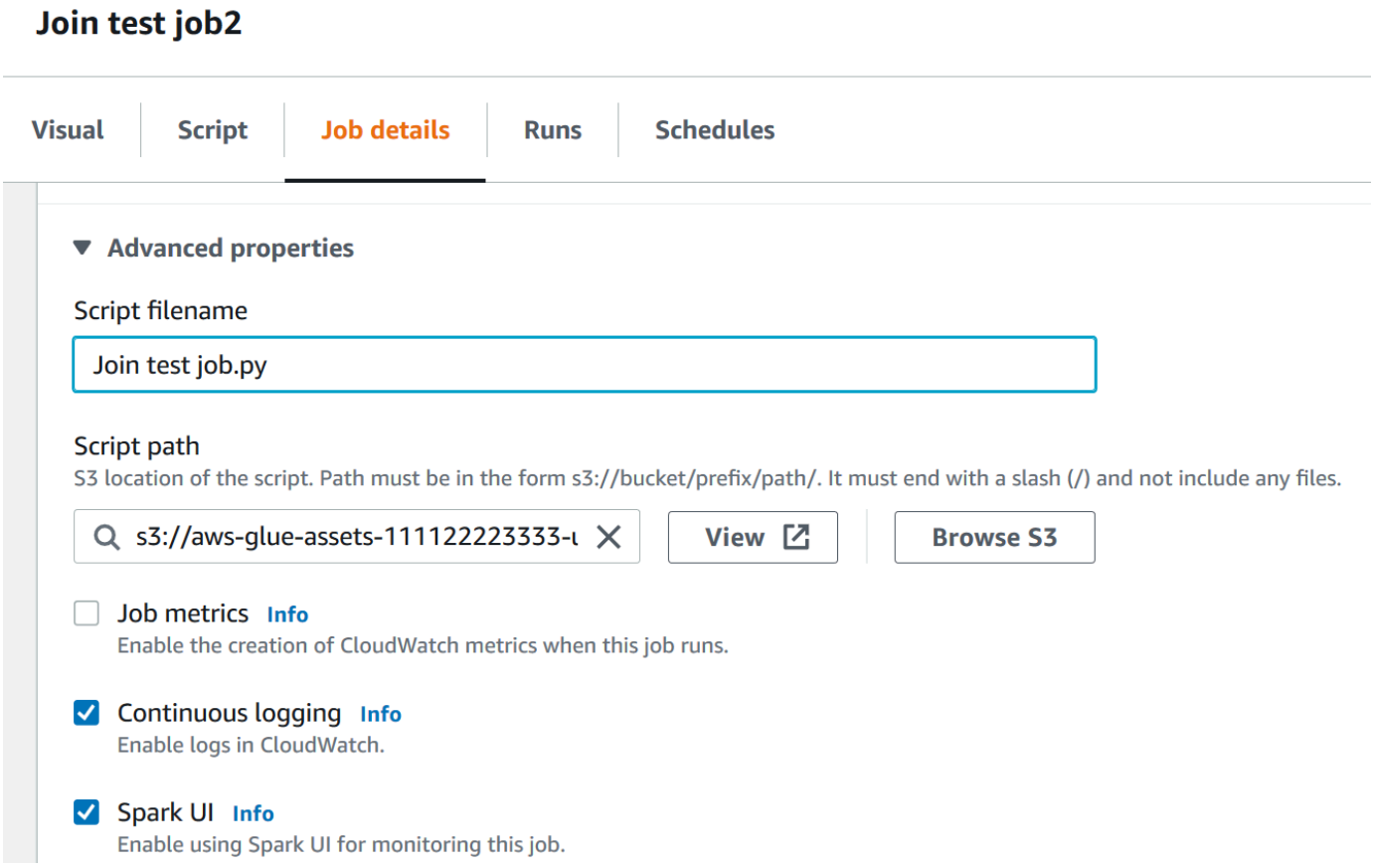
   如果选择 Confirm (确认)，Visual (可视化) 选项卡不再显示在编辑器中。您可以使用 AWS Glue Studio 可使用脚本编辑器修改脚本、修改任务详细信息或调度，或查看任务运行。

   > **Note**
   >
   > 在保存任务之前，转换为仅脚本任务不是永久性操作。如果您刷新控制台网页，或者在保存之前关闭任务并在可视编辑器中重新打开它，则仍然可以在可视编辑器中编辑各个节点。

4. 根据需要编辑脚本。

   编辑完脚本后，选择 Save (保存) 保存任务并将任务从可视化对象永久转换为仅脚本。

5. （可选）您可以在 Script (脚本) 选项卡上选择 Download (下载) 按钮，从 AWS Glue Studio 控制台下载脚本。选择此按钮后，将打开一个新的浏览器窗口，其中显示脚本在 Amazon S3 中的位置。任务的 Job details (任务详细信息) 选项卡中的 Script filename 和 Script path 参数确定脚本文件在 Amazon S3 中的名称和位置。



保存任务时，AWS Glue 将任务脚本保存在这些字段指定的位置。如果您在 Amazon S3 中修改此位置的脚本文件，AWS Glue Studio 将在您下次编辑任务时加载修改后的脚本。

在 AWS Glue Studio 中创建和编辑 Scala 脚本

当您选择用于创建任务的脚本编辑器时，默认情况下，任务编程语言设置为 Python 3。如果您选择编写新脚本而不是上载脚本，AWS Glue Studio 会开启一个新的脚本，以 Python 编写样板文本。如果要改为编写 Scala 脚本，则必须首先将脚本编辑器配置为使用 Scala。

> **Note**
>
> 如果选择 Scala 作为任务的编程语言，并使用可视化编辑器设计任务，则生成的任务脚本将使用 Scala 编写，因此无需进一步操作。

要在 AWS Glue Studio 中编写新的 Scala 脚本

1. 创建新任务，方法是选择 Spark script editor (Spark 脚本编辑器) 选项。

2. 在 Options (选项) 下面，选择 Create a new script with boilerplate code (使用样板代码创建新脚本)。
3. 选择 Job details (任务详细信息) 选项卡，将 Language (语言) 设置为 `Scala`（而不是 `Python 3`）。

> **Note**
>
> 当您选择 Spark script editor (Spark 脚本编辑器) 选项创建任务，Type (类型) 属性将自动设置为 `Spark`。

4. 选择 Script (脚本) 选项卡。
5. 删除 Python 样板文本。您可以用以下 Scala 样板文本替换。

```
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. 在编辑器中编写您的 Scala 任务脚本。您可以根据需要添加其他 `import` 语句。

在 AWS Glue Studio 中创建和编辑 Python Shell 任务

选择 Python shell 脚本编辑器来创建任务时，可以上载现有 Python 脚本或编写新脚本。如果选择编写新脚本，样板代码将添加到新的 Python 任务脚本中。

创建新的 Python Shell 任务

请参阅在 AWS Glue Studio 中启动作业中的说明。

Python Shell 任务支持的任务属性与 Spark 任务支持的任务属性不同。下面的列表描述了 Job details (任务详细信息) 选项卡上对 Python Shell 任务的可用任务参数所做的更改。

- 任务的 Type (类型) 属性将自动设置为 `Python Shell`，而且无法更改。
- 有面向任务的 Python version (Python 版本) 属性，而不是 Language (语言)。目前，Python Shell 任务在 AWS Glue Studio 中使用 Python 3.6。
- Glue version (Glue 版本) 属性不可用，因为它不适用于 Python Shell 任务。
- 显示 Data processing units (数据处理单元) 属性，而不是 Worker type (工件类型) 和 Number of workers (工件数)。此任务属性确定 Python Shell 在运行任务时使用的数据处理单元（DPU）数量。
- Job bookmark (任务书签) 属性不可用，因为 Python Shell 任务不支持该属性。
- 在 Advanced properties (高级属性) 下面，以下属性不适用于 Python Shell 任务。
  - 作业指标
  - 连续日志记录
  - Spark UI 和 Spark UI 日志路径
  - 标题 Libraries (库) 下面的 Dependent jars path (从属 jars 路径)

更改任务图中节点的父节点

您可以更改节点的父节点，以便在任务图中移动节点或更改节点的数据源。

更改父节点

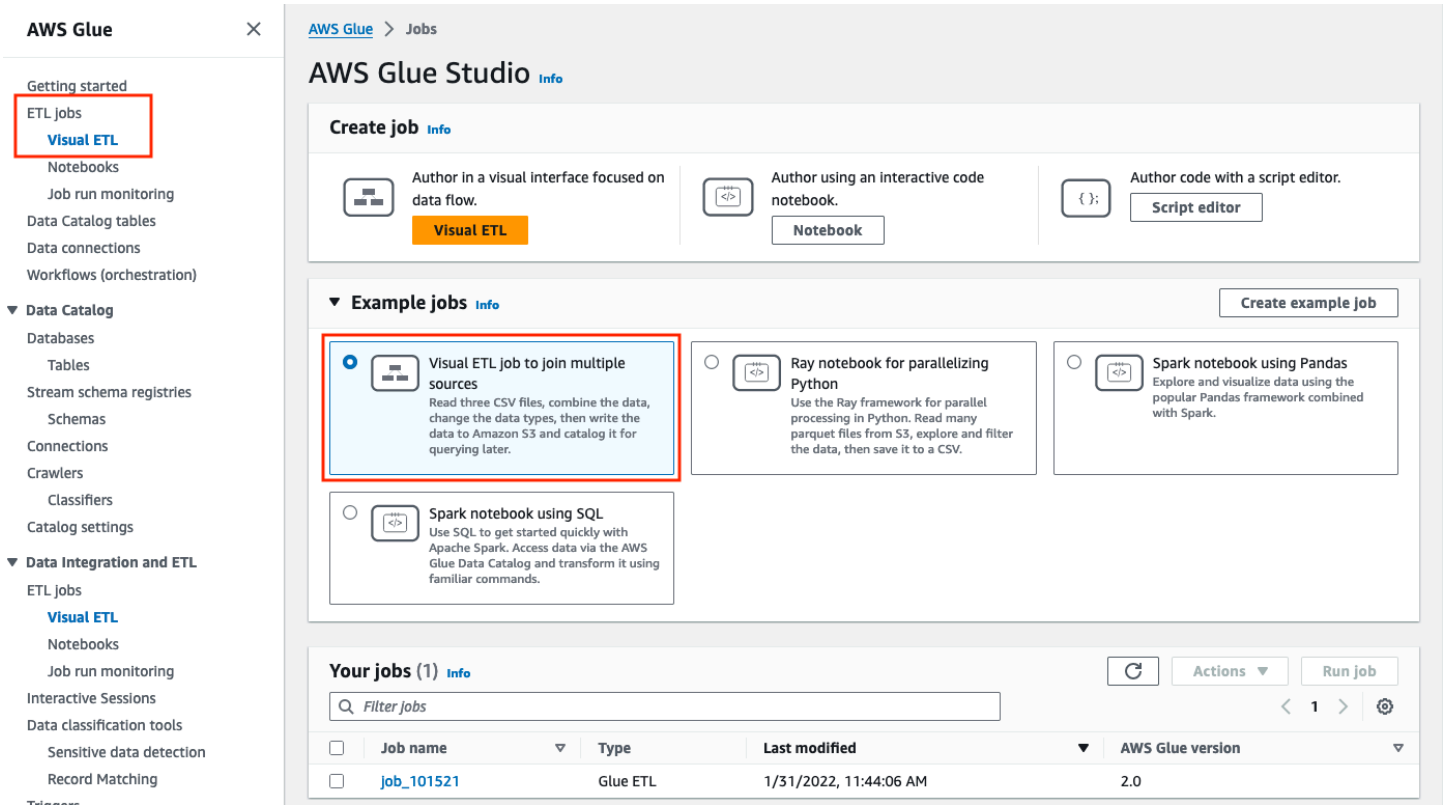
1. 选择任务图中要修改的节点。
2. 在节点详细信息面板中，在标题 Node parents (父节点) 下的 Node properties (节点属性) 选项卡上删除节点的当前父节点。
3. 从列表中选择新的父节点。
4. 根据需要修改节点的其他属性以匹配新选定的父节点。

如果您错误修改了节点，您可以使用 Undo (撤消) 按钮以反转操作。

从任务图中删除节点

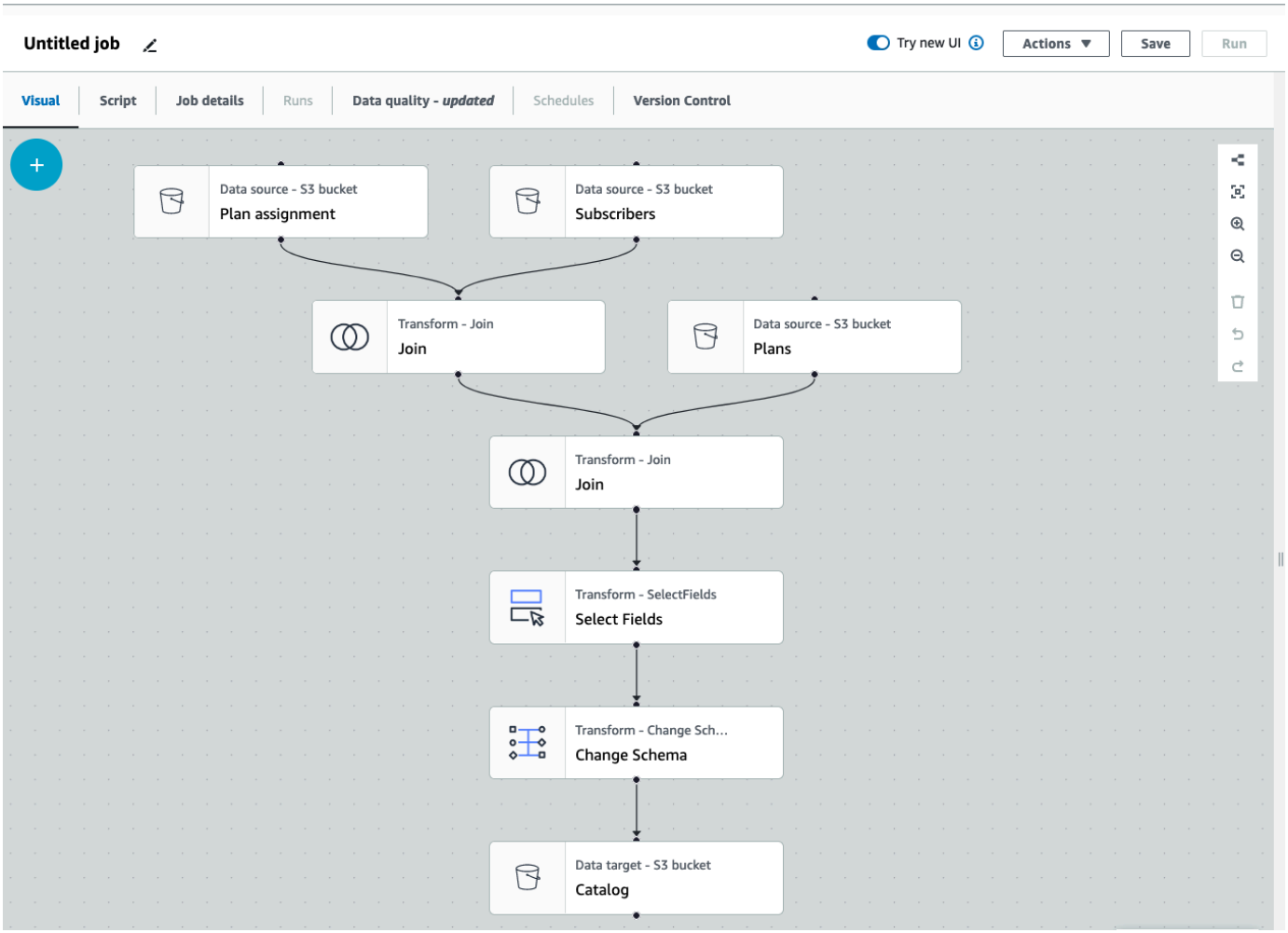
使用 Visual ETL 作业时，您可以从画布中移除节点，而不必重新添加或重构连接到已移除节点的任何节点。

在下面的示例中，您可以选择 ETL 作业 > Visual ETL，然后在示例作业中选择 要加入多个源的 Visual ETL 作业。选择创建示例作业以创建作业，然后按照以下步骤操作。



从画布中移除节点

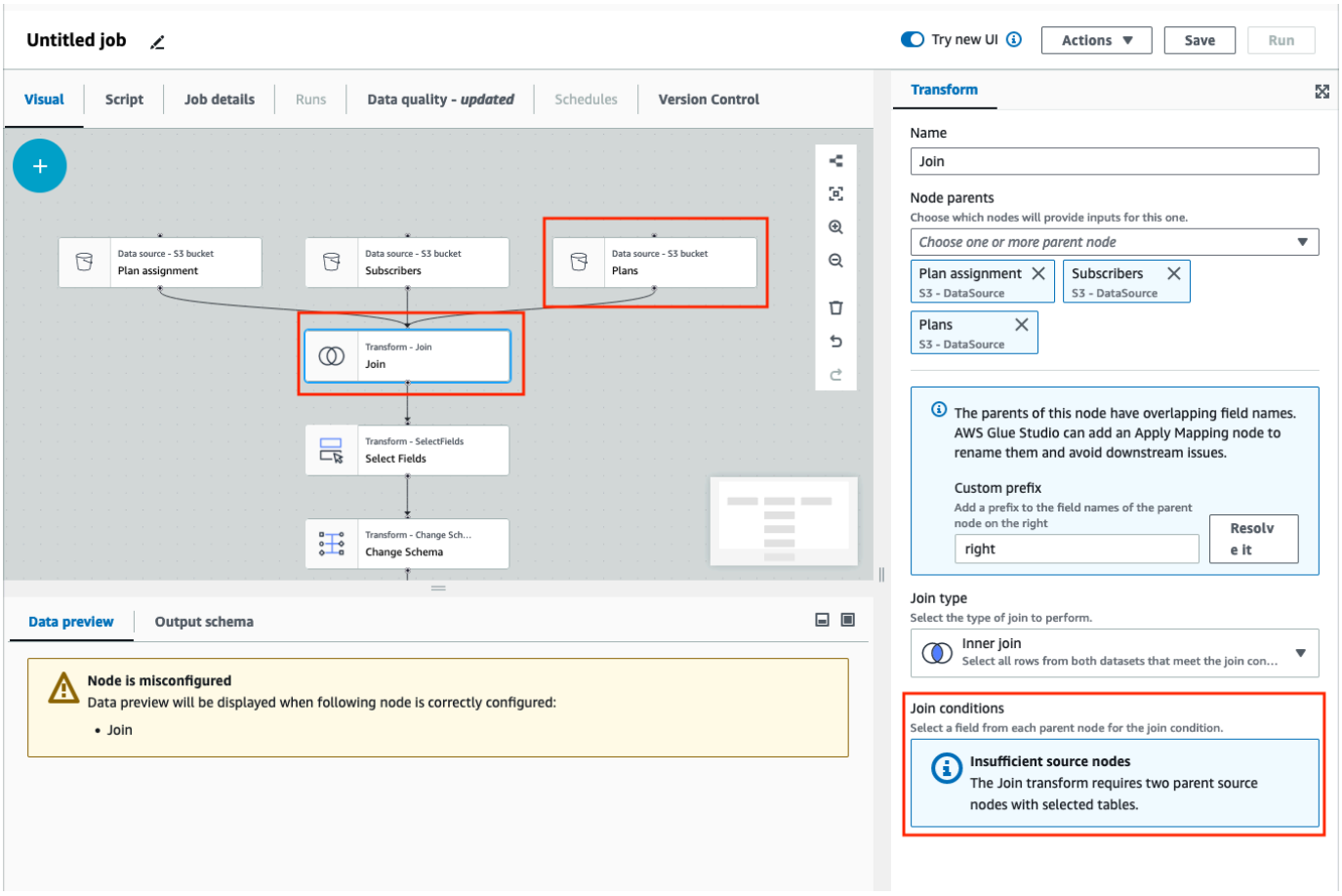
1. 在 AWS Glue 控制台中，从导航菜单中选择 Visual ETL，然后选择一个现有的作业。作业画布显示如下所示的示例作业。



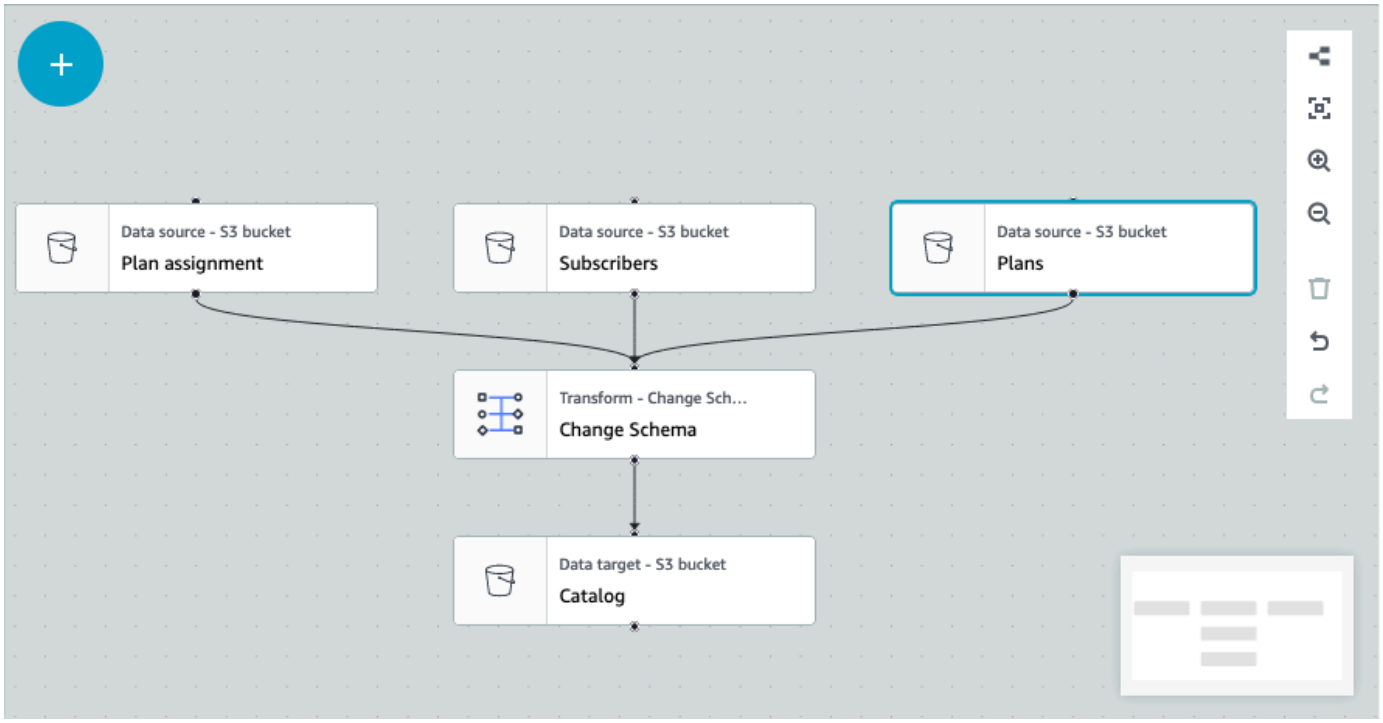
2. 选择要删除的节点。画布将放大到该节点。在画布右侧的工具栏中，选择垃圾桶图标。这将移除该节点，任何连接到该节点的节点都将移动，以取代其在工作流中的位置。在此示例中，第一个 Join 节点已从画布中删除。

   如果删除工作流中的节点，则 AWS Glue 会重新排列这些节点，确保其组织方式不会导致工作流无效。您可能仍需要更正节点的配置。

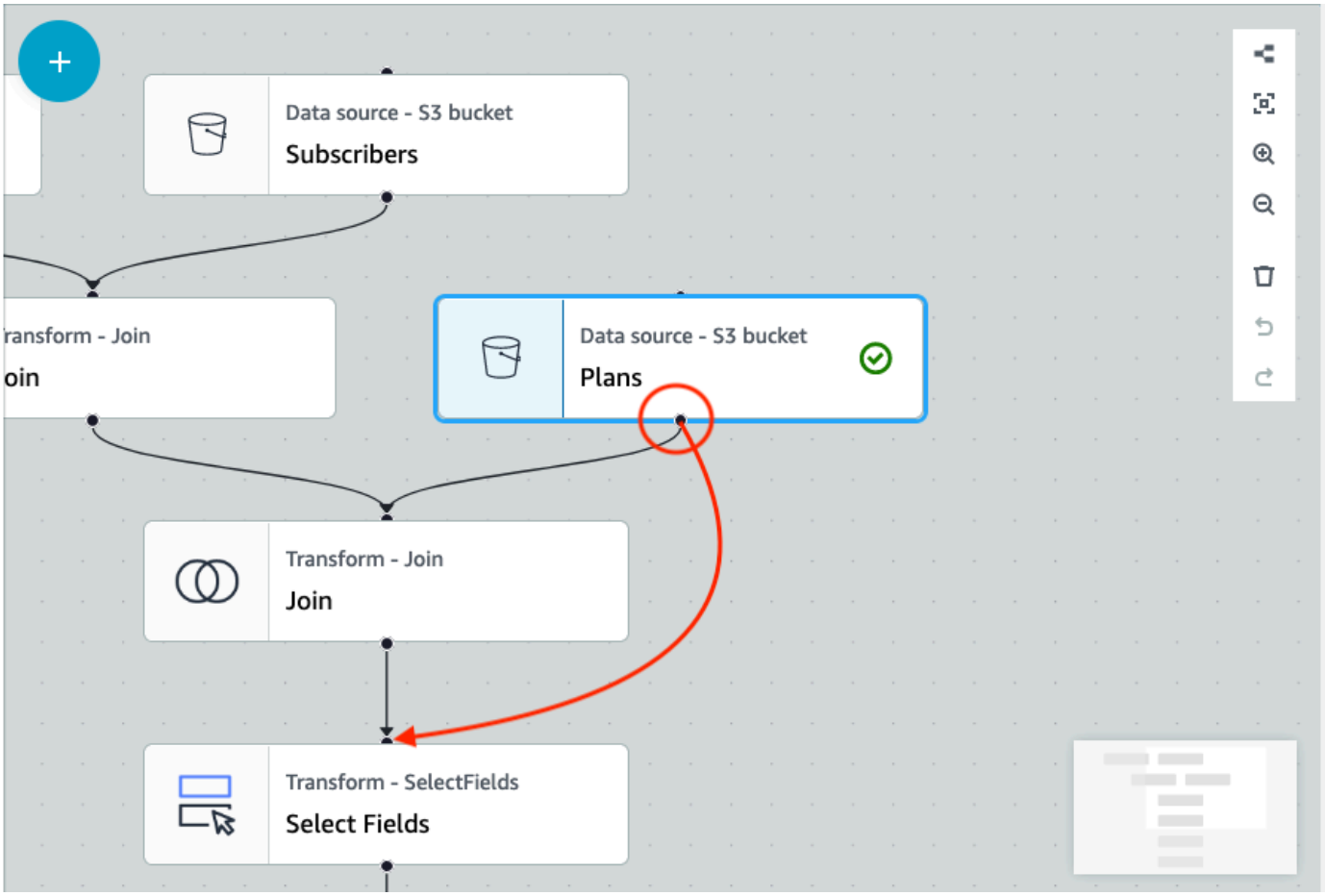
   在示例中，Subscribers 节点下的 Join 节点已被移除。因此，Plans 源节点已移至顶层，并且仍与子节点 Join 相连。Join 节点现在需要额外的配置，因为 Join 需要两个带有选定表的父级源节点。画布右侧的转换选项卡在加入条件下显示缺少的要求。



3. 删除第二个 Join 节点和 Select Fields 节点。删除节点后，工作流将如下所示。



4. 要修改节点连接，请单击节点的手柄并将连接拖动到新节点。这样，您将可以删除节点并重新排列逻辑流中的节点。在此示例中，通过单击 Plans 节点上的手柄并将连接拖动到 Join 节点来建立新的连接，如红色箭头所示。



5. 如果您需要撤消任何操作，请选择画布右侧工具栏中垃圾桶图标正下方的撤销图标。

将源和目标参数添加到 AWS Glue 数据目录节点

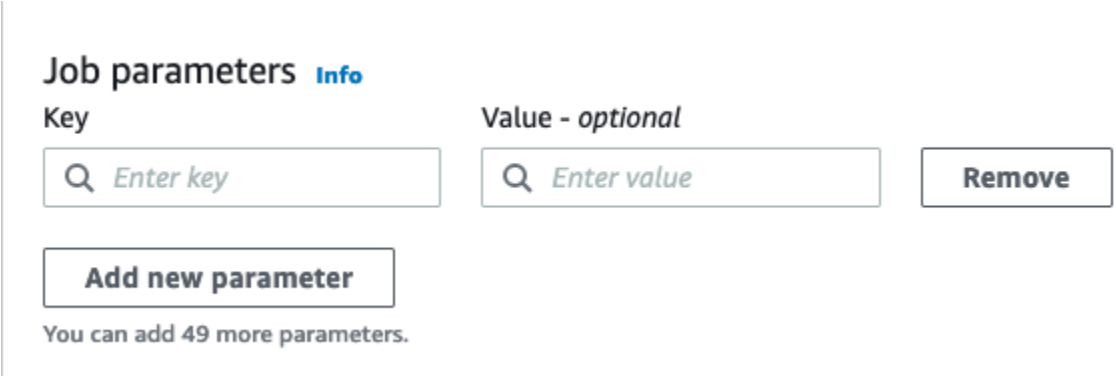
AWS Glue Studio 允许您对视觉作业进行参数化。由于生产和开发环境中的目录表名称可能不同，因此您可以为将在作业运行时运行的数据库和表定义和选择运行时参数。

使用 AWS Glue 数据目录节点时，作业参数允许您参数化源和目标，并将这些参数保存在作业中。当您将源和目标指定为参数时，就实现了作业的可重用性，尤其是在多个环境中使用同一个作业时。通过节省管理源和目标的时间和精力，此功能有助于跨部署环境提升代码。此外，您指定的自定义参数将覆盖 AWS Glue 作业特定运行的任何默认参数。

添加源和目标参数

无论您使用 AWS Glue 数据目录节点作为源或目标，都可以在 Job details（作业详情）选项卡的 Advanced properties（高级属性）部分中定义运行时参数。

1. 选择 AWS Glue 数据目录节点作为源节点或目标节点。
2. 选择 Job details（任务详细信息）选项卡。
3. 选择 Advanced properties（高级属性）。
4. 在 Job parameters（作业参数）部分中，输入键值。例如，`--db.source` 将是数据库源的参数。您可以为键输入任何名称，只要键名称后面带有“破折号”即可。

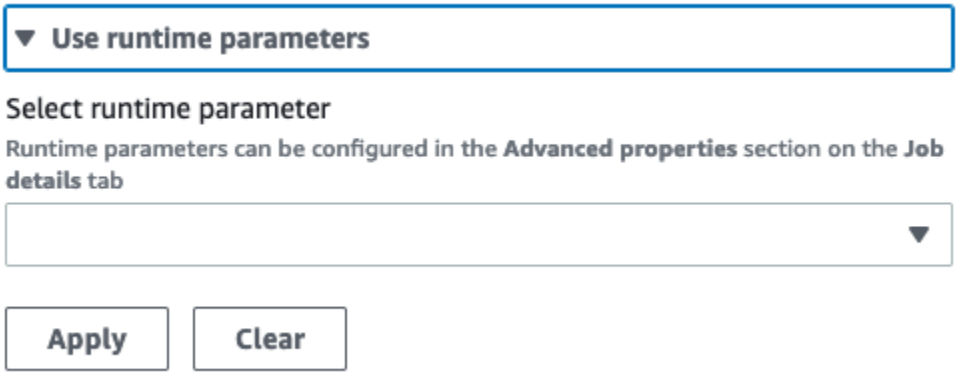


5. 输入值。例如，`databasename` 将是正在参数化的数据库的值。
6. 如果您要添加更多参数，请选择 Add new parameter（添加新参数）。最多允许 50 个参数。一旦定义了键值对，就可以在 AWS Glue 数据目录节点中使用该参数。

选择运行时参数

> **Note**
>
> 无论 AWS Glue 数据目录节点是源节点还是目标节点，为数据库和表选择运行时参数的过程均相同。

1. 选择 AWS Glue 数据目录节点作为源节点或目标节点。
2. 在 Data source properties - Data Catalog（数据源属性 - 数据目录）选项卡中的 Database（数据库）下方，选择 Use runtime parameters（使用运行时参数）。



3. 从下拉菜单中选择参数。例如，如果选择为源数据库定义的参数，该数据库将在您选择 Apply（应用）时自动填充到数据库下拉菜单中。
4. 在 Table（表）部分中，选择已定义为源表的参数。在选择 Apply（应用）时，该表将自动填充为要使用的表。
5. 保存并运行作业时，AWS Glue Studio 将在作业运行期间引用选定的参数。

在 AWS Glue 中使用 Git 版本控制系统

> **Note**
>
> AWS Glue Studio 中目前不支持笔记本电脑进行版本控制。但是，支持 AWS Glue 作业脚本和可视化 ETL 作业的版本控制。

如果您有远程存储库，并且想要使用存储库管理 AWS Glue 作业，则可以使用 AWS Glue Studio 或 AWS CLI，将更改同步到您的存储库和 AWS Glue 中的作业。以这种方式同步更改时，就是在将作业从 AWS Glue Studio 推送到存储库，或者从存储库中提取到 AWS Glue Studio。

通过在 AWS Glue Studio 中集成 Git，您可以：

- 与 AWS CodeCommit、GitHub、GitLab 和 Bitbucket 等 Git 版本控制系统集成
- 编辑 AWS Glue Studio 中的 AWS Glue 作业（无论是使用可视化作业还是脚本作业）并将它们同步到存储库
- 参数化作业中的源和目标
- 从存储库中提取作业并在 AWS Glue Studio 中进行编辑
- 使用 AWS Glue Studio 中的多分支工作流从分支提取和/或推送到分支，以此测试作业

- 从存储库下载文件并将任务上传到 AWS Glue Studio 以创建跨账户作业
- 使用选择的自动化工具（例如 Jenkins、AWS CodeDeploy 等）

这段视频演示了如何将 AWS Glue 与 Git 集成并构建持续的协作代码管道。

IAM 权限

确保作业具有下列 IAM 权限之一。有关如何设置 IAM 权限的更多信息，请参阅为 AWS Glue Studio 设置 IAM 权限。

- `AWSGlueServiceRole`
- `AWSGlueConsoleFullAccess`

Git 集成至少需要以下操作：

- `glue:UpdateJobFromSourceControl` — 能够使用版本控制系统中存在的作业更新 AWS Glue
- `glue:UpdateSourceControlFromJob` — 能够使用存储在 AWS Glue 中的作业更新版本控制系统
- `s3:GetObject` — 能够在推送到版本控制系统的同时检索作业脚本
- `s3:PutObject` — 能够在从源代码控制系统提取作业时更新脚本

先决条件

要将作业推送到源代码控制存储库，您需要：

- 您的管理员已经创建的存储库
- 存储库中的一个分支
- 个人访问令牌（对于 Bitbucket，这是存储库访问令牌）
- 存储库所有者的用户名
- 在存储库中设置权限以允许 AWS Glue Studio 读取和写入存储库
  - GitLab – 将令牌范围设置为 api、read_repository 和 write_repository
  - Bitbucket – 将权限设置为：
    - 工作区成员资格 – 读取、写入
    - 项目 – 写入，管理员读取
    - 存储库 – 读取、写入、管理、删除

> **Note**
>
> 使用 AWS CodeCommit 时，不需要个人访问令牌和存储库所有者。请参阅 Git 和 AWS CodeCommit 入门。

在 AWS Glue Studio 中使用源代码控制存储库中的作业

要从源代码控制存储库提取不在 AWS Glue Studio 中的作业，然后在 AWS Glue Studio 中使用该作业，先决条件将取决于作业的类型。

对于可视化作业：

- 您需要与作业名称相匹配的文件夹和作业定义 JSON 文件

  例如，请参阅下面的作业定义。存储库中的分支应包含路径 `my-visual-job/my-visual-job.json`，其中文件夹和 JSON 文件与作业名称匹配

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

对于脚本作业：

- 您需要文件夹、作业定义的 JSON 文件和脚本
- 该文件夹和 JSON 文件应与作业名称相匹配。脚本名称需要匹配作业定义中 `scriptLocation` 以及文件扩展名

  例如，在下面的作业定义中，存储库中的分支应包含路径 `my-script-job/my-script-job.json` 和 `my-script-job/my-script-job.py`。脚本名称应与包括脚本的扩展名的 `scriptLocation` 中的名称相匹配

```
{
  "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

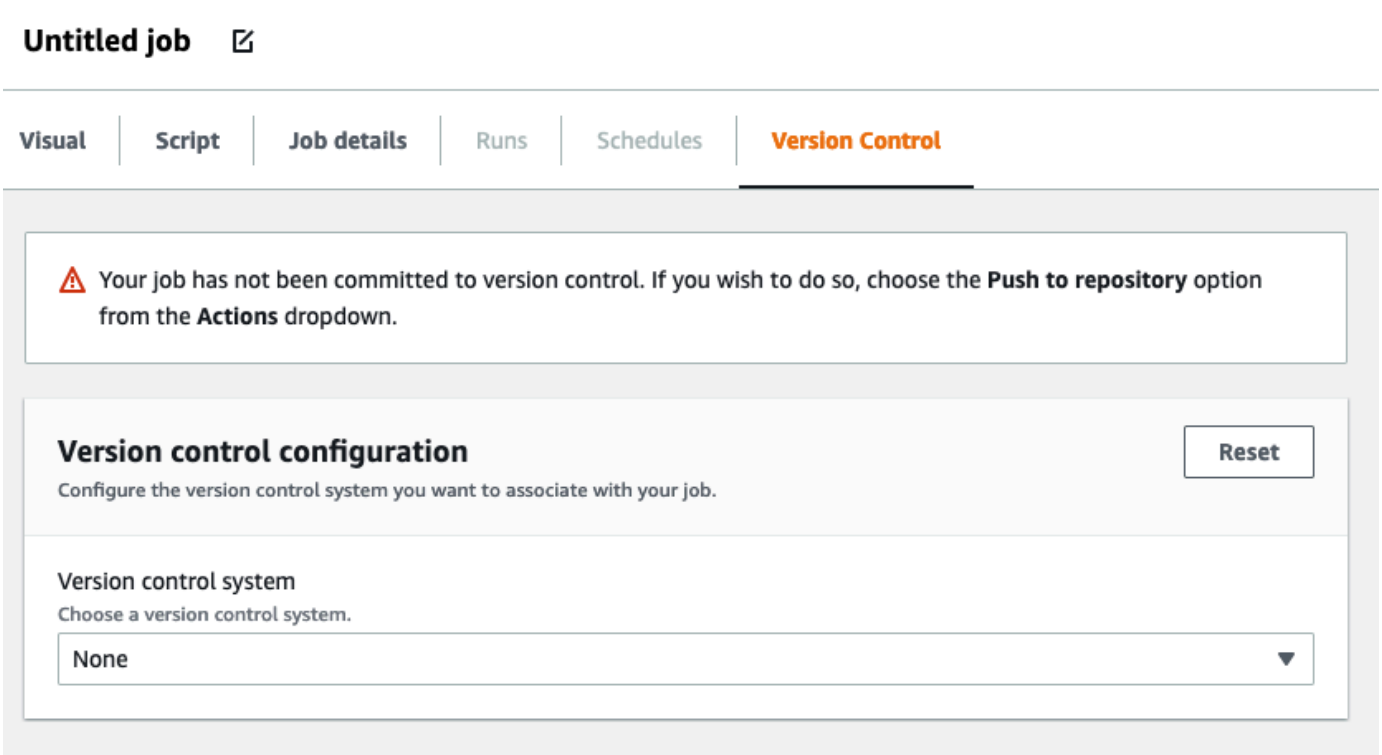
- AWS Glue 目前不支持从 GitLab-Groups 推送/拉取。

将版本控制存储库连接到 AWS Glue

您可以输入版本控制存储库的详细信息并在 AWS Glue Studio 作业编辑器的 Version Control（版本控制）选项卡中管理它们。要与 Git 存储库集成，您必须在每次登录 AWS Glue Studio 时连接到存储库。

要连接 Git 版本控制系统，请执行以下操作：

1. 在 AWS Glue Studio 中，开始新作业，然后选择 Version Control（版本控制）选项卡。



2. 在版本控制系统中，通过单击下拉菜单从可用选项中选择 Git 服务。

   - AWS CodeCommit
   - GitHub
   - GitLab
   - Bitbucket

3. 根据您选择的 Git 版本控制系统，需要填写的字段有所不同。

   对于 AWS CodeCommit：

   通过为作业选择存储库和分支来完成存储库配置：

   - 存储库 — 如果您在 AWS CodeCommit 中设置了存储库，则从下拉菜单中选择存储库。您的存储库将自动填充到列表中
   - 分支 — 从下拉菜单中选择分支
   - 文件夹 — 可选 - 输入保存作业的文件夹的名称。如果留空，则会自动创建文件夹。文件夹名称默认为作业名称

   对于 GitHub：

填写以下字段以完成 GitHub 配置：

- 个人访问令牌 — 这是 GitHub 存储库提供的令牌。有关个人访问令牌的更多信息，请参阅 GitHub 文档
- 存储库所有者 — 这是 GitHub 存储库的所有者。

从 GitHub 中选择储存库和分支以完成存储库配置。

- 存储库 — 如果您已在 GitHub 中设置存储库，请从下拉菜单中选择存储库。您的存储库将自动填充到列表中
- 分支 — 从下拉菜单中选择分支
- 文件夹 — 可选 - 输入保存作业的文件夹的名称。如果留空，则会自动创建文件夹。文件夹名称默认为作业名称

对于 GitLab：

> **Note**
>
> AWS Glue 目前不支持从 GitLab-Groups 推送/拉取。

- 个人访问令牌 – 这是 GitHub 存储库提供的令牌。有关个人访问令牌的更多信息，请参阅 GitLab 个人访问令牌
- 存储库所有者 – 这是 GitHub 存储库的所有者。

从 GitHub 中选择储存库和分支以完成存储库配置。

- 存储库 – 如果您已在 GitHub 中设置存储库，请从下拉菜单中选择存储库。您的存储库将自动填充到列表中
- 分支 — 从下拉菜单中选择分支
- 文件夹 — 可选 - 输入保存作业的文件夹的名称。如果留空，则会自动创建文件夹。文件夹名称默认为作业名称

对于 Bitbucket：

- 应用程序密码 – Bitbucket 使用应用程序密码而不是存储库访问令牌。有关应用程序密码的更多信息，请参阅应用程序密码。
- 存储库所有者 – 这是 Bitbucket 存储库的所有者。在 Bitbucket 中，所有者是存储库的创建者。

从 Bitbucket 中选择工作区、储存库、分支和文件夹以完成存储库配置。

- 工作区 – 如果您已在 Bitbucket 中设置工作区，请从下拉菜单中选择工作区。您的工作区会自动填充
- 存储库 – 如果您在 Bitbucket 中设置了存储库，则从下拉菜单中选择存储库。您的存储库会自动填充
- 分支 – 从下拉菜单中选择分支。您的分支会自动填充
- 文件夹 — 可选 - 输入保存作业的文件夹的名称。如果留空，则会使用作业名称自动创建文件夹。

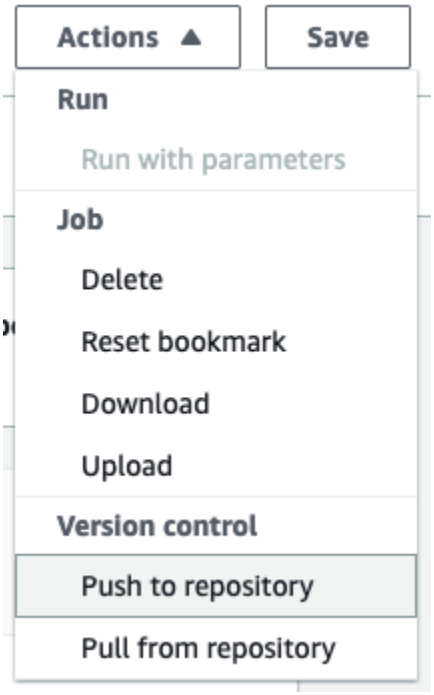
4. 在 AWS Glue Studio 作业的顶部，选择 Save（保存）

推送 AWS Glue 作业到源存储库

输入版本控制系统的详细信息后，可以在 AWS Glue Studio 中编辑作业并将作业推送到您的源存储库。如果您不熟悉 Git 概念，如推送和提取，请参阅 Git 和 AWS CodeCommit 入门上的此教程。

要将作业推送到存储库，您需要输入版本控制系统的详细信息并保存作业。

1. 在 AWS Glue Studio 作业中，选择 Actions（操作）。这将打开其他菜单选项。



2. 选择 Push to repository（推送到存储库）。

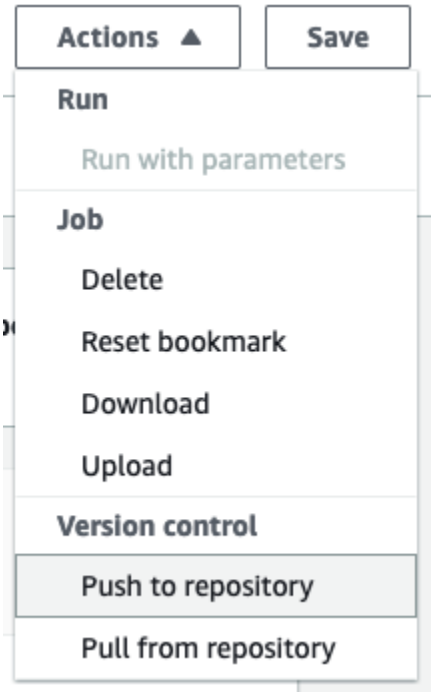
   此操作将保存作业。推送到存储库时，AWS Glue Studio 推送上次保存的更改。如果存储库中的作业被您或其他用户修改，并且与 AWS Glue Studio 中的作业不同步，则在您从 AWS Glue Studio 推送作业时，存储库中的作业会被保存在 AWS Glue Studio 中的作业覆盖。

3. 选择 Confirm（确认）以完成操作。这会在存储库中创建新的提交。如果您使用 AWS CodeCommit，一条确认消息将显示指向 AWS CodeCommit 上最新提交的链接。

从源存储库提取 AWS Glue 作业

将 Git 存储库的详细信息输入到 Version control（版本控制）选项卡之后，您也可以从存储库中提取作业并在 AWS Glue Studio 中进行编辑。

1. 在 AWS Glue Studio 作业中，选择 Actions（操作）。这将打开其他菜单选项。



2. 选择 Pull from repository（从存储库中提取）。
3. 选择确认。这会从存储库中获取最新的提交，并在 AWS Glue Studio 中更新您的作业。
4. 在 AWS Glue Studio 中编辑您的作业。如果您进行了更改，则可以通过选择 Actions（操作）下拉菜单中的 Push to repository（推送到存储库）来将作业同步到存储库。

使用 AWS Glue Studio 笔记本编写代码

数据工程师可以使用 AWS Glue Studio 中交互式笔记本界面或 AWS Glue 中的交互式会话，从而比以前更快、更轻松地编写 AWS Glue 任务。

主题

- 笔记本使用概览
- 在 AWS Glue Studio 中使用笔记本创建 ETL 任务
- 笔记本编辑器组件
- 保存笔记本和任务脚本
- 管理笔记本会话
- CodeWhisperer 与一起使用 AWS Glue Studio notebooks

笔记本使用概览

AWS Glue Studio 允许您在基于 Jupyter 笔记本的笔记本界面中以交互方式编写任务。通过 AWS Glue Studio 中的笔记本，您可以在无需运行完整任务的情况下编辑任务脚本并查看输出；您可以在无需运行完整任务的情况下编辑数据集成代码并查看输出；您还可以添加标记并将笔记本保存为 .ipynb 文件和任务脚本。您可以在无需本地安装软件或管理服务器的情况下开启笔记本。当您对代码感到满意时，只需点击按钮，AWS Glue Studio 就可以将笔记本转换为 Glue 作业。

使用笔记本的一些益处包括：

- 没有要预置或管理的集群
- 没有要付费的空闲集群
- 无需预先配置
- 无需安装 Jupyter 笔记本
- 与 AWS Glue ETL 相同的运行时/平台

当您通过 AWS Glue Studio 开启笔记本时，所有配置步骤都已为您完成，让您在几秒钟后即可浏览数据并开始开发任务脚本。AWS Glue Studio 使用 AWS Glue Jupyter 内核配置 Jupyter notebook。使用此笔记本无需配置 VPC、网络连接或开发端点。

要使用笔记本界面创建任务，请执行以下操作：

- 配置必要的 IAM 权限。
- 开启笔记本会话，创建任务
- 在笔记本的单元格中写代码
- 运行并测试代码，查看输出
- 保存任务

保存笔记本后，笔记本即为完整 AWS Glue 任务。您可以管理任务的所有方面，例如调度任务运行、设置任务参数以及查看笔记本的任务运行历史记录。

在 AWS Glue Studio 中使用笔记本创建 ETL 任务

在 AWS Glue Studio 控制台中开启使用笔记本

1. 将 AWS Identity and Access Management 策略附上 AWS Glue Studio 用户并为 ETL 作业和笔记本创建 IAM 角色。

2. 按照 为 IAM 角色授予权限 中的描述，为笔记本配置额外的 IAM 安全措施
3. 请访问 https://console.aws.amazon.com/gluestudio/ 打开 AWS Glue Studio 控制台。

> **Note**
>
> 请确保您的浏览器不会阻止第三方 Cookie。浏览器的默认设置，或用户启用的设置阻止第三方 Cookie 时，都将阻止笔记本启动。有关管理 Cookie 的更多信息，请参阅：

- Chrome
- Firefox
- Safari

4. 选择左侧导航菜单中的 Jobs（任务）链接。
5. 选择 Jupyter notebook（Jupyter 笔记本），然后选择 Create（创建）以开启新的笔记本会话。
6. 在 Create job in Jupyter notebook（在 Jupyter 笔记本中创建任务）页面上，提供任务名称并选择要使用的 IAM 角色。请选择创建作业。

   等待很短一段时间后，将出现笔记本编辑器。
7. 添加代码后，您必须执行单元格才能初始化会话。有多种方法可以执行单元格：

   - 按下播放按钮。
   - 使用键盘快捷键：
     - 在 MacOS 上，Command 键 + Enter 键以运行单元格。
     - 在 Windows 上，Shift 键 + Enter 键以运行单元格。

   有关使用 Jupyter notebook 界面编写代码的信息，请参阅 Jupyter notebook 用户文档 。
8. 要测试脚本，请运行整个脚本或单个单元格。任何命令输出都将显示在单元格下方的区域中。
9. 完成笔记本开发后，您可以保存任务，然后运行。您可以在 Script（脚本）选项卡中找到脚本。您添加到笔记本的任意魔术都将被删除，并且不会被保存为所生成 AWS Glue 任务的脚本的一部分。AWS Glue Studio 会自动将 `job.commit()` 添加到从笔记本内容生成的脚本的末尾。

   有关运行任务的更多信息，请参阅 启动任务运行。

笔记本编辑器组件

笔记本编辑器界面有以下主要部分。

- 笔记本界面（主面板）和工具栏
- 任务编辑选项卡

笔记本编辑器

AWS Glue Studio 笔记本编辑器基于 Jupyter notebook 应用程序。AWS Glue Studio 笔记本界面与 Juypter Notebooks 提供的界面类似，笔记本用户界面一节对此进行了描述。交互式会话使用的笔记本是 Jupyter notebook。

尽管 AWS Glue Studio 笔记本与 Juptyer Notebooks 相似，但在几个关键方面有所不同：

- 目前，AWS Glue Studio 笔记本无法安装扩展
- 不能使用多个选项卡；任务和笔记本之间存在一一对应关系
- AWS Glue Studio 笔记本没有与 Jupyter 笔记本相同的顶部文件菜单
- 目前，AWS Glue Studio 笔记本只能运行 AWS Glue 内核。请注意，您不能自行更新内核。

AWS Glue Studio 任务编辑选项卡

用于与 ETL 任务交互的选项卡位于笔记本页面的顶部。这些选项卡类似于 AWS Glue Studio 的可视化任务编辑器中显示的选项卡，并且执行的操作也相同。

- Notebook（笔记本）– 使用此选项卡可使用笔记本界面查看任务脚本。
- Job details（任务详细信息）– 配置任务运行的环境和属性。
- Runs（运行）– 查看有关此任务以前运行的信息。
- Schedules（调度）– 配置在特定时间运行任务的调度表。

保存笔记本和任务脚本

您可以随时保存笔记本和正在创建的任务脚本。只需选择右上角的 Save（保存）按钮，就像使用可视化编辑器或脚本编辑器一样。

当您选择 Save（保存）时，笔记本文件将保存在默认位置：

- 默认情况下，任务脚本将保存到 Job Details（任务详细信息）选项卡中指示的 Amazon S3 位置，该选项卡位于任务详细信息属性 Script path（脚本路径）中的 Advanced properties（高级属性）下。任务脚本将保存在名为 `Scripts` 的子文件夹中。
- 默认情况下，笔记本文件 (`.ipynb`) 将保存到 Job Details（任务详细信息）选项卡中指示的 Amazon S3 位置，该选项卡位于任务详细信息 Script path（脚本路径）中的 Advanced properties（高级属性）下。笔记本文件将保存在名为 `Notebooks` 的子文件夹中。

> **Note**
>
> 保存任务时，任务脚本仅包含笔记本中的代码单元格。任务脚本中不包括 Markdown 单元格和魔术命令。但是，`.ipynb` 文件将包含任何 Markdown 和魔术命令。

保存任务后，可以使用在笔记本中创建的脚本运行任务。

管理笔记本会话

AWS Glue Studio 中的笔记本基于 AWS Glue 的交互式会话功能。使用交互式会话需要成本。为了帮助管理成本，您可以监控为您的账户创建的会话，并为所有会话配置原定设置。

更改所有笔记本会话的原定设置超时

默认情况下，如果预置的 AWS Glue Studio 笔记本已启动且未执行任何单元格，则笔记本将在 12 小时后超时。没有与之相关的成本，且超时不可配置。

执行单元格后，将启动一个交互式会话。此会话的默认超时时间为 48 小时。通过在执行单元格前传递 `%idle_timeout` 魔术命令，可以配置此超时。

在 AWS Glue Studio 中修改笔记本的原定设置会话超时

1. 在笔记本中，在单元格中输入 `%idle_timeout` 魔术命令并以分钟为单位指定超时值。
2. 例如：`%idle_timeout 15` 会将默认超时更改为 15 分钟。如果 15 分钟内未使用会话，则会话将自动停止。

安装其他 Python 模块

如果要使用 pip 将其他模块安装到会话中，可以使用 `%additional_python_modules` 将其添加到会话中：

```
%additional_python_modules awswrangler, s3://mybucket/mymodule.whl
```

additional_python_modules 的所有参数都将传递给 `pip3 install -m <>`

要查看可用 Python 模块的列表，请参阅 Using Python libraries with AWS Glue。

更改 AWS Glue 配置

您可以使用魔术来控制 AWS Glue 任务配置值。如果要更改任务配置值，您必须在笔记本中使用正确的魔法。请参阅 Magics supported by AWS Glue interactive sessions for Jupyter。

> **Note**
>
> 正在运行的会话的重写属性不再可用。要更改会话的配置，您可以停止会话，设置新的配置，然后启动新会话。

AWS Glue 支持各种工件类型。可以使用 `%worker_type` 设置工件类型。例如：`%worker_type G.2X` 。原定设置为 G.1X。

您还可以使用 `%number_of_workers` 指定工件数量。例如，要指定 40 个工件：`%number_of_workers 40`。

有关更多信息，请参阅 定义任务属性

停止笔记本会话

要停止笔记本会话，请使用魔术命令 `%stop_session`。

如果您在 AWS 控制台中离开笔记本，您将收到一条警告消息，您可以在其中选择停止会话。

CodeWhisperer 与一起使用 AWS Glue Studio notebooks

AWS Glue Studio 允许您在基于 Jupyter 笔记本的笔记本界面中以交互方式编写任务。使用 CodeWhisperer 可改善AWS Glue Studio笔记本中的创作体验。

Amazon CodeWhisperer 扩展支持通过生成代码建议和提出与代码问题相关的改进建议来编写代码。

什么是亚马逊 CodeWhisperer？

Amazon CodeWhisperer 是一项由机器学习提供支持的服务，可帮助提高开发人员的工作效率。CodeWhisperer 通过根据开发人员在自然语言中的注释以及他们在 IDE 中的代码生成代码推荐来实现这一目标。在预览期间，亚马逊 CodeWhisperer 提供 Java、Python JavaScript、C# 和 TypeScript 编程语言版本。该服务与 Amazon SageMaker Studio JupyterLab、 Amazon SageMaker 笔记本实例和其他集成开发环境 (IDE) 集成。

有关更多信息，请参阅 CodeWhisperer 使用进行设置AWS Glue Studio。

控制台上的 AWS Glue 作业运行状态

您可以查看 AWS Glue 提取、转换和加载（ETL）任务在运行时或停止后的状态。您可以使用 AWS Glue 控制台查看状态。有关作业运行状态的更多信息，请参阅 the section called “作业运行状态”。

访问任务监控控制面板

您可以在 AWS Glue 导航窗格中选择 Monitoring (监控) 链接，从而访问任务监控控制面板。

任务监控控制面板概览

任务监控控制面板提供任务运行的总体摘要，以及状态 Running (正在运行)、Canceled (已取消)、Success (成功) 或者 Failed (失败)。其他磁贴提供总体任务运行成功率、任务的预估 DPU 使用率，以及按任务类型、工件类型和天细分的任务状态计数。

磁贴中的图形是交互式。您可以选择图形中的任意数据块来运行筛选条件，仅显示页面底部 Job runs (任务运行) 表中的任务。

您可以使用 Date range (日期范围) 选择器更改此页面上显示的信息的日期范围。更改日期范围时，信息磁贴会进行调整，显示代表当前日期之前指定天数的值。如果您从日期范围选择器中选择 Custom (自定义)，您还可以使用特定日期范围。

任务运行视图

> **Note**
>
> 您可以在 90 天内访问工作流和任务运行的任务运行历史记录。

Job runs (任务运行) 资源列表显示符合指定日期范围和筛选条件的任务。

您可以根据其他条件（如状态、工件类型、任务类型和任务名称）筛选任务。在表格顶部的筛选条件框中，您可以输入要用作筛选条件的文本。当您输入文本时，将使用包含匹配文本的行更新表结果。

您可以从任务监控控制面板上的图形中选择元素，查看任务的子集。例如，如果您选择 Job runs summary (任务运行摘要) 磁贴中正在运行的任务的数量，则 Job runs (任务运行) 列表仅显示当前状态为 Running 的任务。如果您选择 Worker type breakdown (工件类型细分) 条形图，则 Job runs (任务运行) 列表中仅显示具有匹配工件类型和状态的任务运行。

Job runs (任务运行) 资源列表显示任务运行的详细信息。可以通过选择列标题对表中的行进行排序。此表包含以下信息：

属性	描述
作业名称	作业的名称。
类型	任务环境的类型： • Glue ETL：在由 AWS Glue 管理的 Apache Spark 环境中运行。 • Glue Streaming：在 Apache Spark 环境中运行，并在数据流上执行 ETL。 • Python Shell：以 Shell 运行 Python 脚本。
开始时间	此任务运行的启动日期和时间。
结束时间	此任务运行的完成日期和时间。
运行状态	任务运行的当前状态。值可以是： • STARTING • RUNNING • STOPPING • STOPPED • SUCCEEDED • FAILED • TIMEOUT
运行时间	任务运行使用资源的时间长度（以秒为单位）。

属性	描述
容量	此任务运行时可分配的 AWS Glue 数据处理单元（DPU）的最大数量。有关容量规划的更多信息，请参阅《AWS Glue 开发人员指南》中的 DPU 容量规划监控。

属性	描述
工作线程类型	任务运行时分配的预定义工件的类型。值可以是 G.1X、G.2X、G.4X 或者 G.8X。 • **G.1X** – 当您选择这种类型时，您还提供了 Number of workers (工件数量) 的值。每个工作线程映射到 1 个 DPU（4 个 vCPU，16 GB 内存）和 84GB 磁盘（大约 34GB 可用空间）。我们建议内存密集型作业使用该工作线程类型。这是针对 AWS Glue 2.0 版或更高版本任务的默认工件类型。 • **G.2X** – 当您选择这种类型时，您还提供了 Number of workers (工件数量) 的值。每个工作线程映射到 2 个 DPU（8 个 vCPU，32 GB 内存）和 128GB 磁盘（大约 77GB 可用空间）。我们建议将此工件类型用于内存密集型任务和运行机器学习转换的任务。 • **G.4X** – 当您选择这种类型时，您还提供了 Number of workers (工件数量) 的值。每个工作线程映射到 4 个 DPU（16 个 vCPU，64 GB 内存）和 256GB 磁盘（大约 235GB 可用空间）。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作线程类型仅适用于以下 AWS 区域的 AWS Glue 3.0 版或更高版本的 Spark ETL 作业：美国东部（俄亥俄州）、美国东部（弗吉尼亚州北部）、美国西部（俄勒冈州）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）、加拿大（中部）、欧洲地区（法兰克福）、欧洲地区（爱尔兰）和欧洲地区（斯德哥尔摩）。 • **G.8X** – 当您选择这种类型时，您还提供了 Number of workers (工件数量) 的值。每个工作线程映射到 8 个 DPU（32 个 vCPU，128 GB 内存）和 512GB 磁盘（大约 487GB 可

属性	描述
	用空间）。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作线程类型仅适用于 AWS Glue 3.0 版或更高版本的 Spark ETL 作业，其区域与 G.4X 工作线程类型支持的 AWS 区域相同。
DPU 小时	任务运行使用的 DPU 的估计数量。DPU 是处理能力的相对衡量标准。DPU 用于确定任务运行成本。有关更多信息，请参阅 AWS Glue 价格页面。

您可以在列表中选择任何任务运行并查看其他信息。选择任务运行，然后执行以下任一操作：

- 选择 Actions (操作) 菜单和 View job (查看任务) 选项，在可视化编辑器中查看任务。
- 选择 Actions (操作) 菜单和 Stop run (停止运行) 选项，停止任务的当前运行。
- 选择 View CloudWatch logs (查看 CloudWatch 日志) 按钮，查看该任务的任务运行日志。
- 选择查看详细信息可查看“作业运行详细信息”页面。

查看任务运行日志

您可通过多种方式查看任务日志：

- 在 Monitoring (监控) 页面中的 Job runs (任务运行) 表中，选择任务运行，然后选择 View CloudWatch logs (查看 CloudWatch 日志)。
- 在可视化任务编辑器中，在任务的 Runs (运行) 选项卡上，选择超链接以查看日志：
  - Logs (日志) – 链接到为任务运行启用连续日志记录时写入的 Apache Spark 任务日志。当您选择此链接时，它会将您转到 `/aws-glue/jobs/logs-v2` 日志组中的 Amazon CloudWatch 日志。默认情况下，日志会排除无用的 Apache Hadoop YARN 检测信号和 Apache Spark 驱动程序或执行程序日志消息。有关连续日志记录的更多信息，请参阅的《AWS Glue 开发人员指南》中的连续日志记录 AWS Glue 任务。
  - Error logs (错误日志) – 链接到写入此任务运行的 `stderr` 的日志。当您选择此链接时，它会将您转到 `/aws-glue/jobs/error` 日志组中的 Amazon CloudWatch 日志。您可以使用这些日志查看有关任务运行期间遇到的错误的详细信息。

- Output logs (输出日志) – 链接到写入此任务运行的 `stdout` 的日志。当您选择此链接时，它会将您转到 `/aws-glue/jobs/output` 日志组中的 Amazon CloudWatch 日志。您可以使用这些日志，查看有关在 AWS Glue Data Catalog 中创建的表和遇到的错误的详细信息。

查看任务运行的详细信息

您可以在 Monitoring (监控) 页面上的 Job runs (任务运行) 列表中选择任务，然后选择 View run details (查看运行详细信息)，查看该任务运行的详细信息。

任务运行详细信息页面上显示的信息包括：

属性	描述
作业名称	作业的名称。
运行状态	任务运行的当前状态。值可以是： • STARTING • RUNNING • STOPPING • STOPPED • SUCCEEDED • FAILED • TIMEOUT
Glue 版本	作业运行使用的 AWS Glue 版本。
最近的尝试	此作业运行的自动重试次数。
开始时间	此任务运行的启动日期和时间。
结束时间	此任务运行的完成日期和时间。
开始时间	准备运行作业运行所花费的时间。
执行时间	运行作业脚本花费的时间。
触发器名称	与作业关联的触发器的名称。

属性	描述
上次修改日期	上次修改作业的日期。
安全配置	作业的安全配置，包括 Amazon S3 加密、CloudWatch 加密和作业书签加密设置。
超时	作业运行超时阈值。
已分配容量	此任务运行时可分配的 AWS Glue 数据处理单元（DPU）的最大数量。有关容量规划的更多信息，请参阅《AWS Glue 开发人员指南》中的 DPU 容量规划监控。
最大容量	任务运行可用的最大容量。
工作线程数	作业运行所用的工作线程数。
工作线程类型	为任务运行分配的预定义工件的类型。值可以是 `G.1X` 或者 `G.2X`。 • **`G.1X`** – 当您选择这种类型时，您还提供了 Number of workers (工件数量) 的值。每个工作线程映射到 1 个 DPU（4 个 vCPU，16 GB 内存，64 GB 磁盘），并且每个工作线程提供 1 个执行程序。我们建议内存密集型作业使用该工作线程类型。这是针对 AWS Glue 2.0 版或更高版本任务的默认工件类型。 • **`G.2X`** – 当您选择这种类型时，您还提供了 Number of workers (工件数量) 的值。每个工作线程映射到 2 个 DPU（8 个 vCPU，32 GB 内存，128 GB 磁盘），并且每个工作线程提供 1 个执行程序。我们建议将此工件类型用于内存密集型任务和运行机器学习转换的任务。
日志	指向连续日志记录（`/aws-glue/jobs/logs-v2`）的作业日志链接

属性	描述
输出日志	指向作业输出日志文件（`/aws-glue/jobs/output`）的链接。
错误日志	指向作业错误日志文件（`/aws-glue/jobs/error`）的链接。

您还可以查看以下附加项目，这些项目在您查看最近任务运行的信息时可用。有关更多信息，请参阅 the section called “查看最近任务运行的信息”。

- 输入参数
- 连续日志
- 指标 – 您可以直观地查看基本指标。有关所包含指标的更多信息，请参阅 the section called “查看 Spark 作业运行的 Amazon CloudWatch 指标”。
- Spark UI – 您可以在 Spark UI 中直观地查看任务的 Spark 日志。有关使用 Spark Web UI 的更多信息，请参阅 the section called “使用 Spark UI 进行监控”。按照 the section called “为作业启用 Spark UI” 中描述的过程启用此功能。

查看 Spark 作业运行的 Amazon CloudWatch 指标

在任务运行的详细信息页面上的 Run details (运行详细信息) 部分下面，您可以查看任务指标。AWS Glue Studio 将任务指标发送到 Amazon CloudWatch，用于每次任务运行。

AWS Glue 每 30 秒将指标报告到 Amazon CloudWatch AWS Glue 指标表示先前报告的值的增量值。在适当时，指标控制面板会聚合（合计）30 秒值以获取整个最后一分钟的值。但是，AWS Glue 传递给 Amazon CloudWatch 的 Apache Spark 指标通常是表示在报告它们时的当前状态的绝对值。

> **Note**
>
> 您必须配置您的账户才能访问 Amazon CloudWatch。

指标提供有关任务运行的信息，例如：

- ETL Data Movement (ETL 数据移动) – 从 Amazon S3 中读取或向其中写入的字节数。

- Memory Profile: Heap used (内存配置文件：使用的堆) – Java 虚拟机（JVM）堆使用的内存字节数。
- Memory Profile: heap usage (内存配置文件：堆使用情况) – JVM 堆使用的内存所占的比例（比例：0–1）。
- CPU Load (CPU 负载) – 使用的 CPU 系统负载所占的比例（比例：0–1）。

查看 Ray 作业运行的 Amazon CloudWatch 指标

在任务运行的详细信息页面上的 Run details (运行详细信息) 部分下面，您可以查看任务指标。AWS Glue Studio 将任务指标发送到 Amazon CloudWatch，用于每次任务运行。

AWS Glue 每 30 秒将指标报告到 Amazon CloudWatch AWS Glue 指标表示先前报告的值的增量值。在适当时，指标控制面板会聚合（合计）30 秒值以获取整个最后一分钟的值。但是，AWS Glue 传递给 Amazon CloudWatch 的 Apache Spark 指标通常是表示在报告它们时的当前状态的绝对值。

> **Note**
>
> 您必须配置您的账户才能访问 Amazon CloudWatch，如中所述。

在 Ray 作业中，您可以查看以下聚合指标图表。借助这些功能，您可以建立集群和任务的配置文件，也可以访问有关每个节点的详细信息。支持这些图表的时间序列数据可在 CloudWatch 中找到，以供进一步分析。

任务配置文件：任务状态

显示系统中 Ray 任务的数量。每个任务生命周期都有自己的时间序列。

任务配置文件：任务名称

显示系统中 Ray 任务的数量。仅显示待处理任务和活动任务。每种类型的任务（按名称）都有自己的时间序列。

集群配置文件：正在使用的 CPU

显示使用的 CPU 内核数。每个节点都有自己的时间序列。节点由 IP 地址标识，IP 地址是临时的，仅用于识别。

集群配置文件：对象存储内存使用情况

显示 Ray 对象缓存的内存使用情况。每个内存位置（物理内存、缓存在磁盘上以及溢出在 Amazon S3 中）都有自己的时间序列。对象存储管理集群中所有节点的数据存储。有关更多信息，请参阅 Ray 文档中的 Objects。

集群配置文件：节点数

显示为集群配置的节点数量。

节点详细信息：CPU 使用情况

以百分比形式显示每个节点上的 CPU 使用率。每个系列都显示节点上所有内核的 CPU 使用率的汇总百分比。

节点详细信息：内存使用情况

显示每个节点的内存使用情况（以 GB 为单位）。每个系列都显示节点上所有进程之间聚合的内存，包括 Ray 任务和 Plasma 存储进程。这不会反映存储到磁盘或溢出到 Amazon S3 的对象。

节点详细信息：磁盘使用情况

显示每个节点的磁盘使用情况（以 GB 为单位）。

节点详细信息：磁盘 I/O 速度

以 KB/s 为单位显示每个节点上的磁盘 I/O。

节点详细信息：网络 I/O 吞吐量

以 KB/s 为单位显示每个节点上的网络 I/O。

节点详细信息：Ray 组件的 CPU 使用情况

以所占核心的分数来显示 CPU 使用率。每个节点上的每个 ray 组件都有自己的时间序列。

节点详细信息：Ray 组件的内存使用情况

以 GiB 为单位显示内存使用情况。每个节点上的每个 ray 组件都有自己的时间序列。

检测和处理敏感数据

检测 PII 转换可识别数据源中的个人身份信息（PII）。选择要识别的 PII 实体、希望如何扫描数据以及如何处理检测 PII 转换识别的 PII 实体。

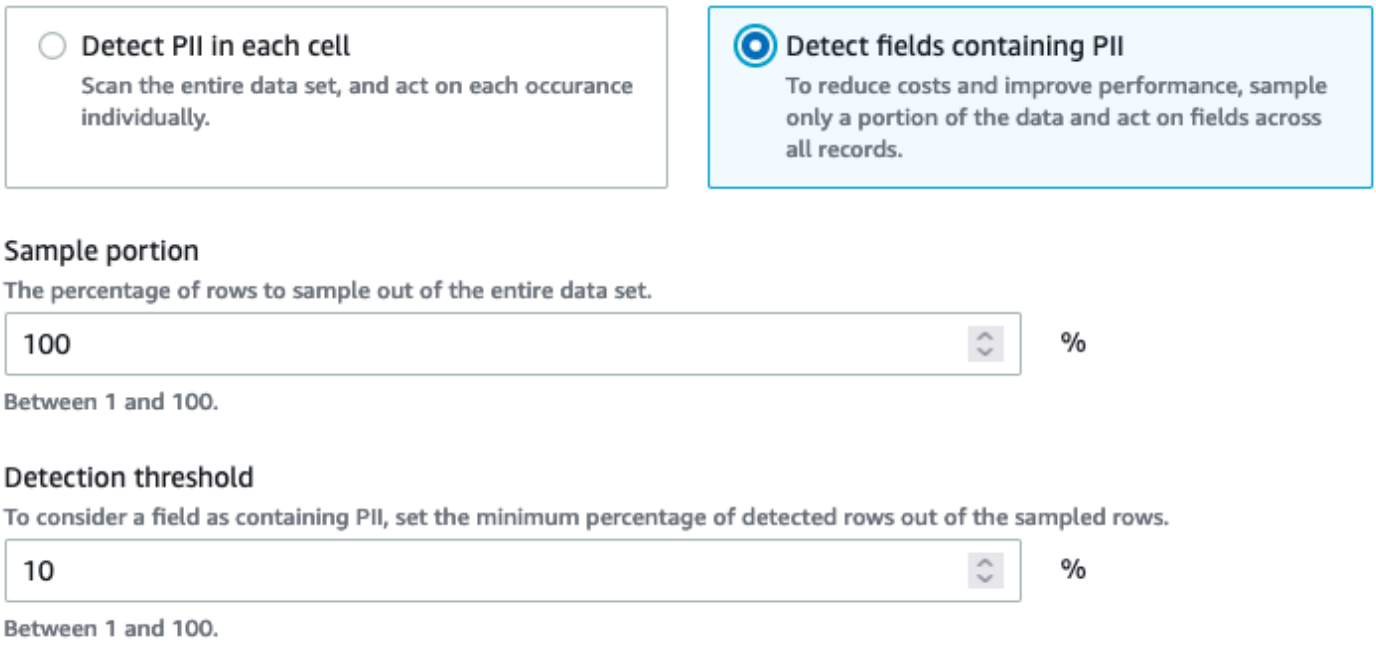
检测 PII 转换提供检测、掩盖或删除您定义的或由 AWS预定义的实体的功能。这赋能您提高合规性并减少责任。例如，您可能希望确保您的数据中不存在可以读取的个人身份信息，并希望用固定字符串（例如 xxx-xx-xxxx）、电话号码或地址来掩盖社会安全号码。

要在 AWS Glue Studio 之外处理敏感数据，请参阅 在 AWS Glue Studio 外部使用敏感数据检测

主题

选择希望如何扫描数据

当您扫描数据集中的个人身份信息（PII）等敏感数据时，您可以选择检测每行中的 PII，也可以检测包含 PII 数据的列。



当您选择 Detect PII in each cell（在每个单元格中检测 PII）时，您就是选择扫描数据源中的所有行。这是一次全面扫描，以确保识别 PII 实体。

当您选择 Detect fields containing PII（检测包含 PII 的字段）时，您就是选择扫描行样本以查找 PII 实体。这种方法可以将成本和资源保持在较低水平，同时还可以识别发现 PII 实体的字段。

当您选择检测包含 PII 的字段时，可以对部分行进行抽样来降低成本并提高性能。选择此选项将允许您指定其他选项：

- Sample portion（抽样比例）：此选项允许您指定要抽样的行的百分比。例如，如果输入“50”，即表示您指定希望 PII 实体扫描 50% 的行。
- Detection threshold（检测阈值）：此选项允许您指定将整个列标识为具有 PII 实体时，包含 PII 实体的行的百分比。例如，如果输入“10”，即表示指定扫描到行中 PII 实体“美国电话号码”的数量必须达到 10% 或以上，才能将该字段标识为具有 PII 实体“美国电话号码”。如果包含 PII 实体的行百分比小于 10%，则该字段将不会标注为包含 PII 实体“美国电话号码”。

选择要检测的 PII 实体

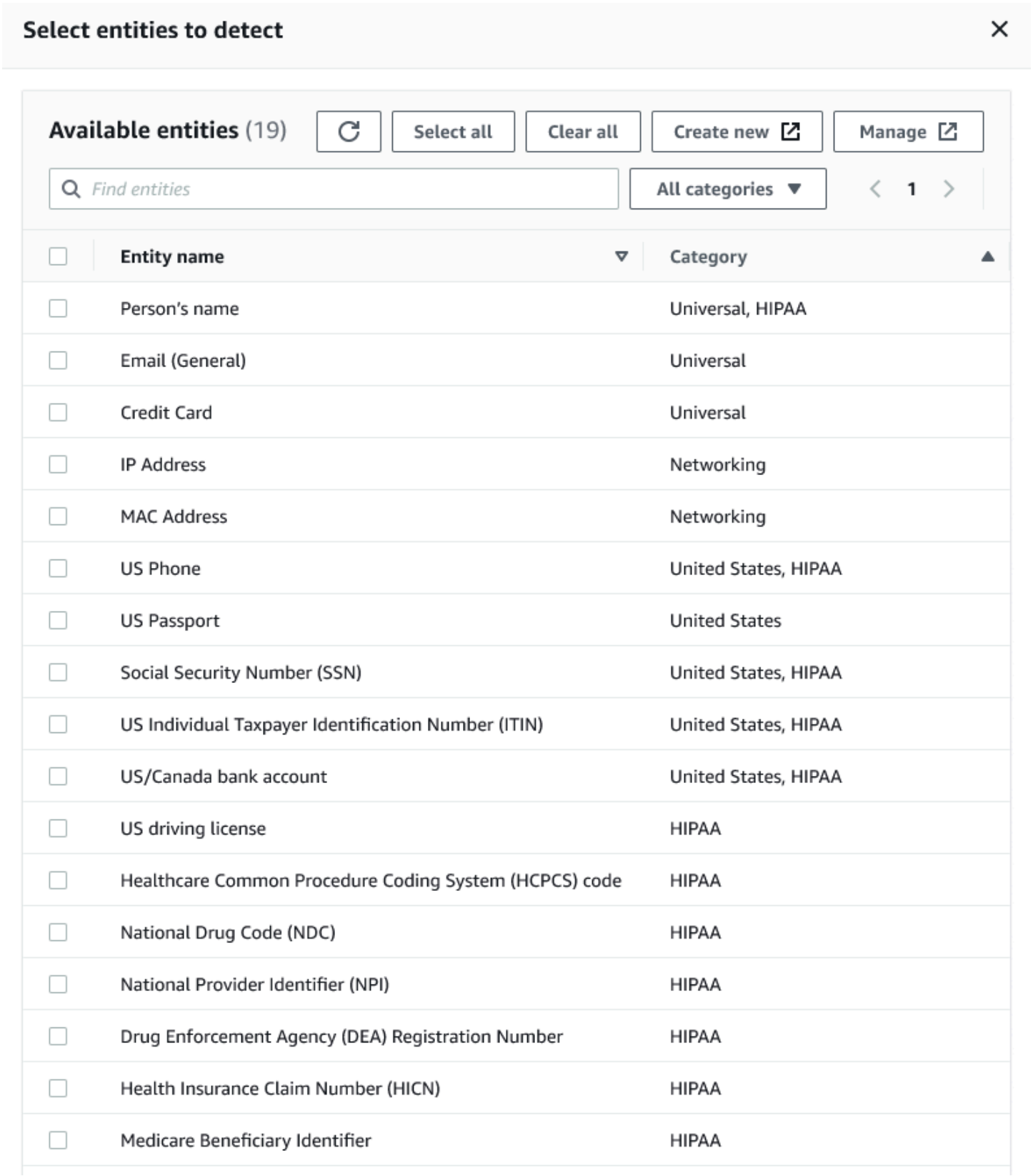
如果选择 Detect PII in each cell（在每个单元格中检测 PII），您可以选择以下三个选项之一：

- 所有可用的 PII 模式-这包括 AWS 实体。
- 选择类别 - 当您选择类别时，PII 模式将自动在您选择的类别中包括模式。
- 选择特定模式 - 仅能检测到您选择的模式。

有关托管的敏感数据类型的完整列表，请参阅 Managed data types。

选择所有可用的 PII 模式

如果您选择 “所有可用的 PII 模式”，请选择预定义的实体。 AWS您可以选择一个、多个或所有实体。


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

选择类别

如果您选择了 Select categories（选择类别）作为要检测的 PII 模式，则您可以选择下拉菜单中的选项。注意，某些实体可能属于多个类别。例如，人员姓名是属于通用和 HIPAA 类别的实体。

- 通用（例如：电子邮件、信用卡）
- HIPAA [例如：美国驾照、医疗保健通用程序编码系统 (HCPCS) 代码]
- 联网（例如：IP 地址、MAC 地址）
- 阿根廷
- 澳大利亚
- 奥地利
- 比利时
- 波斯尼亚
- 保加利亚
- 加拿大
- 智利
- 哥伦比亚
- 克罗地亚
- 塞浦路斯
- 捷克
- 丹麦
- 爱沙尼亚
- 芬兰
- 法国
- 德国
- 希腊
- 匈牙利
- 爱尔兰
- 韩国
- 日本
- 墨西哥

- 荷兰
- 新西兰
- 挪威
- 葡萄牙
- 罗马尼亚
- 新加坡
- 斯洛伐克
- 斯洛文尼亚
- 西班牙
- 瑞典
- 瑞士
- 土耳其
- 乌克兰
- 美国
- 英国
- 委内瑞拉

选择特定模式

如果选择 Select specific patterns（选择特定模式）作为要检测的 PII 模式，您可以搜索或浏览已创建的模式列表，或者创建新的检测实体模式。

以下步骤介绍了如何新建用于检测敏感数据的自定义模式。您将通过输入自定义模式的名称、添加正则表达式以及定义上下文字词来创建自定义模式。

1. 若要创建新模式，请单击 Create new（新建）按钮。

2. 在创建检测实体页面中，输入实体名称和正则表达式。AWS Glue 将使用正则表达式 (Regex) 来匹配实体。

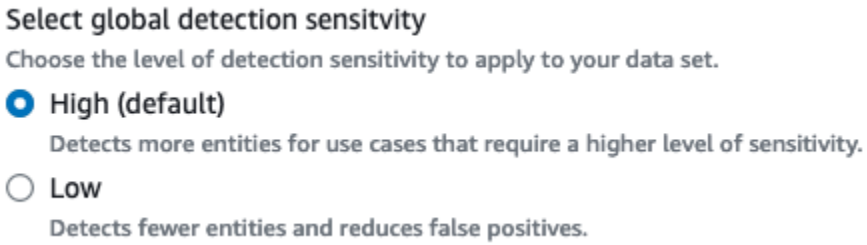
3. 单击 Validate（验证）。如果验证成功，您将看到一条确认消息，指出字符串是有效的正则表达式。如果验证不成功，您将看到一条消息，指出字符串不符合正确的格式和可接受的字符文本、运算符或结构。
4. 除了正则表达式之外，您还可以选择添加上下文字词。上下文字词可以提高匹配的概率。在字段名称没有描述实体的情况下，这些功能非常有用。例如，社会安全号码可以被命名为“SSN”或“SS”。添加这些上下文字词有助于匹配实体。
5. 单击 Create（创建）以创建检测实体。任何创建的实体在 AWS Glue Studio 控制台中可见。单击左侧导航菜单中的 Detection entities（检测实体）。

   您可以从 Detection entities（检测实体）页面编辑、删除或创建检测实体。您还可以使用搜索字段搜索模式。

指定检测灵敏度级别

使用检测敏感数据功能时，您可以设置灵敏度级别。

- 高 –（默认）适用于需要更高灵敏度级别的应用场景，会检测出更多实体。2023 年 11 月之后创建的所有 AWS Glue 作业都将自动启用此设置。
- 低 – 会减少检测出的实体数量并减少误报。

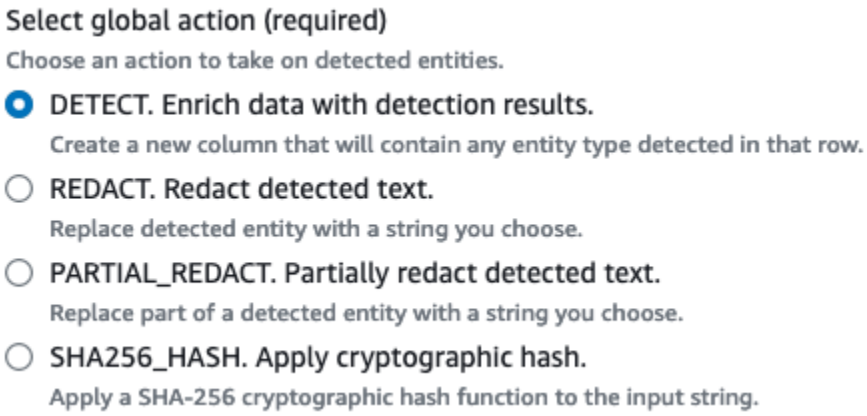


选择如何处理已识别的 PII 数据

如果选择整个数据来源中的 PII，则可以选择应用某个全局操作：

- Enrich data with detection results（使用检测结果丰富数据）：如果选择在每个单元格中检测 PII，则可以将检测到的实体存储到新列中。
- Redact detected text（编校检测到的文本）：可以使用在可选的替换文本输入字段中指定的字符串替换检测到的 PII 值。如果未指定字符串，则检测到的 PII 实体将替换为“*******”。

- 部分编辑检测到的文本：可以使用选定的字符串来部分替换检测到的 PII 值。提供两种选项：只留结尾不掩蔽，或通过显式正则表达式模式来掩蔽。AWS Glue 2.0 未提供此功能。
- Apply cryptographic hash（应用加密哈希）：您可以将检测到的 PII 值传递给 SHA-256 加密哈希函数，并将该值替换为该函数的输出。



AWS Glue 版本 2.0 与 3.0 及以上版本的区别

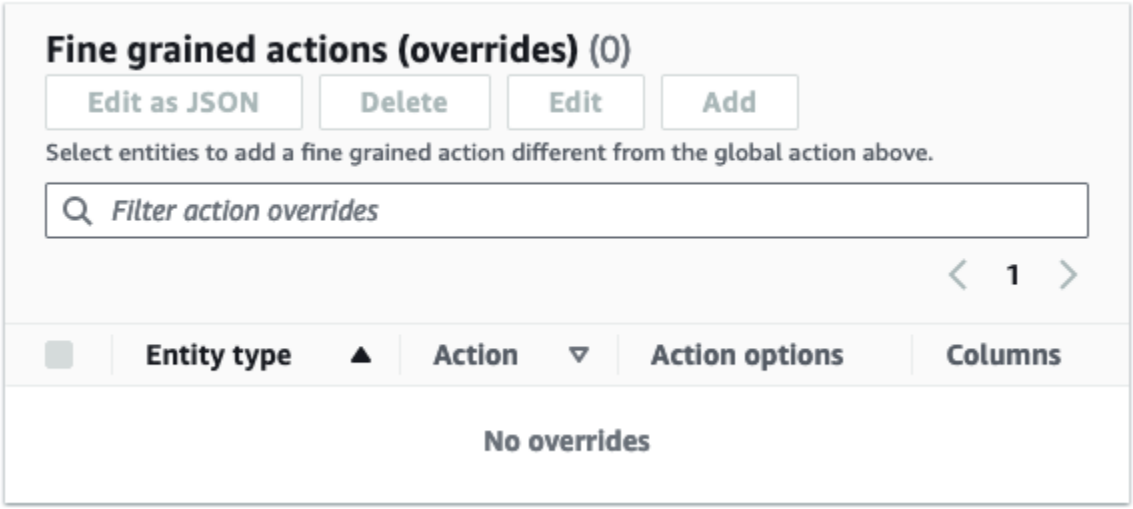
AWS Glue2.0 作业将在补充列中返回一个 DataFrame 包含检测到的 PII 信息的新任务。任何编辑或散列化处理都可通过视觉对象选项卡中的 AWS Glue 脚本查看。

AWS Glue3.0 和 4.0 任务将返回一个 DataFrame 带有相同补充列的新任务。此外还包含一个新键“actionUsed”，值可能为 `DETECT`、`REDACT`、`PARTIAL_REDACT` 或 `SHA256_HASH`。如果选择了屏蔽操作，则 DataFrame 会返回屏蔽敏感数据的数据。

添加精细操作覆盖

可以将其他检测和操作设置添加到精细操作覆盖表中，从而让您能够实现以下目的：

- 在检测范围中包含或排除特定的列 – 数据来源上的推断 Schema 将使用可用列来填充表。
- 指定比使用全局操作时更精细的特定设置 – 例如，您可以为不同的实体类型指定不同的掩蔽文本设置。
- 指定与全局操作不同的操作 – 如果要对不同的敏感数据类型应用不同的操作，则可以通过此设置来完成。请注意，不能在同一列上使用两个不同的 edit-in-place 操作（密文和哈希），但可以始终使用 detect。



使用 AWS Glue Studio 管理 ETL 任务

您可以使用 AWS Glue Studio 中的简单图形界面，管理您的 ETL 任务。使用导航菜单，选择 Jobs (任务) 查看 Jobs (任务) 页面。在此页面上，您可以看到您使用 AWS Glue Studio 或 AWS Glue 控制台创建的所有任务。您可以在此页面上查看、管理和运行您的任务。

在此页面上，您还可以执行以下任务：

- 启动任务运行
- 计划任务运行
- 管理任务计划
- 停止任务运行
- 查看您的作业
- 查看最近任务运行的信息
- 查看任务脚本
- 修改任务属性
- 保存任务
- 克隆任务
- 删除任务

启动任务运行

在 AWS Glue Studio 中，您可以按需运行您的任务。任务可以多次运行，每次您运行任务时，AWS Glue 会收集有关任务活动和绩效的信息。此信息称为任务运行，由任务运行 ID 标识。

您可以通过以下方式在 AWS Glue Studio 中启动任务：

- 在 Jobs (任务) 页面上，选择要启动的任务，然后选择 Run job (运行任务) 按钮。
- 如果您在可视化编辑器中查看任务并且任务已保存，则可以选择 Run (运行) 按钮启动任务运行。

有关任务运行的更多信息，请参阅《AWS Glue 开发人员指南》中的在 AWS Glue Glue 控制台上处理任务。

计划任务运行

在 AWS Glue Studio 中，您可以创建计划，让您的任务在特定时间运行。您可以指定约束条件，例如任务运行次数、它们在一周中的哪几天运行，以及具体在什么时间运行。这些约束基于 `cron`，与 `cron` 具有相同的限制。例如，如果您选择在每月第 31 天运行您的任务，请记住，有些月份没有 31 天。有关 `cron` 的更多信息，请参阅《AWS Glue 开发人员指南》中的 Cron 表达式。

按照计划运行任务

1. 使用以下某种方法创建任务计划：
   - 在 Jobs (任务) 页面上，选择要为其创建计划的任务，选择 Actions (操作)，然后选择 Schedule job (计划任务)。
   - 如果您在可视化编辑器中查看任务并且任务已保存，则选择 Schedules (计划) 选项卡。然后，选择 Create Schedule (创建计划)。
2. 在 Schedule job run (计划任务运行) 页面上，输入以下信息：
   - Name (名称)：输入您的任务计划的名称。
   - Frequency (频率)：输入任务计划的频率。您可以选择以下选项：
     - Hourly (每小时)：任务将每小时运行一次，从特定的分钟开始。您可以指定任务运行小时的具体 Minute (分钟)。默认情况下，当您选择每小时时，任务将在小时之初（分钟 0）开始运行。
     - Daily (每天)：任务将每天运行，从某个时间开始。您可以指定任务运行小时的具体 Minute (分钟) 以及任务的 Start hour (开始小时)。小时使用 23 小时制时钟指定，其中您使用数字 13

到 23 表示下午小时。分钟和小时的默认值为 0，这意味着如果您选择 Daily (每天)，则默认情况下，任务将在午夜运行。

- Weekly (每周)：任务将在每周的一天或多天内运行。除前面针对“Daily (每天)”介绍的相同设置外，您还可以选择任务运行的具体周时间。您可以选择一天或多天。
- Monthly (每月)：任务将在每月的特定日期运行。除前面针对“Daily (每天)”介绍的相同设置外，您还可以选择任务运行的具体日期。将日期指定为 1 到 31 之间的数值。如果您选择一个月中不存在的日期，例如 2 月 30 日，则任务在该月不会运行。
- 自定义：使用 `cron` 语法输入任务计划的表达式。Cron 表达式允许您创建更复杂的计划，例如每月的最后一天（而不是该月的特定日期），或者每三个月的第 7 天和第 21 天。

  请参阅《AWS Glue 开发人员指南》中的 Cron 表达式

- Description (描述)：您可以有选择地为任务计划输入描述。如果您计划为多项任务使用相同的计划，则描述有助于更轻松地确定任务计划的用途。

3. 选择 Create schedule (创建计划)，保存任务计划。
4. 创建计划后，控制台页面的顶部会显示一条成功消息。您可以选择此横幅中的 Job details (任务详细信息)，查看任务详细信息。这样将打开可视化任务编辑器页面，其中 Schedules (计划) 选项卡处于选中状态。

管理任务计划

为任务创建计划后，您可以在可视化编辑器中打开任务，然后选择 Schedules (任务) 选项卡以管理计划。

在可视化编辑器的 Schedules (计划) 选项卡，您可以执行以下任务：

- 创建新计划

  选择 Create schedule (创建计划)，然后输入计划的信息，如 the section called “计划任务运行” 中所述。

- 编辑现有计划。

  选择要编辑的计划，然后依次选择 Action (操作)、Edit schedule (编辑计划)。当您选择编辑现有计划时，Frequency (频率) 显示为 Custom (自定义)，并且计划显示为 `cron` 表达式。您可以修改 `cron` 表达式，也可以使用 Frequency (频率) 按钮指定新计划。完成更改后，选择 Update schedule (更新计划)。

- 暂停活动计划。

选择活动计划，然后依次选择 Action (操作)、Pause schedule (暂停计划)。计划会立即停用。选择刷新（重新加载）按钮，查看更新的任务计划状态。

- 恢复暂停的计划。

  选择停用计划，然后依次选择 Action (操作)、Resume schedule (恢复计划)。计划会立即激活。选择刷新（重新加载）按钮，查看更新的任务计划状态。

- 删除计划。

  选择要删除的计划，然后依次选择 Action (操作)、Delete schedule (删除计划)。计划会立即删除。选择刷新（重新加载）按钮，查看更新的任务计划列表。该计划将显示 Deleting (删除) 状态，直到它完全删除。

停止任务运行

您可以在任务完成任务运行之前将其停止。如果您知道任务未正确配置，或者任务花费太长时间而未完成，则可以选择此选项。

在 Monitoring (监控) 页面中的 Job runs (任务运行) 列表中，选择要停止的任务，然后依次选择 Actions (操作)、Stop run (停止运行)。

查看您的作业

您可以在 Jobs (任务) 页面查看您的所有任务。您可以在导航窗格中选择 Jobs (任务) 以访问此页面。

在 Jobs (任务) 页面上，您可以查看账户中创建的所有任务。Your jobs (您的任务) 列表会显示任务名称、类型、上次任务运行的状态，以及任务创建和上次修改的日期。您可以选择任务名称，查看相关任务的详细信息。

您还可以使用监控控制面板查看所有任务。您可以在导航窗格中选择 Monitoring (监控) 以访问控制面板。

自定义任务显示

您可以在 Jobs (任务) 页面的 Your jobs (您的任务) 部分中自定义任务的显示方式。此外，您可以在搜索文本字段中输入文本，以便仅显示名称包含该文本的任务。

如果选择 Your jobs (您的任务) 部分中的设置图标

，

您可以自定义 AWS Glue Studio 显示表中信息的方式。您可以选择在显示中将文本换行，更改页面上显示的任务数，以及指定要显示的列。

查看最近任务运行的信息

在源位置添加新数据时，任务可以多次运行。每次运行任务时，都会为任务运行分配唯一 ID，并收集有关该任务运行的信息。您可以使用以下方法查看此类信息：

- 选择可视化编辑器的 Runs (运行) 选项卡，查看当前显示的任务的任务运行信息。

  在 Runs (运行) 选项卡（ Recent job runs (最近任务运行) 页面）上，每次任务运行都有一张卡。Runs (运行) 选项卡上显示的信息包括：

  - 任务运行 ID
  - 此任务的尝试运行次数
  - 任务运行的状态
  - 任务运行的开始和结束时间
  - 任务运行的运行时
  - 指向任务日志文件的链接
  - 指向任务日志文件的链接
  - 失败任务返回的错误
- 您可以选择一个任务运行以查看有关任务的其他信息，包括以下信息：
  - 输入参数
  - 连续日志
  - 指标 – 您可以直观地查看基本指标。有关所包含指标的更多信息，请参阅 the section called “查看 Spark 作业运行的 Amazon CloudWatch 指标”。
  - Spark UI – 您可以在 Spark UI 中直观地查看任务的 Spark 日志。有关使用 Spark Web UI 的更多信息，请参阅 the section called “使用 Spark UI 进行监控”。按照 the section called “为作业启用 Spark UI” 中描述的过程启用此功能。

您可以选择查看详细信息，以在作业运行详细信息页面上查看类似的信息。您还可以通过监控页面导航到作业运行详细信息页面。在导航窗格中，选择 Monitoring (监控)。向下滚动到 Job runs (任务运行) 列表。选择任务，然后选择 View run details (查看运行详细信息)。内容在查看任务运行的详细信息中有描述。

有关任务日志的更多信息，请参阅查看任务运行日志。

查看任务脚本

提供任务中所有节点的信息后，AWS Glue Studio 会生成任务使用的脚本，用于从源中读取数据、转换数据以及将数据写入目标位置。如果保存任务，您可以随时查看此脚本。

查看任务生成的脚本

1. 在导航窗格中，选择 Jobs (任务)。
2. 在 Jobs (任务) 页面中的 Your Jobs (您的任务) 列表中，选择要查看的任务的名称。或者，您可以在列表中选择一项任务，选择 Actions (操作) 菜单，然后选择 Edit job (编辑任务)。
3. 在可视化编辑器页面上，选择 Script (脚本) 选项卡以查看任务任务脚本。

   如果您要编辑任务脚本，请参阅AWS Glue 编程指南。

修改任务属性

任务图中的节点定义了任务执行的操作，但也可以为任务配置多个属性。这些属性用于确定任务运行环境、任务使用的资源、阈值设置、安全设置等。

自定义任务运行环境

1. 在导航窗格中，选择 Jobs (任务)。
2. 在 Jobs (任务) 页面中的 Your Jobs (您的任务) 列表中，选择要查看的任务的名称。
3. 在可视化编辑器页面上，选择任务编辑窗格顶部的 Job details (任务详细信息)。
4. 根据需要修改任务属性。

   有关任务属性的更多信息，请参阅《AWS Glue 开发人员指南》中的定义任务属性。
5. 如果您需要指定以下附加任务属性，则展开 Advanced properties (高级属性) 部分：

   - Script filename (脚本文件名) – 在 Amazon S3 中存储任务脚本的文件的名称。
   - Script path (脚本路径) – 任务脚本的 Amazon S3 存储位置。
   - Job metrics (任务指标) – (不适用于 Python Shell 任务) 当此任务运行时启用 Amazon CloudWatch 指标创建。
   - Continuous logging (连续日志记录) – (不适用于 Python Shell 任务) 打开 CloudWatch 的连续日志记录，以便在任务完成之前可以查看日志
   - Spark UI 和 Spark UI logs path (Spark UI 日志路径) – (不适用于 Python Shell 任务) 使用 Spark UI 监控此任务，并指定 Spark UI 日志的位置。

- Maximum concurrency (最大并发) – 设置此任务业允许的并发运行的最大数量。
- Temporary path (临时路径) – 在 Amazon S3 中提供工作目录的位置，以便当 AWS Glue 运行脚本时在该位置写入临时中间结果。
- Delay notification threshold (minutes) (延迟通知阈值（分钟）) – 指定任务的延迟阈值。如果任务运行时间长于阈值指定的时间，则 AWS Glue 将任务的延迟通知发送给 CloudWatch。
- Security configuration (安全配置) 和 Server-side encryption (服务器端加密) – 使用这些字段选择任务的加密选项。
- Use Glue Data Catalog as the Hive metastore (使用 Glue 数据目录作为 Hive 元存储) – 选择此选项，如果您要使用 AWS Glue Data Catalog 作为 Apache 蜂巢元数据仓的替代方案。
- Additional network connection (附加网络连接) – 对于 VPC 中的数据源，您可以指定类型为 `Network`，确保您的任务通过 VPC 访问您的数据。
- Python library path (Python 库路径)、Dependent jars path (从属 jars 路径)（不适用于 Python Shell 任务）或 Referenced files path (引用的文件路径) – 使用这些字段指定任务运行脚本时使用的其他文件的位置。
- Job Parameters (任务参数) – 您可以添加作为命名参数传递给脚本的一组键值对。在 Python 对 AWS Glue API 的调用中，最好按名称显式传递参数。有关在任务脚本中使用参数的更多信息，请参阅《AWS Glue 开发人员指南》中的在 AWS Glue 中传递和访问 Python 参数。
- Tags (标签) – 您可以将标签添加到任务，帮助您组织和识别它们。

6. 修改任务属性后，保存任务。

将 Spark 随机播放文件存储在 Amazon S3 上

某些 ETL 任务需要读取和合并来自多个分区的信息，例如，在使用连接转换时。此操作称为随机排序。在随机排序过程中，数据会写入磁盘并通过网络传输。借助 AWS Glue 3.0 版，您可以将 Amazon S3 配置为这些文件的存储位置。AWS Glue 提供一个随机播放管理器，用于在 Amazon S3 中写入和读取随机文件。与本地磁盘（或者针对 Amazon EC2 进行了严格优化的 Amazon EBS）相比，从 Amazon S3 写入和读取随机放置文件的速度较慢（降低 5%-20%）。不过，Amazon S3 提供无限存储容量，因此您不必担心“`No space left on device`”错误运行您的任务。

将您的任务配置为使用 Amazon S3 对文件进行随机排序

1. 在 Jobs (任务) 页面中的 Your Jobs (您的任务) 列表中，选择要修改的任务的名称。
2. 在可视化编辑器页面上，选择任务编辑窗格顶部的 Job details (任务详细信息)。

   向下滚动到 Job parameters (任务参数) 部分。

3. 指定以下键/值对。

   - `--write-shuffle-files-to-s3` — `true`

     这是用于配置 AWS Glue 中随机排序管理器的主要参数，使用 Amazon S3 存储桶来写入和读取随机数据。此参数的默认值为 `false`。

   - （可选）`--write-shuffle-spills-to-s3` – `true`

     此参数允许您将溢出文件卸载到 Amazon S3 存储桶，从而为AWS Glue。只有将大量数据溢出到磁盘的大型工作负载才需要这样做。此参数的默认值为 `false`。

   - （可选）`--conf spark.shuffle.glue.s3ShuffleBucket` – `S3://<shuffle-bucket>`

     此参数指定在写入临时排序文件时要使用的 Amazon S3 存储桶。如果未设置此参数，则位置是 `shuffle-data` 文件夹中指定的临时路径（`--TempDir`）位置。

     > **Note**
     >
     > 确保 Shell 存储桶位于任务运行所在的相同 AWS 区域。
     > 此外，随机排序服务不会在任务完成运行后清理文件，因此您应该在 Shell 存储桶位置上配置 Amazon S3 存储生命周期策略。有关更多信息，请参阅《Amazon S3 用户指南》中的对象生命周期管理。

保存任务

红色 Job has not been saved (尚未保存任务) 标注将显示在 Save (保存) 按钮左侧，直到您保存任务。

保存您的任务

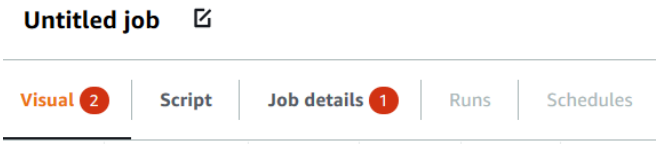
1. 请在 Visual (视觉对象) 和 Job details (任务详细信息) 选项卡中提供所有必需的信息。
2. 选择保存按钮。

   保存任务后，“not saved (未保存)”标注将更改为显示上次保存任务的时间和日期。

如果您在退出 AWS Glue Studio 时未保存您的任务，您下次登录 AWS Glue Studio 时，会显示通知。通知指示存在未保存的任务，并询问您是否要恢复。如果选择恢复任务，则可以继续对其进行编辑。

排查保存任务时的错误

如果选择 Save (保存) 按钮，但您的任务缺少了一些必需的信息，那么缺少信息的选项卡上会出现一个红色标注。标注中的数字表示检测到的缺失字段数。

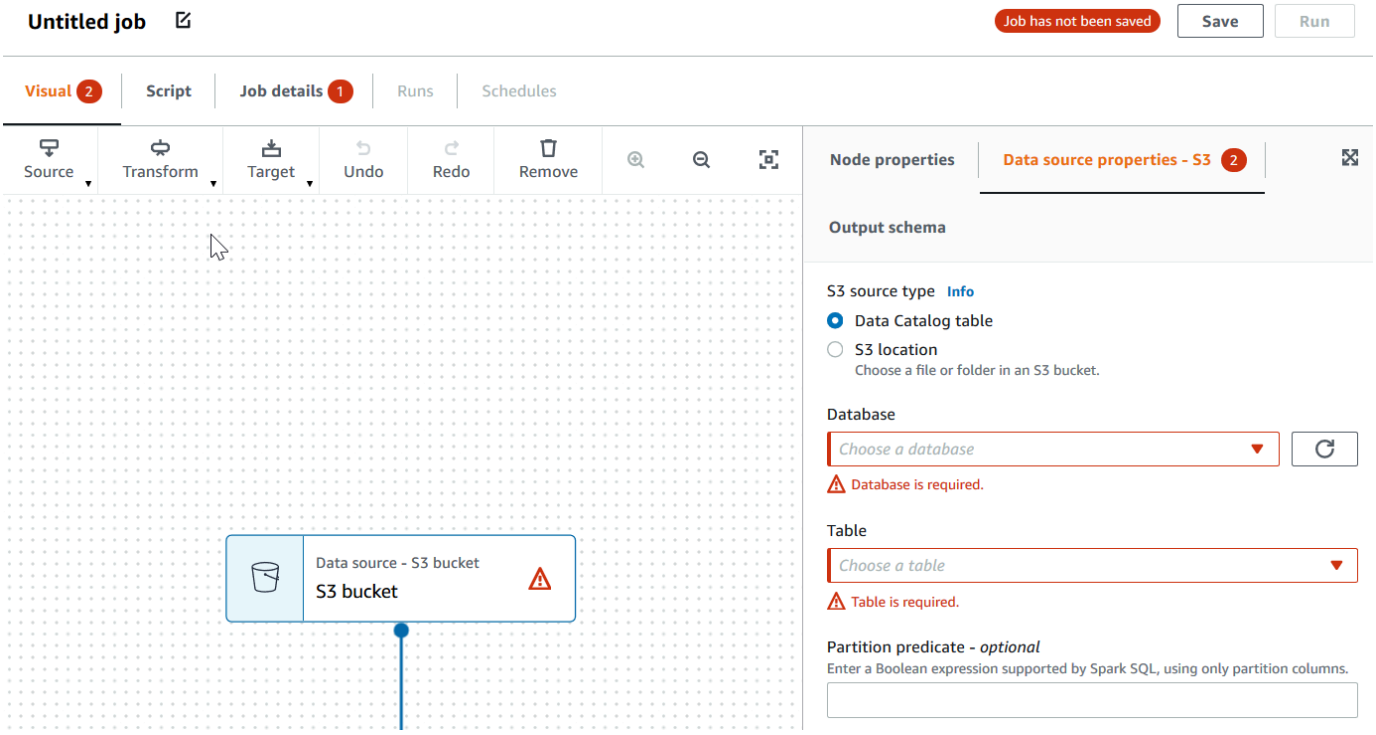


- 如果可视编辑器中的节点配置不正确，则 Visual (视觉对象) 选项卡显示红色标注，出现错误的节点显示警告符号

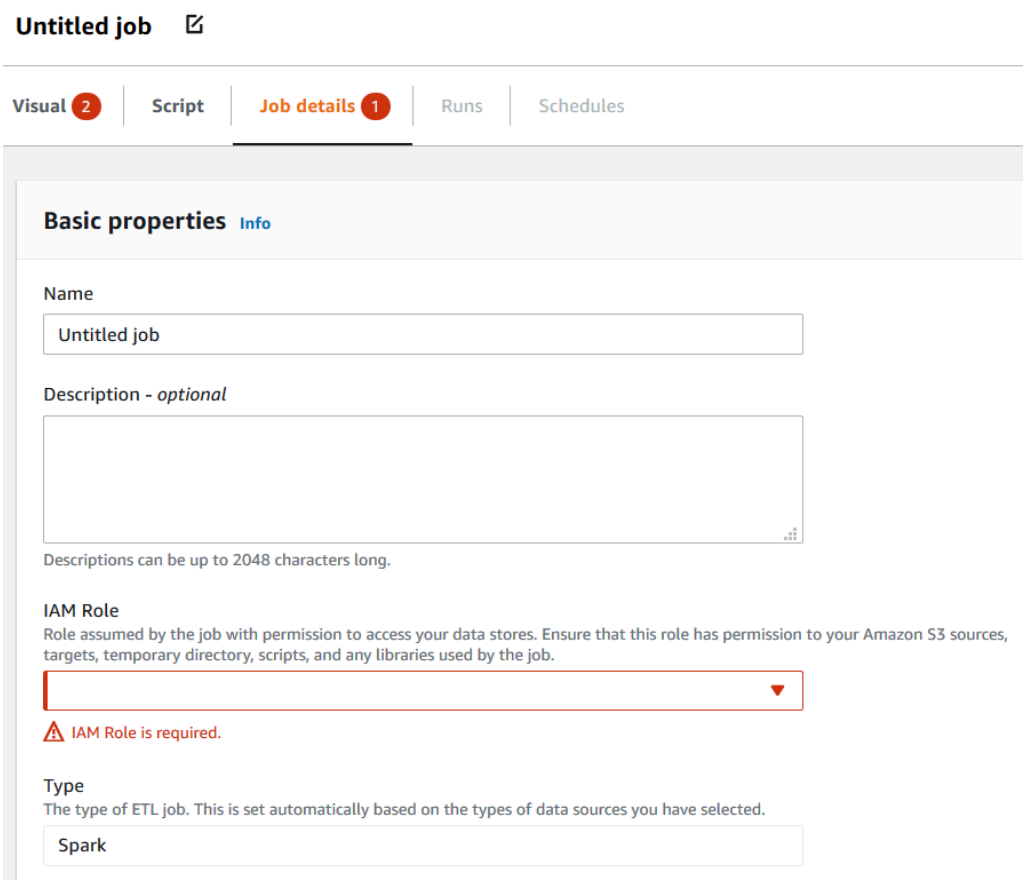
  ⚠

  。

  1. 选择节点。在节点详细信息面板中，缺少信息或不正确信息所在的选项卡上会显示一个红色标注。
  2. 在节点详细信息面板中选择显示红色标注的选项卡，然后找到突出显示的问题字段。字段下面的错误消息提供了有关问题的其他信息。



- 如果任务属性存在问题，则 Job details (任务详细信息) 选项卡显示红色标注。选择该选项卡并找到突出显示的问题字段。字段下面的错误消息提供了有关问题的其他信息。



克隆任务

您可以使用 Clone job (克隆任务) 操作将现有任务复制到新任务。

通过复制现有任务创建新任务

1. 在 Jobs (任务) 页面中的 Your jobs (您的任务) 列表中，选择要复制的任务。
2. 从 Actions (操作) 菜单中，选择 Clone job (克隆任务)。
3. 输入新任务的名称。然后，您可以保存或编辑任务。

删除任务

您可以删除不再需要的任务。您可以在单个操作中删除一个或多个任务。

要从 AWS Glue Studio 中删除任务

1. 在 Jobs (任务) 页面中的 Your jobs (您的任务) 列表中，选择要删除的任务。

2. 从 Actions (操作) 菜单中选择 Delete job (删除任务)。
3. 验证您要删除任务，方法是输入 **`delete`**。

在可视化编辑器中查看任务的 Job details (任务详细信息) 选项卡时，也可以删除保存的任务。

处理 AWS Glue 作业

以下几个部分提供有关 AWS Glue中的 ETL 和 Ray 作业的信息。

主题

AWS Glue 版本

可以在添加或更新作业时配置 AWS Glue 版本参数。AWS Glue 版本决定了 AWS Glue 支持的 Apache Spark 和 Python 版本。Python 版本指示了 Spark 类型的任务支持的版本。下表列出了可用的 AWS Glue 版本、相应的 Spark 和 Python 版本以及其他功能更改。

AWS Glue 版本

AWS Glue 版本	支持的运行时环境版本	支持的 Java 版本	功能更改
AWS Glue4.0	Spark 环境版本 • Spark 3.3.0 • Python 3.10	Java 8	AWS Glue 4.0 是 AWS Glue 的最新版本。此 AWS Glue 版本内置了多项优化和升级，例如： • Spark 功能从 Spark 3.1 到 Spark 3.3 进行了多项升级： • 与 Pandas 配对时的功能进行了多项改进。有关更

AWS Glue 版本	支持的运行时环境版本	支持的 Java 版本	功能更改
			多信息，请参阅 Python 3.3 中的新增功能。 • 在 Amazon EMR 上开发的其他优化。 • 升级到 EMR 文件系统（EMRFS）2.53。 • 从 Log4j 1.x 迁移到 Log4j 2 • Python 模块在 AWS Glue 3.0 基础上进行了多项更新，例如 Boto 的升级版本。 • 升级了多个连接器，其中包括默认的 Amazon Redshift 连接器。请参阅 附录 C：连接器升级。 • 升级了多个 JDBC 驱动程序。请参阅 附录 B：JDBC 驱动程序升级。 • 使用新的 Amazon Redshift 连接器和 JDBC 驱动程序进行了更新。 • 为 Apache Hudi、Delta Lake

AWS Glue 版本	支持的运行时环境版本	支持的 Java 版本	功能更改
			和 Apache Iceberg 提供了本机开放式数据湖框架支持。 • 为基于 Amazon S3 的 Cloud Shuffle 存储插件（Apache Spark 插件）提供了本机支持，从而可以使用 Amazon S3 实现随机排序和弹性存储功能。 限制 以下是 AWS Glue 4.0 的限制： • AWS Glue 机器学习和个人身份信息（PII）转换在 AWS Glue 4.0 中尚不可用。 有关迁移到 AWS Glue 4.0 版本的更多信息，请参阅将 AWS Glue for Spark 作业迁移到 AWS Glue 版本 4.0。

AWS Glue 版本	支持的运行时环境版本	支持的 Java 版本	功能更改
	Ray 环境版本 • Ray 2.4.0 Python 3.9	不适用	使用 AWS Glue for Ray 构建和运行分布式 Python 应用程序。 • 支持 Python 3.9 中的 Ray-2.4.0 数据分布（`ray[data]`）。有关此 Ray 版本的更多信息，请参阅 Ray 存储库中的 Ray-2.4.0。GitHub • 支持在 `Ray2.4` 运行时环境中安装其他 Python 库。有关更多信息，请参阅 the section called “用于 Ray 作业的其他 Python 模块”。 • 将来自 Ray 作业的日志和指标与 Amazon 集成 CloudWatch。有关更多信息，请参阅 the section called “Ray 错误故障排除” 和 the section called “Ray 作业指标”。 • 在 AWS Glue Studio每个作业运行页面上聚合和可

AWS Glue 版本	支持的运行时环境版本	支持的 Java 版本	功能更改
			视化 Ray 作业的指标。 • 支持将文件分发到集群中的每个工作目录，将对象从 Ray 对象存储溢出到 Amazon S3，以及控制分配给 Ray 任务的最小工作节点数。有关更多信息，请参阅 the section called “Ray 作业参数”。 AWS Glue 4.0 中对 Ray 作业的限制 • AWS Glue 在本版本中，Ray 的交互式会话仍处于预览状态。 • AWS Glue for Ray 目前无法与 Amazon VPC 集成。如果没有公共路由， AWS 则无法访问中 VPC 中的资源。有关在 Amazon VPC 中使用的 AWS Glue 更多信息，请参阅the section called “VPC 端点 (AWS PrivateLink)”。

AWS Glue 版本	支持的运行时环境版本	支持的 Java 版本	功能更改
			• AWS Glue for Ray 在美国东部（弗吉尼亚北部）、美国东部（俄亥俄州）、美国西部（俄勒冈）、亚太地区（东京）和欧洲（爱尔兰）上市。

AWS Glue 版本	支持的运行时环境版本	支持的 Java 版本	功能更改

AWS Glue3.0	• Spark 3.1.1
	• Python 3.7
		Java 8	除了 Spark 引擎升级到 3.0 之外，此 AWS Glue 版本还内置了一些优化和升级，例如：
			• 针对 Spark 3.0 构建 AWS Glue ETL 库，Spark 3.0 是 Spark 的主要版本。
			• AWS Glue 3.0 支持流式传输任务。
			• 包括针对性能和可靠性的新 AWS Glue Spark 运行时优化：
			  • 基于 Apache Arrow 的更快内存列式处理，用于读取 CSV 数据。
			  • 基于 SIMD 执行矢量化读取 CSV 数据。
			  • Spark 升级还包括在 Amazon EMR 上开发的其他优化。
			  • 将 EMRFS 从 2.38 升级到 2.46，为 Amazon S3 访问启用新功能和错误修复。

AWS Glue 版本	支持的运行时环境版本	支持的 Java 版本	功能更改
			• 升级了新 Spark 版本所需的几个依赖项。请参阅 附录 A：显著依赖项升级。 • 为我们本机支持的数据源升级了 JDBC 驱动程序。请参阅 附录 B：JDBC 驱动程序升级。 限制 以下是 AWS Glue 3.0 的限制： • AWS Glue 机器学习转换在 AWS Glue 3.0 中尚不可用。 • 如果某些自定义 Spark 连接器依赖于 Spark 2.4 并且与 Spark 3.1 不兼容，则它们不能与 AWS Glue 3.0 一起使用。 有关迁移到 AWS Glue 版本 3.0 的更多信息，请参阅将 AWS Glue for Spark 作业迁移到 AWS Glue 版本 3.0。

AWS Glue 版本	支持的运行时环境版本	支持的 Java 版本	功能更改
AWS Glue 2.0（已弃用，终止支持）	• Spark 2.4.3 • Python 3.7	不适用	除了 AWS Glue 1.0 版本提供的功能外，AWS Glue 2.0 版本还提供： • 用于在 AWS Glue 中运行 Apache Spark ETL 任务并减少启动时间的升级基础设施。 • 默认日志记录现在是实时的，具有驱动程序和执行程序的单独流以及输出和错误。 • 支持在任务级别指定其他 Python 模块或不同版本。 **Note** 由于底层架构更改，AWS Glue 2.0 版本与 AWS Glue 1.0 版本的某些依赖项和版本不同。在跨主要 AWS Glue 发布版本迁移之前，请验证您的

AWS Glue 版本	支持的运行时环境版本	支持的 Java 版本	功能更改
			AWS Glue 任务。 有关 AWS Glue 2.0 版本功能和限制的更多信息，请参阅运行 Spark ETL 作业，缩短启动时间。

AWS Glue 版本	支持的运行时环境版本	支持的 Java 版本	功能更改
AWS Glue 1.0（已弃用，终止支持）	• Spark 2.4.3 • Python 2.7 • Python 3.6	不适用	您可以在 AWS Glue ETL 任务中维护 Parquet 和 ORC 格式的任务书签（使用 AWS Glue 版本 1.0）。以前，您只能在 AWS Glue ETL 任务中为常见的 Amazon S3 源格式添加书签，例如 JSON、CSV、Apache Avro 和 XML。 为 ETL 输入和输出设置格式选项时，您可以指定使用 Apache Avro 读取器/写入器格式 1.8 来支持 Avro 逻辑类型读取和写入（使用 AWS Glue 版本 1.0）。以前，只支持版本 1.7 Avro 读取器/写入器格式。 DynamoDB 连接类型支持写入器选项（使用 AWS Glue 1.0 版本）。 限制 以下是 AWS Glue 1.0 的限制：

AWS Glue 版本	支持的运行时环境版本	支持的 Java 版本	功能更改
			• AWS Glue 版本 0.9 和 1.0 在亚太地区（雅加达）（ap-southeast-3 ）、中东（阿联酋）（me-central-1 ）或后续其他新区域不可用。
AWS Glue 0.9（已弃用，终止支持）	• Spark 2.2.1 • Python 2.7	不适用	在未指定 AWS Glue 版本的情况下，默认创建的任务为 AWS Glue 0.9。 限制 以下是 AWS Glue 0.9 的限制： • AWS Glue 版本 0.9 和 1.0 在亚太地区（雅加达）（ap-southeast-3 ）、中东（阿联酋）（me-central-1 ）或后续其他新区域不可用。

运行 Spark ETL 作业，缩短启动时间

AWS Glue 2.0 版及更高版本提供在 AWS Glue 中以更短的启动时间运行 Apache Spark ETL（提取、转换和加载）任务的升级基础设施。由于等待时间缩短，数据工程师可以提高工作效率，并增加与 AWS Glue 的互动性。减少任务开始时间的差异可以帮助您达到或超过使数据可用于分析的 SLA。

要将此功能与 AWS Glue ETL 任务结合使用，请在创建任务时为 `Glue version` 选择 **`2.0`** 或更高版本。

主题

支持的新功能

本节介绍 AWS Glue 2.0 版及更高版本支持的新功能。

支持在任务层面指定其他 Python 模块

AWS Glue 2.0 版及更高版本还允许您在任务层面提供其他 Python 模块或不同版本。您可以结合使用 `--additional-python-modules` 选项与一系列逗号分隔的 Python 模块，以添加新模块或更改现有模块的版本。

例如，要更新或添加新的 `scikit-learn` 模块，请使用以下键/值：`"--additional-python-modules", "scikit-learn==0.21.3"`。

此外，在 `--additional-python-modules` 选项中，您可以指定指向 Python Wheel 模块的 Amazon S3 路径。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）来安装其他模块。您可以将 `python-modules-installer-option` 指定的其他选项传递到 pip3，用于安装模块。pip3 的任何不兼容或限制都将适用。

AWS Glue 2.0 版中已提供的 Python 模块

AWS Glue 2.0 版支持以下开箱即用的 Python 模块：

- 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

日志记录行为

AWS Glue 2.0 版及更高版本支持不同的默认日志记录行为。差异包括：

- 实时进行日志记录。
- 驱动程序和执行程序有单独的流。
- 对于每个驱动程序和执行程序，有两个流，即输出流和错误流。

驱动程序和执行程序流

驱动程序流由任务运行 ID 标识。执行程序流由任务 <*run id*>_<*executor task id*> 标识。例如：

- `"logStreamName": "jr_8255308b426fff1b4e09e00e0bd5612b1b4ec848d7884cebe61ed33a31789..._g-f65f617bd31d54bd94482af755b6cdf464542..."`

输出和错误流

输出流具有来自代码的标准输出（stdout）。错误流具有来自代码/库的日志消息。

- 日志流：

- 驱动程序日志流具有 <*jr*>，其中 <*jr*> 是任务运行 ID。
- 执行程序日志流具有 <*jr*>_<*g*>，其中 <*g*> 是执行程序的任务 ID。您可以在驱动程序错误日志中查找执行程序任务 ID。

AWS Glue 2.0 版的默认日志组如下：

- 输出为 `/aws-glue/jobs/logs/output`
- 错误为 `/aws-glue/jobs/logs/error`

提供安全配置时，日志组名称更改为：

- `/aws-glue/jobs/`<*security configuration*>`-role/`<*Role Name*>`/output`
- `/aws-glue/jobs/`<*security configuration*>`-role/`<*Role Name*>`/error`

在控制台上，Logs (日志) 链接指向输出日志组，Error (错误) 链接指向错误日志组。启用连续日志记录时，Logs (日志) 链接指向连续日志组，Output (输出) 链接指向输出日志组。

日志记录规则

> **Note**
>
> 连续日志记录的默认日志组名为 `/aws-glue/jobs/logs-v2`。

在 AWS Glue 2.0 版及更高版本中，连续日志记录的行为与 AWS Glue 1.0 版相同：

- 默认日志组：`/aws-glue/jobs/logs-v2`
- 驱动程序日志流：<*jr*>-driver
- 执行程序日志流：<*jr*>-<*executor ID*>

  日志组名称可通过设置 `--continuous-log-logGroupName` 更改

  日志流名称可通过设置 `--continous-log-logStreamPrefix` 添加前缀

不支持的特征

不支持以下 AWS Glue 功能：

- 开发终端节点
- AWS Glue 2.0 版及更高版本不会在 Apache YARN 上运行，因此 YARN 设置不适用
- AWS Glue 2.0 版及更高版本没有 Hadoop Distributed File System (HDFS)
- AWS Glue 2.0 版及更高版本不使用动态分配，因此 ExecutorAllocationManager 指标不可用
- 对于 AWS Glue 2.0 版或更高版本任务，您可以指定工件数量和工件类型，但不指定 `maxCapacity`。
- AWS Glue 2.0 版及更高版本不支持 `s3n` 开箱即用。我们建议使用 `s3` 或 `s3a`。如果任务出于任何原因需要使用 `s3n`，您可以传递以下附加参数：

```
--conf spark.hadoop.fs.s3n.impl=com.amazon.ws.emr.hadoop.fs.EmrFileSystem
```

将 AWS Glue for Spark 作业迁移到 AWS Glue 版本 3.0

本主题介绍 AWS Glue 版本 0.9、1.0、2.0 和 3.0 之间的变化，允许您将 Spark 应用程序和 ETL 任务迁移到 AWS Glue 3.0。

要将此功能与 AWS Glue ETL 任务结合使用，请在创建任务时为 `Glue version` 选择 **`3.0`**。

主题

支持的新功能

此部分介绍 AWS Glue 版本 3.0 的新功能和优势。

- 它基于 Apache Spark 3.1.1，拥有来自开源 Spark 的优化，通过 AWS Glue 和 EMR 服务（如自适应查询执行、矢量化读取器以及优化的随机排序和分区合并）开发。

- 升级了 JDBC 驱动程序，适用于所有 Glue 原生源，包括 MySQL、Microsoft SQL Server、Oracle、PostgreSQL、MongoDB，并且升级了 Spark 3.1.1 引入的 Spark 库和依赖项。
- 使用升级的 EMRFS 优化了 Amazon S3 访问，并且默认启用了 Amazon S3 优化的输出提交程序。
- 使用分区索引、下推谓词、分区列表和升级的 Hive 元存储客户端优了化数据目录访问。
- 通过单元级筛选条件和数据湖事务，与受监管目录表的 Lake Formation 集成。
- 使用新的 Spark 执行程序内存指标和 Spark 结构化流式传输指标，提高了 Spark 3.1.1 的 Spark 用户界面体验。
- 减少了启动延迟，改善了整体任务完成时间和交互性，类似于 AWS Glue 2.0。
- Spark 任务以 1 秒为增量计费，按 10 倍缩短最短计费持续时间（从最短 10 分钟到最短 1 分钟），类似于 AWS Glue 2.0。

用于迁移到 AWS Glue 3.0 的操作

对于现有任务，请将 `Glue version` 从以前的版本更改为任务配置中的 `Glue 3.0`。

- 在控制台中，为 `Glue version` 选择 `Spark 3.1, Python 3 (Glue Version 3.0) or Spark 3.1, Scala 2 (Glue Version 3.0)`。
- 在 AWS Glue Studio 中，为 `Glue version` 选择 `Glue 3.0 - Supports spark 3.1, Scala 2, Python 3`。
- 在 API 中，为 `UpdateJob` API 中的 `GlueVersion` 参数选择 **`3.0`**。

对于新任务，请在创建任务时选择 `Glue 3.0`。

- 在控制台中，为 `Glue version` 选择 `Spark 3.1, Python 3 (Glue Version 3.0) or Spark 3.1, Scala 2 (Glue Version 3.0)`。
- 在 AWS Glue Studio 中，为 `Glue version` 选择 `Glue 3.0 - Supports spark 3.1, Scala 2, Python 3`。
- 在 API 中，为 `CreateJob` API 中的 `GlueVersion` 参数选择 **`3.0`**。

要查看 AWS Glue 3.0 的 Spark 事件日志，请使用 CloudFormation 或 Docker 为 Glue 3.0 启动升级的 Spark 历史记录服务器。

迁移核对清单

查看此核对清单以进行迁移。

- 您的任务是否依赖于 HDFS？ 如果是，请尝试使用 S3 替换 HDFS。
  - 在任务脚本代码中搜索以 hdfs:// 或者 / 开头且作为 DFS 路径的文件系统路径。
  - 检查您的默认文件系统是否已使用 HDFS 配置。如果已显式配置，则需要删除 fs.defaultFS 配置。
  - 检查您的任务是否包含任何 dfs.* 参数。如果它包含任何参数，则需要验证是否可以禁用参数。
- 您的任务是否依赖于 YARN？ 如果是，请检查您的任务是否包含以下参数，以验证影响。如果它包含任何参数，则需要验证是否可以禁用参数。
  - 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
    ```

- 您的任务是否依赖于 Spark 2.2.1 或 Spark 2.4.3？ 如果是，请检查您的任务是否使用 Spark 3.1.1 中更改的功能以验证影响。
  - https://spark.apache.org/docs/latest/sql-migration-guide.html#upgrading-from-spark-sql-22-to-23

    例如，percentile_approx 函数或具有 SparkSession.builder.getOrCreate() 的 SparkSession（当存在现有 SparkContext 时）。
  - https://spark.apache.org/docs/latest/sql-migration-guide.html#upgrading-from-spark-sql-23-to-24

    例如，array_contains 函数，或者具有 spark.sql.caseSensitive=true 的 CURRENT_DATE、CURRENT_TIMESTAMP 函数。
- 在 Glue 3.0 中，您的任务的额外 jars 是否冲突？

  - 对于 AWS Glue 0.9/1.0：现有 AWS Glue 0.9/1.0 任务中提供的额外 jars 可能会因 Glue 3.0 中的升级或新依赖项而导致类路径冲突。您可以使用 `--user-jars-first` AWS Glue 任务参数或者为依赖项填充阴影，避免 AWS Glue 3.0 中的类路径冲突。
  - 对于 AWS Glue 2.0：您仍然可以使用 `--user-jars-first` AWS Glue 任务参数或者为依赖项填充阴影，避免 AWS Glue 3.0 中的类路径冲突。
- 您的任务是否依赖于 Scala 2.11？
  - AWS Glue 3.0 使用 Scala 2.12，因此如果您的库依赖于 Scala 2.11，您需要使用 Scala 2.12 重新构建您的库。
- 您的任务的外部 Python 库是否依赖于 Python 2.7/3.6？
  - 使用 `--additional-python-modules` 参数，而不是在 Python 库路径中设置 egg/wheel/zip 格式文件。
  - 将依赖库从 Python 2.7/3.6 更新为 Python 3.7，因为 Spark 3.1.1 删除了 Python 2.7 支持。

从 AWS Glue 0.9 迁移到 AWS Glue 3.0

迁移时，请注意以下更改：

- AWS Glue 0.9 使用开源 Spark 2.2.1，AWS Glue 3.0 使用 EMR 优化的 Spark 3.1.1。
  - 仅有几项 Spark 更改可能需要修订脚本，确保不会引用已删除的功能。
  - 例如，Spark 3.1.1 不启用 Scala 无类型的 UDF，但 Spark 2.2 确实允许它们。
- AWS Glue 3.0 中的所有任务均会执行，显著缩短启动时间。Spark 任务以 1 秒为增量计费，按 10 倍缩短最短计费持续时间，因为启动延迟从最长 10 分钟到最长 1 分钟。
- 日志记录行为自 AWS Glue 2.0 起已更改。
- 多个依赖项更新，在 附录 A：显著依赖项升级 中突出显示。
- Scala 也从 2.11 更新到 2.12，Scala 2.12 不向后兼容 Scala 2.11。
- Python 3.7 还是 Python 脚本使用的默认版本，因为 AWS Glue 0.9 仅使用了 Python 2。
  - Spark 3.1.1 不支持 Python 2.7。
  - 额外 Python 模块安装的新机制可用。
- AWS Glue 3.0 不会在 Apache YARN 上运行，因此 YARN 设置不适用。
- AWS Glue 3.0 没有 Hadoop Distributed File System (HDFS)。
- 现有 AWS Glue 0.9 任务中提供的任何额外 jars 都可能会带来导致冲突的依赖项，因为 3.0 中的几个依赖项从 0.9 升级。您可以使用 `--user-jars-first` AWS Glue 任务参数，避免 AWS Glue 3.0 中的类路径冲突。

- AWS Glue 3.0 尚不支持动态分配，因此 ExecutorAllocationManager 指标不可用。
- 在 AWS Glue 版本 3.0 任务中，您可以指定工件数量和工件类型，但不能指定 `maxCapacity`。
- AWS Glue 3.0 尚不支持机器学习转换。
- AWS Glue 3.0 尚不支持开发终端节点。

请参阅 Spark 迁移文档：

- 请参阅从 Spark SQL 2.2 升级到 2.3
- 请参阅从 Spark SQL 2.3 升级到 2.4
- 请参阅从 Spark SQL 2.4 升级到 3.0
- 请参阅从 Spark SQL 3.0 升级到 3.1
- 请参阅自 Spark 3.0 以来预期的日期时间行为更改。

从 AWS Glue 1.0 迁移到 AWS Glue 3.0

迁移时，请注意以下更改：

- AWS Glue 1.0 使用开源 Spark 2.4，AWS Glue 3.0 使用 EMR 优化的 Spark 3.1.1。
  - 仅有几项 Spark 更改可能需要修订脚本，确保不会引用已删除的功能。
  - 例如，Spark 3.1.1 不启用 Scala 无类型的 UDF，但 Spark 2.4 确实允许它们。
- AWS Glue 3.0 中的所有任务均会执行，显著缩短启动时间。Spark 任务以 1 秒为增量计费，按 10 倍缩短最短计费持续时间，因为启动延迟从最长 10 分钟到最长 1 分钟。
- 日志记录行为自 AWS Glue 2.0 起已更改。
- 多个依赖项更新，突出显示
- Scala 也从 2.11 更新到 2.12，Scala 2.12 不向后兼容 Scala 2.11。
- Python 3.7 还是 Python 脚本使用的默认版本，因为 AWS Glue 0.9 仅使用了 Python 2。
  - Spark 3.1.1 不支持 Python 2.7。
  - 额外 Python 模块安装的新机制可用。
- AWS Glue 3.0 不会在 Apache YARN 上运行，因此 YARN 设置不适用。
- AWS Glue 3.0 没有 Hadoop Distributed File System (HDFS)。
- 现有 AWS Glue 1.0 任务中提供的任何额外 jars 都可能会带来导致冲突的依赖项，因为 3.0 中的几个依赖项从 1.0 升级。您可以使用 `--user-jars-first` AWS Glue 任务参数，避免 AWS Glue 3.0 中的类路径冲突。

- AWS Glue 3.0 尚不支持动态分配，因此 ExecutorAllocationManager 指标不可用。
- 在 AWS Glue 版本 3.0 任务中，您可以指定工件数量和工件类型，但不能指定 `maxCapacity`。
- AWS Glue 3.0 尚不支持机器学习转换。
- AWS Glue 3.0 尚不支持开发终端节点。

请参阅 Spark 迁移文档：

- 请参阅从 Spark SQL 2.4 升级到 3.0
- 请参阅自 Spark 3.0 以来预期的日期时间行为更改。

从 AWS Glue 2.0 迁移到 AWS Glue 3.0

迁移时，请注意以下更改：

- 所有现有任务参数和 AWS Glue 2.0 中存在的主要功能将存在于 AWS Glue 3.0。
  - 预设情况下，AWS Glue 3.0 中将启用经 EMRFS S3 优化的提交程序，用于将 Parquet 数据写入 Amazon S3。但是，您仍然可以通过将 `--enable-s3-parquet-optimized-committer` 设置为 `false` 来禁用。
- AWS Glue 2.0 使用开源 Spark 2.4，AWS Glue 3.0 使用 EMR 优化的 Spark 3.1.1。
  - 仅有几项 Spark 更改可能需要修订脚本，确保不会引用已删除的功能。
  - 例如，Spark 3.1.1 不启用 Scala 无类型的 UDF，但 Spark 2.4 确实允许它们。
- AWS Glue 3.0 还更新了 EMRFS、JDBC 驱动程序，并且对 AWS Glue 提供的 Spark 进行了额外优化。
- AWS Glue 3.0 中的所有任务均会执行，显著缩短启动时间。Spark 任务以 1 秒为增量计费，按 10 倍缩短最短计费持续时间，因为启动延迟从最长 10 分钟到最长 1 分钟。
- Spark 3.1.1 不支持 Python 2.7。
- 多个依赖项更新，在 附录 A：显著依赖项升级 中突出显示。
- Scala 也从 2.11 更新到 2.12，Scala 2.12 不向后兼容 Scala 2.11。
- 现有 AWS Glue 2.0 任务中提供的任何额外 jars 都可能会带来导致冲突的依赖项，因为 3.0 中的几个依赖项从 2.0 升级。您可以使用 `--user-jars-first` AWS Glue 任务参数，避免 AWS Glue 3.0 中的类路径冲突。
- AWS Glue 3.0 在驱动程序/执行器配置方面的 Spark 任务并行性与 AWS Glue 2.0 有所不同，提高了性能并更好地利用了可用资源。`spark.driver.cores` 和 `spark.executor.cores` 都配置为 AWS Glue 3.0 上的内核数（标准和 `G.1X` 工件上为 4 个，`G.2X` 工件上为 8 个）。这些配置不

会更改 AWS Glue 任务的工件类型或硬件。您可以使用这些配置来计算分区或拆分的数量，以匹配 Spark 应用程序中的 Spark 任务并行性。

通常，与 AWS Glue 2.0 相比，作业的性能要么相似，要么有所提高。如果作业运行速度较慢，则可以通过传递以下作业参数来提高作业并行度：

- 键：`--executor-cores` 值：`<############>`
- 该值不能超过工作线程类型上 vCPU 数量的 2 倍，即 `G.1X` 上为 8、`G.2X` 上为 16、`G.4X` 上为 32，`G.8X` 上为 64。更新此配置时应谨慎行事，因为它可能会影响作业性能，因为并行度增加会导致内存和磁盘面临压力，并可能限制源系统和目标系统。

- AWS Glue 3.0 使用的是 Spark 3.1，改变了从/到 parquet 文件加载/保存时间戳的行为。有关更多详细信息，请参阅 从 Spark SQL 3.0 升级到 3.1。

  在读取/写入包含时间戳列的 parquet 数据时，建议设置以下参数。设置这些参数可以解决 Spark 2 到 Spark 3 升级期间发生的 AWS Glue 动态帧和 Spark 数据帧的日历不兼容问题。使用 CORRECTED（更正）选项将按原样读取日期时间值；使用 LEGACY（旧式）选项将根据读取期间的日历差异重新设置日期时间值的基础。

  ```
  - 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]
  ```

请参阅 Spark 迁移文档：

- 请参阅从 Spark SQL 2.4 升级到 3.0
- 请参阅自 Spark 3.0 以来预期的日期时间行为更改。

附录 A：显著依赖项升级

以下是依赖项升级：

依赖关系	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

依赖关系	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
Scala	2.1.1	2.1.1	2.1.1	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
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
箭头	不适用	0.10.0	0.10.0	2.0.0
AWS Glue Catalog 客户端	不适用	不适用	1.10.0	3.0.0

附录 B：JDBC 驱动程序升级

以下是 JDBC 驱动程序升级：

驱动程序	过去 AWS Glue 版本中的 JDBC 驱动程序版本	AWS Glue 3.0 中的 JDBC 驱动程序版本
MySQL	5.1	8.0.23
Microsoft SQL Server	6.1.0	7.0.0
Oracle 数据库	11.2	21.1
PostgreSQL	42.1.0	42.2.18
MongoDB	2.0.0	4.0.0

将 AWS Glue for Spark 作业迁移到 AWS Glue 版本 4.0

本主题介绍 AWS Glue 版本 0.9、1.0、2.0 和 3.0 之间的变化，允许您将 Spark 应用程序和 ETL 任务迁移到 AWS Glue 4.0。还介绍了 AWS Glue 4.0 中的功能以及使用它的优点。

要将此功能与 AWS Glue ETL 任务结合使用，请在创建任务时为 `Glue version` 选择 **`4.0`**。

主题

支持的新功能

此部分介绍 AWS Glue 版本 4.0 的新功能和优势。

- 它基于 Apache Spark 3.3.0，拥有对 AWS Glue 和 Amazon EMR 的优化，如自适应查询运行、矢量化读取器以及优化的随机排序和分区合并。
- 升级了 JDBC 驱动程序，适用于所有 AWS Glue 原生源，包括 MySQL、Microsoft SQL Server、Oracle、PostgreSQL、MongoDB，并且升级了 Spark 3.3.0 引入的 Spark 库和依赖项。
- 更新了新的 Amazon Redshift 连接器和 JDBC 驱动程序。
- 使用升级的 EMR 文件系统（EMRFS）优化了 Amazon S3 访问，并且默认启用了 Amazon S3 优化的输出提交程序。
- 使用分区索引、下推谓词、分区列表和升级的 Hive 元存储客户端优了化数据目录访问。
- 通过单元级筛选条件和数据湖事务，与受监管目录表的 Lake Formation 集成。
- 减少了启动延迟，改善了整体任务完成时间和交互性。
- Spark 任务以 1 秒为增量计费，按 10 倍缩短最短计费持续时间（从最短 10 分钟到最短 1 分钟）。
- 本机支持 Apache Hudi、Delta Lake 和 Apache Iceberg 的开放数据湖框架。
- 本机支持基于 Amazon S3 的 Cloud Shuffle 存储插件（Apache Spark 插件），从而使用 Amazon S3 进行洗牌和弹性存储容量。

从 Spark 3.1.1 到 Spark 3.3.0 的主要增强功能

请注意以下增强功能：

- 行级运行时筛选（SPARK-32268）。
- ANSI 增强功能（SPARK-38860）。
- 错误消息改进（SPARK-38781）。
- 支持 Parquet 矢量化阅读器（SPARK-34863）的复杂类型。
- 支持 Spark SQL（SPARK-37273）的隐藏文件元数据。
- 为 Python/Pandas UDF 提供分析器（SPARK-37443）。
- 推出 Trigger.AvailableNow 来分批运行像 Trigger.Once 这样的流式查询（SPARK-36533）。
- 更全面的 Datasource V2 下推功能（SPARK-38788）。
- 从 log4j 1 迁移到 log4j 2（SPARK-37814）。

其他显著的变化

注意以下更改：

- 重大更改
  - 删除文档和 Python/Docs 中对 Python 3.6 支持的引用（SPARK-36977）。
  - 通过将内置 pickle 替换为 cloudpickle 来移除命名的元组黑客攻击（SPARK-32079）。
  - 将 pandas 的最低版本提高到 1.0.5（SPARK-37465）。

用于迁移到 AWS Glue 4.0 的操作

对于现有任务，请将 `Glue version` 从以前的版本更改为任务配置中的 `Glue 4.0`。

- 在 AWS Glue Studio 中，为 `Glue version` 选择 `Glue 4.0 - Supports Spark 3.3, Scala 2, Python 3`。
- 在 API 中，为 `UpdateJob` API 操作中的 `GlueVersion` 参数选择 **`4.0`**。

对于新任务，请在创建任务时选择 `Glue 4.0`。

- 在控制台中，为 `Glue version` 选择 `Spark 3.3, Python 3 (Glue Version 4.0) or Spark 3.3, Scala 2 (Glue Version 3.0)`。

- 在 AWS Glue Studio 中，为 `Glue version` 选择 `Glue 4.0 - Supports Spark 3.3, Scala 2, Python 3`。
- 在 API 中，为 `CreateJob` API 操作中的 `GlueVersion` 参数选择 **`4.0`**。

要查看从 AWS Glue 2.0 或更早版本到 AWS Glue 4.0 的 Spark 事件日志，请使用 AWS CloudFormation 或 Docker 为 AWS Glue 4.0 启动已升级的 Spark 历史服务器。

迁移核对清单

查看此核对清单以进行迁移：

> **Note**
>
> 有关与 AWS Glue 3.0 相关的核对清单项目，请参阅迁移核对清单。

- 您的任务的外部 Python 库是否依赖于 Python 2.7/3.6？
  - 将依赖库从 Python 2.7/3.6 更新为 Python 3.10，因为 Spark 3.3.0 完全删除了 Python 2.7 和 3.6 支持。

从 AWS Glue 3.0 迁移到 AWS Glue 4.0

迁移时，请注意以下更改：

- 所有现有任务参数和 AWS Glue 3.0 中存在的主要功能将存在于 AWS Glue 4.0。
- AWS Glue 3.0 使用 Amazon EMR 优化的 Spark 3.1.1，AWS Glue 4.0 使用 Amazon EMR 优化的 Spark 3.3.0。

  仅有几项 Spark 更改可能需要修订脚本，确保不会引用已删除的功能。
- AWS Glue 4.0 还包括对 EMRFS 和 Hadoop 的更新。有关特定版本，请参阅附录 A：显著依赖项升级。
- ETL 任务中提供的 AWS 开发工具包现已从 1.11 升级到 1.12。
- 所有 Python 作业都将使用 Python 版本 3.10。以前，AWS Glue 3.0 中使用 Python 3.7。

  因此，AWS Glue 一些即用的 pymodule 得到了升级。
- Log4j 已升级到 Log4j2。
  - 有关 Log4j2 迁移路径的信息，请参阅 Log4j 文档。

  - 必须改为将任何自定义 log4j.properties 文件重命名为 log4j2.properties 文件，并使用相应的 log4j2 属性。
- 有关迁移某些连接器的信息，请参阅AWS Glue 4.0 版的连接器和 JDBC 驱动程序迁移。
- AWS 加密 SDK 从 1.x 升级到 2.x。使用 AWS Glue 安全配置的 AWS Glue 作业和依赖于运行时提供的 AWS 加密 SDK 依赖项的作业会受到影响。请参阅 AWS Glue 作业迁移说明。

  您可以安全地将 AWS Glue 2.0/3.0 作业升级到 AWS Glue 4.0 作业，因为 AWS Glue 2.0/3.0 已经包含 AWS 加密 SDK 桥接版本。

请参阅 Spark 迁移文档：

- 从 Spark SQL 3.1 升级到 3.2
- 从 Spark SQL 3.2 升级到 3.3

从 AWS Glue 2.0 迁移到 AWS Glue 4.0

迁移时，请注意以下更改：

> **Note**
>
> 有关与 AWS Glue 3.0 相关的迁移步骤，请参阅从 AWS Glue 3.0 迁移到 AWS Glue 4.0。

- 所有现有任务参数和 AWS Glue 2.0 中存在的主要功能将存在于 AWS Glue 4.0。
- 预设情况下，自 AWS Glue 3.0，将启用经 EMRFS S3 优化的提交程序，用于将 Parquet 数据写入 Amazon S3。但是，您仍然可以通过将 `--enable-s3-parquet-optimized-committer` 设置为 `false` 来禁用。
- AWS Glue 2.0 使用开源 Spark 2.4，AWS Glue 4.0 使用 Amazon EMR 优化的 Spark 3.3.0。
  - 仅有几项 Spark 更改可能需要修订脚本，确保不会引用已删除的功能。
  - 例如，Spark 3.3.0 不启用 Scala 无类型的 UDF，但 Spark 2.4 允许它们。
- ETL 任务中提供的 AWS 开发工具包现已从 1.11 升级到 1.12。
- AWS Glue 4.0 还更新了 EMRFS、JDBC 驱动程序，并且对 AWS Glue 提供的 Spark 进行了额外优化。
- Scala 也从 2.11 更新到 2.12，Scala 2.12 不向后兼容 Scala 2.11。
- Python 3.10 还是 Python 脚本使用的默认版本，因为 AWS Glue 2.0 仅使用了 Python 3.7和 2.7。

  - Spark 3.3.0 不支持 Python 2.7。任何在作业配置中请求 Python 2 的任务都会因为 IllegalArgumentException 而失败。
  - 自 AWS Glue 2.0 以来，提供了一种安装额外 Python 模块的新机制。
- 多个依赖项更新，在 附录 A：显著依赖项升级 中突出显示。
- 现有 AWS Glue 2.0 任务中提供的任何额外 JAR 文件都可能会带来导致冲突的依赖项，因为 4.0 中的几个依赖项从 2.0 升级。您可以使用 `--user-jars-first` AWS Glue 作业参数，避免 AWS Glue 4.0 中的类路径冲突。
- AWS Glue 4.0 使用 Spark 3.3。从 Spark 3.1 开始，从/到 parquet 文件加载/保存时间戳的行为发生了变化。有关更多详细信息，请参阅 从 Spark SQL 3.0 升级到 3.1。

  在读取/写入包含时间戳列的 parquet 数据时，建议设置以下参数。设置这些参数可以解决 Spark 2 到 Spark 3 升级期间发生的 AWS Glue 动态帧和 Spark 数据帧的日历不兼容问题。使用 CORRECTED（更正）选项将按原样读取日期时间值；使用 LEGACY（旧式）选项将根据读取期间的日历差异重新设置日期时间值的基础。

  ```
  - 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]
  ```
- 有关迁移某些连接器的信息，请参阅AWS Glue 4.0 版的连接器和 JDBC 驱动程序迁移。
- AWS 加密 SDK 从 1.x 升级到 2.x。使用 AWS Glue 安全配置的 AWS Glue 作业和依赖于运行时提供的 AWS 加密 SDK 依赖项的作业会受到影响。请参阅以下有关 AWS Glue 工作迁移的说明：
  - 您可以安全地将 AWS Glue 2.0 作业升级到 AWS Glue 4.0 作业，因为 AWS Glue 2.0 已经包含 AWS 加密 SDK 桥接版本。

请参阅 Spark 迁移文档：

- 从 Spark SQL 2.4 升级到 3.0
- 从 Spark SQL 3.1 升级到 3.2
- 从 Spark SQL 3.2 升级到 3.3
- 自 Spark 3.0 以来预期的日期时间行为更改。

从 AWS Glue 1.0 迁移到 AWS Glue 4.0

迁移时，请注意以下更改：

- AWS Glue 1.0 使用开源 Spark 2.4，AWS Glue 4.0 使用 Amazon EMR 优化的 Spark 3.3.0。
  - 仅有几项 Spark 更改可能需要修订脚本，确保不会引用已删除的功能。
  - 例如，Spark 3.3.0 不启用 Scala 无类型的 UDF，但 Spark 2.4 允许它们。
- AWS Glue 4.0 中的所有任务均会执行，显著缩短启动时间。Spark 任务以 1 秒为增量计费，按 10 倍缩短最短计费持续时间，因为启动延迟从最长 10 分钟到最长 1 分钟。
- 在 AWS Glue 4.0 中，记录行为发生了显著变化，Spark 3.3.0 的最低要求为 Log4j2。
- 多个依赖项更新，在附录中突出显示。
- Scala 也从 2.11 更新到 2.12，Scala 2.12 不向后兼容 Scala 2.11。
- Python 3.10 还是 Python 脚本使用的默认版本，因为 AWS Glue 0.9 仅使用了 Python 2。

  Spark 3.3.0 不支持 Python 2.7。任何在作业配置中请求 Python 2 的任务都会因为 IllegalArgumentException 而失败。
- 从 AWS Glue 2.0 开始，提供通过 pip 安装额外 Python 模块的新机制。有关更多信息，请参阅使用 pip 在 AWS Glue 2.0+ 中安装其他 Python 模块。
- AWS Glue 4.0 不会在 Apache YARN 上运行，因此 YARN 设置不适用。
- AWS Glue 4.0 没有 Hadoop Distributed File System (HDFS)。
- 现有 AWS Glue 1.0 任务中提供的任何额外 JAR 文件都可能会带来导致冲突的依赖项，因为 4.0 中的几个依赖项从 1.0 升级。为了避免这个问题，我们默认使用 `--user-jars-first`AWS Glue 作业参数启用 AWS Glue 4.0。
- AWS Glue 4.0 现在支持 Auto Scaling。因此，启用 Auto Scaling 后，ExecutorAllocationManager 指标将可用。
- 在 AWS Glue 版本 4.0 任务中，您可以指定工件数量和工件类型，但不能指定 `maxCapacity`。
- AWS Glue 4.0 尚不支持机器学习转换。
- 有关迁移某些连接器的信息，请参阅 AWS Glue 4.0 版的连接器和 JDBC 驱动程序迁移。
- AWS 加密 SDK 从 1.x 升级到 2.x。使用 AWS Glue 安全配置的 AWS Glue 作业和依赖于运行时提供的 AWS 加密 SDK 依赖项的作业会受到影响。有关 AWS Glue 工作迁移的信息，请参阅这些说明。
  - 您无法将 AWS Glue 0.9/1.0 作业直接迁移到 AWS Glue 4.0 作业。这是因为当直接升级到 2.x 或更高版本并立即启用所有新功能时，AWS 加密开发工具包将无法解密在早期版本的 AWS 加密开发工具包下加密的加密文字。
  - 为了安全升级，我们首先建议您迁移到包含 AWS 加密开发工具包桥接版本的 AWS Glue 2.0/3.0 作业。运行一次作业即可使用 AWS 加密开发工具包桥接版本。
  - 完成后，您可以安全地将 AWS Glue 2.0/3.0 作业迁移到 AWS Glue 4.0。

请参阅 Spark 迁移文档：

- 从 Spark SQL 2.4 升级到 3.0
- 从 Spark SQL 3.0 升级到 3.1
- 从 Spark SQL 3.1 升级到 3.2
- 从 Spark SQL 3.2 升级到 3.3
- 自 Spark 3.0 以来预期的日期时间行为更改。

从 AWS Glue 0.9 迁移到 AWS Glue 4.0

迁移时，请注意以下更改：

- AWS Glue 0.9 使用开源 Spark 2.2.1，AWS Glue 4.0 使用 Amazon EMR 优化的 Spark 3.3.0。
  - 仅有几项 Spark 更改可能需要修订脚本，确保不会引用已删除的功能。
  - 例如，Spark 3.3.0 不启用 Scala 无类型的 UDF，但 Spark 2.2 允许它们。
- AWS Glue 4.0 中的所有任务均会执行，显著缩短启动时间。Spark 任务以 1 秒为增量计费，按 10 倍缩短最短计费持续时间，因为启动延迟从最长 10 分钟到最长 1 分钟。
- 自 AWS Glue 4.0 以来，日志记录行为发生了重大变化，Spark 3.3.0 的最低要求为 Log4j2，如此处所述 (https://spark.apache.org/docs/latest/core-migration-guide.html#upgrading-from-core-32-to-33)。
- 多个依赖项更新，在附录中突出显示。
- Scala 也从 2.11 更新到 2.12，Scala 2.12 不向后兼容 Scala 2.11。
- Python 3.10 还是 Python 脚本使用的默认版本，因为 AWS Glue 0.9 仅使用了 Python 2。
  - Spark 3.3.0 不支持 Python 2.7。任何在作业配置中请求 Python 2 的任务都会因为 IllegalArgumentException 而失败。
  - 提供一种通过 pip 安装额外 Python 模块的新机制。
- AWS Glue 4.0 不会在 Apache YARN 上运行，因此 YARN 设置不适用。
- AWS Glue 4.0 没有 Hadoop Distributed File System (HDFS)。
- 现有 AWS Glue 0.9 任务中提供的任何额外 JAR 文件都可能会带来导致冲突的依赖项，因为 3.0 中的几个依赖项从 0.9 升级。您可以使用 `--user-jars-first` AWS Glue 任务参数，避免 AWS Glue 3.0 中的类路径冲突。
- AWS Glue 4.0 现在支持 Auto Scaling。因此，启用 Auto Scaling 后，ExecutorAllocationManager 指标将可用。

- 在 AWS Glue 版本 4.0 任务中，您可以指定工件数量和工件类型，但不能指定 `maxCapacity`。
- AWS Glue 4.0 尚不支持机器学习转换。
- 有关迁移某些连接器的信息，请参阅 AWS Glue 4.0 版的连接器和 JDBC 驱动程序迁移。
- AWS 加密 SDK 从 1.x 升级到 2.x。使用 AWS Glue 安全配置的 AWS Glue 作业和依赖于运行时提供的 AWS 加密 SDK 依赖项的作业会受到影响。有关 AWS Glue 工作迁移的信息，请参阅这些说明。
  - 您无法将 AWS Glue 0.9/1.0 作业直接迁移到 AWS Glue 4.0 作业。这是因为当直接升级到 2.x 或更高版本并立即启用所有新功能时，AWS 加密开发工具包将无法解密在早期版本的 AWS 加密开发工具包下加密的加密文字。
  - 为了安全升级，我们首先建议您迁移到包含 AWS 加密开发工具包桥接版本的 AWS Glue 2.0/3.0 作业。运行一次作业即可使用 AWS 加密开发工具包桥接版本。
  - 完成后，您可以安全地将 AWS Glue 2.0/3.0 作业迁移到 AWS Glue 4.0。

请参阅 Spark 迁移文档：

- 从 Spark SQL 2.2 升级到 2.3
- 从 Spark SQL 2.3 升级到 2.4
- 从 Spark SQL 2.4 升级到 3.0
- 从 Spark SQL 3.0 升级到 3.1
- 从 Spark SQL 3.1 升级到 3.2
- 从 Spark SQL 3.2 升级到 3.3
- 自 Spark 3.0 以来预期的日期时间行为更改。

AWS Glue 4.0 版的连接器和 JDBC 驱动程序迁移

有关已升级的 JDBC 和数据湖连接器的版本，请参阅：

- 附录 B：JDBC 驱动程序升级
- 附录 C：连接器升级

Hudi

- Spark SQL 支持方面的改进：

  - 通过 `Call Procedure` 命令，增加了对升级、降级、引导、清理和修复的支持。可以在 Spark SQL 中使用 `Create/Drop/Show/Refresh Index` 语法。
  - 与 Spark SQL 相比，通过 Spark DataSource 使用之间的性能差距已经缩小。过去，数据源写入速度比 SQL 快。
  - 所有内置密钥生成器都实现了更高性能的 Spark 特定 API 操作。
  - 将批量 `insert` 操作中的 UDF 转换替换为 RDD 转换，以降低使用 SerDe 的成本。
  - 带有 Hudi 的 Spark SQL 要求 `primaryKey` 在 SQL 语句中由 `tblproperites` 或选项指定。对于更新和删除操作，`preCombineField` 也是必需的。
- 从 0.10.0 版本开始创建的任何没有 `primaryKey` 的 Hudi 表都需要使用自版本 0.10.0 起的 `primaryKey` 字段重新创建。

PostgreSQL

- 多个漏洞（CVE）已得到解决。
- 本机支持 Java 8。
- 如果任务是使用数组的数组，则除字节数组外，可以将此场景视为多维数组。

MongoDB

- 当前的 MongoDB 连接器支持 Spark 3.1 或更高版本以及 MongoDB 版本 4.0 或更高版本。
- 由于连接器升级，一些属性名称发生了变化。例如，URI 属性名更改为 `connection.uri`。有关当前选项的更多信息，请参阅 MongoDB Spark 连接器博客。
- 使用 Amazon DocumentDB 托管的 MongoDB 4.0 有一些功能差异。有关更多信息，请参阅以下主题：
  - 功能差异：Amazon DocumentDB 和 MongoDB
  - 支持的 MongoDB API、操作和数据类型。
- “partitioner”（分区程序）选项仅限于 `ShardedPartitioner`、`PaginateIntoPartitionsPartitioner`、和 `SinglePartitionPartitioner`。它不能为 Amazon DocumentDB 使用默认 `SamplePartitioner` 和 `PaginateBySizePartitioner`，因为阶段运算符不支持 MongoDB API。有关更多信息，请参阅支持的 MongoDB API、操作和数据类型。

Delta Lake

- Delta Lake 现在支持在 SQL 中进行时空旅行，以便轻松查询较旧的数据。在此更新中，时空旅行现在既可以在 Spark SQL 中使用，也可以通过 DataFrame API 使用。已在 SQL 中添加了对当前版本的 TIMESTAMP 的支持。
- Spark 3.3 推出了用于运行流式查询的 Trigger.AvailableNow，等同于 `Trigger.Once` 批量查询。在使用 Delta 表作为流媒体源时，此支持也可用。
- 支持 SHOW COLUMNS 返回表中的列表。
- 在 Scala 和 Python DeltaTable API 中支持 DESCRIBE DETAIL。它使用 DeltaTable API 或 Spark SQL 检索有关 Delta 表的详细信息。
- 支持从 SQL 删除、合并和更新命令返回操作指标。以前这些 SQL 命令返回一个空的 DataFrame，现在它们返回一个 DataFrame，其中包含有关所执行操作的有用指标。
- 优化性能改进：
  - 将配置选项 `spark.databricks.delta.optimize.repartition.enabled=true` 设置为在“优化”命令使用 `repartition(1)` 而不是 `coalesce(1)`，以提高压缩许多小文件时的性能。
  - 通过使用基于队列的方法对压缩作业进行并行化来提高性能。
- 其他显著的变化：
  - 支持在 VACUUM 和 OPTIMIZE SQL 命令中使用变量。
  - 对带有目录表的 CONVERT TO DELTA 的改进包括：
    - 如果未提供分区架构，则自动填充目录中的分区架构。
    - 使用目录中的分区信息来查找要提交的数据文件，而不是进行完整的目录扫描。不是提交表目录中的所有数据文件，而是只提交活动分区目录下的数据文件。
  - 当未使用 DROP COLUMN 和 RENAME COLUMN 时，支持对启用列映射的表进行更改数据馈送 (CDF) 批量读取。有关更多信息，请参阅 Delta Lake 文档。
  - 通过在第一阶段启用架构修剪来提高更新命令的性能。

Apache Iceberg

- 为扫描计划和 Spark 查询添加了多项性能改进。
- 添加了一个通用 REST 目录客户端，该客户端使用基于更改的提交来解决服务端的提交冲突。
- 支持 SQL 时空旅行查询的 `AS OF` 语法。
- 为 MERGE 和 UPDATE 查询添加了读取时合并支持。
- 添加了对使用 Z 顺序重写分区的支持。

- 为 Puffin 添加了规范和实施，这是一种用于大型统计数据和索引 blob 的格式，例如 Theta sketch 或布隆过滤器。
- 添加了用于增量使用数据的新接口（追加和更改日志扫描）。
- 增加了对 FileIO 接口的批量操作和远程读取的支持。
- 增加了更多元数据表，以显示元数据树中的删除文件。
- 删除表的行为发生了变化。在 Iceberg 0.13.1 中，运行 `DROP TABLE` 会将表从目录中移除，同时删除表内容。在 Iceberg 1.0.0 中，`DROP TABLE` 仅将表从目录中删除。要删除表格内容，请使用 `DROP TABLE PURGE`。
- 在 Iceberg 1.0.0 中，Park 矢量化读取默认启用。如果要禁用矢量化读取，请设置 `read.parquet.vectorization.enabled` 为 `false`。

Oracle

更改很小。

MySQL

更改很小。

Amazon Redshift

AWS Glue 4.0 采用了新的 Amazon Redshift 连接器和新的 JDBC 驱动程序。有关增强功能以及如何从先前 AWS Glue 版本迁移的信息，请参阅the section called “Redshift 连接”。

附录 A：显著依赖项升级

以下是依赖项升级：

依赖关系	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	2.1.1	2.1.1
Jackson	2.13.3	2.10.x	2.7.x	2.7.x

依赖关系	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
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 数据目录客户端	3.7.0	3.0.0	1.10.0	不适用
Python	3.10	3.7	2.7 和 3.6	2.7 和 3.6
Boto	1.26	1.18	1.12	不适用

附录 B：JDBC 驱动程序升级

以下是 JDBC 驱动程序升级：

驱动程序	过去 AWS Glue 版本中的 JDBC 驱动程序版本	AWS Glue 3.0 中的 JDBC 驱动程序版本	AWS Glue 4.0 中的 JDBC 驱动程序版本
MySQL	5.1	8.0.23	8.0.23
Microsoft SQL Server	6.1.0	7.0.0	9.4.0
Oracle 数据库	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

附录 C：连接器升级

以下是连接器升级：

驱动程序	AWS Glue 3.0 中的连接器版本	AWS Glue 4.0 中的连接器版本
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

将 AWS Glue for Ray 从预览版迁移到 Ray2.4 运行时环境

> **Warning**
>
> 当您将 AWS Glue for Ray（预览版）作业保存到 AWS Glue Studio 中时，它会自动升级到 `Ray2.4` 运行时。如果您的脚本遇到兼容性问题，请联系支持人员。

您应该在将 AWS Glue for Ray（预览版）迁移到 AWS Glue for Ray 期间创建 AWS Glue 作业。这将涉及对您的作业配置进行一些并发更改。

- 在 `Runtime` 字段中，提供 `Ray2.4` 运行时值。这会将底层 Ray 版本从 2.0.0 升级到 2.4.0。
- 不再提供预览版中默认包含的某些 Python 库。如果您的工作利用了适用于 Pandas（awswrangler）、dask、modin 或 pymars 的 AWS 开发工具包，则需要将它们作为其他库包含在内。有关包含其他 Python 库的更多信息，请参阅 the section called “用于 Ray 作业的其他 Python 模块”。
- 如果您使用的是 `--additional-python-modules` 参数，则用于支持此工作流程的参数已分解为 `--pip-install` 和 `--s3-py-modules`。有关这些参数的更多信息，请参阅 the section called “用于 Ray 作业的其他 Python 模块”。
- 如果您使用的是 `--auto-scaling-ray-min-workers` 参数，则该参数已被重命名 `--min-workers`。

AWS Glue 版本支持策略

AWS Glue 是一项无服务器数据集成服务，可轻松发现、准备和组合数据，以用于分析、机器学习和应用程序开发。AWS Glue 任务包含在 AWS Glue 中执行数据集成工作的业务逻辑。AWS Glue 中有三种类型的作业：Spark（批次和流式处理）、Ray 和 Python shell。定义作业时，需要指定 AWS Glue 版本，其在底层 Spark、Ray 或 Python 运行时环境中配置版本。例如：AWS Glue 版本 2.0 Spark 作业支持 Spark 2.4.3 和 Python 3.7。

支持策略

AWS Glue 偶尔会停止对旧 AWS Glue 版本的支持。但如果您在已弃用的版本上运行任务，将不再能够获得技术支持。AWS Glue 将不再对已弃用的版本应用安全补丁或其他更新。如果您在已弃用的版本上运行任务，AWS Glue 也不会遵守 SLA 的要求。

当终止支持 2.0 或更高 AWS Glue 版本时，您将无法创建作业，只能编辑或运行作业。

下列 AWS Glue 版本已经或计划终止支持。终止支持从指定日期午夜（太平洋时区）生效。

类型	Glue 版本	终止支持
Spark	Spark 2.2、Scala 2（Glue 版本 0.9）	6/1/2022
Spark	Spark 2.2、Python 2（Glue 版本 0.9）	6/1/2022
Spark	Spark 2.4、Python 2（Glue 版本 1.0）	6/1/2022
Spark	Spark 2.4、Python 3（Glue 版本 1.0）	9/30/2022
Spark	Spark 2.4、Scala 2（Glue 版本 1.0）	9/30/2022
Spark	Glue 版本 2.0	1/31/2024
类型	Python 版本	终止支持
Python shell	Python 2（Glue 版本 1.0）	6/1/2022

类型	Notebook 版本	终止支持
开发终端节点	Zeppelin notebook	9/30/2022

AWS 强烈建议您将任务迁移到受支持的版本。

有关将 Spark 任务迁移到最新 AWS Glue 版本的信息，请参阅将 AWS Glue 任务迁移到 AWS Glue 4.0 版本。

要将 Python shell 任务迁移到最新 AWS Glue 版本，请执行以下操作：

- 在控制台中，选择 `Python 3 (Glue Version 4.0)`。
- 在 CreateJob/UpdateJob API 中，将 `GlueVersion` 参数设置为 `2.0`，并将 `PythonVersion` 参数下的 `3` 设置为 `Command`。`GlueVersion` 配置不会影响 Python Shell 作业的行为，因此递增 `GlueVersion` 没有好处。
- 您需要使任务脚本与 Python 3 兼容。

> **Note**
>
> 在 2022 年 8 月推出印度尼西亚雅加达（ap-southeast-3）区域之前推出的所有 AWS 区域都有允许运行 AWS Glue 0.9/1.0 版本作业的客户名单。在这些较旧的区域中，您可以使用空值创建作业，根据区域的不同，该作业将默认为版本 0.9/1.0。对于任何以后推出的 AWS 区域，您必须在 API 中明确设置 AWS Glue 版本。AWS Glue 不再接受空参数。如果您在参数中传递 0.9 或 1.0，则会遇到错误“不支持 Glue 版本 0.9（或）1.0”。

在 Spark 中处理职位 AWS Glue

提供 AWS Glue 有关 Spark ETL 任务的信息。

主题

- AWS Glue 作业参数
- AWS Glue Spark 和 PySpark jobs
- 在 AWS Glue 中流式处理 ETL 作业
- 与 AWS Lake Formation FindMatches 匹配的记录

- 将 Apache Spark 程序迁移到 AWS Glue

AWS Glue 作业参数

创建 AWS Glue 作业时，需要设置一些标准字段，例如Role和WorkerType。您可以通过 Argument 字段（控制台中的作业参数）提供其他配置信息。在这些字段中，您可以为 AWS Glue 作业提供本主题中列出的参数（参数）。有关 Glue Job AWS API 的更多信息，请参阅the section called “作业”。

设置作业参数

您可以在控制台的 Job details（作业详细信息）选项卡的 Job Parameters（作业参数）标题下配置作业。您也可以 AWS CLI 通过设置DefaultArguments或在作业NonOverridableArguments上配置作业，或者在作业运行Arguments时设置来配置作业。每次运行作业时，都会传入在作业上设置的参数，而在作业运行中设置的参数将仅在单独运行时传入。

例如，以下是运行作业的语法，使用 --arguments 设置作业参数。

```
$ aws glue start-job-run --job-name "CSV to CSV" --arguments='--scriptLocation="s3://
my_glue/libraries/test_lib.py"'
```

访问作业参数

在编 AWS 写 Glue 脚本时，你可能需要访问作业参数值来改变自己代码的行为。我们在库中提供了执行此操作的辅助方法。这些方法可以解析覆盖作业参数值的作业运行参数值。解析在多个位置设置的参数时，作业 NonOverridableArguments 将覆盖作业运行 Arguments，这将覆盖作业 DefaultArguments。

在 Python 中：

在 Python 作业中，我们提供了一个名为 getResolvedParameters 的函数。有关更多信息，请参阅 the section called “getResolvedOptions”。sys.argv 变量中有作业参数。

在 Scala 中：

在 Scala 作业中，我们提供了一个名为 GlueArgParser 的对象。有关更多信息，请参阅 the section called “GlueArgParser”。sysArgs 变量中有作业参数。

作业参数参考

AWS Glue 可识别多个参数名称，这些参数名称可用于设置作业和作业运行的脚本环境：

--additional-python-modules

以逗号分隔的列表，表示要安装的一组 Python 包。您可以从 PyPI 安装包，也可以提供自定义分发。PyPI 包条目的格式为 *package*==*version*，带有目标包的 PyPI 名称和版本。自定义分发条目是到分发的 S3 路径。

条目使用 Python 版本匹配以匹配包和版本。这意味着您需要使用两个等号，例如 ==。还有其他版本匹配运算符，有关更多信息，请参阅 PEP 440。

要将模块安装选项传递给 pip3，请使用 --python-modules-installer-option 参数。

--auto-scale-within-microbatch

默认值为 False。此参数只能用于 AWS Glue 流式处理作业，该任务以一系列微批处理流式处理数据，并且必须启用自动缩放。将此值设置为 false 时，它会计算已完成的微批次的批次持续时间的指数移动平均值，并将该值与窗口大小进行比较，以确定是纵向扩展还是缩减执行程序数量。只有在微批次完成时才会进行扩展。将此值设置为 true 时，在微批批次期间，当 Spark 任务数在 30 秒内保持不变，或者当前批处理大于窗口大小时，它会纵向扩展。如果执行程序闲置超过 60 秒，或者批处理持续时间的指数移动平均线很低，则执行程序的数量将下降。

--class

用作 Scala 脚本之入口点的 Scala 类。仅在 `--job-language` 设置为 `scala` 时适用。

--continuous-log-conversionPattern

为已启用连续日志记录的作业指定自定义转换日志模式。转换模式仅适用于驱动程序日志和执行程序日志。它不会影响 AWS Glue 进度条。

--continuous-log-logGroup

为启用持续 CloudWatch 日志记录的任务指定自定义 Amazon 日志组名称。

--continuous-log-logStreamPrefix

为启用持续 CloudWatch 日志记录的作业指定自定义日志流前缀。

--customer-driver-env-vars 和 **--customer-executor-env-vars**

这些参数将在操作系统中分别为每个工作线程（驱动程序或执行程序）设置环境变量。在 AWS Glue 之上构建平台和自定义框架时，你可以使用这些参数，让你的用户在上面写作业。启用这两个标志后，可为驱动程序和执行程序分别设置不同的环境变量，而不必在作业脚本本身中注入相同的逻辑。

示例用法

以下是使用这些参数的示例：

```
"–customer-driver-env-vars", "CUSTOMER_KEY1=VAL1,CUSTOMER_KEY2=\"val2,val2 val2\"",
"–customer-executor-env-vars", "CUSTOMER_KEY3=VAL3,KEY4=VAL4"
```

在作业运行参数中设置这些值等同于运行以下命令：

在驱动程序中：

- export CUSTOMER_KEY1=VAL1
- export CUSTOMER_KEY2="val2,val2 val2"

在执行程序中：

- export CUSTOMER_KEY3=VAL3

然后在作业脚本本身中，您可以使用 `os.environ.get("CUSTOMER_KEY1")` 或 `System.getenv("CUSTOMER_KEY1")` 来检索环境变量。

强制实施语法

定义环境变量时应遵循以下标准：

- 每个键必须具有 `CUSTOMER_ prefix`。

  例如：对于 `"CUSTOMER_KEY3=VAL3,KEY4=VAL4"`，`KEY4=VAL4` 将被忽略并且不会设置。
- 每个键和值对都必须用一个逗号分隔。

  例如：`"CUSTOMER_KEY3=VAL3,CUSTOMER_KEY4=VAL4"`
- 如果“值”包含空格或逗号，则必须用引号括起来。

  例如：`CUSTOMER_KEY2=\"val2,val2 val2\"`

此语法与设置 bash 环境变量的标准非常接近。

`--datalake-formats`

在 AWS Glue 3.0 及更高版本中支持。

指定要使用的数据湖框架。 AWS Glue 会将您指定的框架所需的 JAR 文件添加到中`classpath`。有关更多信息，请参阅 在 AWS Glue ETL 任务中使用数据湖框架。

您可以指定以下一个或多个值，用逗号分隔：

- `hudi`

- delta
- iceberg

例如，传递以下参数以指定所有三个框架。

```
'--datalake-formats': 'hudi,delta,iceberg'
```

--disable-proxy-v2

禁用服务代理以允许对 Amazon S3 的 AWS 服务调用 CloudWatch，以及通过您的 VPC 从您的脚本AWS Glue发起的服务调用。有关更多信息，请参阅将 AWS 调用配置为通过您的 VPC。要禁用服务代理，请将此参数的值设置为 true。

--enable-auto-scaling

此值设置为 true 时，将打开自动扩缩和按工件计费的功能。

--enable-continuous-cloudwatch-log

允许对 AWS Glue 作业进行实时的连续日志记录。您可以在中查看 Apache Spark 的实时作业日志。 CloudWatch

--enable-continuous-log-filter

在创建或编辑为连续日志记录启用的作业时，指定标准筛选器 (true) 或无筛选器 (false)。选择标准筛选器可筛选掉无用的 Apache Spark 驱动程序/执行程序和 Apache Hadoop YARN 检测信号日志消息。选择无筛选器可提供所有日志消息。

--enable-glue-datacatalog

允许你将 AWS Glue 数据目录用作 Apache Spark Hive 元数据仓。要启用此功能，请将值设置为 true。

--enable-job-insights

通过 AWS Glue 作业运行见解启用额外的错误分析监控。有关更多信息，请参阅 the section called “通过 AWS Glue 任务运行洞察进行监控”。默认情况下，该值将设置为 true，并将启用任务运行洞察。

此选项仅可用于 AWS Glue 版本 2.0 和 3.0。

--enable-metrics

为此作业运行启用作业分析指标的集合。这些指标可在AWS Glue控制台和 Amazon CloudWatch 控制台上找到。此参数的值无关紧要。要启用此功能，您可以为该参数提供任意值，但为了清楚起见，建议使用 true。要禁用此功能，请从作业配置中移除此参数。

--enable-observability-metrics

启用一组可观察性指标，以深入了解AWS Glue控制台和控制台下的 “作业运行监控” 页面上运行的每个作业中 Amazon CloudWatch 发生的情况。要启用此功能，请将值设置为 true。要禁用此功能，将其设置为 `false` 或从作业配置中移除此参数。

--enable-rename-algorithm-v2

将 EMRFS 重命名算法版本设置为版本 2。当 Spark 任务使用动态分区覆盖模式时，可能会创建重复分区。例如，您最终可以得到 `s3://bucket/table/location/p1=1/p1=1` 之类的重复分区。在此处，P1 是被覆盖的分区。重命名算法版本 2 可以解决此问题。

此选项仅适用于 AWS Glue 版本 1.0。

--enable-s3-parquet-optimized-committer

启用经 EMRFS S3 优化的提交程序，用于将 Parquet 数据写入 Amazon S3。您可以在创建或更新 AWS Glue 作业时通过 AWS Glue 控制台提供参数/值对。将值设置为 **`true`** 将启用提交程序。默认情况下，该标志在 AWS Glue 3.0 中处于打开状态，在 Glue 2.0 中 AWS 处于关闭状态。

有关更多信息，请参阅使用经 EMRFS S3 优化的提交程序。

--enable-spark-ui

当设置为 `true` 时，打开该功能以使用 Spark UI 监控和调试 AWS Glue ETL 任务。

--executor-cores

可以并行运行的 Spark 任务数量。 AWS Glue 3.0及以上版本支持此选项。该值不能超过工作线程类型上 vCPU 数量的 2 倍，即 `G.1X` 上为 8、`G.2X` 上为 16、`G.4X` 上为 32，`G.8X` 上为 64。更新此配置时应谨慎行事，因为它可能会影响作业性能，因为任务并行度增加会导致内存和磁盘面临压力，并可能限制源系统和目标系统（例如：这将导致 Amazon RDS 上更多的并发连接）。

--extra-files

在执行脚本之前 AWS Glue 复制到脚本工作目录中的其他文件（如配置文件）的 Amazon S3 路径。多个值必须是以逗号（`,`）分隔的完整路径。仅支持单个文件而不是目录路径。Python Shell 作业类型不支持此选项。

--extra-jars

在执行脚本之前 AWS Glue 添加到 Java `.jar` 路径中的其他 Java 文件的 Amazon S3 路径。多个值必须是以逗号（`,`）分隔的完整路径。

--extra-py-files

在执行脚本之前 AWS Glue 添加到 Python 路径中的其他 Python 模块的 Amazon S3 路径。多个值必须是以逗号（,）分隔的完整路径。仅支持单个文件而不是目录路径。

--job-bookmark-option

控制作业书签的行为。可以设置以下选项值。

--job-bookmark-option 值	描述
job-bookmark-enable	追踪以前处理数据。当作业运行时，处理自上一个检查点以来的新数据。
job-bookmark-disable	始终处理整个数据集。您负责管理上一个作业运行的输出。
job-bookmark-pause	处理上次成功运行以来的增量数据或以下子选项标识的范围内的数据，但不更新最后一个书签的状态。您负责管理上一个作业运行的输出。这两个子选项的说明如下： • **job-bookmark-from** <from-value> 是运行 ID，它表示在最后一次成功运行之前所处理的所有输入，包括指定的运行 ID。对应的输入将被忽略。 • **job-bookmark-to** <to-value> 是运行 ID，它表示在最后一次成功运行之前所处理的所有输入，包括指定的运行 ID。作业会处理对应的输入（不包括由 <from-value> 标识的输入）。任何晚于此输入的输入也会被排除在外，不进行处理。 指定此选项集时，作业书签状态不更新。 子选项是可选的。但是，在使用时，必须提供两个子选项。

例如，要启用作业书签，请传递以下参数。

```
'--job-bookmark-option': 'job-bookmark-enable'
```

--job-language

脚本编程语言。此值必须为 scala 或 python。如果此参数不存在，默认值为 python。

--python-modules-installer-option

纯文本字符串，它定义了在使用 --additional-python-modules 安装模块时要传递给 pip3 的选项。像在命令行中一样提供选项，用空格分隔这些选项并以短划线作为前缀。有关使用情况的更多信息，请参阅 the section called “使用 pip 在 AWS Glue 2.0+ 中安装其他 Python 模块”。

> **Note**
>
> 当你使用 Python 3.9 时， AWS Glue 作业不支持此选项。

--scriptLocation

ETL 脚本所在的 Amazon Simple Storage Service（Amazon S3）位置（采用格式 s3://path/to/my/script.py）。此参数会覆盖 JobCommand 对象中设置的脚本位置。

--spark-event-logs-path

指定 Amazon S3 路径。使用 Spark UI 监控功能时，AWS Glue 会每 30 秒刷新 Spark 事件日志到 Amazon S3 路径的存储桶，该存储桶可用作存储 Spark UI 事件的临时目录。

--TempDir

指定可用作任务的临时目录的存储桶的 Amazon S3 路径。

例如，要设置临时目录，请传递以下参数。

```
'--TempDir': 's3-path-to-directory'
```

> **Note**
>
> 如果区域中尚不存在存储桶，则 AWS Glue 会为任务创建临时存储桶。此存储桶可能允许公开访问。您可以修改 Amazon S3 中的存储桶以设置公开访问数据块，也可以在该区域中的所有任务完成之后删除存储桶。

--use-postgres-driver

将此值设置为 `true` 时，系统会优先考虑类路径中的 Postgres JDBC 驱动程序，以避免与 Amazon Redshift JDBC 驱动程序发生冲突。此选项仅适用于 AWS Glue 版本 2.0。

--user-jars-first

将此值设置为 `true` 时，系统会优先考虑类路径中客户的额外 JAR 文件。此选项仅适用于 AWS Glue 版本 2.0 或更高版本。

--conf

控制 Spark 配置参数。适用于高级使用案例。

--encryption-type

传统参数。应使用安全配置来配置相应的行为。有关安全配置的更多信息，请参阅 the section called “加密 AWS Glue 写入的数据”。

AWS Glue 在内部使用以下参数，您永远不要使用它们：

- `--debug` - 供 AWS Glue 内部使用。请勿设置。
- `--mode` - 供 AWS Glue 内部使用。请勿设置。
- `--JOB_NAME` - 供 AWS Glue 内部使用。请勿设置。
- `--endpoint` - 供 AWS Glue 内部使用。请勿设置。

AWS Glue 支持使用 Python 的 `site` 模块引导环境，使用 `sitecustomize` 执行特定于站点的自定义。建议仅在高级用例中引导您自己的初始化函数，AWS Glue 4.0 支持尽力而为。

环境变量前缀 `GLUE_CUSTOMER` 保留给客户使用。

AWS Glue Spark 和 PySpark jobs

以下各节提供有关 AWS Glue Spark 和 PySpark 作业的信息。

主题

- 在 AWS Glue 中添加 Spark 和 PySpark 作业
- 使用作业书签跟踪已处理的数据

- AWS Glue Spark Shuffle 插件与 Amazon S3
- 监控 AWS Glue Spark 作业

在 AWS Glue 中添加 Spark 和 PySpark 作业

以下几个部分提供有关在 AWS Glue 中添加 Spark 和 PySpark 作业的信息。

主题

- 在中配置 Spark 作业的作业属性 AWS Glue
- 在 AWS Glue 控制台中编辑 Spark 脚本
- 作业（旧版）

在中配置 Spark 作业的作业属性 AWS Glue

AWS Glue 作业将封装连接到源数据的脚本，处理该脚本，然后将其写入数据目标。通常，作业运行提取、转换和加载 (ETL) 脚本。作业还可以运行通用 Python 脚本（Python shell 作业）。AWS Glue 触发器 可以根据计划或事件或者按需启动作业。您可以监控作业运行以了解运行时指标（例如完成状态、持续时间和开始时间）。

您可以使用 AWS Glue 生成的脚本，也可以提供您自己的脚本。使用源架构和目标位置或架构，AWS Glue代码生成器可以自动创建 Apache Spark API (PySpark) 脚本。您可以将此脚本用作起点，并对其进行编辑以满足您的目标。

AWS Glue 可以用多种数据格式编写输出文件，包括 JSON、CSV、ORC（优化的行列式）、Apache Parquet 和 Apache Avro。对于某些数据格式，可以编写常见的压缩格式。

AWS Glue 中有三种类型的作业：Spark、Streaming ETL 和 Python shell。

- Spark 作业在由 AWS Glue管理的 Apache Spark 环境中运行。它将批量处理数据。
- 流式处理 ETL 作业与 Spark 作业类似，只不过前者在数据流上执行 ETL。它使用 Apache Spark Structured Streaming 框架。某些 Spark 作业功能不可用于流式处理 ETL 作业。
- Python shell 任务将 Python 脚本作为 shell 运行，支持的 Python 版本具体取决于您使用的 AWS Glue 版本。您可以使用这些作业来计划和运行不需要 Apache Spark 环境的任务。

定义 Spark 作业的作业属性

当您在 AWS Glue 控制台中定义任务时，您会提供属性的值来控制 AWS Glue 的运行时环境。

以下列表描述了 Spark 作业的属性。有关 Python Shell 作业的属性，请参阅 定义 Python shell 作业的作业属性。有关流式处理 ETL 作业的属性，请参阅 the section called “定义串流 ETL 作业的作业属性”。

属性将按照在 AWS Glue 控制台上的 Add job (添加任务) 向导中显示的顺序列出。

名称

提供最大长度为 255 个字符的 UTF-8 字符串。

描述

可选的描述最长为 2048 个字符。

IAM 角色

指定用来为用于运行任务和访问数据存储的资源进行授权的 IAM 角色。有关在 AWS Glue 中运行作业的权限的更多信息，请参阅 AWS Glue 的身份和访问管理。

类型

ETL 作业的类型。这将根据您选择的数据来源类型自动设置。

- Spark 会使用作业命令 `glueetl` 运行 Apache Spark ETL 脚本。
- Spark 流式处理使用作业命令 `gluestreaming` 运行 Apache Spark 流式处理 ETL 脚本。有关更多信息，请参阅 the section called “流式处理 ETL 作业”。
- Python Shell 使用作业命令 `pythonshell` 运行 Python 脚本。有关更多信息，请参阅 在 AWS Glue 中为 Python shell 作业配置作业属性。

AWS Glue 版本

AWS Glue 版本决定了可用于任务的 Apache Spark 和 Python 的版本，如下表中所述。

AWS Glue 版本	支持的 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

AWS Glue 版本	支持的 Spark 和 Python 版本
1.0	• Spark 2.4.3 • Python 2.7 • Python 3.6
0.9	• Spark 2.2.1 • Python 2.7

工作线程类型

以下工作线程类型可供使用：

AWS Glue 工作人员上可用的资源以 DPU 为单位进行衡量。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。

- **G.1X** – 当您选择这种类型时，您还提供了 Number of workers (工件数量) 的值。每个工作线程映射到 1 个 DPU（4 个 vCPU，16GB 内存）和 84GB 磁盘（大约 34GB 可用空间）。我们建议将这种工作线程类型用于数据转换、联接和查询等工作负载，以提供一种可扩展且经济实惠的方式来运行大多数作业。
- **G.2X** – 当您选择这种类型时，您还提供了 Number of workers (工件数量) 的值。每个工作线程映射到 2 个 DPU（8 个 vCPU，32GB 内存）和 128GB 磁盘（大约 77GB 可用空间）。我们建议将这种工作线程类型用于数据转换、联接和查询等工作负载，以提供一种可扩展且经济实惠的方式来运行大多数作业。
- **G.4X** – 当您选择这种类型时，您还提供了 Number of workers (工件数量) 的值。每个工作线程映射到 4 个 DPU（16 个 vCPU，64 GB 内存）和 256GB 磁盘（大约 235GB 可用空间）。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作人员类型仅适用于以下 AWS 区域的 3.0 或更高AWS Glue版本的 Spark ETL 职位：美国东部（俄亥俄州）、美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）、加拿大（中部）、欧洲（法兰克福）、欧洲（爱尔兰）和欧洲（斯德哥尔摩）。
- **G.8X** – 当您选择这种类型时，您还提供了 Number of workers (工件数量) 的值。每个工作线程映射到 8 个 DPU（32 个 vCPU，128 GB 内存）和 512GB 磁盘（大约 487GB 可用空间）。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作器类型仅适用于 3.0 或更高AWS Glue版本的 Spark ETL 作业，其 AWS 区域与该G.4X工作人员类型支持的区域相同。

- **G.025X** – 当您选择这种类型时，您还提供了 Number of workers (工件数量) 的值。每个工作线程映射到 0.25 个 DPU（2 个 vCPU，4GB 内存）和 84GB 磁盘（大约 34GB 可用空间）。我们建议为低容量串流任务使用此 Worker 类型。此 Worker 类型仅适用于 AWS Glue 版本 3.0 串流任务。

您将根据用于运行 ETL 作业的 DPU 数量按小时支付费用。有关更多信息，请参阅 AWS Glue 价格页面。

对于 AWS Glue 版本 1.0 或更低版本的任务，当您使用控制台配置任务并指定 Standard (标准) 的 Worker type (工件类型) 时，则会设置 Maximum capacity (最大容量)，并且 Number of workers (工件数量) 会成为 Maximum capacity (最大容量) 值 - 1。如果您使用 AWS Command Line Interface (AWS CLI) 或 AWS SDK，则可以指定最大容量参数，也可以同时指定工作器类型和工作人员数量。

对于 AWS Glue 2.0 版或更高版本的作业，则不能指定最大容量。相反，您应该指定 Worker type (工件类型) 和 Number of workers (工件数量)。

Language

ETL 脚本中的代码定义了作业的逻辑。该脚本可以用 Python 或 Scala 进行编码。您可以选择作业运行的脚本是由 AWS Glue 生成还是由您提供。您需要在 Amazon Simple Storage Service（Amazon S3）中提供脚本名称和位置。确认没有与路径中的脚本目录同名的文件。要了解有关编写脚本的更多信息，请参阅AWS Glue 编程指南。

请求的工作线程数量

对于大多数工作线程类型，您必须指定作业运行时分配的工作线程数。

作业书签

指定当作业运行时，AWS Glue 如何处理状态信息。您可以让它记住之前处理过的数据、更新状态信息或忽略状态信息。有关更多信息，请参阅 the section called “使用作业书签跟踪已处理的数据”。

Flex 执行

使用 AWS Studio 或 API 配置作业时，可以指定标准或灵活的任务执行类。您的任务可能具有不同程度的优先级和时间敏感度。标准执行类非常适合需要快速任务启动和专用资源的时间敏感型工作负载。

灵活的执行类适用于非紧急任务，例如预生产任务、测试和一次性数据加载。使用 AWS Glue 版本 3.0 或更高版本以及 G.1X 或 G.2X 工作线程类型的任务支持灵活的任务运行。

灵活的任务运行基于在任何时间点运行的工作线程数量计费。对于正在运行的灵活任务运行，可以添加或移除工作线程数。每个工作线程将贡献它在任务运行期间所运行的时间，而不是计费为 `Max Capacity * Execution Time` 的简单计算。账单是 (`Number of DPUs per worker * time each worker ran`) 的总和。

有关更多信息，请参阅 AWS Studio 中的帮助面板，或任务和任务运行。

重试次数

指定 AWS Glue 在失败时自动重新启动任务的次数（从 0 到 10）。达到超时限制的任务不会重新启动。

作业超时

设置最大执行时间 (以分钟为单位)。批处理任务的默认值为 2880 分钟（48 小时）。当任务执行时间超过此限制时，任务运行状态更改为 `TIMEOUT`。

流式传输作业的超时值必须小于 7 天或 10080 分钟。如果将该值留空，则如果您尚未设置维护窗口，则该作业将在 7 天后重新启动。如果您设置了维护时段，则该窗口将在 7 天后的维护时段内重新启动。

> **有关作业超时的最佳实践**
>
> 作业按执行时间计费。为避免产生意外收费，请配置与作业的预期执行时间相适应的超时值。

高级属性

脚本文件名

作业的唯一脚本名称。不能命名为 Untitled job。

脚本路径

脚本的 Amazon S3 位置。路径必须采用 `s3://bucket/prefix/path/` 格式。它必须以斜杠（/）结尾，并且不包含任何文件。

作业指标

此任务运行时，开启或关闭创建 Amazon CloudWatch 指标。要查看分析数据，则必须启用此选项。有关如何启用和可视化指标的更多信息，请参阅 作业监控和调试。

作业可观测性指标

当此作业运行时，开启创建其他可观测性 CloudWatch 指标的功能。有关更多信息，请参阅 the section called “使用 AWS Glue 可观测性指标进行监控”。

连续日志记录

开启持续登录到 Amazon CloudWatch。如果未启用此选项，则只有在作业完成后日志才可用。有关更多信息，请参阅 the section called “AWS Glue 任务的连续日志记录”。

Spark UI

允许使用 Spark UI 监控此任务。有关更多信息，请参阅 为 AWS Glue 作业启用 Apache Spark Web UI。

Spark UI 日志路径

启用 Spark UI 时写入日志的路径。

Spark UI 日志记录和监控配置

请选择以下选项之一：

- 标准：使用 AWS Glue 作业运行 ID 作为文件名写入日志。在 AWS Glue 控制台中打开 Spark 用户界面监控。
- 旧版：使用“spark-application-{timestamp}”作为文件名写入日志。不打开 Spark UI 监控。
- 标准和旧版：将日志写入标准位置和旧版位置。在 AWS Glue 控制台中打开 Spark 用户界面监控。

最大并发数量

设置此作业允许的并发运行的最大数量。默认 为 1。达到此阈值时，将返回一个错误。可以指定的最大值由服务限制控制。例如，如果在启动新实例时，作业的前一运行仍在运行，则可能需要返回错误以防止同一作业的两个实例同时运行。

临时路径

在 Amazon S3 中提供工作目录的位置，以便当 AWS Glue 运行脚本时在该位置写入临时中间结果。确认没有与路径中的临时目录同名的文件。在 AWS Glue 读取和写入 Amazon Redshift 时使用此目录，或者通过某些 AWS Glue 转换使用。

> **Note**
>
> 如果区域中尚不存在存储桶，则 AWS Glue 会为任务创建临时存储桶。此存储桶可能允许公开访问。您可以修改 Amazon S3 中的存储桶以设置公开访问数据块，也可以在该区域中的所有任务完成之后删除存储桶。

延迟通知阈值（分钟）

设置发送延迟通知之前的阈值（以分钟为单位）。您可以设置此阈值，以在 RUNNING、STARTING 或 STOPPING 任务运行超过预期的分钟数时发送通知。

安全配置

从列表中选择安全配置。安全配置会指定如何对 Amazon S3 目标上的数据加密：不加密、使用 AWS KMS托管式密钥的服务器端加密（SSE-KMS）或者使用 Amazon S3 托管式加密密钥的服务器端加密（SSE-S3）。

服务器端加密

如果选择此选项，在将 ETL 任务写入 Amazon S3 时，数据将会使用 SSE-S3 加密进行静态加密。Amazon S3 数据目标和任何写入 Amazon S3 临时目录的数据都将被加密。此选项作为作业参数传递。有关更多信息，请参阅 Amazon Simple Storage Service 用户指南中的借助使用 Amazon S3 托管式加密密钥的服务器端加密 (SSE-S3) 保护数据。

> **Important**
>
> 如果指定了安全配置，则忽略此选项。

使用 Glue 数据目录作为 Hive 元存储

选择此项以使用 AWS Glue 数据目录作为 Hive 元存储。用于任务的 IAM 角色必须具有 `glue:CreateDatabase` 权限。在数据目录中创建一个名为“default”的数据库（如果该数据库不存在）。

连接

选择一个 VPC 配置来访问位于您的虚拟私有云（VPC）中的 Amazon S3 数据来源。您可以在 AWS Glue中创建和管理网络连接。有关更多信息，请参阅 连接到数据。

库

Python 库路径、从属 JAR 路径和参考文件路径

如果脚本需要这些选项，请指定。当您定义任务时，可以为这些选项定义使用逗号分隔的 Amazon S3 路径。当您运行任务时，可以覆盖这些路径。有关更多信息，请参阅 提供您自己的自定义脚本。

任务参数

作为命名参数传递给脚本的一组键值对。这些是在运行脚本时使用的默认值，但您可以在触发器中或运行任务时覆盖它们。您必须为键名使用前缀 `--`；例如：`--myKey`。使用时，您可以将作业参数作为地图传递 AWS Command Line Interface。

有关示例，请参阅在 AWS Glue 中传递和访问 Python 参数中的 Python 参数。

标签

使用 Tag key (标签键) 和可选的 Tag value (标签值) 来标记作业。创建标签键后，它们是只读的。对某些资源使用标签可帮助您整理和标识资源。有关更多信息，请参阅 AWS Glue 中的 AWS 标签。

针对访问 Lake Formation 托管表的作业的限制

在创建从管理的表中读取或写入的作业时，请记住以下注意事项和限制 AWS Lake Formation：

- 使用单元格级筛选条件访问表的任务中，不支持以下功能：
  - 任务书签和有界执行
  - 下推谓词
  - 服务器端目录分区谓词
  - enableUpdateCatalog

在 AWS Glue 控制台中编辑 Spark 脚本

脚本包含从源中提取数据、转换数据并将其加载到目标中的代码。 AWS Glue 在启动作业时运行脚本。

AWS Glue ETL 脚本可使用 Python 或 Scala 编码。Python 脚本使用的语言是 PySpark Python 方言的扩展，用于提取、转换和加载 (ETL) 作业。脚本包含扩展构造，用于处理 ETL 转换。当您为作业自动生成源代码逻辑时，会创建脚本。您可以编辑此脚本，也可以提供自己的脚本来处理您的 ETL 作业。

有关如何在 AWS Glue 中定义和编辑脚本的信息，请参阅 AWS Glue 编程指南。

其他库或文件

如果您的脚本需要额外的库或文件，您可以指定它们，如下所示：

Python 库路径

以逗号分隔的到脚本所需的 Python 库的 Amazon Simple Storage Service（Amazon S3）路径。

> **Note**
> 只能使用纯 Python 库。尚不支持依赖于 C 扩展的库，如 pandas Python 数据分析库。

从属 jars 路径

脚本所需的以逗号分隔的到 JAR 文件的 Amazon S3 路径。

> **Note**
> 目前，只能使用纯 Java 或 Scala (2.11) 库。

引用的文件路径

以逗号分隔的到脚本所需的其他文件（例如，配置文件）的 Amazon S3 路径。

作业（旧版）

脚本中包含用于执行提取、转换和加载 (ETL) 工作的代码。您可以提供您自己的脚本，或者 AWS Glue 可以通过您的指导生成脚本。有关创建您自己的脚本的信息，请参阅提供您自己的自定义脚本。

您可以在 AWS Glue 控制台中编辑脚本。当您编辑脚本时，您可以添加源、目标和转换。

编辑脚本

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。然后选择 Jobs 选项卡。
2. 在列表中选择一个作业，然后选择 Action、Edit script 以打开脚本编辑器。

您还可以从任务详细信息页面访问脚本编辑器。选择 Script (脚本) 选项卡，然后选择 Edit script (编辑脚本)。

脚本编辑器

利用 AWS Glue 脚本编辑器，您可以在脚本中插入、修改和删除源、目标和转换。脚本编辑器显示脚本和图表，可以帮助您直观呈现数据流。

要为脚本创建图表，请选择 Generate diagram (生成示意图)。AWS Glue 使用脚本中以 ## 开头的注释行来呈现示意图。要在图表中正确地表示脚本，必须将注释中的参数和 Apache Spark 代码中的参数保持同步。

脚本编辑器允许您在脚本中定位光标的任何位置添加代码模板。在编辑器的顶部，选择以下选项：

- 要向脚本中添加源表，请选择 Source。
- 要向脚本中添加目标表，请选择 Target。
- 要向脚本中添加目标位置，请选择 Target location。
- 要向脚本中添加转换，请选择 Transform。有关脚本中调用的函数的信息，请参阅在 AWS Glue ETL 脚本中编程 PySpark。
- 要向脚本中添加 Spigot，请选择 Spigot。

在插入的代码中，修改注释和 Apache Spark 代码中的 `parameters`。例如，如果您添加 Spigot 转换，请验证 `path` 在 `@args` 注释行和 `output` 代码行中都被替换。

Logs 选项卡显示在作业运行时与其关联的日志。将会显示最新的 1000 行。

Schema (架构) 选项卡显示选定源和目标的架构（在数据目录中可用时）。

使用作业书签跟踪已处理的数据

AWS Glue 通过保存作业运行的状态信息来跟踪上次运行 ETL 作业期间已处理的数据。此持久状态信息称为作业书签。作业书签可帮助 AWS Glue 维护状态信息，并可防止重新处理旧数据。有了作业书签，您可以在按照计划的时间间隔重新运行时处理新数据。作业书签包含作业的各种元素的状态，如源、转换和目标。例如，您的 ETL 任务可能会读取 Amazon S3 文件中的新分区。AWS Glue 跟踪任务已成功处理哪些分区，以防止任务的目标数据存储中出现重复处理和重复数据。

为 JDBC 数据源、关系化转换和一些 Amazon Simple Storage Service（Amazon S3）源实施了任务书签。下表列出了 AWS Glue 支持任务书签的 Amazon S3 源格式。

AWS Glue 版本	Amazon S3 源格式
版本 0.9	JSON、CSV、Apache Avro、XML
版本 1.0 及更高版本	JSON、CSV、Apache Avro、XML、Parquet、ORC

有关 AWS Glue 版本的信息，请参阅定义 Spark 作业的作业属性。

通过 AWS Glue 脚本访问作业书签功能时，还具有其他功能。浏览生成的脚本时，您可能会看到与此功能相关的转换上下文。有关更多信息，请参阅 the section called “使用作业书签”。

主题

- 在 AWS Glue 中使用作业书签
- 作业书签功能的操作详细信息

在 AWS Glue 中使用作业书签

在启动作业时，作业书签选项作为参数传递。下表描述了在 AWS Glue 控制台中用于设置任务书签的选项。

作业书签	描述
Enable	使作业在运行后更新状态，以跟踪之前处理的数据。如果作业的源支持作业书签，它将跟踪已处理的数据，当作业运行时，它将处理自上一检查点以来的新数据。
禁用	不使用作业书签，作业始终处理整个数据集。您负责管理上一个作业运行的输出。这是默认模式。
Pause	处理上次成功运行以来的增量数据或以下子选项标识的范围内的数据，无需更新最后一个书签的状态。您负责管理上一个作业运行的输出。这两个子选项是： • job-bookmark-from <from-value> 是运行 ID，它表示直到最后一次成功运行之前处理的所有输入，包括指定的运行 ID。对应的输入将被忽略。 • job-bookmark-to <to-value> 是运行 ID，它表示直到最后一次成功运行之前处理的所有输入，包括指定的运行 ID。由作业处理对应的输入，不

作业书签	描述
	包括 <from-value> 标识的输入。任何晚于此输入的输入也会被排除在外，不进行处理。 指定此选项集时，作业书签状态不更新。 子选项是可选的，但是使用时，需要提供两个子选项。

有关命令行上传递给作业的参数的详细信息（特别是关于作业书签的详细信息），请参阅AWS Glue 作业参数。

对于 Amazon S3 输入源，AWS Glue 任务书签将通过检查对象的上次修改时间来验证哪些对象需要重新处理。如果您的输入源数据在上次作业运行后已修改，则再次运行作业时将重新处理这些文件。

对于 JDBC 源，以下规则将适用：

- 对于每个表，AWS Glue 使用一个或多个列作为书签键来确定新数据和处理过的数据。书签键将合并成一个复合键。
- AWS Glue 默认情况下将使用主键作为书签键，前提是它按顺序递增或递减（没有间隙）。
- 可以指定要用作 AWS Glue 脚本中书签键的列。有关在 AWS Glue 脚本模式中使用作业书签的更多信息，请参阅 the section called “使用作业书签”。
- AWS Glue 不支持使用区分大小写的列作为作业书签键。

您可以将您的 AWS Glue Spark ETL 任务的任务书签倒回之前的任何任务运行。您可以通过将您的作业书签倒回之前的任何作业运行来更好的支持数据回填场景，从而使后续作业运行只再处理已做上标签的作业运行的数据。

如果您打算使用相同的作业重新处理所有数据，请重置作业书签。要重置作业书签状态，请使用 AWS Glue 控制台、ResetJobBookmark 动作（Python：重置作业书签）API 操作或 AWS CLI。例如，使用 AWS CLI 输入以下命令：

```
aws glue reset-job-bookmark --job-name my-job-name
```

在回放或重置书签时，AWS Glue 不会清除目标文件，因为可能有多个目标，并且不会使用任务书签跟踪目标。仅使用任务书签跟踪源文件。在回卷和重新处理源文件时，您可以创建不同的输出目标，以避免输出中出现重复数据。

AWS Glue 按作业跟踪作业书签。如果您删除作业，作业书签也会被删除。

有些情况下，您可能已启用 AWS Glue 作业书签，但您的 ETL 作业还在重新处理早期运行中已处理的数据。有关解决该错误的常见原因的信息，请参阅对 Spark 中的错误 AWS Glue 进行故障排除。

作业书签功能的操作详细信息

本节将介绍有关使用作业书签的更多操作详情。

作业书签存储作业的状态。每个状态实例的键由作业名称和版本号组成。当脚本调用 `job.init` 时，它将检索其状态并始终获取最新版本。在一个状态中，有多个状态元素，这些元素特定于脚本中的每个源、转换和接收器实例。这些状态元素由附加至脚本中相应元素（源、转换或接收器）的转换上下文来标识。从用户脚本中调用 `job.commit` 时，将自动保存状态元素。脚本将从参数中获取作业书签的作业名称和控制选项。

作业书签中的状态元素是特定于源、转换或接收器的数据。例如，假设您要从上游任务或进程不断写入数据的 Amazon S3 位置读取增量数据。在这种情况下，脚本必须确定目前已处理的内容。Amazon S3 源的任务书签实现将保存信息，这样，当任务再次运行时，它可以使用保存的信息仅筛选新对象并重新计算任务的下次运行状态。时间戳用于筛选新文件。

除了状态元素外，作业书签还有运行编号、尝试次数 和版本号。运行编号跟踪作业的运行，尝试次数记录作业运行的尝试次数。作业运行编号是随着每次成功运行单调增加的数字。尝试次数跟踪每次运行的尝试，仅在失败尝试后有运行时才增加。版本号单调增加并跟踪作业书签的更新。

在 AWS Glue 服务数据库中，所有转换的书签状态都作为键值对存储在一起：

```
{
  "job_name" : ...,
  "run_id": ...,
  "run_number": ..,
  "attempt_number": ...
  "states": {
    "transformation_ctx1" : {
      bookmark_state1
    },
    "transformation_ctx2" : {
      bookmark_state2
    }
```

```
  }
}
```

最佳实践

以下是使用作业书签的最佳实践。

- 请勿在启用书签的情况下更改数据源属性。例如，假定 datasource0 指向 Simple Storage Service (Amazon S3) 输入路径 A，并且该任务从启用书签情况下运行数轮的源进行读取。如果您将 datasource0 的输入路径更改为 Simple Storage Service (Amazon S3) 路径 B 而未更改 `transformation_ctx`，则 AWS Glue 任务将使用存储的旧书签状态。这将导致丢失或跳过输入路径 B 中的文件，因为 AWS Glue 将假定这些文件在以前的运行中已经处理过。
- 将目录表和书签搭配使用以实现更好的分区管理。书签既适用于来自数据目录的数据源，也适用于来自选项的数据源。但是，使用来自选项方法很难删除/添加新的分区。将目录表和爬网程序搭配使用可以提供更好的自动化，以跟踪新添加的分区并让您通过下推谓词灵活选择特定的分区。
- 使用适合大型数据集的 AWS Glue Simple Storage Service (Amazon S3 文件列表器。书签将列出每个输入分区下的所有文件并进行筛选，因此，如果单个分区下的文件过多，书签可以运行到驱动程序 OOM 中。使用 AWS Glue Simple Storage Service (Amazon S3) 文件列表器，以避免一次在内存中列出所有文件。

AWS Glue Spark Shuffle 插件与 Amazon S3

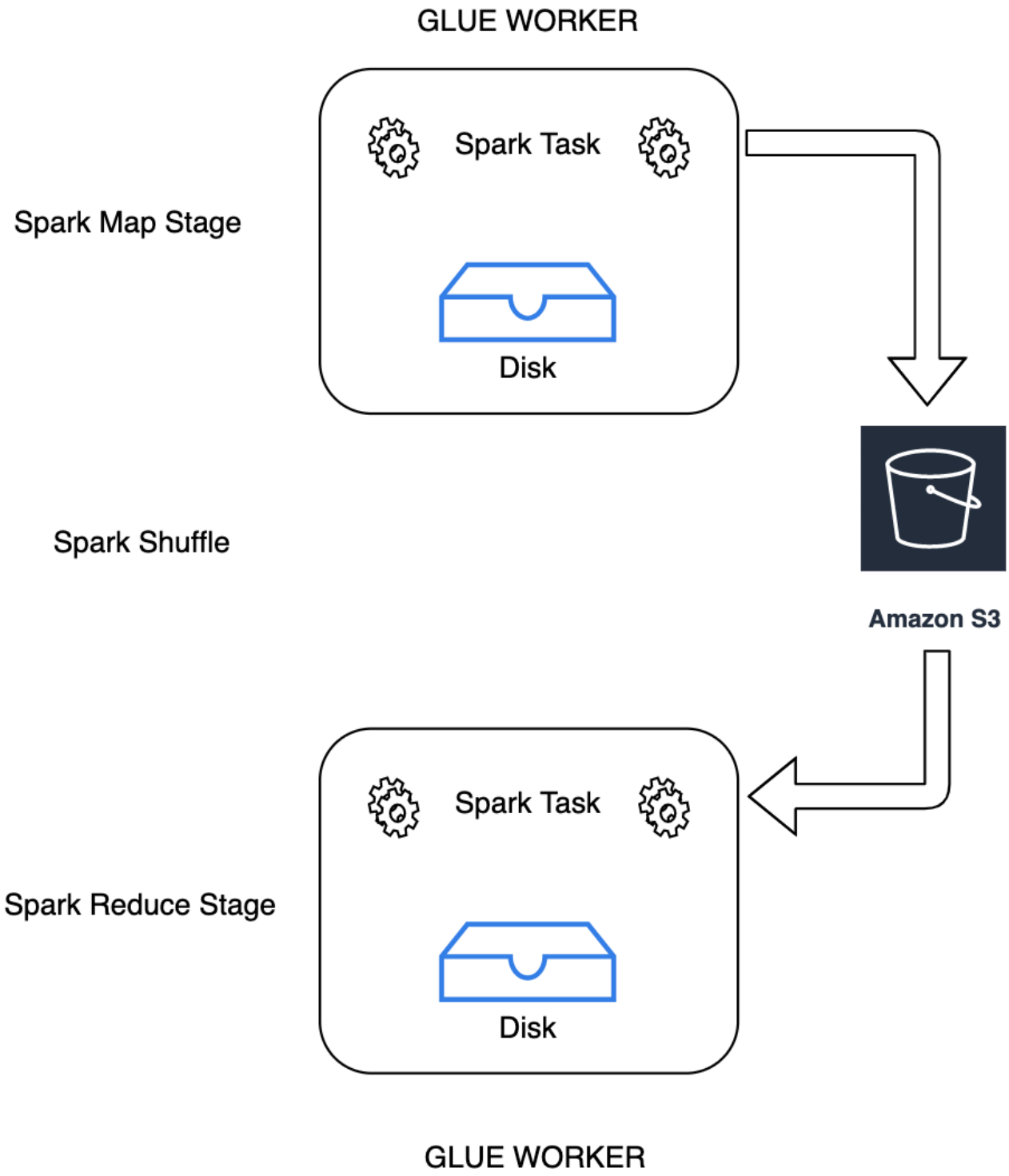
每当数据在分区之间重新排列时，随机排序是 Spark 任务中的一个重要步骤。这是必需的，因为 `join`、 `groupByKey`、`reduceByKey` 和 `repartition` 等广泛的转换需要来自其他分区的信息才能完成处理。Spark 从每个分区收集所需的数据，并将其合并到一个新的分区中。在随机排序过程中，数据会写入磁盘并通过网络传输。因此，随机排序操作绑定到本地磁盘容量。当执行程序上没有足够的磁盘空间并且没有恢复时，Spark 会抛出一个 `No space left on device` 或 `MetadataFetchFailedException` 错误。

> **Note**
>
> Amazon S3 中的 AWS Glue Spark Shuffle Plugin 仅支持 AWS Glue ETL 作业。

解决方案

通过 AWS Glue，您现在可以使用 Amazon S3 存储 Spark Shuffle 数据。Amazon S3 是一种对象存储服务，提供行业领先的可扩展性、数据可用性、安全性和性能。此解决方案可分解 Spark 任务的计算

和存储，并提供完整的弹性和低成本的随机排序存储，使您能够可靠地运行随机排序最密集的工作负载。



我们即将推出适用于 Apache Spark 的 Cloud Shuffle Storage 插件，以使用 Amazon S3。如果已知任务受大规模随机排序操作的本地磁盘容量限制，则可以启用 Amazon S3 随机排序来可靠、无故障地运

行 AWS Glue 任务。在某些情况下，如果您有大量的小分区或随机排序文件写入 Amazon S3，则随机排序到 Amazon S3 的速度比本地磁盘（或 EBS）慢一些。

使用 Cloud Shuffle Storage Plugin 的先决条件

要将 Cloud Shuffle Storage Plugin 用于 AWS Glue ETL 作业，需要满足以下条件：

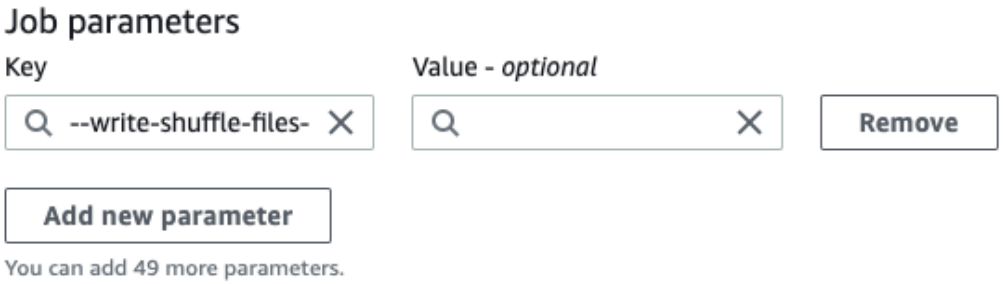
- 与您的作业运行位于同一区域的 Amazon S3 存储桶，用于存储随机排序和溢出的数据。可使用 `--conf spark.shuffle.glue.s3ShuffleBucket=s3://`*`shuffle-bucket`*`/`*`prefix`*`/` 指定随机排序存储的 Amazon S3 前缀，如以下示例所示：

```
--conf spark.shuffle.glue.s3ShuffleBucket=s3://glue-shuffle-123456789-us-east-1/glue-shuffle-data/
```

- 在前缀上设置 Amazon S3 存储生命周期策略（例如 `glue-shuffle-data`），因为作业完成后，随机排序管理器不会清理文件。作业完成后，应删除中间随机排序和溢出的数据。用户可以在前缀上设置短生命周期策略。有关设置 Amazon S3 生命周期策略的说明，请参阅《Amazon Simple Storage Service 用户指南》中的 Setting lifecycle configuration on a bucket。

从 AWS 控制台使用 AWS Glue Spark 随机排序管理器

在配置作业时使用 AWS Glue 控制台或 AWS Glue Studio 设置 AWS Glue Spark 随机排序管理器：选择 --write-shuffle-files-to-s3 作业参数启用作业的 Amazon S3 随机排序。



使用 AWS Glue Spark Shuffle 插件

以下任务参数启用并调整 AWS Glue 随机排序管理器。这些参数是标志，因此不考虑所提供的任何值。

- `--write-shuffle-files-to-s3` – 主标志，启用 AWS Glue Spark 随机排序管理器，以使用 Amazon S3 存储桶写入和读取随机排序数据。如果标志未指定，则不使用随机排序管理器。

- `--write-shuffle-spills-to-s3`— (仅在 AWS Glue 版本 2.0 上支持) 。一个可选标志，允许您将溢出文件卸载到 Amazon S3 存储桶，从而为您的 Spark 作业提供额外的弹性。只有将大量数据溢出到磁盘的大型工作负载才需要这样做。如果未指定该标志，则不会写入任何中间溢出文件。
- `--conf spark.shuffle.glue.s3ShuffleBucket=s3://<shuffle-bucket>` – 另一个可选标志，用于指定您在其中写入随机排序文件的 Amazon S3 存储桶。默认情况下，`--TempDir/shuffle-data`. AWS Glue 3.0+ 支持通过使用逗号分隔符指定存储桶，将随机排序文件写入多个存储桶，如 `--conf spark.shuffle.glue.s3ShuffleBucket=s3://`*`shuffle-bucket-1`*`/`*`prefix`*`,s3://`*`shuffle-bucket-2`*`/`*`prefix`*`/` 中所示。使用多个存储桶可以提高性能。

您需要提供安全配置设置，才能对随机排序数据启用静态加密。有关安全配置的更多信息，请参阅 the section called “设置加密”。AWS Glue 支持 Spark 提供的所有其他随机排序相关配置。

Cloud Shuffle Storage 插件的软件二进制文件

您还可以在 Apache 2.0 许可证下下载适用于 Apache Spark 的 Cloud Shuffle Storage 插件的软件二进制文件，然后在任何 Spark 环境中运行它。新插件对 Amazon S3 提供开箱即用支持，也可以轻松配置为使用其他形式的云存储，例如 Google Cloud Storage 和 Microsoft Azure Blob Storage。有关更多信息，请参阅适用于 Apache Spark 的 Cloud Shuffle Storage 插件。

注释和限制

以下是 AWS Glue 随机排序管理器的注释或限制：

- 作业完成后，AWS Glue 随机排序管理器不会自动删除存储在 Amazon S3 存储桶中的 (临时) 随机排序数据文件。为确保数据保护，请在启用 Cloud Shuffle Storage Plugin 之前按照 使用 Cloud Shuffle Storage Plugin 的先决条件 中的说明进行操作。
- 如果数据出现误差，则可以使用此功能。

适用于 Apache Spark 的 Cloud Shuffle 存储插件

Cloud Shuffle 存储插件是一款与 `ShuffleDataIO` API 兼容的 Apache Spark 插件，它允许在云存储系统 (例如 Amazon S3) 上存储 shuffle 数据。它可以帮助您补充或替换本地磁盘存储容量以进行大型随机播放，这些操作通常由 Spark 应用程序中的 `join`、`reduceByKey`、`groupByKey` 和 `repartition` 等转换触发，从而减少无服务器数据分析任务和管道的常见故障或性价比/性能失调。

AWS Glue

AWS Glue 版本 3.0 和 4.0 预装了插件，无需任何额外步骤即可在 Amazon S3 上进行重组。有关更多信息，请参阅 AWS Glue 搭载 Amazon S3 的 Spark shuffle 插件，以为您的 Spark 应用程序启用该功能。

其他 Spark 环境

该插件要求在其他 Spark 环境中设置以下 Spark 配置：

- `--conf spark.shuffle.sort.io.plugin.class=com.amazonaws.spark.shuffle.io.cloud.Choppe`
  这通知 Spark 为 Shuffle IO 使用这个插件。
- `--conf spark.shuffle.storage.path=s3://`*`bucket-name`*`/`*`shuffle-file-dir`*：您的随机播放文件的存储路径。

> **Note**
>
> 该插件覆盖了一个 Spark 核心类。因此，需要在 Spark jar 之前加载插件 jar。如果在 AWS Glue 外部使用插件，则可以使用 `userClassPathFirst` 在本地 YARN 环境中执行此操作。

将插件与您的 Spark 应用程序捆绑在一起

在本地开发 Spark 应用程序时，您可以在 Maven `pom.xml` 中添加插件依赖关系，将插件与 Spark 应用程序和 Spark 发行版（版本 3.1 及更高版本）捆绑在一起。有关插件和 Spark 版本的更多信息，请参阅插件版本。

```
<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>
```

您也可以直接从 AWS Glue Maven 构件下载二进制文件，并将它们包含在您的 Spark 应用程序中，如下所示。

```
#!/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 提交示例

```
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> \
```

可选配置

这些是控制 Amazon S3 随机播放行为的可选配置。

- `spark.shuffle.storage.s3.enableServerSideEncryption`：为随机播放和溢出文件启用/禁用 S3 SSE。默认值为 `true`。
- `spark.shuffle.storage.s3.serverSideEncryption.algorithm`：要使用的 SSE 算法。默认值为 `AES256`。
- `spark.shuffle.storage.s3.serverSideEncryption.kms.key`：启用 SSE aws:kms 时的 KMS 密钥 ARN。

除了这些配置之外，您可能还需要设置配置（例如 `spark.hadoop.fs.s3.enableServerSideEncryption` 和其他特定于环境的配置），以确保为您的用例应用适当的加密。

插件版本

与每个 AWS Glue 版本关联的 Spark 版本都支持此插件。下表显示了插件的软件二进制文件的 AWS Glue 版本、Spark 版本和关联的插件版本以及 Amazon S3 的位置。

AWS Glue 版本	Spark 版本	插件版本	Amazon S3 位置
3.0	3.1	3.1-amzn-LATEST	s3://aws-glue-etl-artifacts/release/com/amazonaws/chopper-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

许可证

此插件的软件二进制文件已根据 Apache-2.0 许可证获得许可。

监控 AWS Glue Spark 作业

主题

- Spark 指标可在中找到 AWS Glue Studio
- 使用 Apache Spark Web UI 监控作业
- 通过 AWS Glue 任务运行洞察进行监控
- 使用 Amazon CloudWatch 监控
- 作业监控和调试

Spark 指标可在中找到 AWS Glue Studio

Metrics (指标) 选项卡显示启用任务运行和分析时收集的指标。Spark 作业显示了以下图表：

- ETL 数据移动
- 内存配置文件：驱动程序和执行程序

选择 View additional metrics (查看其他指标) 显示以下图表：

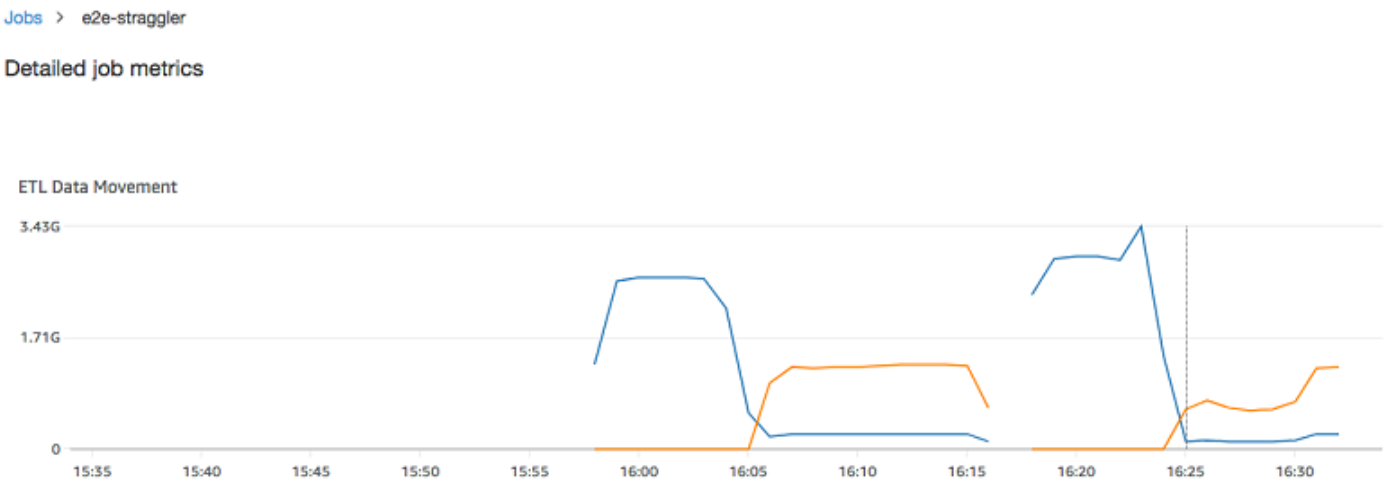
- ETL 数据移动
- 内存配置文件：驱动程序和执行程序
- 执行程序之间的数据随机排序
- CPU 负载：驱动程序和执行程序
- 作业执行：活动执行程序、已完成的阶段和需求最大的执行程序

如果任务配置为收集 CloudWatch 指标，则这些图表的数据将推送到指标。有关如何启用指标和解释图表的更多信息，请参阅作业监控和调试。

Example ETL 数据移动图表

ETL 数据移动图表会显示以下指标：

- 所有执行程序从 Amazon S3 读取的字节数 – glue.ALL.s3.filesystem.read_bytes
- 所有执行程序写入 Amazon S3 的字节数 – glue.ALL.s3.filesystem.write_bytes

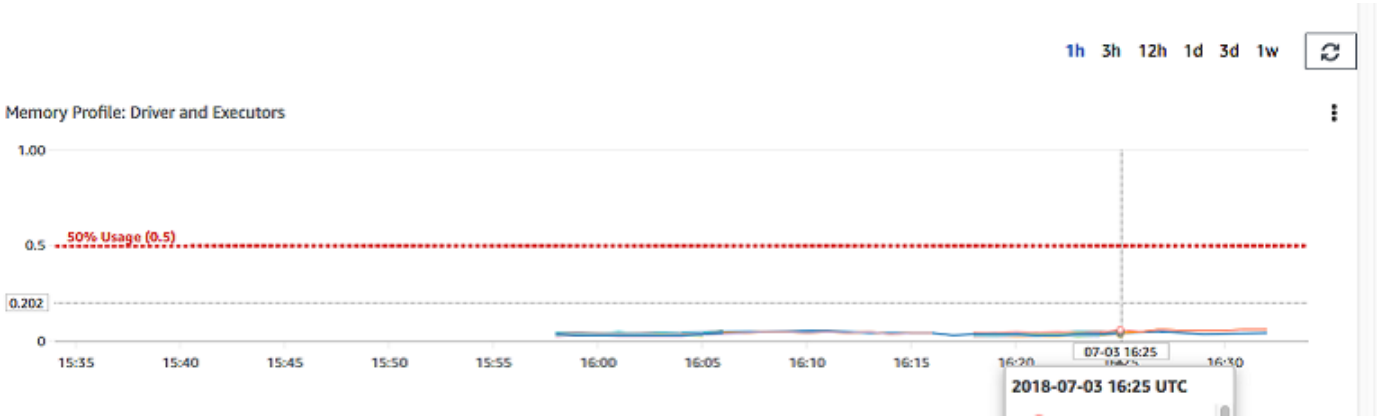


Example 内存配置文件图表

内存配置文件图表会显示以下指标：

- 驱动程序的 JVM 堆用于此驱动程序的内存量（比例：0-1），用 executorId 标识的执行程序，或所有执行程序 –
  - glue.driver.jvm.heap.usage

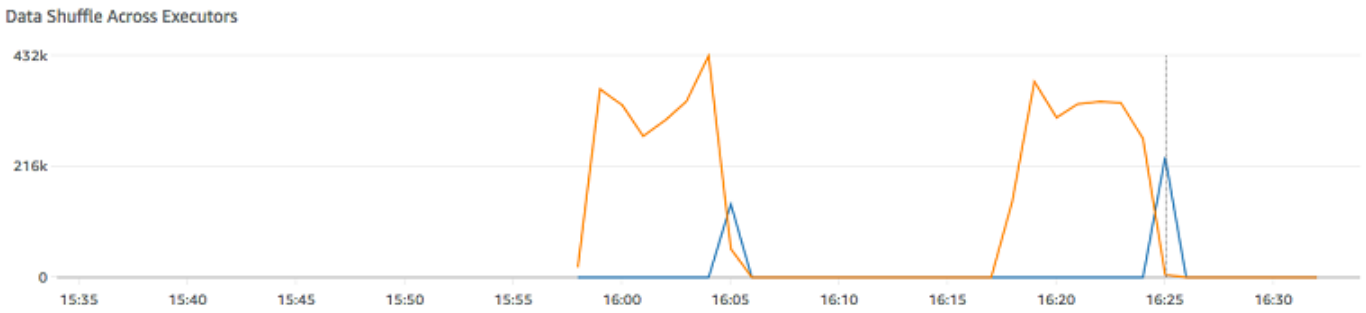
- glue.*executorId*.jvm.heap.usage
- glue.ALL.jvm.heap.usage



Example 执行程序之间的数据随机排序图表

执行程序之间的数据随机排序图表显示了以下指标：

- 所有执行程序用于在它们之间对数据进行随机排序所读取的字节数 -glue.driver.aggregate.shuffleLocalBytesRead
- 所有执行程序用于在它们之间对数据进行随机排序所写入的字节数 -glue.driver.aggregate.shuffleBytesWritten

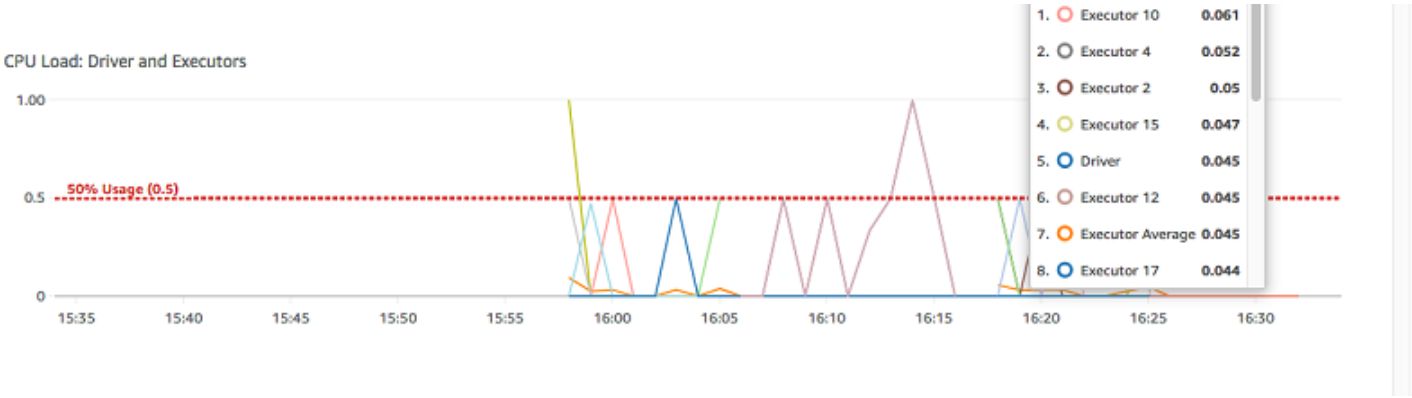


Example CPU 负载图表

CPU 负载图表显示以下指标：

- 驱动程序使用的 CPU 系统负载量（比例：0-1），用 executorId 标识的执行程序，或所有执行程序 –

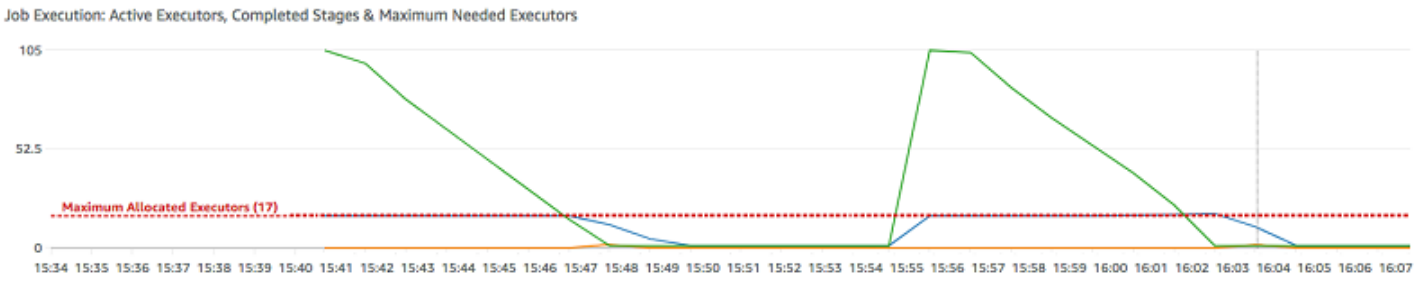
- glue.driver.system.cpuSystemLoad
- glue.*executorId*.system.cpuSystemLoad
- glue.ALL.system.cpuSystemLoad



Example 作业执行图表

作业执行图表显示以下指标：

- 主动运行的执行程序的数量 -glue.driver.ExecutorAllocationManager.executors.numberAllExecutors
- 已完成的阶段数量 -glue.aggregate.numCompletedStages
- 需求最大的执行程序的数量 -glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors



使用 Apache Spark Web UI 监控作业

可以使用 Apache Spark Web UI 监控和调试在 AWS Glue 作业系统上运行的 AWS Glue ETL 作业以及在 AWS Glue 开发终端节点上运行的 Spark 应用程序。可使用 Spark UI 检查每个作业的以下内容：

- 每个 Spark 阶段的事件时间表
- 作业的有向非循环图 (DAG)
- SparkSQL 查询的物理和逻辑计划
- 每个作业的基础 Spark 环境变量

有关使用 Spark Web UI 的更多信息，请参阅 Spark 文档中的 Web UI。有关如何解释 Spark UI 结果以提高作业性能的指导，请参阅 AWS 规范性指南中的 Apache Spark 作业性能调整 AWS Glue 最佳实践。

你可以在 AWS Glue 控制台中看到 Spark 用户界面。当 AWS Glue 作业在 AWS Glue 3.0 或更高版本上运行，并且日志以标准（而不是传统）格式生成（这是较新作业的默认格式）时，此功能可用。如果您的日志文件大于 0.5 GB，则可以为在 AWS Glue 4.0 或更高版本上运行的作业启用滚动日志支持，以简化日志存档、分析和故障排除。

您可以使用AWS Glue控制台或 AWS Command Line Interface (AWS CLI) 启用 Spark 用户界面。在启用 Spark UI 后，AWS Glue 开发端点上的 AWS Glue ETL 任务和 Spark 应用程序可将 Spark 事件日志备份到您在 Amazon Simple Storage Service (Amazon S3) 中指定的位置。您可以将 Amazon S3 中备份的事件日志与 Spark UI 一起使用，既可以在任务运行时实时使用，也可以在任务完成后使用。当日志保留在 Amazon S3 中时， AWS Glue 控制台中的 Spark 用户界面可以查看它们。

权限

要在 AWS Glue 控制台中使用 Spark 用户界面，您可以使用`UseGlueStudio`或添加所有单独的服务 API。需要所有 API 才能完全使用 Spark UI，不过用户可以通过在其 IAM 权限中添加相关服务 API 来访问相应的 SparkUI 功能，从而实现精细访问控制。

`RequestLogParsing` 负责执行日志解析，因此最为关键。其余的 API 用于读取相应的解析数据。例如，`GetStages` 将提供对 Spark 作业所有阶段数据的访问权限。

在示例策略中，映射到 `UseGlueStudio` 的 Spark UI 服务 API 列表如下。以下策略仅提供使用 Spark UI 功能的权限。要添加更多权限，例如 Amazon S3 和 IAM，请参阅为其创建自定义 IAM 策略 AWS Glue Studio。

在示例策略中，映射到 `UseGlueStudio` 的 Spark UI 服务 API 列表如下。使用 Spark UI 服务 API 时，请使用以下命名空间：`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 控制台中的 Spark UI 不适用于 2023 年 11 月 20 日之前的任务运行，因为它们采用的是传统日志格式。
- AWS Glue 控制台中的 Spark UI 支持 AWS Glue 4.0 的滚动日志，例如流媒体作业中默认生成的日志。所有生成的滚动日志事件文件的最大总和为 2 GB。对于不支持滚动日志的 AWS Glue 作业，SparkUI 支持的最大日志事件文件大小为 0.5 GB。
- 无服务器 Spark 用户界面不适用于存储在只能由您的 VPC 访问的 Amazon S3 存储桶中的 Spark 事件日志。

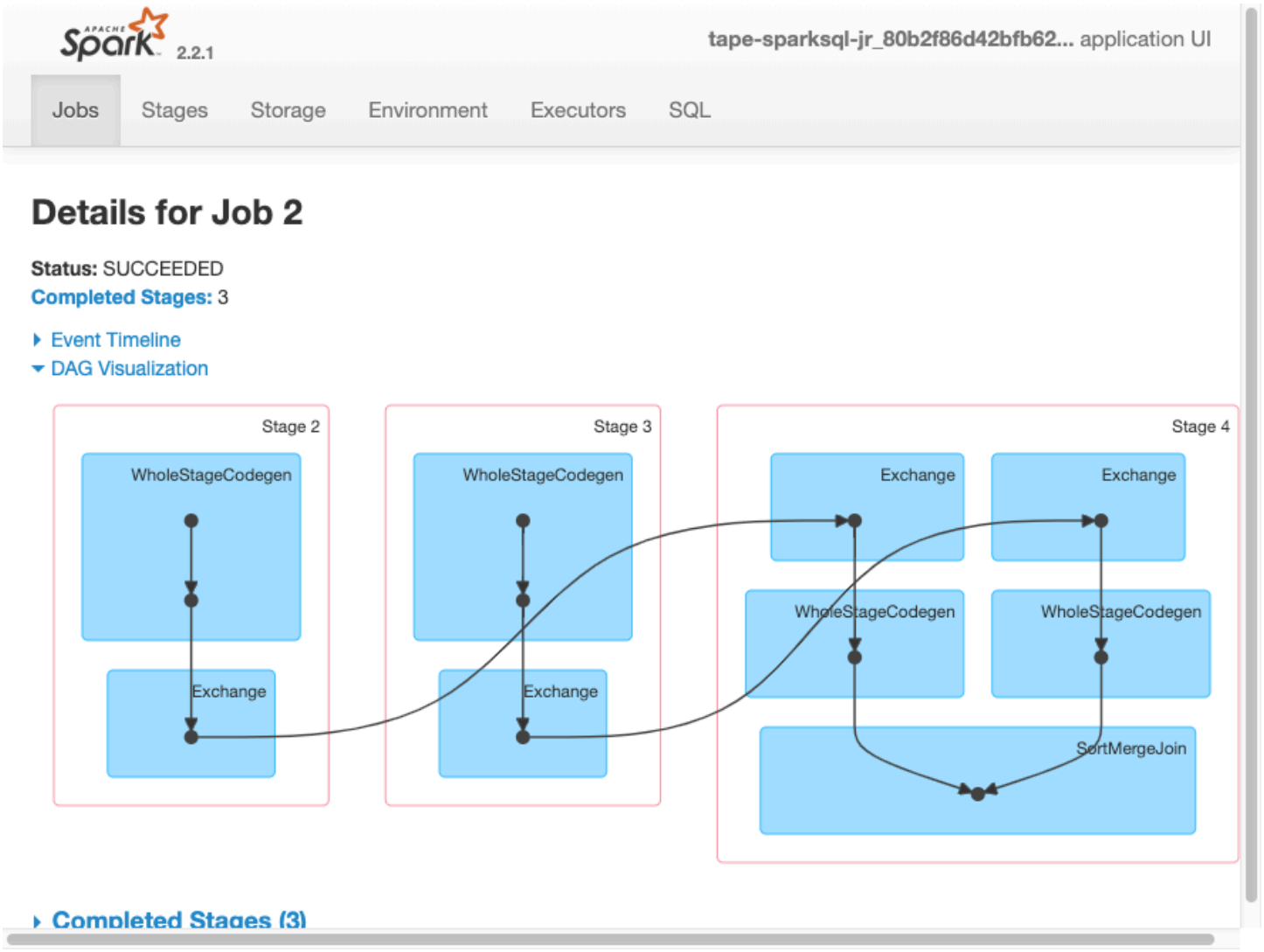
示例：Apache Spark Web UI

此示例演示了如何使用 Spark UI 了解任务性能。屏幕截图显示了由自行管理的 Spark 历史记录服务器提供的 Spark Web UI。 AWS Glue 控制台中的 Spark 用户界面提供了类似的视图。有关使用 Spark Web UI 的更多信息，请参阅 Spark 文档中的 Web UI。

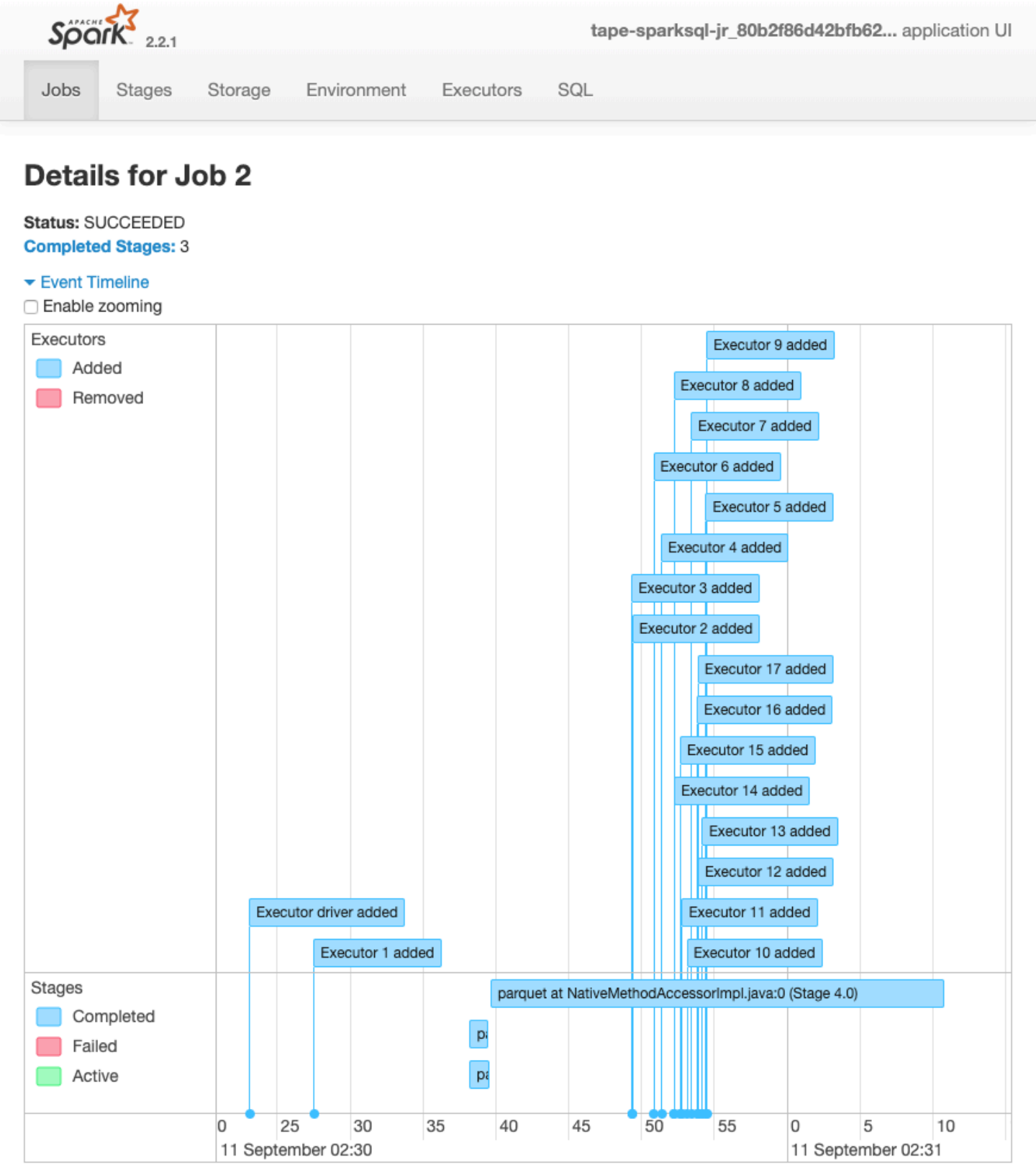
以下是一个示例 Spark 应用程序，其从两个数据源读取数据，执行联接转换，并以 Parquet 格式将其写入 Amazon S3。

```
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()
```

以下 DAG 可视化显示此 Spark 作业中的各个阶段。



以下作业事件时间表显示了各个 Spark 执行程序的启动、执行和终止。


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)

以下屏幕显示了 SparkSQL 查询计划的详细信息：

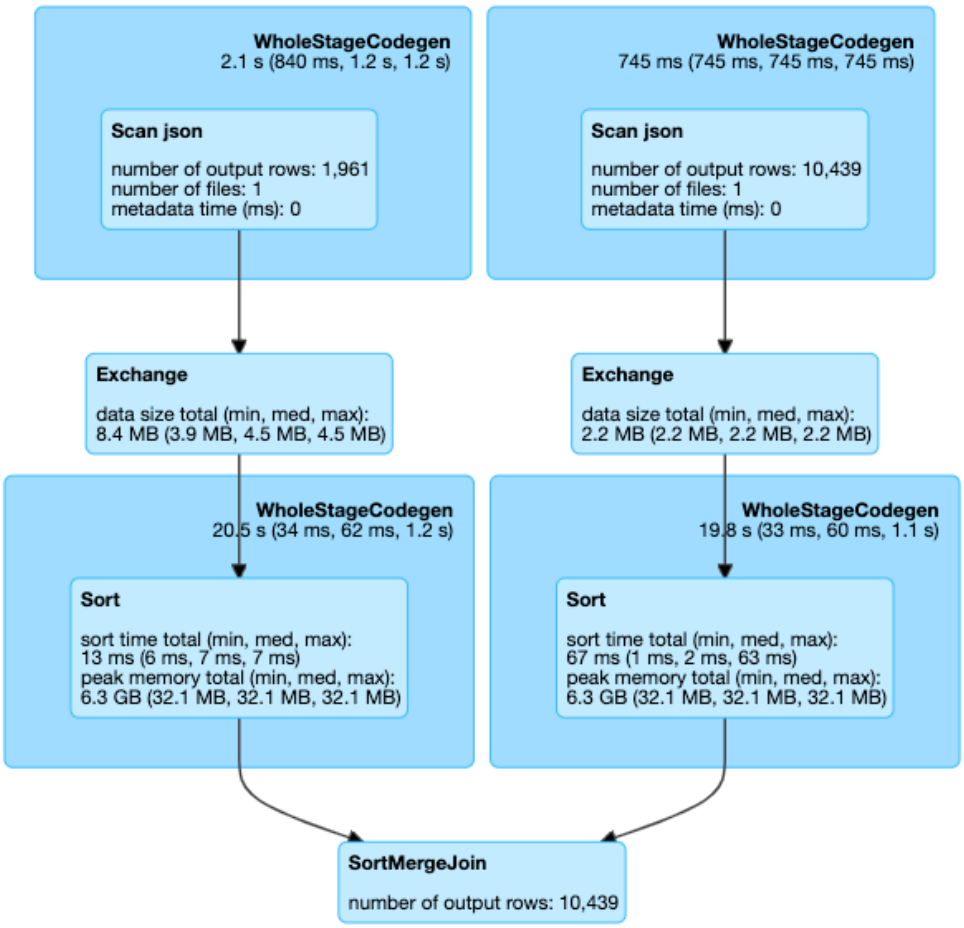
- 已解析的逻辑计划
- 已分析的逻辑计划
- 已优化的逻辑计划
- 执行的物理计划

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



▾ 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)
```

主题

为 AWS Glue 作业启用 Apache Spark Web UI

您可以使用 Apache Spark Web UI 监控和调试在 AWS Glue 作业系统上运行的 AWS Glue ETL 作业。您可以使用 AWS Glue 控制台或 AWS Command Line Interface (AWS CLI) 配置 Spark UI。

AWS Glue 每 30 秒将 Spark 事件日志备份到您指定的 Amazon S3 路径一次。

主题

配置 Spark UI（控制台）

按照以下步骤使用 AWS Management Console配置 Spark UI。创建 AWS Glue 任务时，Spark 用户界面默认处于启用状态。

在创建或编辑任务时启用 Spark UI

1. 登录 AWS Management Console 并打开AWS Glue控制台，网址为 https://console.aws.amazon.com/glue/。
2. 在导航窗格中，选择作业。
3. 选择添加作业，或选择现有的作业。
4. 在作业详细信息中，打开高级属性。
5. 在 Spark UI 选项卡下，选择将 Spark UI 日志写入 Amazon S3。
6. 指定用于存储任务的 Spark 事件日志的 Amazon S3 路径。请注意，如果您在任务中使用安全配置，则加密也将适用于 Spark UI 日志文件。有关更多信息，请参阅 加密 AWS Glue 写入的数据。
7. 在 Spark UI 日志记录和监控配置下：
    - 如果您要生成要在 AWS Glue 控制台中查看的日志，请选择标准。

- 如果要生成可在 Spark 历史记录服务器上查看的日志，请选择传统。
- 您还可以选择同时生成这两种日志。

配置 Spark UI (AWS CLI)

要生成日志以供使用 Spark UI 查看，请在 AWS Glue 控制台中使用将以下作业参数传递 AWS CLI 给 AWS Glue作业。有关更多信息，请参阅 the section called “任务参数”。

```
'--enable-spark-ui': 'true',
'--spark-event-logs-path': 's3://s3-event-log-path'
```

要将日志分发到其遗留位置，请将 `--enable-spark-ui-legacy-path` 参数设置为 `"true"`。如果不需要同时生成两种格式的日志，请移除 `--enable-spark-ui` 参数。

为使用笔记本的会话配置 Spark 用户界面

> **Warning**
>
> AWS Glue 交互式会话目前不支持控制台中的 Spark 用户界面。配置 Spark 历史记录服务器。

如果您使用 AWS Glue 笔记本电脑，请在开始会话之前设置 SparkUI 配置。为此，请使用 `%%configure` 单元格魔术命令：

```
%%configure { “--enable-spark-ui”: “true”, “--spark-event-logs-path”: “s3://path” }
```

启用滚动日志

为 AWS Glue 作业启用 SparkUI 和滚动日志事件文件有以下好处：

- 滚动日志事件文件-启用滚动日志事件文件后， AWS Glue 会为任务执行的每个步骤生成单独的日志文件，从而更轻松地识别和解决特定阶段或转换的问题。
- 更好的日志管理-滚动日志事件文件有助于更有效地管理日志文件。这些日志不是一个可能很大的日志文件，而是根据任务执行阶段拆分为更小、更易于管理的文件。这可以简化日志存档、分析和故障排除。
- 提高容错能力-如果 AWS Glue 作业失败或中断，滚动日志事件文件可以提供有关最后成功阶段的宝贵信息，从而更轻松地从该点恢复作业，而不是从头开始。

- 成本优化-通过启用滚动日志事件文件，您可以节省与日志文件相关的存储成本。与其存储单个可能很大的日志文件，不如存储更小、更易于管理的日志文件，这可能更具成本效益，特别是对于长时间运行或复杂的作业。

在新环境中，用户可以通过以下方式显式启用滚动日志：

```
'–conf': 'spark.eventLog.rolling.enabled=true'
```

或者

```
'–conf': 'spark.eventLog.rolling.enabled=true –conf
spark.eventLog.rolling.maxFileSize=128m'
```

激活滚动日志时，`spark.eventLog.rolling.maxFileSize`指定事件日志文件在滚动之前的最大大小。如果未指定，则此可选参数的默认值为 128 MB。最小值为 10 MB。

所有生成的滚动日志事件文件的最大总和为 2 GB。对于不支持滚动日志的 AWS Glue 作业，SparkUI 支持的最大日志事件文件大小为 0.5 GB。

您可以通过传递额外配置来关闭流式处理任务的滚动日志功能。请注意，非常大的日志文件的维护成本可能很高。

要关闭滚动日志，请提供以下配置：

```
'--spark-ui-event-logs-path': 'true',
'--conf': 'spark.eventLog.rolling.enabled=false'
```

启动 Spark 历史记录服务器

您可以使用 Spark 历史记录服务器在自己的基础设施上可视化显示 Spark 日志。对于在 AWS Glue 4.0 或更高版本上运行并以标准（而不是传统）格式生成日志的 AWS Glue 任务，您可以在 AWS Glue 控制台中看到相同的可视化效果。有关更多信息，请参阅 the section called “使用 Spark UI 进行监控”。

您可以使用在 EC2 实例上托管服务器的 AWS CloudFormation 模板启动 Spark 历史记录服务器，也可以使用 Docker 在本地启动 Spark 历史记录服务器。

主题

- 使用 AWS CloudFormation 启动 Spark 历史记录服务器并查看 Spark UI

- 使用 Docker 启动 Spark 历史记录服务器并查看 Spark UI

使用 AWS CloudFormation 启动 Spark 历史记录服务器并查看 Spark UI

您可以使用 AWS CloudFormation 模板启动 Apache Spark 历史记录服务器并查看 Spark Web UI。这些模板是您应修改以满足要求的示例。

使用 AWS CloudFormation 启动 Spark 历史记录服务器并查看 Spark UI

1. 选择下表中的 Launch Stack (启动堆栈) 按钮之一。这将在 AWS CloudFormation 控制台上启动堆栈。

区域	发布
美国东部（俄亥俄）	Launch Stack
美国东部（弗吉尼亚州北部）	Launch Stack
美国西部（北加利福尼亚）	Launch Stack
美国西部（俄勒冈州）	Launch Stack
非洲（开普敦）	Launch Stack
亚太地区（香港）	Launch Stack
Asia Pacific (Mumbai)	Launch Stack
亚太地区（大阪）	Launch Stack
亚太地区（首尔）	Launch Stack
亚太地区（新加坡）	Launch Stack

区域	发布
亚太地区（悉尼）	Launch Stack
亚太地区（东京）	Launch Stack
加拿大（中部）	Launch Stack
欧洲地区（法兰克福）	Launch Stack
欧洲地区（爱尔兰）	Launch Stack
欧洲地区（伦敦）	Launch Stack
欧洲地区（米兰）	Launch Stack
欧洲地区（巴黎）	Launch Stack
欧洲地区（斯德哥尔摩）	Launch Stack
中东（巴林）	Launch Stack
南美洲（圣保罗）	Launch Stack

2. 在 Specify template (指定模板) 页面上，选择 Next (下一步)。
3. 在 Specify Stack details (指定堆栈详细信息) 页面上，输入 Stack name (堆栈名称)。在 Parameters（参数）下输入其他信息。

   a. Spark UI 配置

      提供以下信息：

      - IP address range (IP 地址范围) – 可用于查看 Spark UI 的 IP 地址范围。如果要限制来自特定 IP 地址范围的访问，则应使用自定义值。

- History server port (历史记录服务器端口) – 用于 Spark UI 的端口。您可以使用默认值。
- Event log directory (事件日志目录) – 选择用于存储来自 AWS Glue 任务或开发终端节点的 Spark 事件日志的位置。您必须将 **s3a://** 用于事件日志路径模式。
- Spark package location (Spark 包位置) – 可以使用默认值。
- Keystore path (密钥库路径) – HTTPS 的 SSL/TLS 密钥库路径。如果要使用自定义密钥库文件，则可在此处指定 S3 路径 `s3://path_to_your_keystore_file`。如果将此参数保留为空，则会生成并使用基于自签名证书的密钥库。
- Keystore password（密钥库密码）– 输入 HTTPS 的 SSL/TLS 密钥库密码。

b. EC2 实例配置

提供以下信息：

- Instance type (实例类型) – 托管 Spark 历史记录服务器的 Amazon EC2 实例的类型。由于此模板在您的账户中启动 Amazon EC2 实例，因此，将向您的账户单独收取 Amazon EC2 费用。
- Latest AMI ID (最新 AMI ID) – Spark 历史记录服务器实例的 Amazon Linux 2 的 AMI ID。您可以使用默认值。
- VPC ID – Spark 历史记录服务器实例的 Virtual Private Cloud（VPC）ID。可以使用您的账户中可用的任何 VPC。建议不要将默认 VPC 用于默认网络 ACL。有关更多信息，请参阅《Amazon VPC 用户指南》中的默认 VPC 与默认子网和创建 VPC。
- Subnet ID (子网 ID) – Spark 历史记录服务器实例的 ID。您可以使用 VPC 中的任意子网。您必须能够从客户端访问子网。如果要通过 Internet 访问，则必须使用在路由表中具有 Internet 网关的公有子网。

c. 选择下一步。

4. 在 Configure stack options（配置堆栈选项）页面上，要使用当前用户凭证确定 CloudFormation 如何在堆栈中创建、修改或删除资源，请选择 Next（下一步）。您还可以在权限部分中指定要使用的角色，而不是当前用户权限，然后选择下一步。
5. 在 Review (检查) 页面上，检查模板。

   选择 I acknowledge that AWS CloudFormation might create IAM resources (我确认 Amazon CloudFormation 可能创建 IAM 资源)，然后选择 Create stack (创建堆栈)。
6. 等待创建堆栈。
7. 打开 Outputs (输出) 选项卡。

   a. 如果您使用的是公有子网，请复制 SparkUiPublicUrl 的 URL。

b. 如果您使用的是私有子网，请复制 SparkUiPrivateUrl 的 URL。

8. 打开 Web 浏览器，然后将 URL 粘贴到其中。这样一来，您便能在指定端口上使用 HTTPS 访问服务器。您的浏览器可能无法识别服务器的证书，在这种情况下，您必须重写其保护并继续进行。

使用 Docker 启动 Spark 历史记录服务器并查看 Spark UI

如果您更喜欢本地访问（不需要 Apache Spark 历史记录服务器的 EC2 实例），也可以使用 Docker 启动 Apache Spark 历史记录服务器并在本地查看 Spark UI。此 Dockerfile 是一个示例，您应修改该示例以满足您的要求。

先决条件

有关如何在笔记本电脑上安装 Docker 的信息，请参阅 Docker Engine 社区。

使用 Docker 启动 Spark 历史记录服务器并在本地查看 Spark UI

1. 从 GitHub 下载文件。

   从 AWS Glue 代码示例下载 Dockerfile 和 `pom.xml`。

2. 确定是要使用您的用户凭证还是联合身份用户凭证来访问 AWS。

   - 要使用当前用户凭证访问 AWS，请获取要用于 `docker run` 命令中的 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 的值。有关更多信息，请参阅《IAM 用户指南》中的管理 IAM 用户的访问密钥。
   - 要使用 SAML 2.0 联合身份用户访问 AWS，请获取 `AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY` 和 `AWS_SESSION_TOKEN` 的值。有关更多信息，请参阅 请求临时安全凭证。

3. 确定事件日志目录的位置，以在 `docker run` 命令中使用。

4. 使用名称 `glue/sparkui` 和标签 `latest` 以及本地目录中的文件构建 Docker 镜像。

   ```
   $ docker build -t glue/sparkui:latest .
   ```

5. 创建并开启 docker 容器。

   在以下命令中，使用先前在步骤 2 和步骤 3 中获得的值。

   a. 要使用您的用户凭证创建 docker 容器，请使用类似于以下内容的命令

```
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. 要使用临时凭证创建 docker 容器，请使用 `org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider` 作为提供程序，并提供在步骤 2 中获得的凭证值。有关更多信息，请参阅 Hadoop：与 Amazon Web Services 集成文档中的 将会话凭证与 TemporaryAWSCredentialsProvider 配合使用。

```
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**
>
> 这些配置参数来自 Hadoop-AWS模块。您可能需要根据自己的使用案例添加特定的配置。例如：隔离区域的用户需要配置 `spark.hadoop.fs.s3a.endpoint`。

6. 在浏览器中打开 `http://localhost:18080` 以在本地查看 Spark UI。

通过 AWS Glue 任务运行洞察进行监控

AWS Glue作业运行见解是一项功能AWS Glue，可简化作业的作业调试和优化。 AWS Glue AWS Glue提供 Spark 用户界面以及用于监控 AWS Glue 作业的CloudWatch 日志和指标。使用此功能，您可以获得有关 AWS Glue 任务执行的以下信息：

- 失败的 AWS Glue 任务脚本的行号。
- 在您的作业失败之前，在 Spark 查询计划中最后执行的 Spark 操作。
- 在按时间排序的日志流中显示与故障相关的 Spark 异常事件。

- 根本原因分析和解决问题的建议措施（如优化脚本）。
- 常见的 Spark 事件（与 Spark 操作有关的日志消息），其中包含解决根本原因的推荐操作。

使用AWS Glue任务日志中的两个新日志流，您可以获得所有这些见解。 CloudWatch

要求

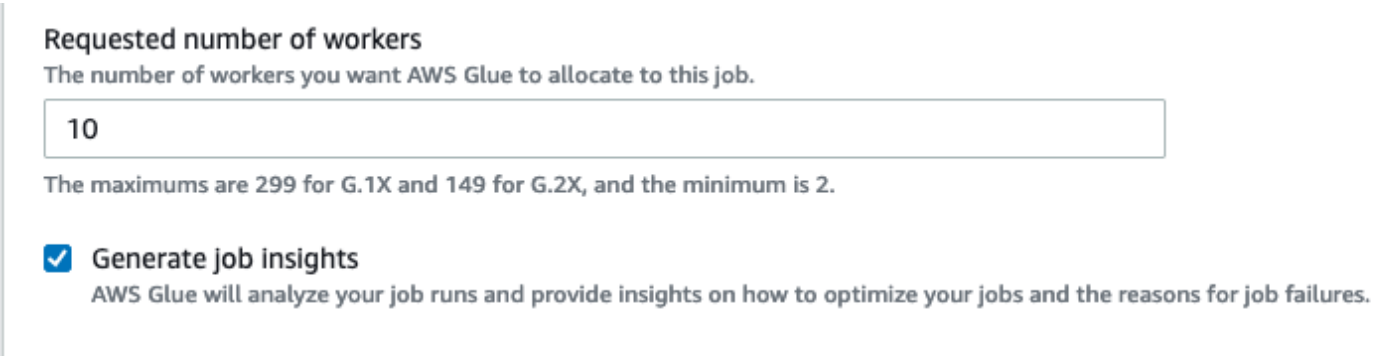
AWS Glue作业运行见解功能适用于 2.0、3.0 和 4.0 AWS Glue 版本。您可以按照现有任务的迁移指南从较早的 AWS Glue 版本进行升级。

为 AWS Glue ETL 作业启用任务运行见解

您可以通过 AWS Glue Studio 或 CLI 启用任务运行洞察。

AWS Glue Studio

通过 AWS Glue Studio 创建任务时，您可以在 Job Details（任务详细信息）选项卡下启用或禁用任务运行洞察。选中 “生成工作见解” 复选框。



命令行

如果通过 CLI 创建任务，则可以通过简单的新 job parameter（任务参数）启动任务运行：`--enable-job-insights = true`。

默认情况下，任务运行洞察日志流在 AWS Glue 连续录入使用的同一个默认日志组下进行创建，也就是 `/aws-glue/jobs/logs-v2/`。您可以使用相同的参数集来设置自定义日志组名称、日志筛选条件和日志组配置以进行连续日志录入。有关更多信息，请参阅启用 AWS Glue 任务的连续日志录入。

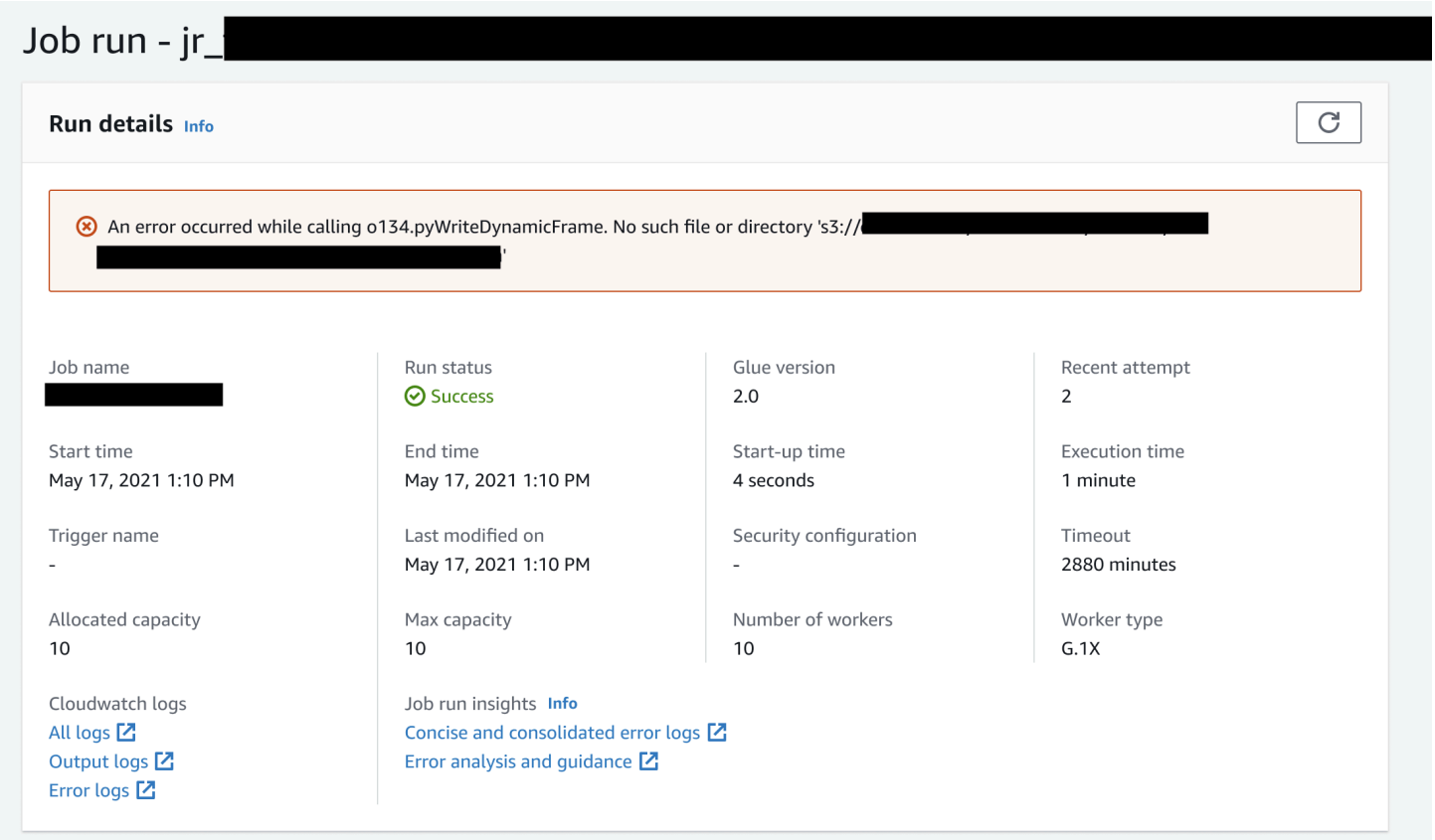
访问任务运行见解日志流 CloudWatch

启用任务运行洞察功能后，任务运行失败时可能会创建两个日志流。当任务成功完成后，两个流都不会生成。

1. 异常分析日志流：<job-run-id>-job-insights-rca-driver。该数据流提供以下功能：
   - 导致失败的 AWS Glue 任务脚本的行号。
   - Spark 查询计划 (DAG) 中最后执行的 Spark 操作。
   - 来自与异常相关的 Spark 驱动程序和执行器的简明时序事件。您可以找到详细信息，例如完整的错误消息、失败的 Spark 任务及其执行者 ID，这些信息可帮助您专注于特定执行者的日志流，以便在需要时进行更深入的调查。
2. 基于规则的洞察流式传输：
   - 根本原因分析和如何修复错误的建议（例如使用特定的任务参数来优化性能）。
   - 作为根本原因分析的基础和建议的行动的相关 Spark 事件。

> **Note**
>
> 第一个流仅在任何异常触发事件可用于失败的任务运行时存在，第二个流仅在任何洞察可用于失败的任务运行时存在。例如，如果您的任务成功完成，则不会生成任何流；如果您的任务失败，但是没有服务定义的规则可以与您的失败场景匹配，那么将只生成第一个流。

如果通过 AWS Glue Studio 创建任务，则在“任务运行详细信息”选项卡（任务运行洞察）下，还可提供指向上述流式传输的链接，即“简明和综合错误日志”和“错误分析和指导”。



AWS Glue 任务运行洞察的示例

在本节中，我们将举例说明任务运行洞察功能如何帮助您解决失败任务中的问题。在本例中，用户忘记在用来分析和构建基于其数据的机器学习模型的 AWS Glue 任务中导入所需的模块 (tensorflow)。

```
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)
```

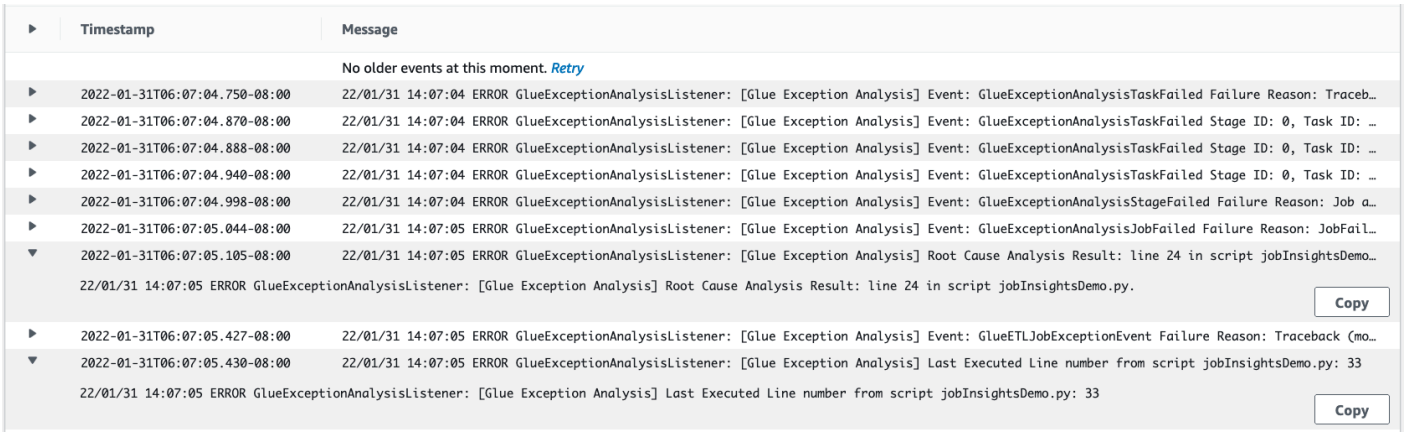
```
data_set_1 = [1, 2, 3, 4]
data_set_2 = [5, 6, 7, 8]

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())
```

如果没有任务运行洞察功能，当任务失败时，您只会看到 Spark 抛出的以下消息:

```
An error occurred while calling o111.collectToPython. Traceback (most
recent call last):
```

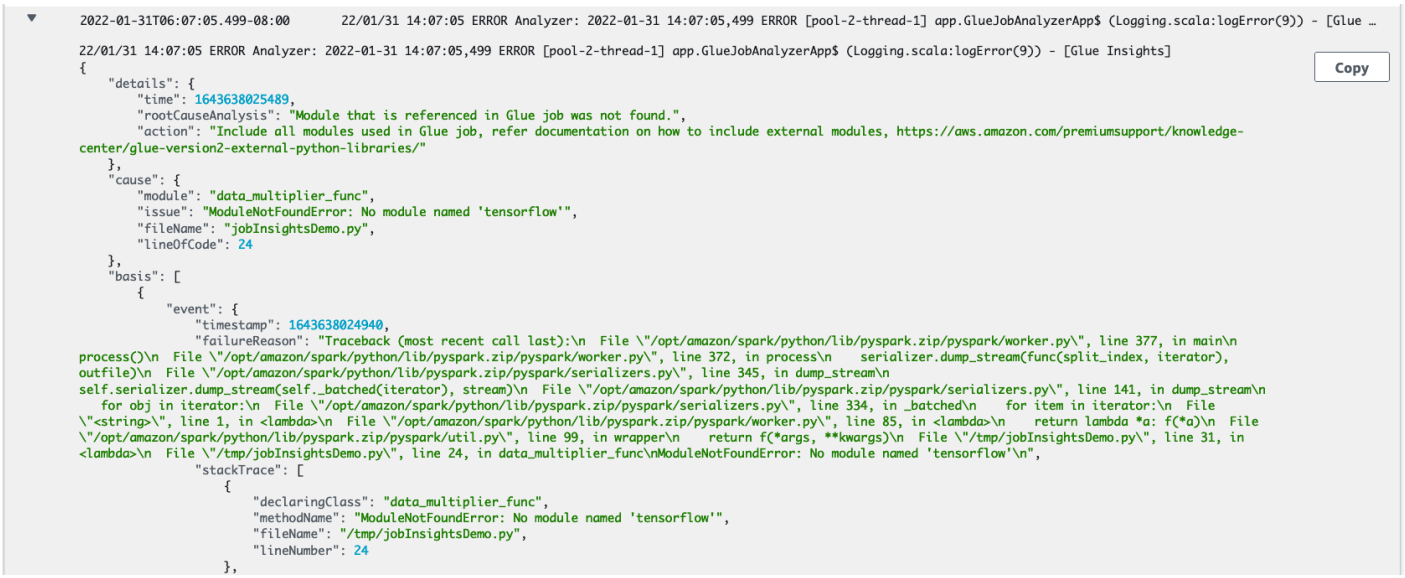
消息不明确，限制了您的调试体验。在这种情况下，此功能可通过两个 CloudWatch 日志流为您提供更多见解：

1. `job-insights-rca-driver` 日志流：
   - 异常事件：此日志流为您提供与从 Spark 驱动程序和不同分布式工作人员收集的故障相关的 Spark 异常事件。这些事件有助于您理解异常在错误代码跨 Spark 任务、执行器和分布在 AWS Glue 工作器中的阶段执行时的时间顺序传播。
   - 行号：这个日志流标识了第 21 行，我们对此进行调用，以导入导致失败的缺失 Python 模块；它还将第 24 行（对 Spark 操作 `collect()` 的调用）标识为脚本中最后执行的一行。



2. `job-insights-rule-driver` 日志流：

   - 根本原因和建议：除了脚本中错误的行号和上次执行的行号之外，此日志流还显示了根本原因分析和建议，供您遵循 AWS Glue 文档并设置必要的任务参数，以便在 AWS Glue 任务中使用额外的 Python 模块。
   - 基础事件：此日志流还显示了使用服务定义的规则评估的 Spark 异常事件，以推断根本原因并提供建议。



使用 Amazon CloudWatch 监控

您可以使用 Amazon CloudWatch 监控 AWS Glue，此工具可从 AWS Glue 收集原始数据，并将其处理为易读的近乎实时的指标。这些统计数据会保存两周，从而使您能够访问历史信息，以更好地了解您的 Web 应用程序或服务的执行情况。默认情况下，AWS Glue 指标数据自动发送到 CloudWatch。有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的什么是 Amazon CloudWatch？和AWS Glue 指标。

持续日志记录

AWS Glue 还支持对 AWS Glue 作业的实时连续日志记录。为任务启用连续日志记录之后，您可在 AWS Glue 控制台或 CloudWatch 控制台控制面板上查看实时日志。有关更多信息，请参阅 AWS Glue 任务的连续日志记录。

可观测性指标

启用作业可观测性指标后，作业运行时将会生成额外的 Amazon CloudWatch 指标。使用 AWS Glue 可观测性指标可深入了解 AWS Glue 内部发生的情况，从而可以改进对问题的分类和分析。

主题

- 使用 Amazon CloudWatch 指标监控 AWS Glue
- 在 AWS Glue 作业配置文件上设置 Amazon CloudWatch 警报
- AWS Glue 任务的连续日志记录
- 使用 AWS Glue 可观测性指标进行监控

使用 Amazon CloudWatch 指标监控 AWS Glue

您可以使用 AWS Glue 作业分析器分析和监控 AWS Glue 操作。它从 AWS Glue 收集原始数据，并将其处理为 Amazon CloudWatch 中存储的易读且近乎实时的指标。这些统计数据将保留并聚合在 CloudWatch 中，以便您可以访问历史信息，更好地了解您的应用程序的运行状况。

> **Note**
>
> 启用任务指标并且创建 CloudWatch 自定义指标时，可能需要支付额外费用。有关更多信息，请参阅 Amazon CloudWatch 定价。

AWS Glue 指标概述

当您与 AWS Glue 交互时，它将指标发送到 CloudWatch。您可以使用 AWS Glue 控制台（首选方法）、CloudWatch 控制台控制面板或 AWS Command Line Interface（AWS CLI）查看这些指标。

使用 AWS Glue 控制台控制面板查看指标

您可以查看作业的指标的摘要或详细图表，或作业运行的详细图表。

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。

2. 在导航窗格中，选择作业运行监控。
3. 在作业运行中，选择操作以停止当前正在运行的作业、查看作业或倒回作业书签。
4. 选择一个作业，然后选择查看运行详细信息以查看有关该作业运行的其他信息。

使用 CloudWatch 控制台控制面板查看指标

指标的分组首先依据服务命名空间，然后依据每个命名空间内的各种维度组合。

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。
3. 选择 Glue 命名空间。

使用 AWS CLI 查看指标

- 在命令提示符处，使用以下命令。

```
aws cloudwatch list-metrics --namespace Glue
```

AWS Glue 每隔 30 秒向 CloudWatch 报告一次指标，CloudWatch 指标控制面板配置为每分钟显示一次指标。AWS Glue 指标表示先前报告的值的增量值。在适当时，指标控制面板会聚合（合计）30 秒值以获取整个最后一分钟的值。

Spark 作业的 AWS Glue 指标行为

在脚本中初始化 `GlueContext` 时启用 AWS Glue 指标，并且一般仅在 Apache Spark 任务结束时更新这些指标。它们表示迄今为止所有已完成的 Spark 任务的聚合值。

但是，AWS Glue 传递给 CloudWatch 的 Spark 指标通常表示在报告它们时的当前状态的绝对值。AWS Glue 每隔 30 秒向 CloudWatch 报告一次指标，并且指标控制面板通常会显示在最后 1 分钟收到的数据点的平均值。

AWS Glue 指标名称均采用以下类型的前缀之一：

- `glue.driver.` – 其名称以此前缀开头的指标表示从 Spark 驱动程序上的所有执行程序聚合的 AWS Glue 指标，或对应于 Spark 驱动程序的 Spark 指标。
- `glue.executorId.` – executorId 是特定 Spark 执行程序的编号。它与日志中列出的执行程序相对应。

- `glue.ALL.` – 其名称以此前缀开头的指标聚合来自所有 Spark 执行程序的值。

AWS Glue 指标

AWS Glue 会每隔 30 秒配置以下指标并将其发送到 CloudWatch，AWS Glue 指标控制面板每分钟报告一次：

指标	描述
`glue.driver.aggregate.bytesRead`	所有执行程序中运行的所有已完成的 Spark 任务从所有数据源读取的字节数。 有效维度：`JobName`（AWS Glue 任务的名称）、`JobRunId`（JobRun ID 或 `ALL`）和 `Type`（计数）。 有效统计数据：SUM 此指标是上次报告值的增量值，因此，在 AWS Glue 指标控制面板上，SUM 统计数据用于聚合。 单位：字节 可用于监控： • 读取的字节数。 • 任务进度。 • JDBC 数据源。 • 任务书签问题。 • 任务运行之间的差异。 此指标可以按 `glue.ALL.s3.filesystem.read_bytes` 指标的方式使用，不同之处在于此指标在 Spark 任务结束时更新并捕获非 S3 数据源。
`glue.driver.aggregate.elapsedTime`	ETL 运行时间（以毫秒为单位，不包括任务的引导启动时间）。

指标	描述
	有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（计数）。 有效统计数据：SUM 此指标是上次报告值的增量值，因此，在 AWS Glue 指标控制面板上，SUM 统计数据用于聚合。 单位：毫秒 可用于确定任务运行的平均时长。 数据的一些使用方式： • 为落后者设置警报。 • 衡量任务运行之间的差异。

指标	描述
glue.driver.aggregate.numCompletedStages	任务中已完成的阶段数量。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（计数）。 有效统计数据：SUM 此指标是上次报告值的增量值，因此，在 AWS Glue 指标控制面板上，SUM 统计数据用于聚合。 单位：计数 可用于监控： • 任务进度。 • 每个阶段的任务执行时间线（与其他指标相关时）。 数据的一些使用方式： • 识别任务执行过程中要求苛刻的阶段。 • 为任务运行之间的关联峰值（要求苛刻的阶段）设置警报。

指标	描述
glue.driver.aggregate.numCompletedTasks	任务中已完成的任务数量。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（计数）。 有效统计数据：SUM 此指标是上次报告值的增量值，因此，在 AWS Glue 指标控制面板上，SUM 统计数据用于聚合。 单位：计数 可用于监控： • 任务进度。 • 某个阶段内的并行度。
glue.driver.aggregate.numFailedTasks	失败的任务数。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（计数）。 有效统计数据：SUM 此指标是上次报告值的增量值，因此，在 AWS Glue 指标控制面板上，SUM 统计数据用于聚合。 单位：计数 可用于监控： • 导致作业任务失败的数据异常。 • 导致作业任务失败的集群异常。 • 导致作业任务失败的脚本异常。 这些数据可用于为增加的故障设置警报，这些故障可能表明数据、集群或脚本出现异常。

指标	描述
glue.driver.aggregate.numKilledTasks	已终止的任务数量。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（计数）。 有效统计数据：SUM 此指标是上次报告值的增量值，因此，在 AWS Glue 指标控制面板上，SUM 统计数据用于聚合。 单位：计数 可用于监控： • 导致终止任务的异常（OOM）的数据偏斜异常。 • 导致终止任务的异常（OOM）的脚本异常。 数据的一些使用方式： • 为指示数据异常的增加故障设置警报。 • 为指示集群异常的增加故障设置警报。 • 为指示脚本异常的增加故障设置警报。

指标	描述
glue.driver.aggregate.recordsRead	所有执行程序中运行的所有已完成的 Spark 任务从所有数据源读取的记录数。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（计数）。 有效统计数据：SUM 此指标是上次报告值的增量值，因此，在 AWS Glue 指标控制面板上，SUM 统计数据用于聚合。 单位：计数 可用于监控： • 读取的记录数。 • 任务进度。 • JDBC 数据源。 • 任务书签问题。 • 几天内的任务运行偏斜。 此指标可以按 glue.ALL.s3.filesystem.read_bytes 指标的方式使用，不同之处在于此指标在 Spark 任务结束时更新。

指标	描述
glue.driver.aggregate.shuffleBytesWritten	自上次报告以来所有执行程序为在它们之间对数据进行随机排序而写入的字节数（由 AWS Glue 指标控制面板聚合为前一分钟内为此目的写入的字节数）。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（计数）。 有效统计数据：SUM 此指标是上次报告值的增量值，因此，在 AWS Glue 指标控制面板上，SUM 统计数据用于聚合。 单位：字节 可用于监控：任务（大型联接、分组依据、重新分区、合并）中的数据随机排序。 数据的一些使用方式： • 在进一步处理之前，对大型输入文件重新分区或解压缩。 • 更均匀地对数据重新分区以避免热键。 • 在联接或分组依据操作之前预筛选数据。

指标	描述
glue.driver.aggregate.shuffleLocalBytesRead	自上次报告以来所有执行程序为在它们之间对数据进行随机排序而读取的字节数（由 AWS Glue 指标控制面板聚合为前一分钟内为此目的读取的字节数）。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（计数）。 有效统计数据：SUM 此指标是上次报告值的增量值，因此，在 AWS Glue 指标控制面板上，SUM 统计数据用于聚合。 单位：字节 可用于监控：任务（大型联接、分组依据、重新分区、合并）中的数据随机排序。 数据的一些使用方式： • 在进一步处理之前，对大型输入文件重新分区或解压缩。 • 使用热键更均匀地对数据重新分区。 • 在联接或分组依据操作之前预筛选数据。

指标	描述
glue.driver.BlockManager.disk.diskSpaceUsed_MB	所有执行程序中所用磁盘空间的兆字节数。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（量规）。 有效统计数据：平均值。这是 Spark 指标，报告为绝对值。 单位：兆字节 可用于监控： • 用于表示缓存 RDD 分区的数据块所用的磁盘空间。 • 用于表示中间随机排序输出的数据块所用的磁盘空间。 • 用于表示直播的数据块所用的磁盘空间。 数据的一些使用方式： • 识别因磁盘使用率增加而导致的任务故障。 • 识别导致溢出或随机排序的大型分区。 • 增加预置 DPU 容量以纠正这些问题。

指标	描述
glue.driver.ExecutorAllocationManager.executors.numberAllExecutors	主动运行的执行程序的数量。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（量规）。 有效统计数据：平均值。这是 Spark 指标，报告为绝对值。 单位：计数 可用于监控： • 任务活动。 • 落后的执行程序（只有几个执行程序在运行） • 当前执行程序级并行度。 数据的一些使用方式： • 如果集群未充分利用，请首先对大型输入文件重新分区或解压缩。 • 识别因落后场景而导致的阶段或任务执行延迟。 • • 与 numberMaxNeededExecutors 进行比较，了解用于预置更多 DPU 的积压。

指标	描述
glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors	为满足当前负载所需的最大（主动运行和待处理）任务执行程序的数量。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（量规）。 有效统计数据：最大值 这是 Spark 指标，报告为绝对值。 单位：计数 可用于监控： • 任务活动。 • 因 DPU 容量导致执行程序不可用或执行程序终止/失败，尚未安排的待处理任务的当前执行级并行度和积压。 数据的一些使用方式： • 识别计划队列的待处理/积压。 • 识别因落后场景而导致的阶段或任务执行延迟。 • 与 numberAllExecutors 进行比较，了解用于预置更多 DPU 的积压。 • 增加预置的 DPU 容量，更正待处理的执行程序积压。

指标	描述
glue.driver.jvm.heap.usage glue.executorId.jvm.heap.usage glue.ALL.jvm.heap.usage	驱动程序的 JVM 堆用于此驱动程序的内存量（比例：0-1），executorId 标识的执行程序，或所有执行程序。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（量规）。 有效统计数据：平均值。这是 Spark 指标，报告为绝对值。 单位：百分比 可用于监控： • 使用 glue.driver.jvm.heap.usage 的驱动程序内存不足状况（OOM）。 • 使用 glue.ALL.jvm.heap.usage 的执行程序内存不足状况（OOM）。 数据的一些使用方式： • 识别占用内存的执行程序 ID 和阶段。 • 识别落后执行程序 ID 和阶段。 • 识别驱动程序内存不足状况（OOM）。 • 识别执行程序内存不足状况（OOM）并获取相应的执行程序 ID，以便能够从执行程序日志中获取堆栈追踪。 • 识别可能有数据偏差导致落后程序或内存不足状况（OOM）的文件或分区。

指标	描述
glue.driver.jvm.heap.used glue.executorId.jvm.heap.used glue.ALL.jvm.heap.used	驱动程序的 JVM 堆所用的内存字节数，executorId 表示的执行程序或所有执行程序。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（量规）。 有效统计数据：平均值。这是 Spark 指标，报告为绝对值。 单位：字节 可用于监控： • 驱动程序内存不足状况（OOM）。 • 执行程序内存不足状况（OOM）。 数据的一些使用方式： • 识别占用内存的执行程序 ID 和阶段。 • 识别落后执行程序 ID 和阶段。 • 识别驱动程序内存不足状况（OOM）。 • 识别执行程序内存不足状况（OOM）并获取相应的执行程序 ID，以便能够从执行程序日志中获取堆栈追踪。 • 识别可能有数据偏差导致落后程序或内存不足状况（OOM）的文件或分区。

指标	描述
glue.driver.s3.filesystem.read_bytes glue.executorId.s3.filesystem.read_bytes glue.ALL.s3.filesystem.read_bytes	自上次报告以来，驱动程序、executorId 标识的执行程序、所有执行程序从 Amazon S3 读取的字节数（由 AWS Glue 指标控制面板聚合为上一分钟内读取的字节数）。 有效维度：JobName、JobRunId 和 Type（量规）。 有效统计数据：SUM 此指标是上次报告值的增量值，因此，在 AWS Glue 指标控制面板上，SUM 统计数据用于聚合。AWS Glue 指标控制面板上曲线下面的区域可用于直观比较两个不同任务运行读取的字节。 单位：字节。 可用于监控： • ETL 数据移动。 • 任务进度。 • 任务书签问题（数据已处理、已重新处理和已跳过）。 • 外部数据源的读取和摄入速率比较。 • 任务运行之间的差异。 生成的数据可用于： • DPU 容量规划。 • 为任务运行和任务阶段的数据读取中的大型峰值或低谷设置警报。

指标	描述
glue.driver.s3.filesystem.write_bytes glue.executorId.s3.filesystem.write_bytes glue.ALL.s3.filesystem.write_bytes	自上次报告以来，驱动程序、executorId 标识的执行程序、所有执行程序从 Amazon S3 写入的字节数（由 AWS Glue 指标控制面板聚合为上一分钟内写入的字节数）。 有效维度：JobName、JobRunId 和 Type（量规）。 有效统计数据：SUM 此指标是上次报告值的增量值，因此，在 AWS Glue 指标控制面板上，SUM 统计数据用于聚合。AWS Glue 指标控制面板上曲线下面的区域可用于直观比较两个不同任务运行写入的字节。 单位：字节 可用于监控： • ETL 数据移动。 • 任务进度。 • 任务书签问题（数据已处理、已重新处理和已跳过）。 • 外部数据源的读取和摄入速率比较。 • 任务运行之间的差异。 数据的一些使用方式： • DPU 容量规划。 • 为任务运行和任务阶段的数据读取中的大型峰值或低谷设置警报。

指标	描述
glue.driver.streaming.numRecords	微批处理中接收的记录数。此指标仅适用于 AWS Glue 流式传输任务（采用 AWS Glue 2.0 版及更高版本）。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（计数）。 有效统计数据：总计、最大值、最小值、平均值、百分比 单位：计数 可用于监控： • 读取的记录数。 • 任务进度。
glue.driver.streaming.batchProcessingTimeInMs	处理批处理所需的时间（以毫秒为单位）。此指标仅适用于 AWS Glue 流式传输任务（采用 AWS Glue 2.0 版及更高版本）。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（计数）。 有效统计数据：总计、最大值、最小值、平均值、百分比 单位：计数 可用于监控： • 任务进度。 • 脚本性能。

指标	描述
glue.driver.system.cpuSystemLoad glue.executorId.system.cpuSystemLoad glue.ALL.system.cpuSystemLoad	驱动程序使用的 CPU 系统负载量（比例：0-1），executorId 标识的执行程序，或所有执行程序。 有效维度：JobName（AWS Glue 任务的名称）、JobRunId（JobRun ID 或 ALL）和 Type（量规）。 有效统计数据：平均值。此指标报告为绝对值。 单位：百分比 可用于监控： • 驱动程序 CPU 负载。 • 执行程序 CPU 负载。 • 检测任务中的 CPU 绑定或 IO 绑定的执行程序或阶段。 数据的一些使用方式： • DPU 容量与 IO 指标（字节读取/随机排序字节、任务并行度）和所需最大执行程序数量指标一起规划。 • 识别 CPU/IO 绑定比率。此操作允许使用 CPU 利用率较低的可拆分数据集，对长时间运行的任务进行重新分区并增加预置容量。

AWS Glue 指标的维度

AWS Glue 指标使用 AWS Glue 命名空间并提供以下维度的指标。

维度	描述
JobName	此维度筛选特定 AWS Glue 任务的所有任务运行的指标。
JobRunId	此维度按 JobRun ID 或 ALL 筛选特定 AWS Glue 任务的所有任务运行的指标。
Type	此维度按 count（总数）或 gauge（在某个时间点的值）筛选指标。

有关更多信息，请参阅 Amazon CloudWatch 用户指南。

在 AWS Glue 作业配置文件上设置 Amazon CloudWatch 警报

AWS Glue 指标还可用于 Amazon CloudWatch。您可以对已计划作业的任何 AWS Glue 指标设置警报。

设置警报的一些常见方案如下所示：

- 作业内存不足 (OOM)：在内存使用量超过 AWS Glue 作业的驱动程序或执行程序的正常平均值时设置警报。
- 落后的执行程序：在 AWS Glue 作业中执行程序的数量在较长一段时间内低于特定阈值时设置警报。
- 数据积压或再处理：使用 CloudWatch 数学表达式比较工作流中单个任务的指标。然后，您可以对生成的表达式值（如作业写入的字节比率和以下作业读取的字节比率）触发警报。

有关设置警报的详细说明，请参阅《Amazon CloudWatch Events 用户指南》中的创建或编辑 CloudWatch 警报。

有关使用 CloudWatch 监控和调试场景的信息，请参阅作业监控和调试。

AWS Glue 任务的连续日志记录

AWS Glue 提供对 AWS Glue 任务进行实时的连续日志记录。您可在 Amazon CloudWatch 中查看实时 Apache Spark 任务日志，包括驱动程序日志、执行程序日志和 Apache Spark 任务进度栏。查看实时日志可让您更好地了解正在运行的任务。

当您启动 AWS Glue 任务时，它会在 Spark 应用程序开始运行之后向 CloudWatch 发送实时日志记录信息（在每次执行程序终止之前，每 5 秒发送一次）。您可在 AWS Glue 控制台或 CloudWatch 控制台控制面板上查看日志。

连续日志记录功能包括以下特性：

- 连续日志记录
- 自定义脚本日志记录程序，用于记录特定于应用程序的消息
- 控制台进度栏，用于跟踪当前 AWS Glue 任务的运行状态

有关如何在 AWS Glue 2.0 版中支持持续日志记录的信息，请参阅运行 Spark ETL 任务，减少启动时间。

您可以将对 CloudWatch 日志组或 IAM 角色的流的访问限制为读取日志。有关限制访问的更多详细信息，请参阅 CloudWatch 文档中的 将基于身份的策略（IAM policy）用于 CloudWatch Logs。

> **Note**
>
> 启用连续日志记录并创建 CloudWatch 日志事件时，可能需要支付额外费用。有关更多信息，请参阅 Amazon CloudWatch 定价。

主题

为 AWS Glue 作业启用连续日志记录

您可使用 AWS Glue 控制台或通过 AWS Command Line Interface (AWS CLI) 启用连续日志记录。

您可以在创建新作业或编辑现有作业时启用连续日志记录，也可通过 AWS CLI 启用此功能。

您还可以指定自定义配置选项，例如 Amazon CloudWatch 日志组名称、AWS Glue 任务运行 ID 驱动程序/执行程序 ID 之前的 CloudWatch 日志流前缀，以及日志消息的日志转换模式。这些配置可帮助您在具有不同到期策略的自定义 CloudWatch 日志组中设置聚合日志，并使用自定义日志流前缀和转换模式对它们进行进一步分析。

主题

- 使用 AWS Management Console
- 使用自定义脚本日志记录程序记录特定于应用程序的消息
- 启用进度栏以显示作业进度
- 采用连续日志记录的安全配置

使用 AWS Management Console

在创建或编辑 AWS Glue 作业时，按照以下步骤使用控制台来启用连续日志记录。

创建带有连续日志记录的新 AWS Glue 作业

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格中，选择 ETL 作业。
3. 选择 Visual ETL。
4. 在作业详细信息选项卡中，展开高级属性部分。
5. 在持续日志记录下，选择启用 CloudWatch 中的日志。

为现有 AWS Glue 作业启用连续日志记录

1. 通过 https://console.aws.amazon.com/glue/ 打开 AWS Glue 控制台。
2. 在导航窗格中，选择作业。
3. 从作业列表选择现有作业。
4. 选择操作和编辑作业。
5. 在作业详细信息选项卡中，展开高级属性部分。
6. 在持续日志记录下，选择启用 CloudWatch 中的日志。

使用 AWS CLI

要启用连续日志记录，您需要将作业参数传入 AWS Glue 作业。通过类似于其他 AWS Glue 作业参数的方法，传递以下特殊作业参数。有关更多信息，请参阅 AWS Glue 作业参数。

```
'--enable-continuous-cloudwatch-log': 'true'
```

您可以指定自定义 Amazon CloudWatch 日志组名称。如果未指定，则默认日志组名称为 /aws-glue/jobs/logs-v2/。

```
'--continuous-log-logGroup': 'custom_log_group_name'
```

您可以指定自定义 Amazon CloudWatch 日志流前缀。如果未指定，则默认日志流前缀为作业运行 ID。

```
'--continuous-log-logStreamPrefix': 'custom_log_stream_prefix'
```

您可以指定自定义连续日志记录转换模式。如果未指定，则默认转换模式为 %d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n。请注意，转换模式仅适用于驱动程序日志和执行程序日志。它不会影响 AWS Glue 进度条。

```
'--continuous-log-conversionPattern': 'custom_log_conversion_pattern'
```

使用自定义脚本日志记录程序记录特定于应用程序的消息

您可以使用 AWS Glue 日志记录程序记录在脚本中记录任何特定于应用程序的消息，这些消息实时发送到驱动程序日志流。

以下示例显示了一个 Python 脚本。

```
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")
```

以下示例显示了一个 Scala 脚本。

```
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")
  }
}
```

启用进度栏以显示作业进度

AWS Glue 在 JOB_RUN_ID-progress-bar 日志流下提供了实时进度栏，用于查看 AWS Glue 作业运行状态。目前，它仅支持初始化 glueContext 的作业。如果您运行纯 Spark 作业而没有初始化 glueContext，则不显示 AWS Glue 进度栏。

进度栏每 5 秒显示下列进度更新。

```
Stage Number (Stage Name): > (numCompletedTasks + numActiveTasks) /
 totalNumOfTasksInThisStage]
```

采用连续日志记录的安全配置

如果为 CloudWatch 日志启用了安全配置，AWS Glue 将为连续日志创建一个名称如下的日志组：

```
<Log-Group-Name>-<Security-Configuration-Name>
```

默认日志组和自定义日志组将如下所示：

- 默认连续日志组将为 /aws-glue/jobs/logs-v2-*<Security-Configuration-Name>*
- 自定义连续日志组将为 *<custom-log-group-name>*-*<Security-Configuration-Name>*

您需要将 logs:AssociateKmsKey 添加到您的 IAM 角色权限（如果您使用 CloudWatch Logs 启用了安全配置）。如果不包括该权限，连续日志记录将禁用。此外，要为 CloudWatch Logs 配置加密，请遵循《Amazon CloudWatch Logs 用户指南》中的使用 AWS Key Management Service 加密 CloudWatch Logs 中日志数据处的说明。

有关创建安全配置的更多信息，请参阅在 AWS Glue 控制台上处理安全配置。

查看 AWS Glue 作业的连续日志记录

您可以使用 AWS Glue 控制台或 Amazon CloudWatch 控制台查看实时日志。

使用 AWS Glue 控制台控制面板查看实时日志

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格中，选择作业。
3. 添加或启动现有作业。选择操作、运行作业。

   在您开始运行某个作业时，系统会将您导航到包含该运行作业相关信息的页面：

   - 日志显示之前的已聚合应用程序日志。
   - 连续日志记录选项卡在运行的作业初始化了 `glueContext` 时，显示一个实时进度栏。
   - 连续日志记录选项卡还包含驱动程序日志，这会捕获实时 Apache Spark 驱动程序日志，以及在作业运行时使用 AWS Glue 应用程序日志记录程序记录的脚本中的应用程序日志。
4. 对于较早的任务，您还可以选择 Logs (日志)，在 Job History (任务历史记录) 视图下面查看实时日志。此操作会将您转到显示所有 Spark 进度栏、执行程序及该任务运行的进度栏日志流的 CloudWatch 控制台。

使用 CloudWatch 控制台控制面板查看实时日志

1. 通过 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格选择日志。
3. 选择 /aws-glue/jobs/logs-v2/ 日志组。
4. 在筛选条件框中，粘贴作业运行 ID。

   您可以查看驱动程序日志、执行程序日志和进度栏（如果使用 标准筛选条件）。

使用 AWS Glue 可观测性指标进行监控

> **Note**
>
> AWS Glue 可观测性指标在 AWS Glue 4.0 及更高版本中可用。

使用 AWS Glue 可观测性指标可深入了解 AWS Glue 内部发生的情况，以便 Apache Spark 作业可以改进对问题的分类和分析。可观测性指标通过 Amazon CloudWatch 控制面板进行可视化显示，有助于

分析错误的根本原因以及诊断性能瓶颈。您可以缩短大规模问题调试所需的时间，从而专注于更快、更有效地解决问题。

AWS Glue 可观测性提供了以下四组 Amazon CloudWatch 指标：

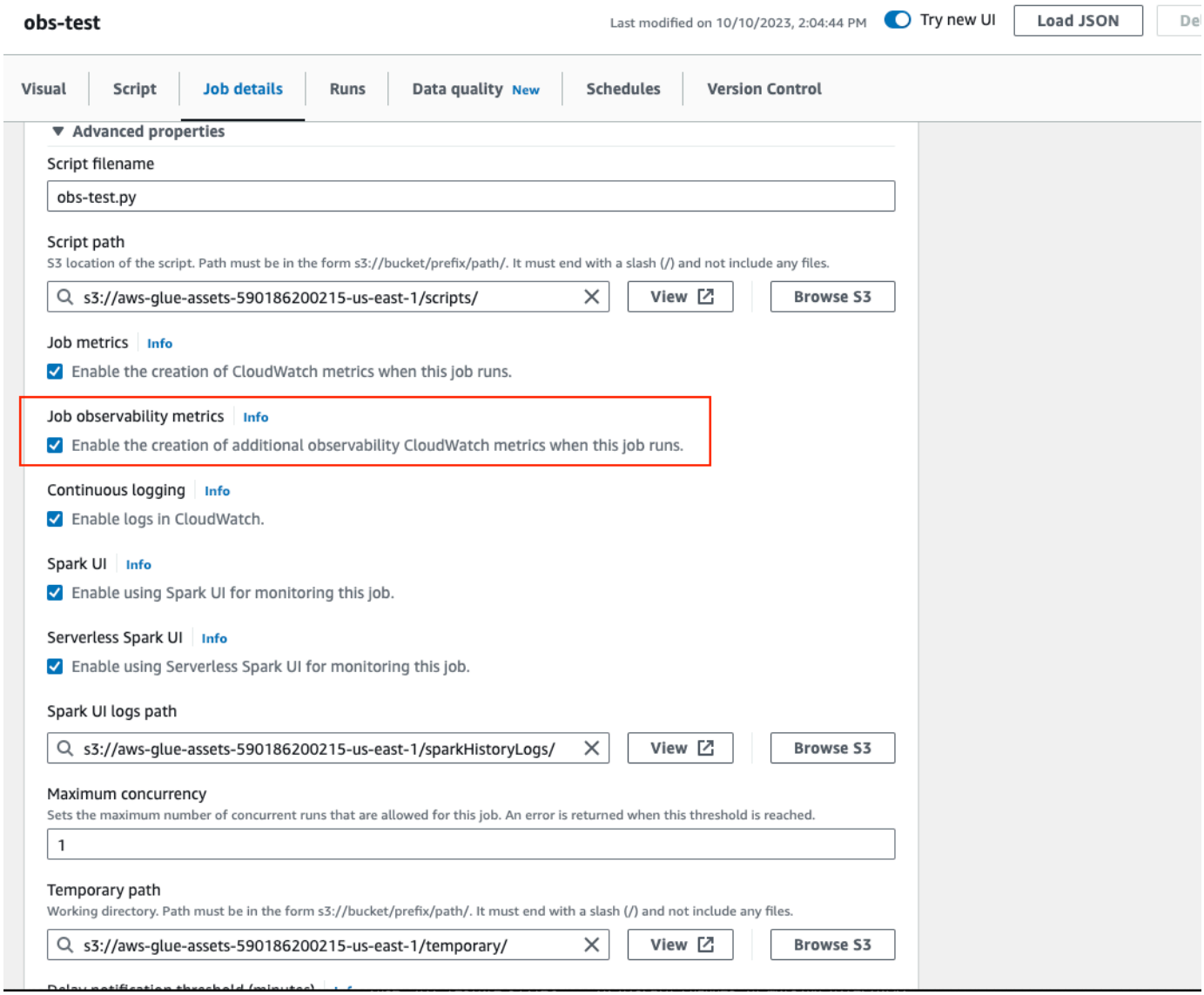
- 可靠性（即错误类别）– 可轻松确定给定时间范围内可能需要解决的最常见故障原因。
- 性能（即偏斜）– 确定性能瓶颈并应用优化技术。例如，性能因作业偏斜而下降时，可能需要启用 Spark 自适应查询执行并微调优化偏斜联接阈值。
- 吞吐量（即每源/接收器吞吐量）– 监控数据读取和写入的趋势。您还可以针对异常配置 Amazon CloudWatch 警报。
- 资源利用率（即 Worker 线程、内存和磁盘利用率）– 高效地查找容量利用率低的作业。您可能需要为这些作业启用 AWS Glue 自动扩缩。

AWS Glue 可观测性指标入门

> **Note**
>
> 默认情况下，新指标在 AWS Glue Studio 控制台中已启用。

在 AWS Glue Studio 中配置可观测性指标：

1. 登录 AWS Glue 控制台并从控制台菜单中选择 ETL 作业。
2. 单击您的作业部分中的作业名称，从而选择作业。
3. 选择 Job details（任务详细信息）选项卡。
4. 滚动到底部并选择高级属性，然后选择作业可观测性指标。



通过 AWS CLI 启用 AWS Glue 可观测性指标：

- 将输入 JSON 文件中的以下键值添加到 `--default-arguments` 映射中：

```
--enable-observability-metrics, true
```

使用 AWS Glue 可观测性

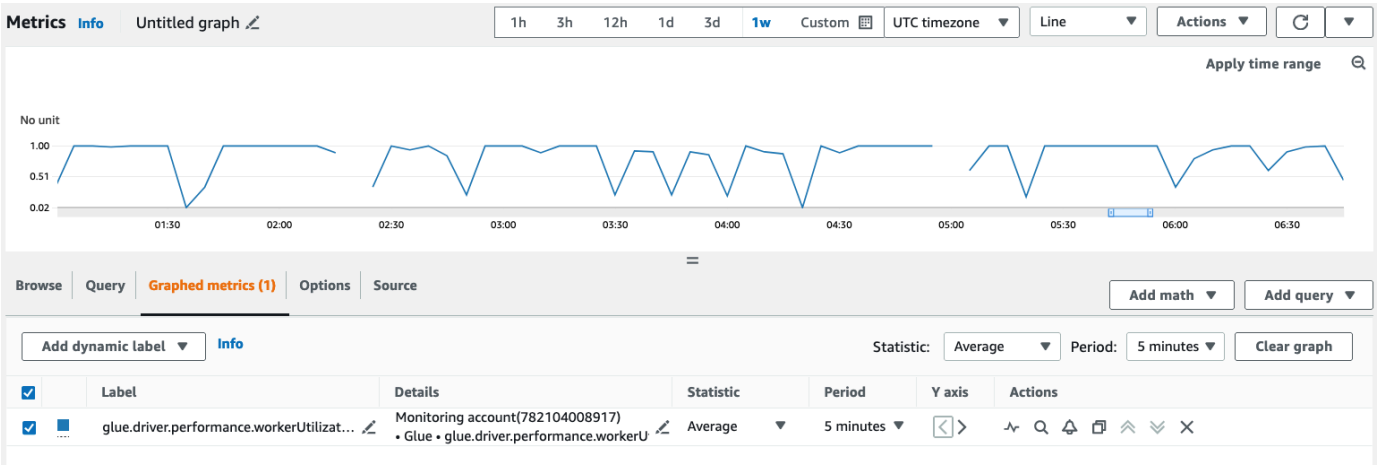
由于 AWS Glue 可观测性指标是通过 Amazon CloudWatch 提供的，因此您可以通过 Amazon CloudWatch 控制台、AWS CLI、SDK 或 API 来查询可观测性指标数据点。有关何时使用 AWS Glue

可观测性指标的示例用例，请参阅 Using Glue Observability for monitoring resource utilization to reduce cost。

在 Amazon CloudWatch 控制台中使用 AWS Glue 可观测性

在 Amazon CloudWatch 控制台中查询和可视化显示指标：

1. 打开 Amazon CloudWatch 控制台，然后选择 所有指标。
2. 在“自定义命名空间”下，选择 AWS Glue。
3. 选择作业可观测性指标、每源可观测性指标或每接收器可观测性指标。
4. 搜索特定的指标名称、作业名称、作业运行 ID，然后将其选中。
5. 在图形化指标选项卡下，配置您的首选统计数据、周期和其他选项。



使用 AWS CLI 查询可观测性指标：

1. 创建指标定义 JSON 文件并将 `your-Glue-job-name` 和 `your-Glue-job-run-id` 替换为您的值。

```
$ 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. 运行 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": [
                0.0671875000000001
            ],
            "StatusCode": "Complete"
        },
        {
            "Id": "avgWorkerUtil_1",
            "Label": "<your-label-for-B>",
            "Timestamps": [
                "2023-10-28T18:50:00+00:00"
            ],
            "Values": [
                0.5959183673469387
            ],
            "StatusCode": "Complete"
        }
    ],
```

```
    "Messages": []
}
```

可观测性指标

AWS Glue 可观测性每隔 30 秒分析以下指标并将其发送到 Amazon CloudWatch，其中一些指标可以在 AWS Glue Studio 作业运行监测页面中看到。

指标	描述	类别
glue.driver.skewness.stage	指标类别：job_performance Spark 分阶段执行偏度：此指标捕获执行偏度，这可能是由输入数据偏度或转换（例如偏斜联接）引起的。该指标值的范围是 [0, 无穷大]，其中 0 表示该阶段中所有任务中最长任务执行时间与中值任务执行时间的比率，小于某个阶段偏度系数。默认的阶段偏度系数为“5”，可通过 spark conf 进行覆盖：spark.metrics.conf.driver.source.glue.jobPerformance.skewnessFactor 阶段偏度值为 1 表示该比率是阶段偏度系数的两倍。 阶段偏度值每 30 秒更新一次，以反映当前偏度。阶段结束时的值反映了最终阶段偏度。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和	job_performance

指标	描述	类别
	ObservabilityGroup（job_performance） 有效统计信息：平均值、最大值、最小值、百分比 单位：计数	
glue.driver.skewness.job	指标类别：job_performance 作业偏度是作业阶段偏度的加权平均值。加权平均值为执行时间更长的阶段提供了更多的权重。这是为了避免极端的情况，即一个非常偏斜的阶段相对于其他阶段实际上运行的时间很短（因此其偏度对整体作业性能并不重要，也没必要费力解决其偏度）。 该指标在每个阶段完成后更新，因此最后一个值反映了实际的整体作业偏度。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（job_performance） 有效统计信息：平均值、最大值、最小值、百分比 单位：计数	job_performance

指标	描述	类别
glue.succeed.ALL	指标类别：错误 成功运行的作业总数，以全面了解失败类别 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（计数）和 ObservabilityGroup（错误） 有效统计信息：SUM 单位：计数	error
glue.error.ALL	指标类别：错误 作业运行错误总数，以全面了解失败类别 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（计数）和 ObservabilityGroup（错误） 有效统计信息：SUM 单位：计数	error

指标	描述	类别
glue.error.[错误类别]	指标类别：错误 这实际上是一组指标，只有在作业运行失败时才会更新。错误分类有助于分类和调试。作业运行失败时，会对导致失败的错误进行分类，并将相应的错误类别指标设置为 1。这有助于执行一段时间内的故障分析，以及对所有作业进行错误分析，以确定最常见的失败类别并开始解决这些问题。AWS Glue 有 28 个错误类别，包括 OUT_OF_MEMORY（驱动程序和执行程序）、PERMISSION、SYNTAX 和 THROTTLING 错误类别。错误类别还包括 COMPILATION、LAUNCH 和 TIMEOUT 错误类别。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（计数）和 ObservabilityGroup（错误） 有效统计信息：SUM 单位：计数	error

指标	描述	类别
glue.driver.workerUtilization	指标类别：resource_utilization 实际使用的已分配工作人员的百分比。如果效果不佳，自动扩缩功能可以提供帮助。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值、最大值、最小值、百分比 单位：百分比	resource_utilization

指标	描述	类别
glue.driver.memory.heap.[available | used]	指标类别：resource_utilization 作业运行期间，驱动程序的可用/已用堆内存。这有助于了解内存使用趋势，尤其是随时间推移而来的内存使用趋势，从而能帮助避免潜在的故障，此外还可以调试与内存相关的故障。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：字节	resource_utilization

指标	描述	类别
glue.driver.memory.heap.used.percentage	指标类别：resource_utilization 作业运行期间，驱动程序的已用堆内存（%）。这有助于了解内存使用趋势，尤其是随时间推移而来的内存使用趋势，从而能帮助避免潜在的故障，此外还可以调试与内存相关的故障。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：百分比	resource_utilization

指标	描述	类别
glue.driver.memory.non-heap.[available | used]	指标类别：resource_utilization 作业运行期间，驱动程序的可用/已用非堆内存。这有助于了解内存使用趋势，尤其是随时间推移而来的内存使用趋势，这有助于避免潜在的故障，此外还可以调试与内存相关的故障。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：字节	resource_utilization

指标	描述	类别
glue.driver.memory.non-heap.used.percentage	指标类别：resource_utilization 作业运行期间，驱动程序的已用非堆内存（%）。这有助于了解内存使用趋势，尤其是随时间推移而来的内存使用趋势，从而能帮助避免潜在的故障，此外还可以调试与内存相关的故障。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：百分比	resource_utilization

指标	描述	类别
glue.driver.memory.total.[available | used]	指标类别：resource_utilization 作业运行期间，驱动程序的可用/已用总内存。这有助于了解内存使用趋势，尤其是随时间推移而来的内存使用趋势，从而能帮助避免潜在的故障，此外还可以调试与内存相关的故障。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：字节	resource_utilization

指标	描述	类别
glue.driver.memory.total.used.percentage	指标类别：resource_utilization 作业运行期间，驱动程序的已用总内存（%）。这有助于了解内存使用趋势，尤其是随时间推移而来的内存使用趋势，从而能帮助避免潜在的故障，此外还可以调试与内存相关的故障。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：百分比	resource_utilization
glue.ALL.memory.heap.[available | used]	指标类别：resource_utilization 执行程序的可用/已用堆内存。ALL 表示所有执行程序。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：字节	resource_utilization

指标	描述	类别
glue.ALL.memory.heap.used.percentage	指标类别：resource_utilization 执行程序的已用堆内存（%）。ALL 表示所有执行程序。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：百分比	resource_utilization
glue.ALL.memory.non-heap.[available | used]	指标类别：resource_utilization 执行程序的可用/已用非堆内存。ALL 表示所有执行程序。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：字节	resource_utilization

指标	描述	类别
glue.ALL.memory.non-heap.used.percentage	指标类别：resource_utilization 执行程序的已用非堆内存（%）。ALL 表示所有执行程序。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：百分比	resource_utilization
glue.ALL.memory.total.[available | used]	指标类别：resource_utilization 执行程序的可用/已用总内存。ALL 表示所有执行程序。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：字节	resource_utilization

指标	描述	类别
glue.ALL.memory.total.used.percentage	指标类别：resource_utilization 执行程序的已用总内存（%）。ALL 表示所有执行程序。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：百分比	resource_utilization
glue.driver.disk.[available_GB | used_GB]	指标类别：resource_utilization 作业运行期间，驱动程序的可用/已用磁盘空间。这有助于了解磁盘使用趋势，尤其是随时间推移而来的内存使用趋势，从而能帮助避免潜在的故障，此外还可以调试与充足磁盘空间相关的故障。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：千兆字节	resource_utilization

指标	描述	类别
glue.driver.disk.used.percentage]	指标类别：resource_utilization 作业运行期间，驱动程序的可用/已用磁盘空间。这有助于了解磁盘使用趋势，尤其是随时间推移而来的内存使用趋势，从而能帮助避免潜在的故障，此外还可以调试与充足磁盘空间相关的故障。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：百分比	resource_utilization
glue.ALL.disk.[available_GB | used_GB]	指标类别：resource_utilization 执行程序的可用/已用磁盘空间。ALL 表示所有执行程序。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：千兆字节	resource_utilization

指标	描述	类别
glue.ALL.disk.used.percentage	指标类别：resource_utilization 执行程序的可用/已用/已用（%）磁盘空间。ALL 表示所有执行程序。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）和 ObservabilityGroup（resource_utilization） 有效统计信息：平均值 单位：百分比	resource_utilization
glue.driver.bytesRead	指标类别：吞吐量 此作业运行中每个输入源以及所有源读取的字节数。这有助于了解数据量及其随着时间的推移而发生的变化，从而帮助解决诸如数据偏斜之类的问题。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）、ObservabilityGroup（resource_utilization）和 Source（源数据位置） 有效统计信息：平均值 单位：字节	吞吐量

指标	描述	类别
glue.driver.[recordsRead | filesRead]	指标类别：吞吐量 此作业运行中每个输入源以及所有源读取的记录/文件数。这有助于了解数据量及其随着时间的推移而发生的变化，从而帮助解决诸如数据偏斜之类的问题。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）、ObservabilityGroup（resource_utilization）和 Source（源数据位置） 有效统计信息：平均值 单位：计数	吞吐量

指标	描述	类别
glue.driver.partitionsRead	指标类别：吞吐量 此作业运行中每个 Amazon S3 输入源以及所有源读取的分区数。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）、ObservabilityGroup（resource_utilization）和 Source（源数据位置） 有效统计信息：平均值 单位：计数	吞吐量

指标	描述	类别
glue.driver.bytesWrittten	指标类别：吞吐量 此作业运行中每个输出接收器以及所有接收器写入的字节数。这有助于了解数据量及其随着时间的推移而发生的变化，从而帮助解决诸如处理偏斜之类的问题。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）、ObservabilityGroup（resource_utilization）和 Sink（接收器数据位置） 有效统计信息：平均值 单位：字节	吞吐量

指标	描述	类别
glue.driver.[recordsWritten | filesWritten]	指标类别：吞吐量 此作业运行中每个输出接收器以及所有接收器写入的记录/文件数。这有助于了解数据量及其随着时间的推移而发生的变化，从而帮助解决诸如处理偏斜之类的问题。 有效维度：JobName（AWS Glue 作业名称）、JobRunId（JobRun ID 或 ALL）、Type（量规）、ObservabilityGroup（resource_utilization）和 Sink（接收器数据位置） 有效统计信息：平均值 单位：计数	吞吐量

错误类别

错误类别	描述
COMPILATION_ERROR	编译 Scala 代码的过程中会出现错误。
CONNECTION_ERROR	连接到服务/远程主机/数据库服务等过程中会出现错误。
DISK_NO_SPACE_ERROR	当驱动程序/执行程序的磁盘中没有剩余空间时，就会出现错误。
OUT_OF_MEMORY_ERROR	当驱动程序/执行程序的内存中没有剩余空间时，就会出现错误。

错误类别	描述
IMPORT_ERROR	导入依赖项时会出现错误。
INVALID_ARGUMENT_ERROR	如果输入参数无效/非法，则会出现错误。
PERMISSION_ERROR	当缺乏服务、数据等权限时，就会出现错误。
RESOURCE_NOT_FOUND_ERROR	当数据、位置等无法退出时，就会出现错误。
QUERY_ERROR	执行 Spark SQL 查询时会出现错误。
SYNTAX_ERROR	当脚本中存在语法错误时，就会出现错误。
THROTTLING_ERROR	达到服务并发限制或超过服务限额限制时会出现错误。
DATA_LAKE_FRAMEWORK_ERROR	Hudi、Iceberg 等 AWS Glue 原生支持的数据湖框架出现错误。
UNSUPPORTED_OPERATION_ERROR	进行不支持的操作时会出现错误。
RESOURCES_ALREADY_EXISTS_ERROR	创建或添加的资源已存在时会出现错误。
GLUE_INTERNAL_SERVICE_ERROR	当出现 AWS Glue 内部服务问题时，就会出现错误。
GLUE_OPERATION_TIMEOUT_ERROR	AWS Glue 操作超时时会出现错误。
GLUE_VALIDATION_ERROR	当无法验证 AWS Glue 作业的所需值时，就会出现错误。
GLUE_JOB_BOOKMARK_VERSION_MISMATCH_ERROR	当同一作业外显子相同的源存储桶并同时写入相同/不同的目标时会出现错误（并发度 > 1）
LAUNCH_ERROR	AWS Glue 作业启动阶段会出现错误。
DYNAMODB_ERROR	由 Amazon DynamoDB 服务引起的一般错误。
GLUE_ERROR	由 AWS Glue 服务引起的一般错误。

错误类别	描述
LAKEFORMATION_ERROR	由 AWS Lake Formation 服务引起的一般错误。
REDSHIFT_ERROR	由 Amazon Redshift 服务引起的一般错误。
S3_ERROR	由 Amazon S3 服务引起的一般错误。
SYSTEM_EXIT_ERROR	通用系统退出错误。
TIMEOUT_ERROR	当作业因操作超时而失败时，就会出现一般错误。
UNCLASSIFIED_SPARK_ERROR	由 Spark 引起的一般错误。
UNCLASSIFIED_ERROR	默认错误类别。

限制

> **Note**
>
> `glueContext` 必须初始化才能发布指标。

在源维度中，该值可以是 Amazon S3 路径或表名，具体视源类型而定。此外，如果源为 JDBC 并且使用了查询选项，则会在源维度中设置查询字符串。如果该值超过 500 个字符，则将其修剪为 500 个字符以内。以下是该值的限制：

- 将删除非 ASCII 字符。
- 如果源名称不包含任何 ASCII 字符，则会将其转换为 <非 ASCII 输入>。

吞吐量指标限制和注意事项

- 支持 DataFrame 和基于 DataFrame 的 DynamicFrame（例如 JDBC，在 Amazon S3 上从 parquet 读取），但不支持基于 RDD 的 DynamicFrame（例如在 Amazon S3 上读取 csv、json 等）。从技术上讲，支持 Spark UI 上显示的所有读取和写入。
- 如果数据来源为目录表且格式为 JSON、CSV、文本或 Iceberg，将发出 `recordsRead` 指标。

- JDBC 和 Iceberg 表中没有 `glue.driver.throughput.recordsWritten`、`glue.driver.throughput.bytesWritten` 和 `glue.driver.throughput.filesWritten` 指标。
- 指标可能会延迟。如果作业在大约一分钟后完成，则 Amazon CloudWatch 指标中可能没有吞吐量指标。

作业监控和调试

您可以收集有关 AWS Glue 任务的指标，并在 AWS Glue 和 Amazon CloudWatch 控制台上显示它们，以确定并修复问题。分析 AWS Glue 作业需要执行以下步骤：

1. 启用指标：
   a. 在作业定义中启用 Job metrics (作业指标) 选项。您可以在 AWS Glue 控制台中启用分析，也可以作为作业的参数。有关更多信息，请参阅定义 Spark 作业的作业属性或AWS Glue 作业参数。
   b. 在作业定义中启用 AWS Glue 可观测性指标选项。您可以在 AWS Glue 控制台中启用可观测性，也可以作为作业的参数。有关更多信息，请参阅使用 AWS Glue 可观测性指标进行监控。
2. 确认作业脚本初始化 `GlueContext`。例如，以下脚本代码段初始化 `GlueContext` 并显示在脚本中放置已分析代码的位置。此常规格式用于后续的调试方案。

```
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. 运行作业。
4. 可视化指标：
   a. 在 AWS Glue 控制台上显示作业指标，并确定驱动程序或执行程序的异常指标。
   b. 在作业运行监测页面、作业运行详细信息页面或 Amazon CloudWatch 上查看可观测性指标。有关更多信息，请参阅 使用 AWS Glue 可观测性指标进行监控。
5. 使用已确定的指标缩小根本原因范围。
6. 也可以选择使用已确定驱动程序或作业执行程序的日志流确认根本原因。

AWS Glue 可观测性指标的用例

- 调试 OOM 异常和作业异常
- 调试要求苛刻的阶段和落后任务
- 监控多个作业的进度
- 监控 DPU 容量规划
- 使用 AWS Glue 可观测性监测资源利用率以降低成本

调试 OOM 异常和作业异常

您可以在 AWS Glue 中调试内存不足 (OOM) 异常和作业异常。以下部分介绍用于调试 Apache Spark 驱动程序或 Spark 执行程序的内存不足异常的方案。

- 调试驱动程序 OOM 异常
- 调试执行程序 OOM 异常

调试驱动程序 OOM 异常

在这种情况下，Spark 任务从 Amazon Simple Storage Service（Amazon S3）中读取大量小文件。它会将这些文件转换为 Apache Parquet 格式，然后将其写出到 Amazon S3。Spark 驱动程序内存不足。输入 Amazon S3 数据在各 Amazon S3 分区中的文件数量已超过 100 万个。

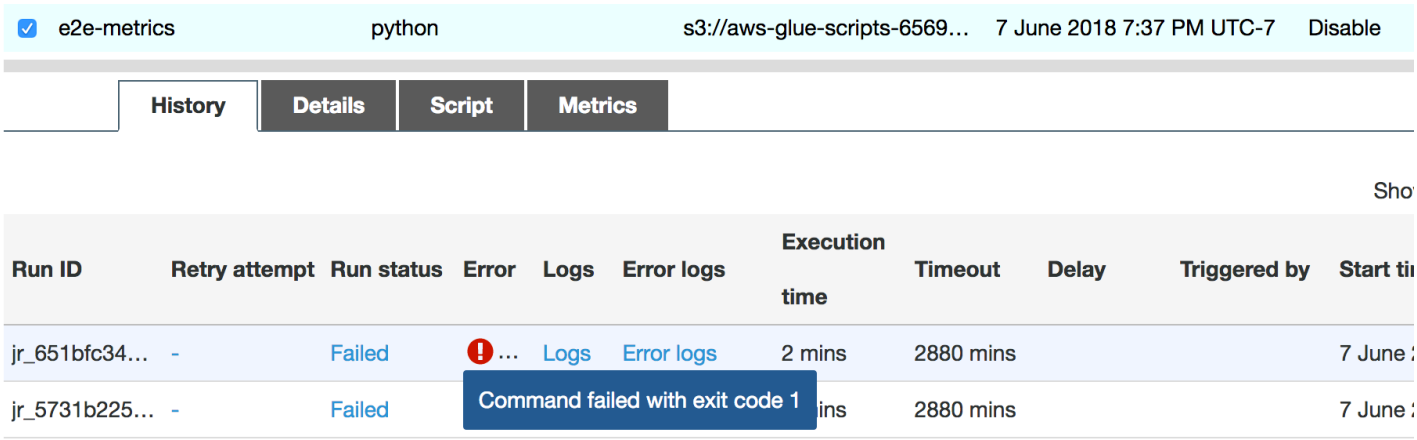
配置的代码如下所示：

```
data = spark.read.format("json").option("inferSchema", False).load("s3://input_path")
data.write.format("parquet").save(output_path)
```

在 AWS Glue 控制台上可视化分析指标

下图显示了以驱动程序和执行程序的百分比表示的内存使用率。此使用率绘制为一个数据点，它是上一分钟报告的值的平均数。您可以在作业的内存配置文件中看到驱动程序内存快速超过 50% 使用率的安全阈值。另一方面，所有执行程序的平均内存使用率仍低于 4%。这清楚地显示了此 Spark 作业中驱动程序执行的异常。

该作业运行很快失败，并在 控制台上的历史记录AWS Glue选项卡中出现以下错误：Command Failed with Exit Code 1 (命令失败，退出代码 1)。此错误字符串表示任务由于系统错误而失败 – 在此示例中，该错误为驱动程序内存不足。



在控制台上，选择 History (历史记录) 选项卡上的 Error logs (错误日志) 链接以确认 CloudWatch Logs 中有关驱动程序 OOM 的查找结果。在作业的错误日志中搜索“**`Error`**”以确认确实是 OOM 异常使作业失败：

```
# java.lang.OutOfMemoryError: Java heap space
# -XX:OnOutOfMemoryError="kill -9 %p"
# Executing /bin/sh -c "kill -9 12039"...
```

在作业的历史记录选项卡上，选择日志。您可以在任务开始时在 CloudWatch Logs 中找到驱动程序执行的以下跟踪。Spark 驱动程序尝试列出所有目录中的所有文件，构造 `InMemoryFileIndex`，并为每个文件启动一个任务。这进而会导致 Spark 驱动程序必须在内存中维持大量状态以跟踪所有任务。它会缓存内存索引的大量文件的完整列表，从而导致驱动程序 OOM。

使用分组修复多个文件的处理

您可以通过使用 . 中的分组AWS Glue 功能修复多个文件的处理。使用动态帧时以及输入数据集包含大量文件（超过 50,000 个）时，将自动启用分组功能。分组使您可以将多个文件合并到一个组中，并且允许任务处理整个组而非单个文件。因此，Spark 驱动程序在内存中存储显著减少的状态以跟踪较少的任务。有关手动为数据集启用分组的更多信息，请参阅以较大的组读取输入文件。

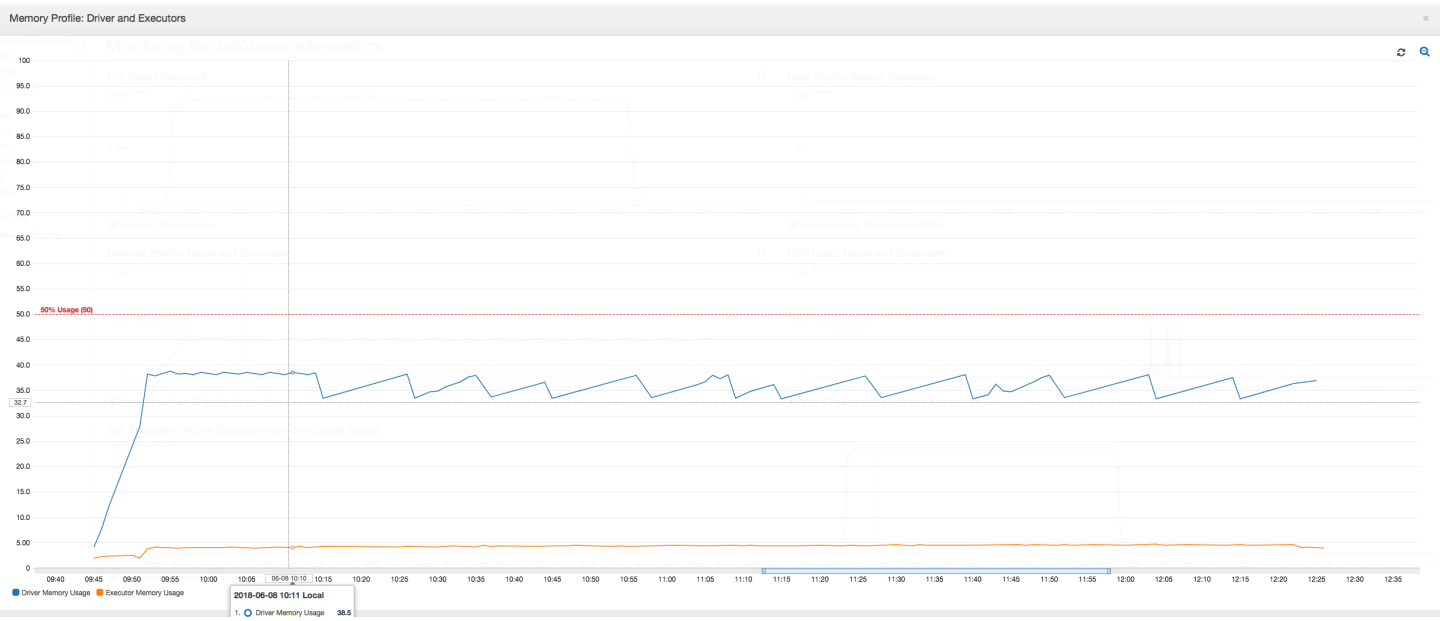
要检查 AWS Glue 作业的内存配置文件，请在启用分组的情况下配置以下代码：

```
df = glueContext.create_dynamic_frame_from_options("s3", {'paths': ["s3://input_path"],
 "recurse":True, 'groupFiles': 'inPartition'}, format="json")
```

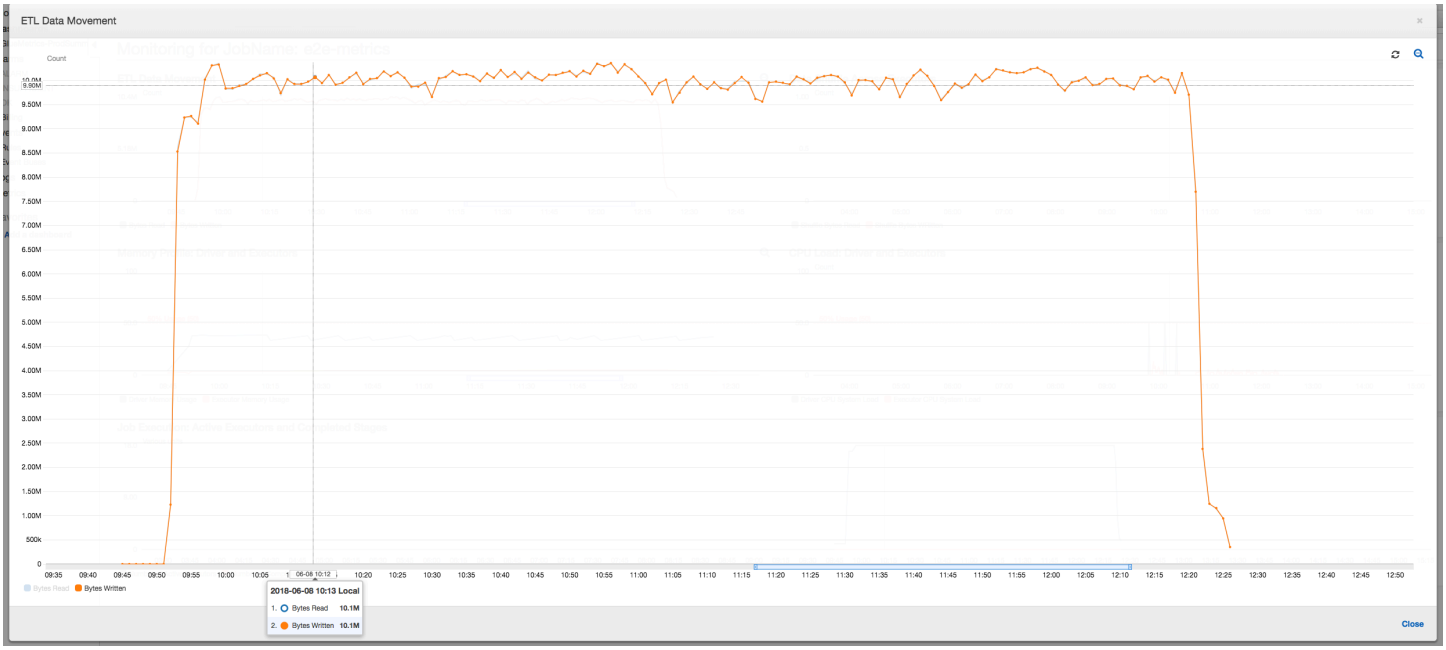
```
datasink = glueContext.write_dynamic_frame.from_options(frame = df, connection_type
 = "s3", connection_options = {"path": output_path}, format = "parquet",
 transformation_ctx = "datasink")
```

您可以在 AWS Glue 作业配置文件中监控内存配置文件和 ETL 数据移动。

在 AWS Glue 任务的整个持续时间内，驱动程序在低于 50% 内存使用率的阈值下运行。执行程序从 Amazon S3 流式传输数据，对其进行处理，然后将其写出到 Amazon S3。因此，它们在任何时间点消耗的内存不到 5%。



下面的数据移动配置文件显示了任务进行时所有执行程序在最后一分钟内读取和写入的 Amazon S3 字节总数。两者当数据在所有执行程序中流式传输时都遵循类似的模式。该作业在不到三个小时内完成处理所有一百万个文件。



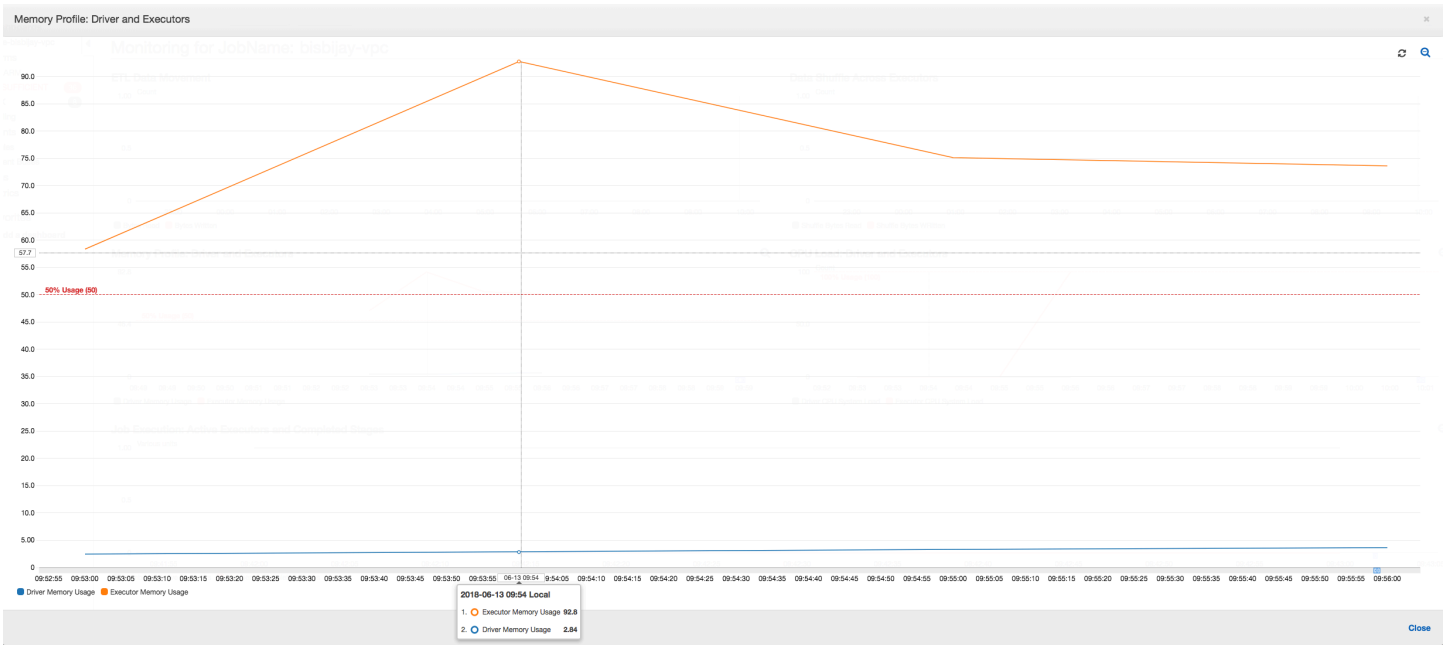
调试执行程序 OOM 异常

在此方案中，您可以了解如何调试 Apache Spark 执行程序中可能出现的 OOM 异常。以下代码使用 Spark 的 MySQL 读取器将大约 3400 万行的一个大型表读入 Spark 数据帧中。然后它以 Parquet 格式将其写出到 Amazon S3。您可以提供连接属性，并使用默认 Spark 配置来读取表。

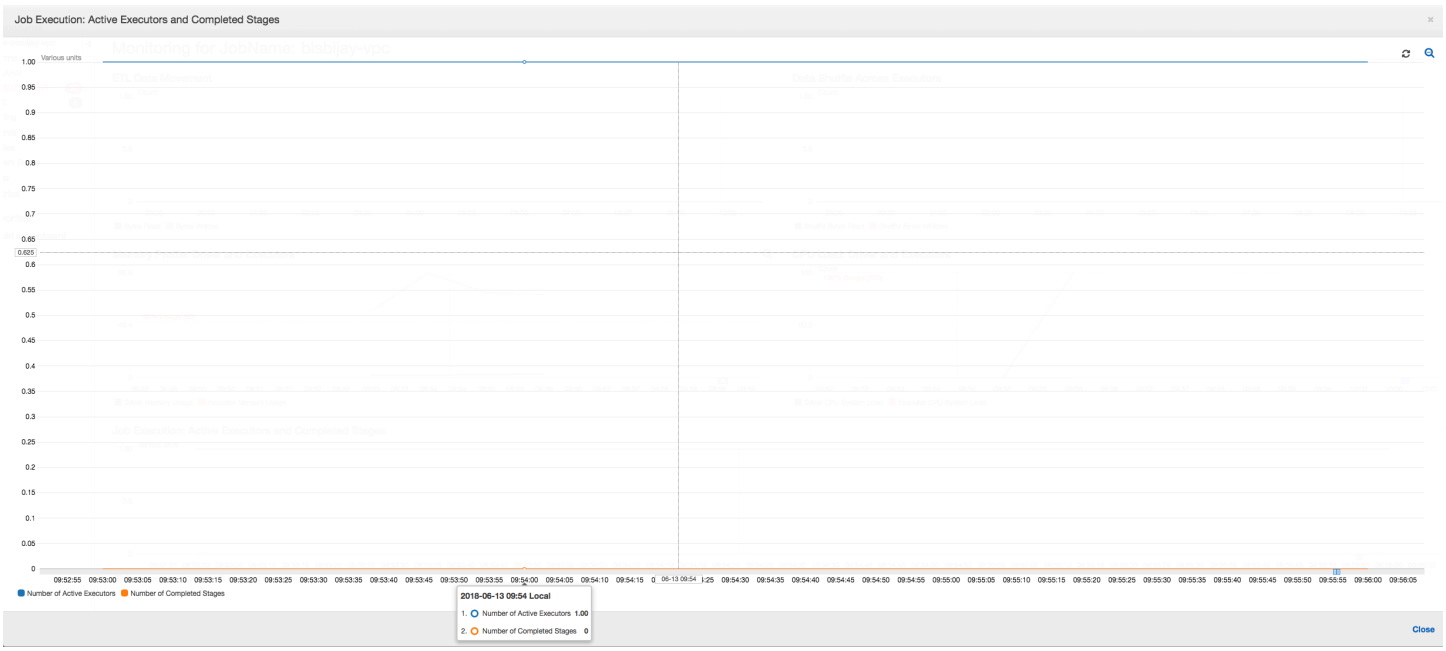
```
val connectionProperties = new Properties()
connectionProperties.put("user", user)
connectionProperties.put("password", password)
connectionProperties.put("Driver", "com.mysql.jdbc.Driver")
val sparkSession = glueContext.sparkSession
val dfSpark = sparkSession.read.jdbc(url, tableName, connectionProperties)
dfSpark.write.format("parquet").save(output_path)
```

在 AWS Glue 控制台上可视化分析指标

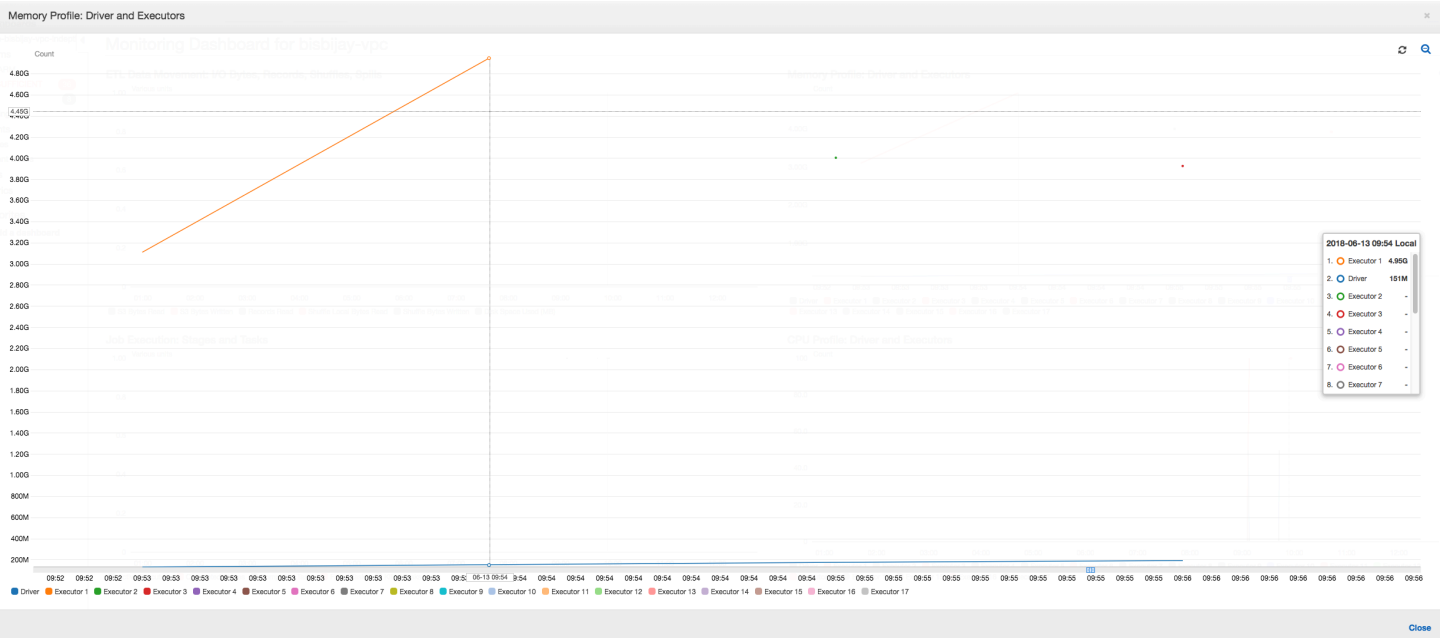
如果内存使用情况图的斜率为正且超过 50%，则如果作业在发出下一个衡量指标之前失败，那么内存耗尽是一个很好的候选原因。下图显示了在执行的一分钟内，所有执行程序的平均内存使用率快速超过 50%。使用率高达 92%，而且运行执行程序的容器被 Apache Hadoop YARN 停止。



如下图所示，在作业失败之前始终会有单个执行程序在运行。这是因为启动了新的执行程序以替换停止的执行程序。默认情况下，不会并行读取 JDBC 数据源，因为它需要对表中的列进行分区并打开多个连接。因此，只有一个执行程序按顺序读入整个表。



如下图所示，Spark 尝试在作业失败之前四次启动新任务。您可以看到三个执行程序的内存配置文件。每个执行程序都会快速耗尽其所有内存。第四个执行程序内存不足，作业失败。因此，不会立即报告其指标。



您可以从 AWS Glue 控制台上的错误字符串确认作业因 OOM 异常而失败，如下图所示。

任务输出日志：要进一步确认执行程序 OOM 异常这一发现，请查看 CloudWatch Logs。当您搜索 **`Error`** 时，您会发现四个执行程序在指标控制面板上显示的大致相同的时间段内被停止。所有执行程序均被 YARN 终止，因为它们超出其内存限制。

执行程序 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.
```

执行程序 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.
```

执行程序 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.
```

执行程序 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.
```

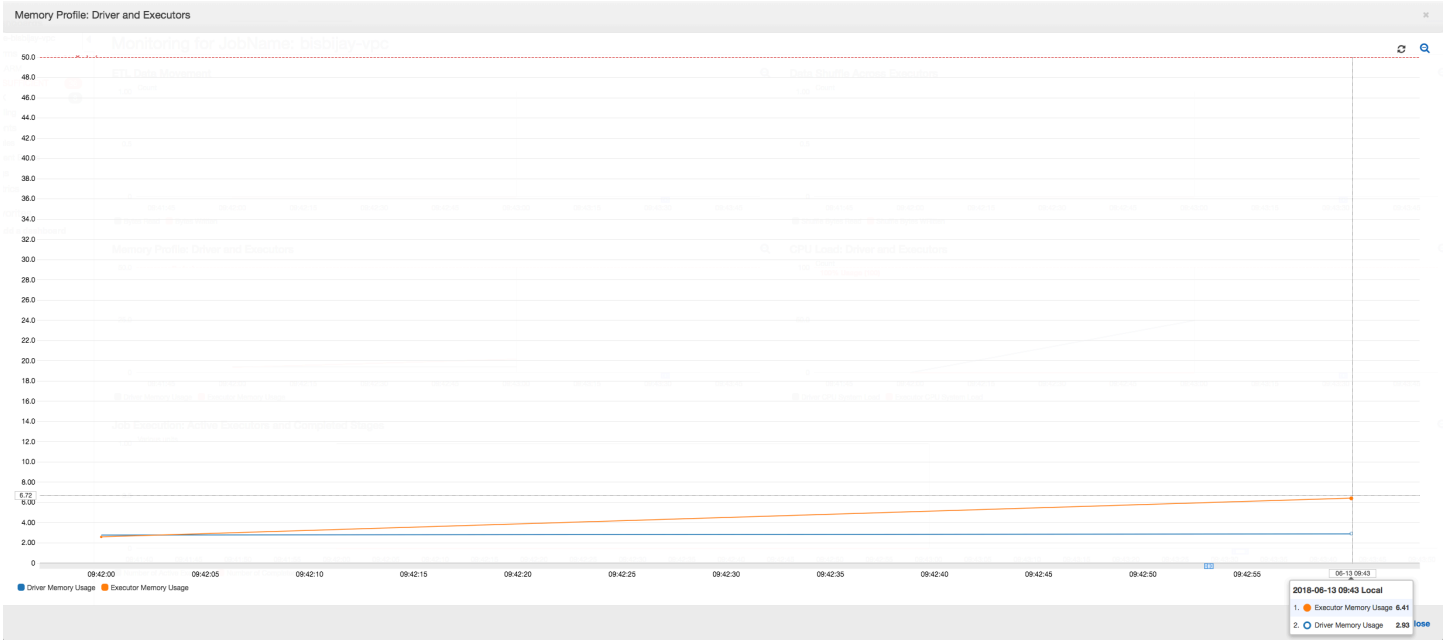
使用 AWS Glue 动态帧修复提取大小设置

执行程序在读取 JDBC 表时内存不足，因为 Spark JDBC 提取大小的默认配置为零。这意味着 Spark 执行程序上的 JDBC 驱动程序尝试从数据库中一起提取 3400 万行并对其进行缓存，即使 Spark 一次流式传输一行。使用 Spark，您可以通过将提取大小参数设置为非零默认值来避免此情况。

您还可以通过使用 AWS Glue 动态帧修复此问题。默认情况下，动态帧使用 1000 行的提取大小，这通常是一个足够的值。因此，执行程序不会占用其总内存的 7% 以上。AWS Glue 作业只需单个执行程序即可在不到两分钟内完成。虽然建议的方法是使用 AWS Glue 动态帧，但也可以使用 Apache Spark `fetchsize` 属性设置提取大小。请参阅 Spark SQL、DataFrame 和 Dataset 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")
```

正常的分析指标：具有 动态帧的执行程序内存AWS Glue永远不会超过安全阈值，如下图所示。它在数据库的行中流式传输数据，并且在任何时间点在 JDBC 驱动程序中仅缓存 1,000 行。不会发生内存不足异常。



调试要求苛刻的阶段和落后任务

您可以使用 AWS Glue 作业分析来确定提取、转换和加载 (ETL) 作业中要求苛刻的阶段和落后任务。在 AWS Glue 作业的某个阶段，落后任务比其余任务需要更长的时间。因此，该阶段需要更长的时间才能完成，这也会延迟作业的总执行时间。

将小型输入文件合并为更大的输出文件

当不同任务中的工作分配不均衡时，可能会出现落后任务，或者数据倾斜会导致一个任务处理更多数据。

您可以分析以下代码 – Apache Spark 中的一种常见模式 – 以将大量小文件合并为更大的输出文件。在本示例中，输入数据集为 32 GB 的 JSON Gzip 压缩文件。输出数据集包含大约 190 GB 的未压缩 JSON 文件。

配置的代码如下所示：

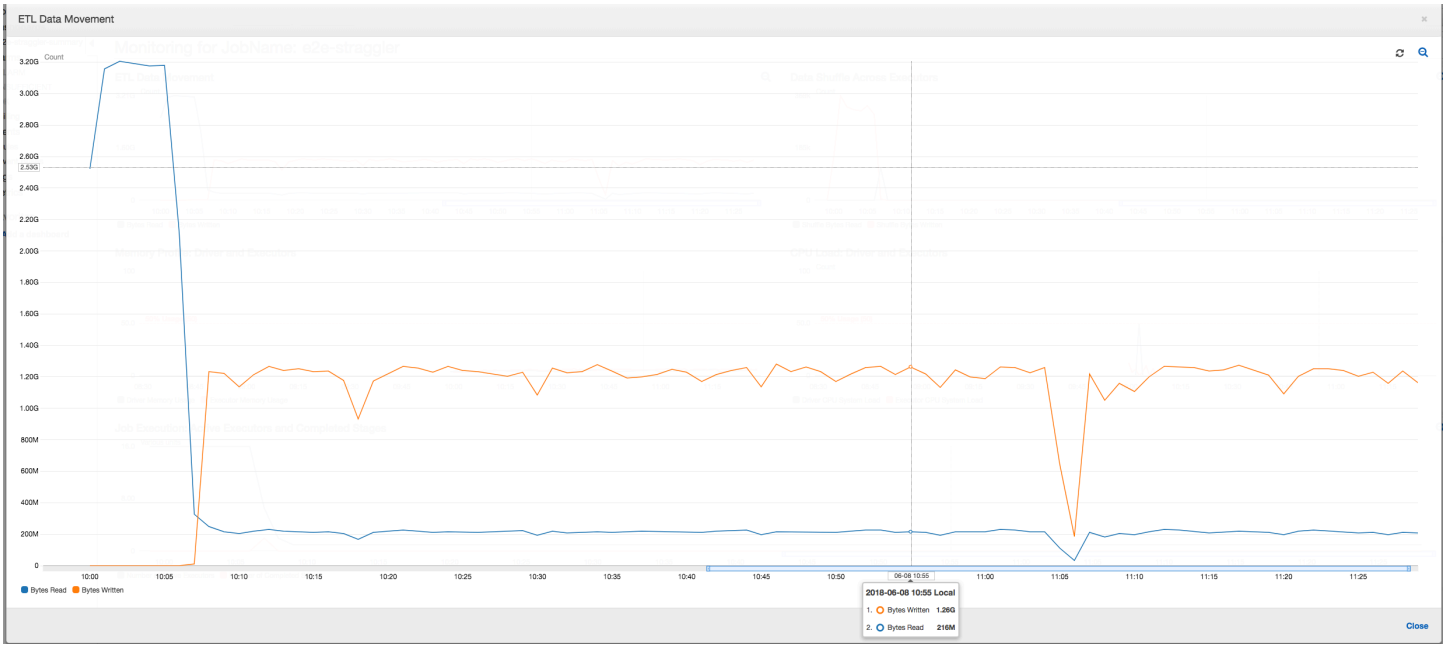
```
datasource0 = spark.read.format("json").load("s3://input_path")
df = datasource0.coalesce(1)
df.write.format("json").save(output_path)
```

在 AWS Glue 控制台上可视化分析指标

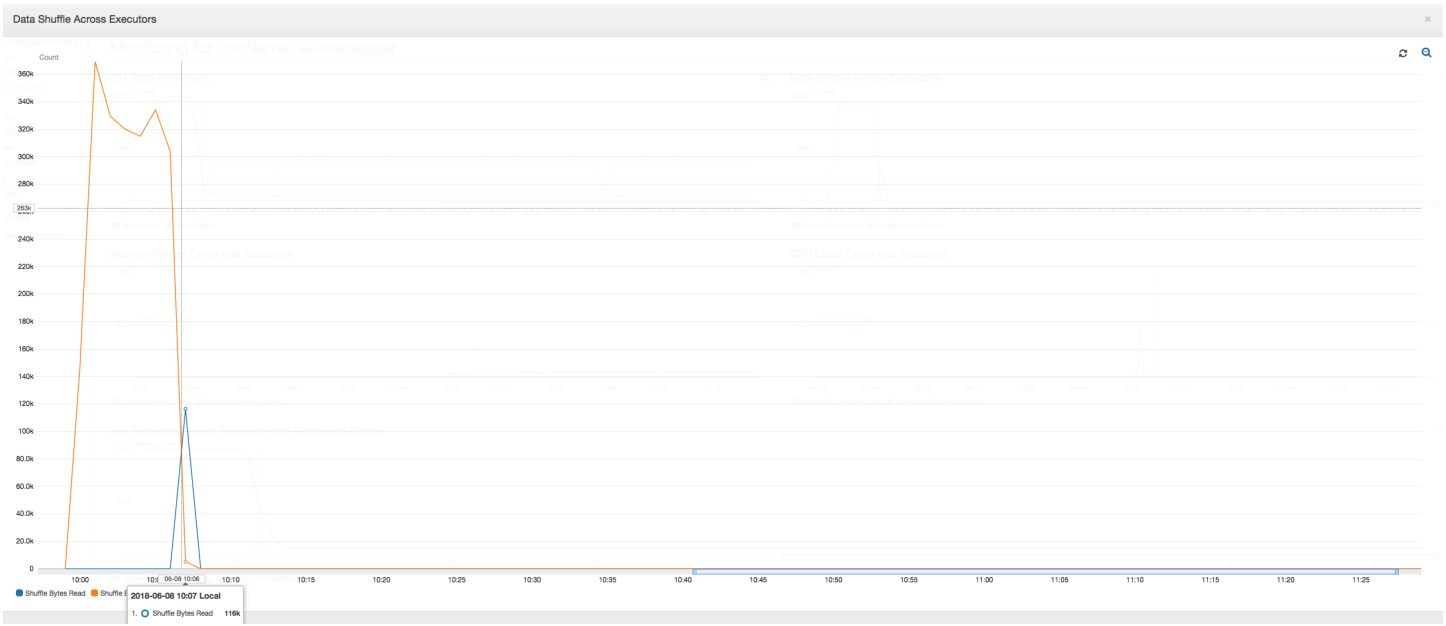
您可以分析您的作业以检查四组不同的指标：

- ETL 数据移动
- 执行程序之间的数据随机排序
- 任务执行
- 内存配置文件

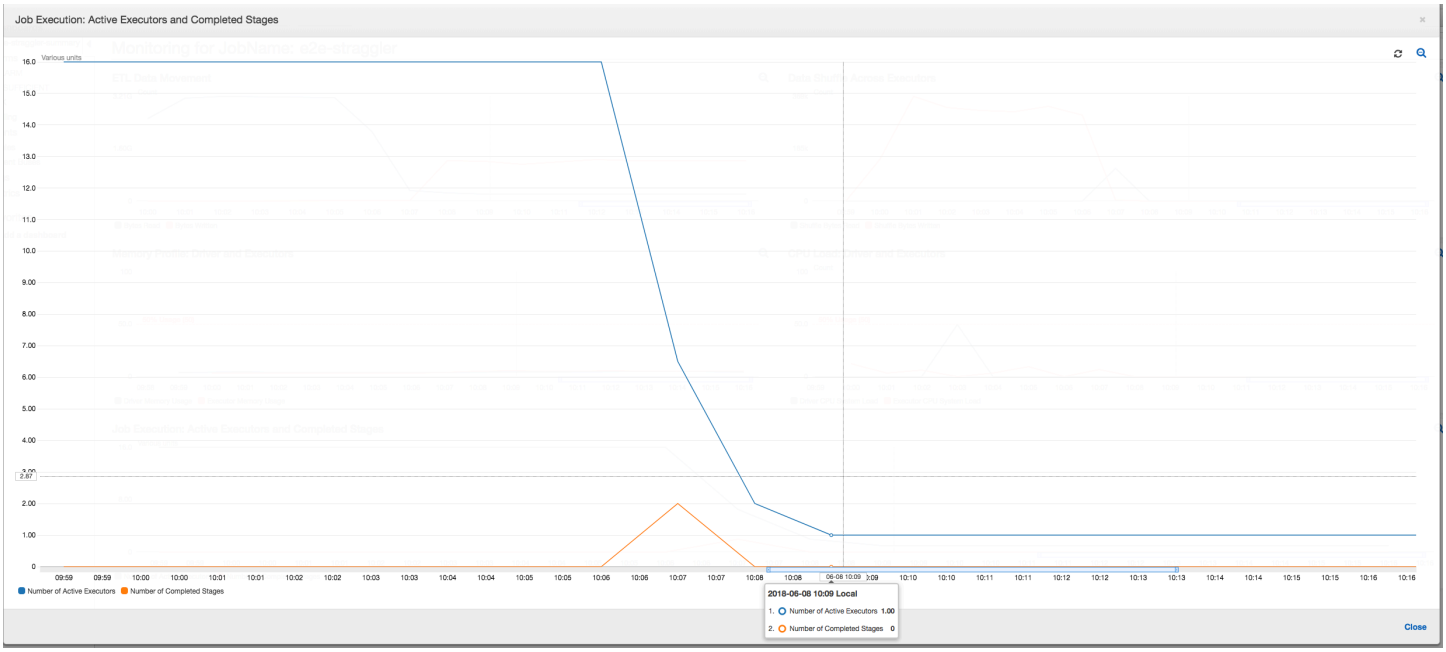
ETL 数据移动：在 ETL 数据移动配置文件中，在前六分钟内完成的第一阶段中的所有执行程序都会相当快速地读取字节。但是，总作业执行时间大约为一小时，主要由数据写入时间组成。



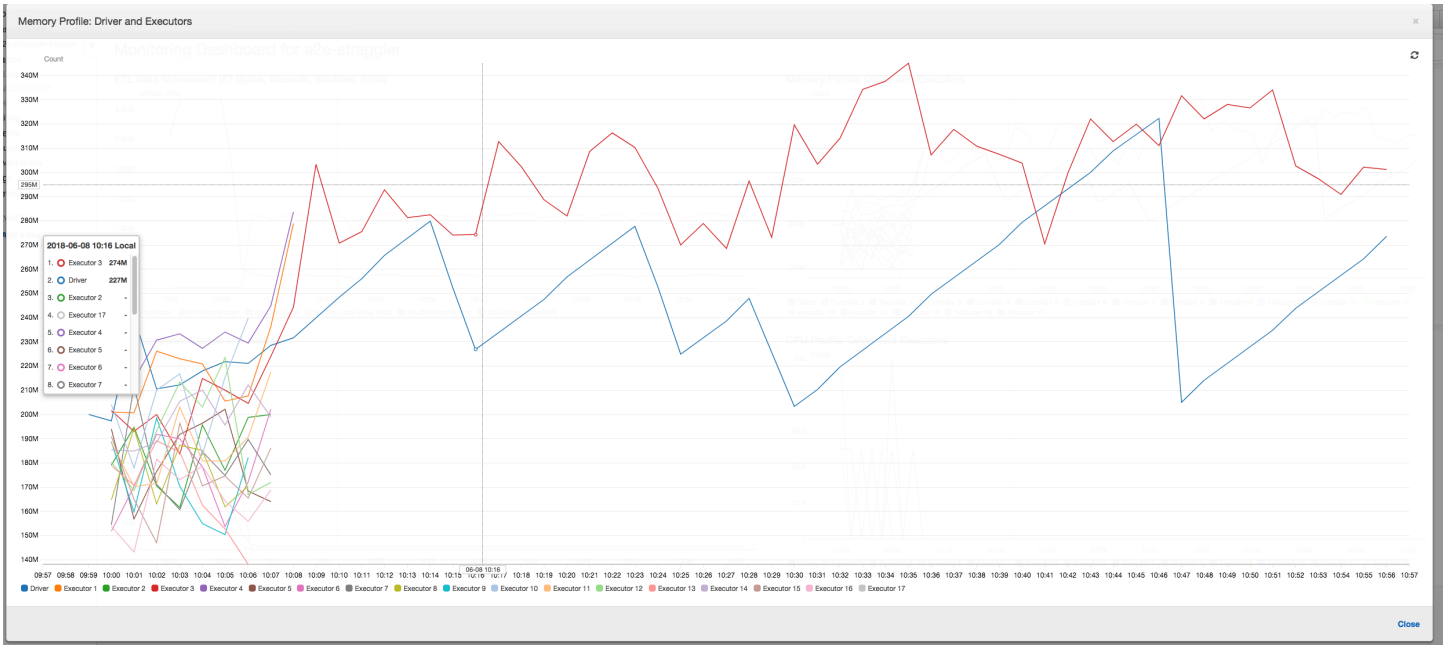
执行程序之间的数据随机排序：随机排序期间读取和写入的字节数还在阶段 2 结束之前显示一个峰值，如Job Execution (作业执行) 和Data Shuffle (数据随机排序) 指标所示。在从所有执行程序对数据进行随机排序之后，读取和写入仅从 3 号执行程序继续。



作业执行：如下图所示，所有其他执行程序均处于空闲状态，并最终在 10:09 之前被释放。此时，执行程序的总数减少到只有一个。这显然表明 3 号执行程序由执行时间最长并且占据大部分作业执行时间的落后任务组成。



内存配置文件：在前两个阶段之后，只有 3 号执行程序在主动消耗内存来处理数据。其余执行程序只是处于空闲状态或在前两个阶段完成后很快被释放。



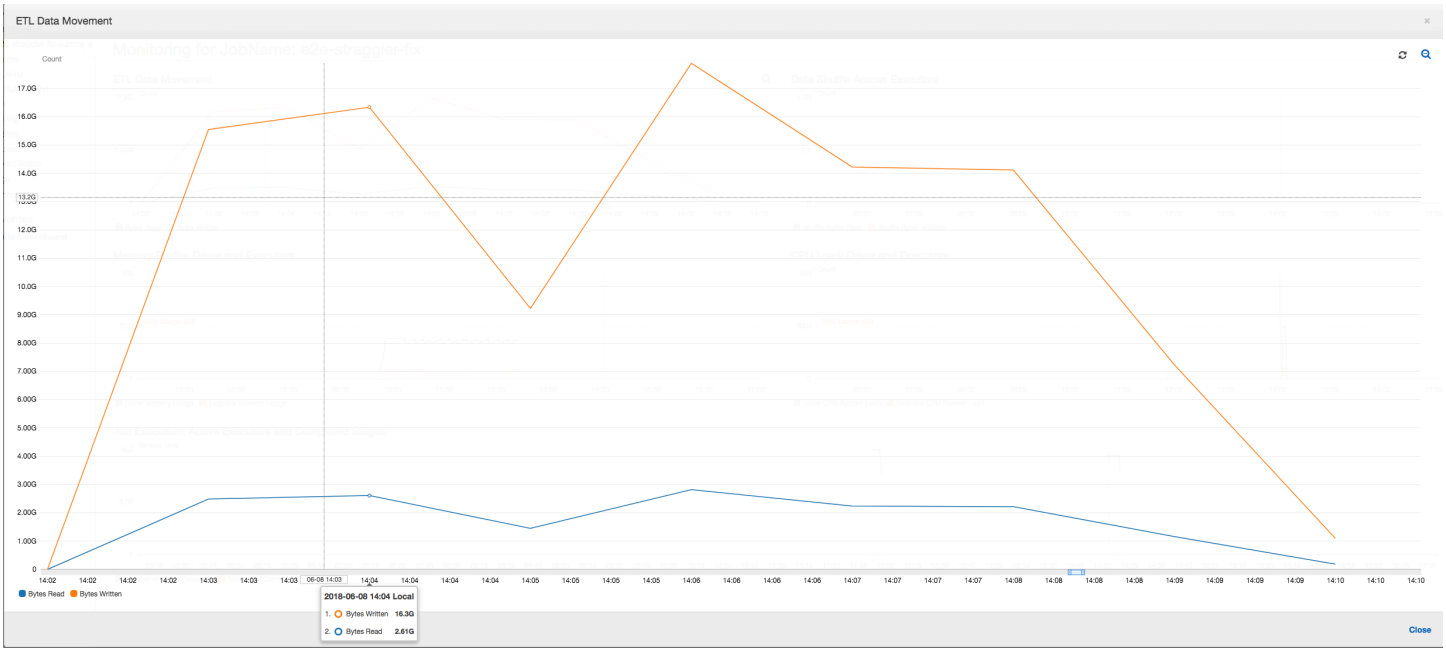
使用分组修复落后的执行程序

您可以通过使用 中的分组AWS Glue功能来避免执行程序落后。使用分组在所有执行程序中均匀分配数据，并使用集群上的所有可用执行程序将文件合并为更大的文件。有关更多信息，请参阅 以较大的组读取输入文件。

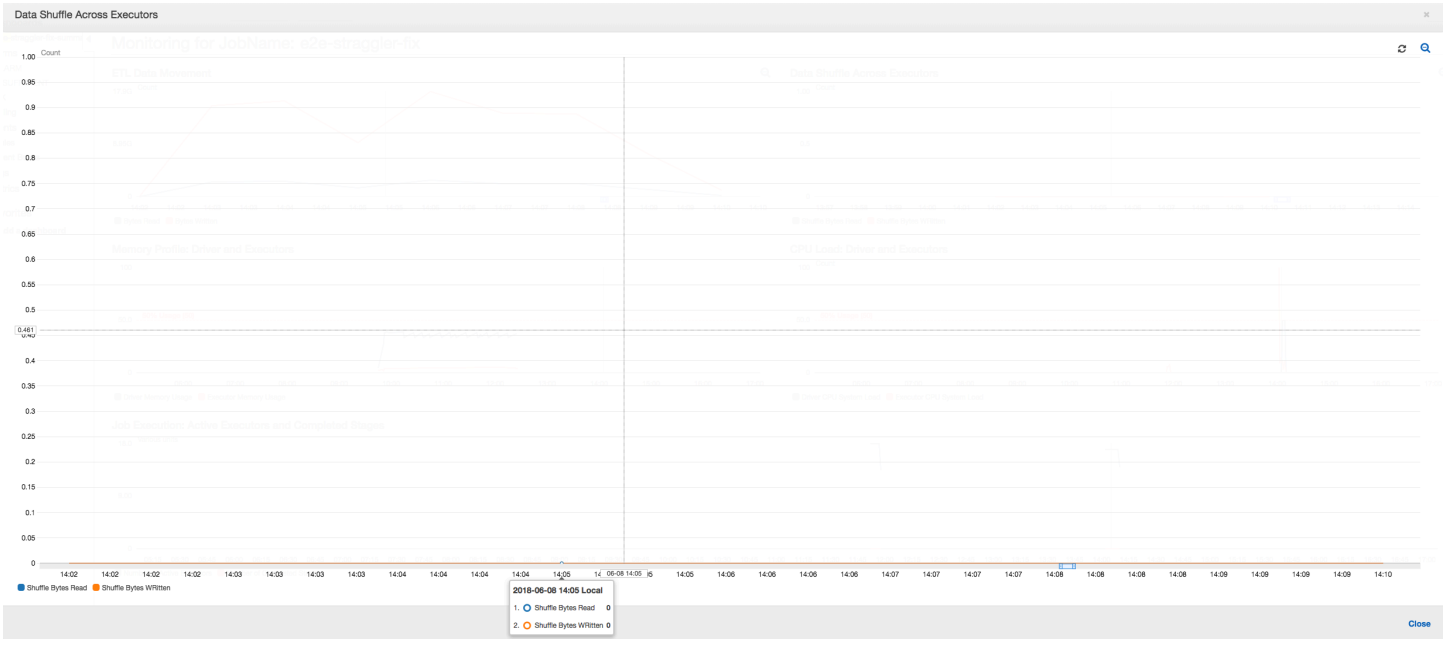
要检查 AWS Glue 作业中的 ETL 数据移动，请在启用分组的情况下配置以下代码：

```
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")
```

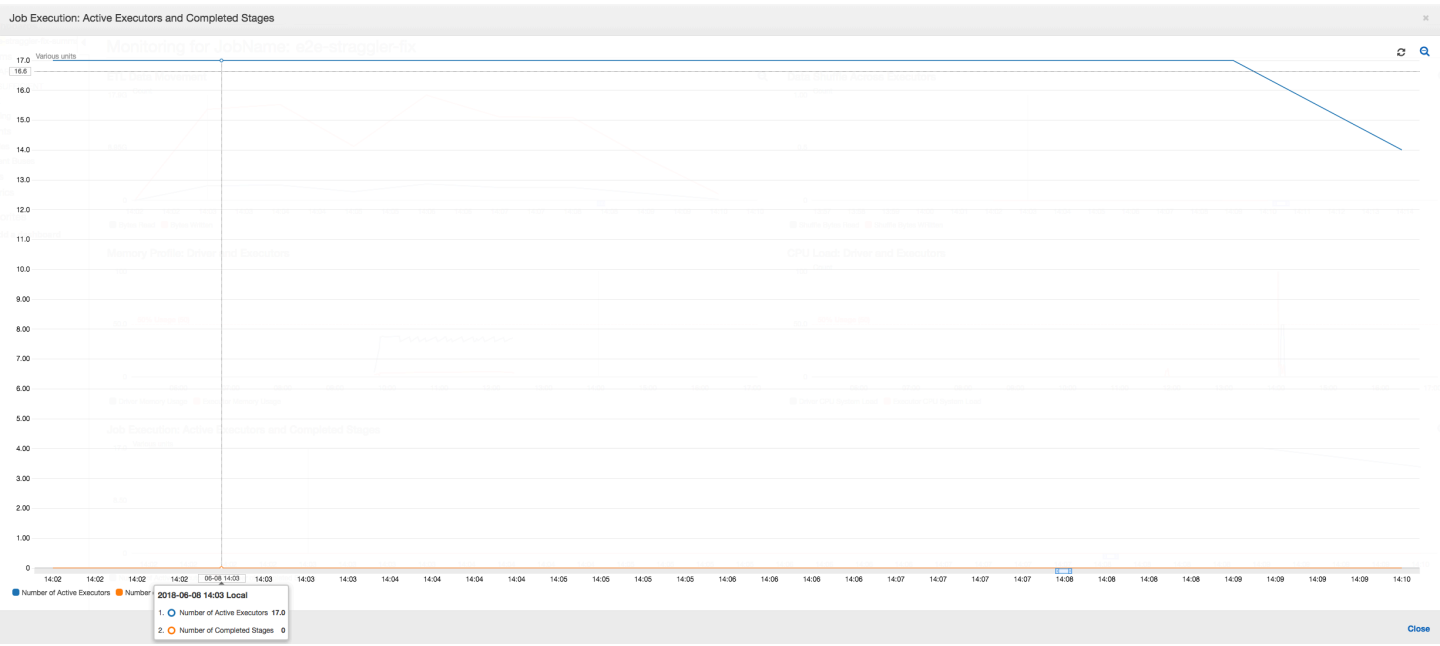
ETL 数据移动：整个作业执行时间内的数据写入与数据读取现在并行进行流式传输。因此，任务在八分钟内完成 – 比以前快得多。



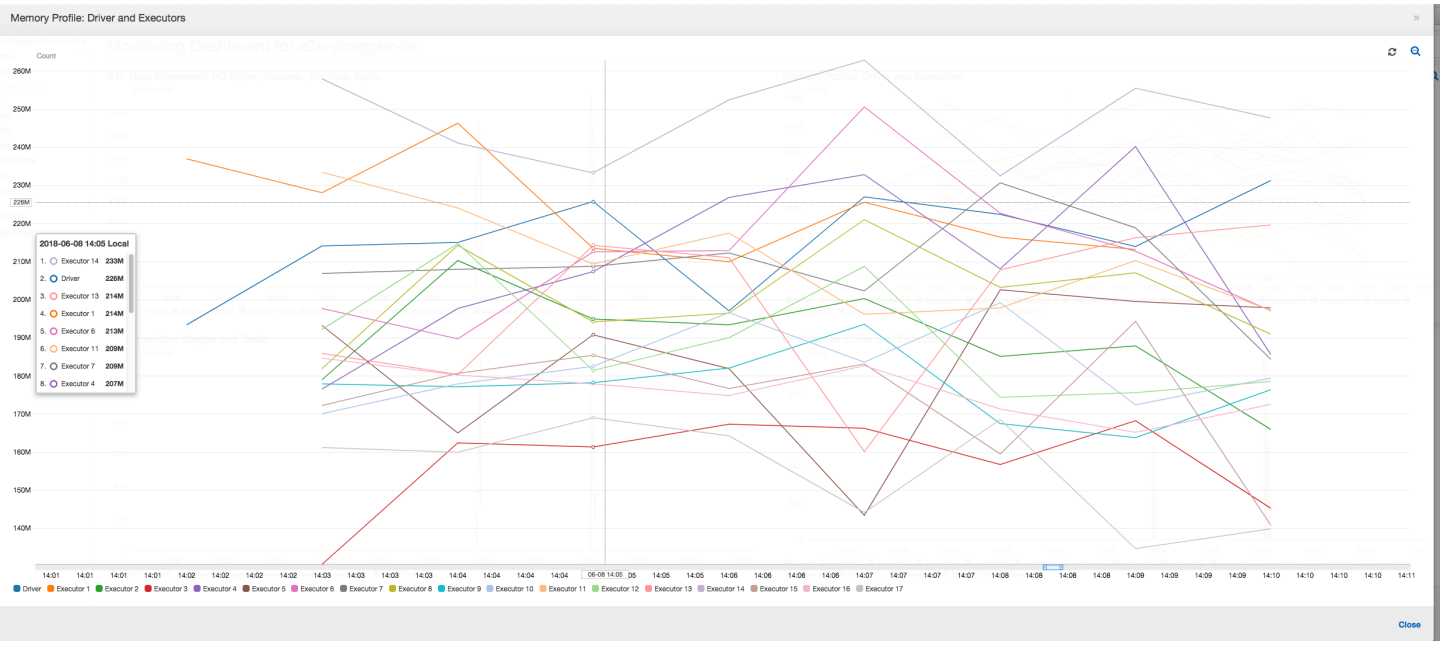
执行程序之间的数据随机排序：由于输入文件在读取期间使用分组功能进行了合并，因此在读取数据后不会进行代价高昂的数据随机排序。



作业执行：作业执行指标显示运行和处理数据的活动执行程序总数保持恒定。作业中没有单个落后者。所有执行程序均处于活动状态，在完成作业之前不会被释放。由于执行程序中没有数据的中间随机排序情况，因此作业中只有一个阶段。



内存配置文件：相关指标显示所有执行程序的活动内存消耗 – 跨所有执行程序重新确认存在活动。随着数据并行流入和写出，所有执行程序的总内存占用空间大致均匀，远低于所有执行程序的安全阈值。



监控多个作业的进度

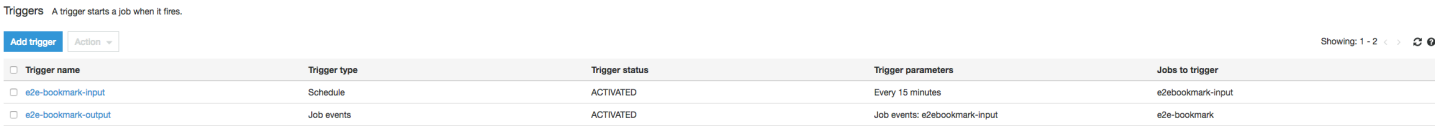
您可以一起分析多个 AWS Glue 作业并监控它们之间的数据流。这是一种常见的工作流模式，需要监控单个作业进度、数据处理积压、数据重新处理和作业书签。

主题

- 分析代码
- 在 AWS Glue 控制台上可视化分析指标
- 修复文件的处理

分析代码

在此工作流程中，您有两个作业：一个输入作业和一个输出作业。输入作业计划使用定期触发器每 30 分钟运行一次。输出作业计划在每次成功运行输入作业后运行。可使用作业触发器控制这些计划作业。



输入任务：此任务从 Amazon Simple Storage Service（Amazon S3）位置读取数据，使用 ApplyMapping 对数据进行转换，并将其写入暂存的 Amazon S3 位置。以下代码是输入作业的分析代码：

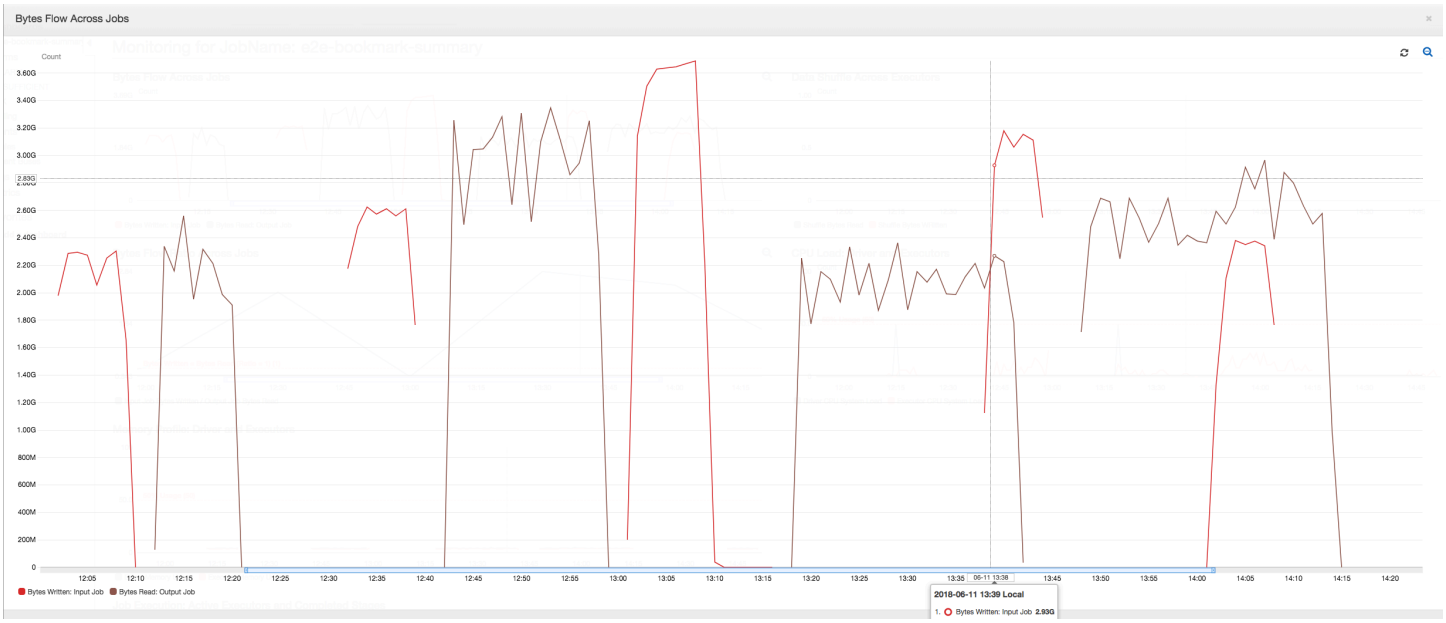
```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
 connection_options = {"paths": ["s3://input_path"],
 "useS3ListImplementation":True,"recurse":True}, format="json")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [map_spec])
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
 connection_type = "s3", connection_options = {"path": staging_path, "compression":
 "gzip"}, format = "json")
```

输出任务：此任务从 Amazon S3 中的暂存位置读取输入任务的输出，再次对其进行转换，并将其写入目标位置：

```
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 控制台上可视化分析指标

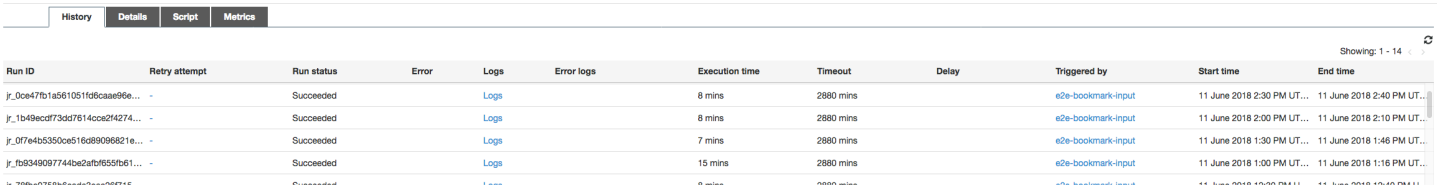
以下控制面板将输入任务中的 Amazon S3 字节写入指标叠加到输出任务的同一时间表的 Amazon S3 字节读取指标上。时间表显示输入和输出作业的不同作业运行。输入作业（以红色显示）每 30 分钟启动一次。输出作业（以棕色显示）在输入作业完成时启动，最大并发数为 1。



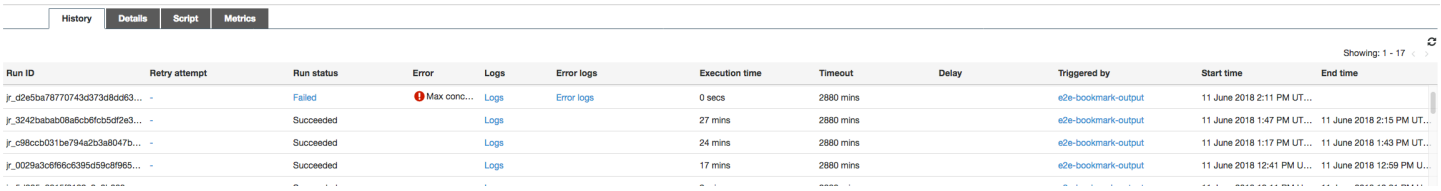
在本示例中，未启用作业书签。没有用于在脚本代码中启用作业书签的转换上下文。

作业历史记录：输入和输出作业具有多个从中午 12:00 启动的作业运行，如历史记录选项卡上所示。

AWS Glue 控制台上的输入任务如下所示：



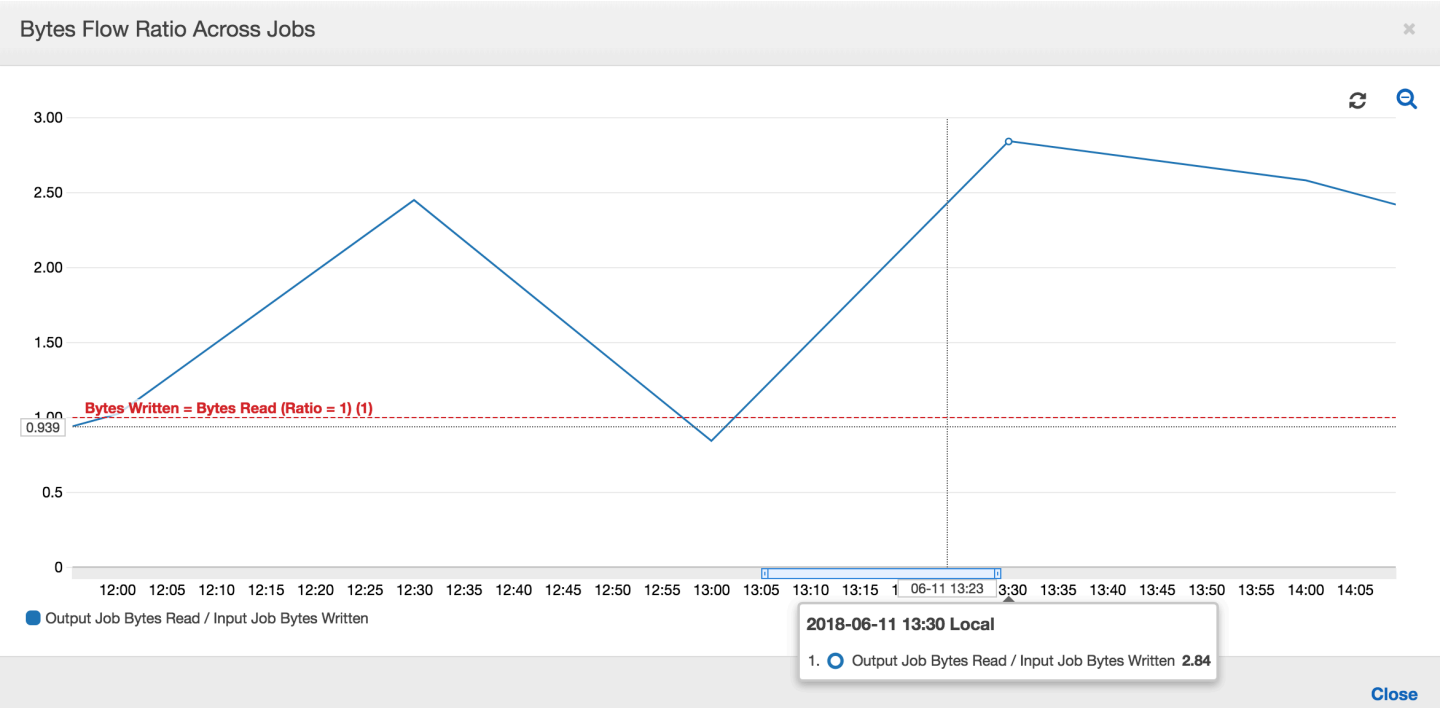
下图显示了输出作业：



第一次作业运行：如下面的数据字节读取和写入图表所示，在 12:00 到 12:30 之间的输入和输出作业的第一次作业运行显示曲线下大致相同的区域。这些区域表示由输入任务写入的 Amazon S3 字节和由

输出任务读取的 Amazon S3 字节。此数据同样通过写入的 Amazon S3 字节的比率（总计超过 30 分钟 – 输入任务的任务触发频率）来确认。中午 12:00 启动的输入作业运行的比率的数据点也是 1。

下图显示了所有作业运行的数据流比率：



第二次作业运行：在第二次作业运行中，输出作业读取的字节数与输入作业写入的字节数相比明显不同。（比较输出作业的两次作业运行的曲线下的区域，或比较输入和输出作业的第二次运行中的区域。）读取和写入的字节比率表明，在 12:30 到 13:00 的第二个 30 分钟跨度内，输出作业读取的数据约为输入作业写入的数据的 2.5 倍。这是因为由于未启用作业书签，导致输出作业重新处理了输入作业的第一次作业运行的输出。比率大于 1 表示输出作业处理的数据存在额外的积压。

第三次作业运行：输入作业在写入的字节数方面相当一致（请参阅红色曲线下的区域）。但是，输入作业的第三次作业运行时间比预期的要长（请参阅红色曲线的长尾）。因此，输出作业的第三次作业运行启动较晚。第三次作业运行在 13:00 到 13:30 之间的剩余 30 分钟内仅处理了暂存位置中累积的一小部分数据。字节流的比率表明，它仅处理了由输入作业的第三次作业运行写入的 0.83 数据（请参阅 13:00 的比率）。

重叠的输入和输出作业：输入作业的第四次作业运行按照计划在输出作业的第三次作业运行完成之前于 13:30 启动。这两次作业运行之间存在部分重叠。但是，输出任务的第三次任务运行仅捕获它于 13:17 左右启动时在 Amazon S3 的暂存位置列出的文件。这包括输入作业的第一次作业运行的所有数据输出。13:30 的实际比率约为 2.75。输出作业的第三次作业运行处理了输入作业的第四次作业运行从 13:30 到 14:00 写入的大约 2.75 倍数据。

如这些图像所示，输出作业正在从输入作业的所有先前作业运行中的暂存位置重新处理数据。因此，输出作业的第四次作业运行时间最长，并与输入作业的整个第五次作业运行重叠。

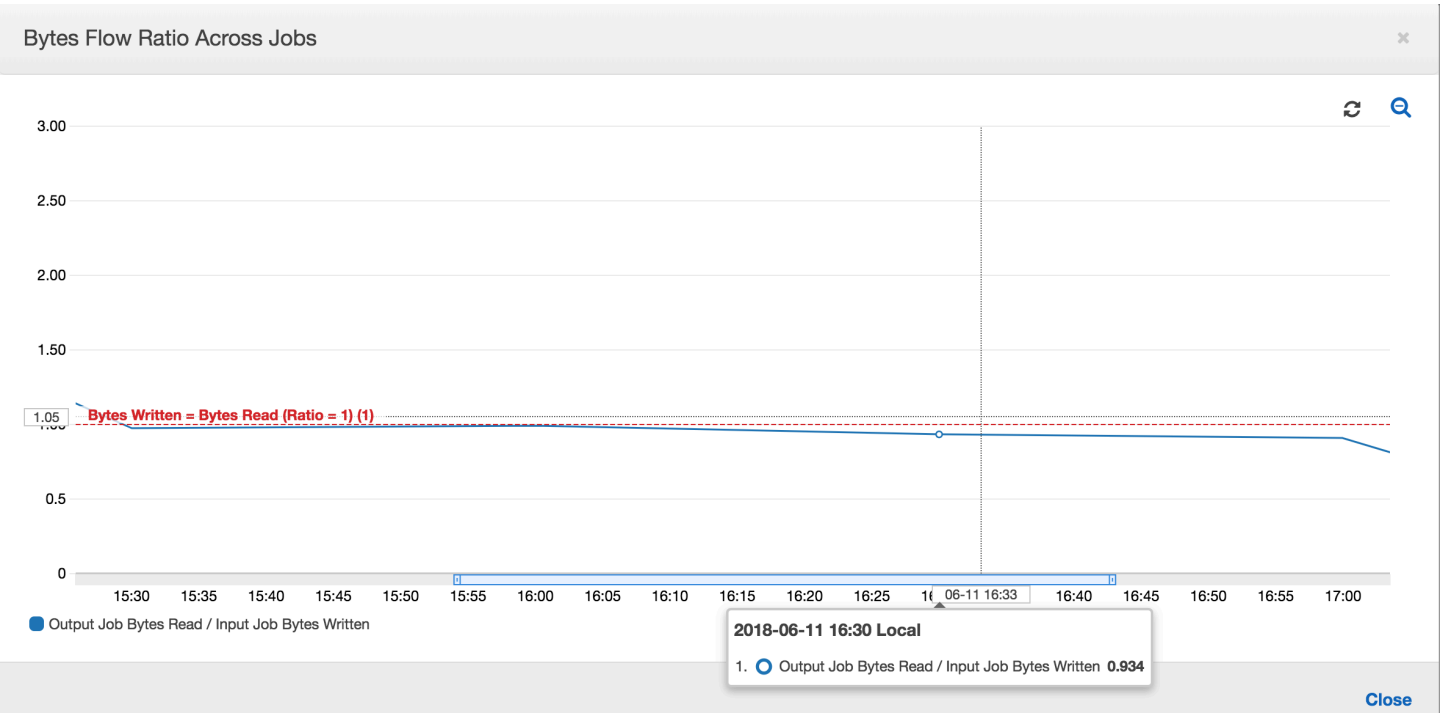
修复文件的处理

您应确保输出作业仅处理输出作业的先前作业运行尚未处理的文件。为此，请启用作业书签并在输出作业中设置转换上下文，如下所示：

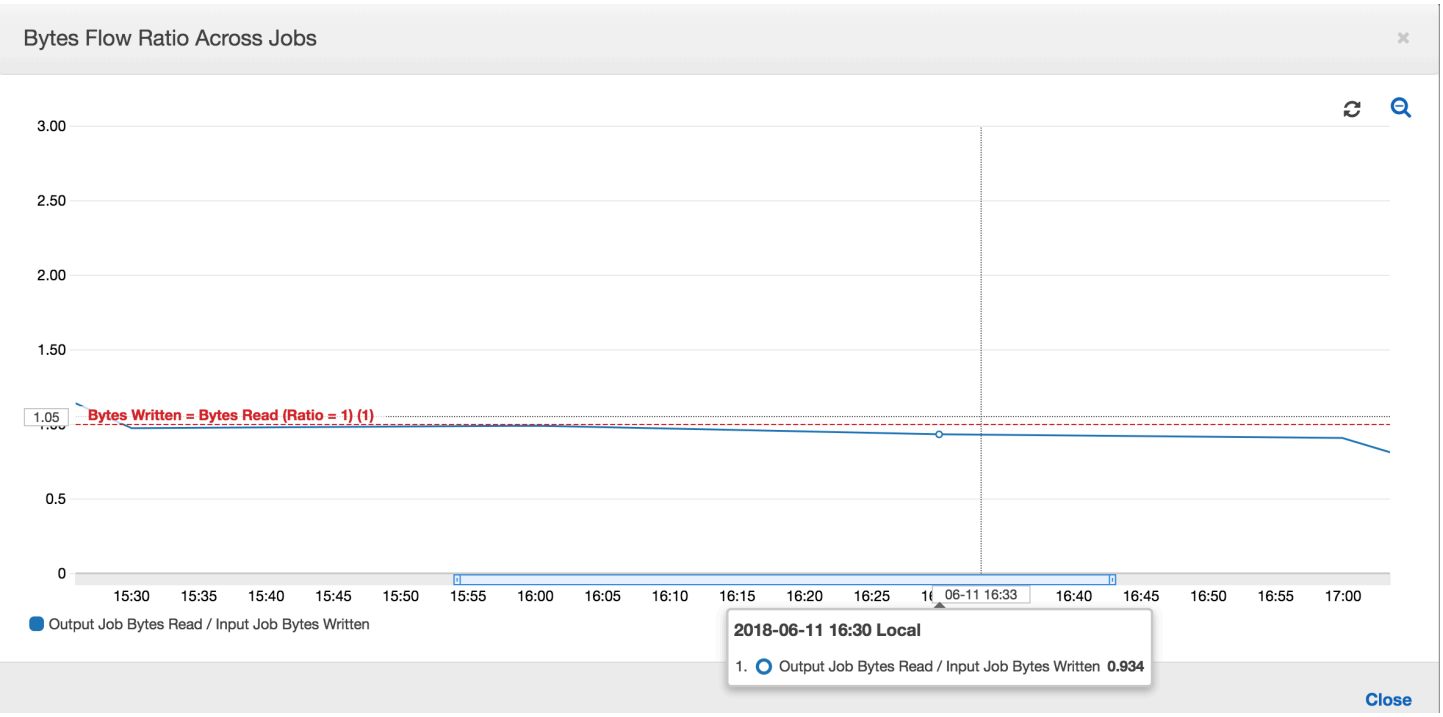
```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
 connection_options = {"paths": [staging_path],
 "useS3ListImplementation":True,"recurse":True}, format="json", transformation_ctx =
 "bookmark_ctx")
```

启用作业书签后，输出作业不会重新处理输入作业的所有先前作业运行的暂存位置中的数据。在显示读取和写入数据的下图中，棕色曲线下的区域相当一致，并与红色曲线类似。

字节流的比率还保持大致接近 1，因为没有处理其他数据。



在下一次输入作业运行开始将更多数据放入暂存位置之前，输出作业的作业运行将启动并捕获暂存位置中的文件。只要它继续执行此操作，就会仅处理从先前输入作业运行中捕获的文件，并且该比率保持接近 1。



假设输入作业花费的时间超过预期，那么，输出作业将从两次输入作业运行中捕获暂存位置中的文件。该输出作业运行的比率将大于 1。但是，输出作业的以下作业运行不会处理已由输出作业的先前作业运行处理的任何文件。

监控 DPU 容量规划

您可以使用 AWS Glue 中的作业指标估算可用于扩展 AWS Glue 作业的数据处理单元 (DPU) 的数量。

> **Note**
>
> 此页面仅适用于 AWS Glue 0.9 和 1.0 版。最新版本的 AWS Glue 包含成本节省功能，介绍了在进行容量规划时的其他注意事项。

主题

分析代码

以下脚本读取包含 428 个 gzip 类型 JSON 文件的 Amazon Simple Storage Service（Amazon S3）分区。该脚本应用映射来更改字段名称，以 Apache Parquet 格式转换这些名称并将其写入 Amazon S3。您可以按默认值预置 10 个 DPU 并运行此任务。

```
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")
```

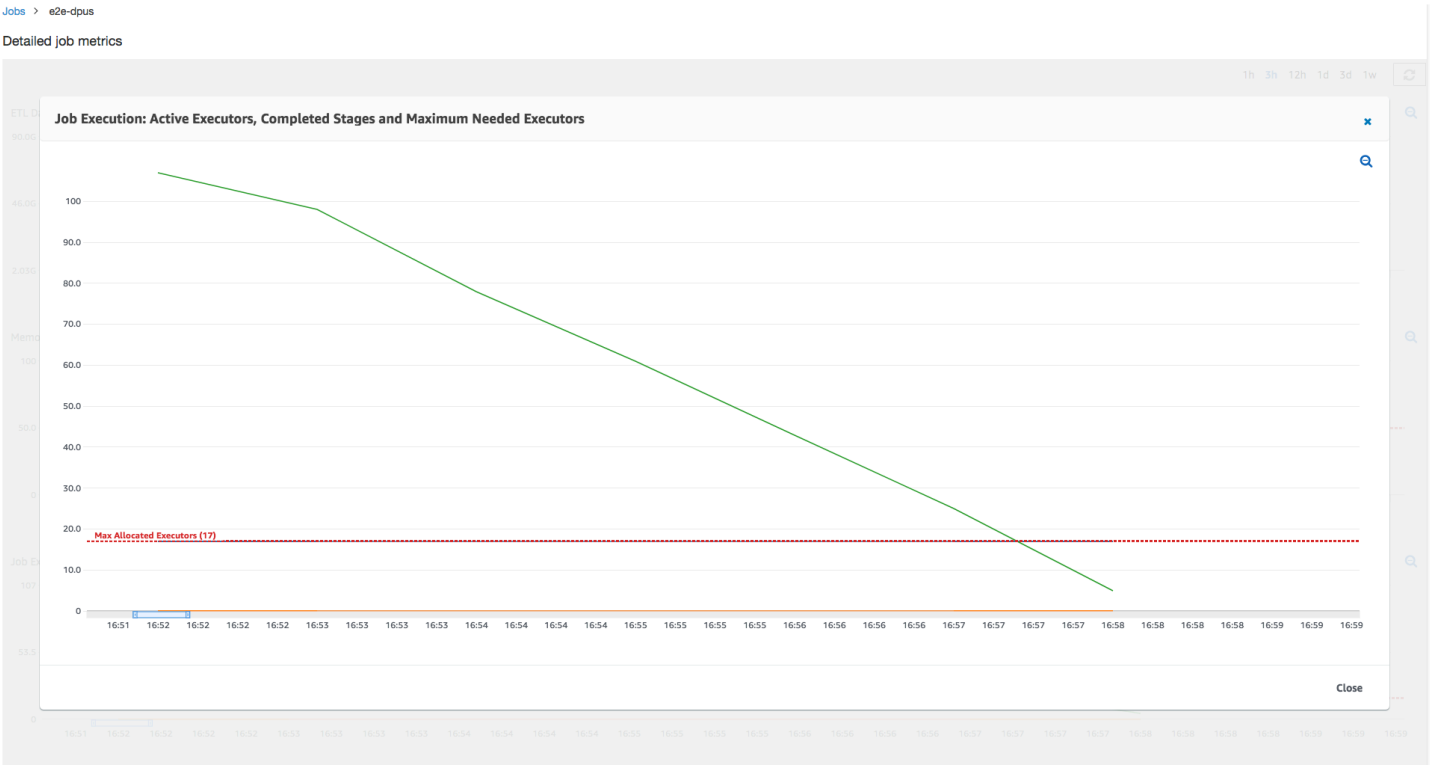
在 AWS Glue 控制台上可视化分析指标

作业运行 1：在此作业运行中，我们将展示如何确定集群中是否存在预配置不足的 DPU。AWS Glue 中的作业执行功能显示主动运行的执行程序的总数、已完成的阶段数和所需的最大执行程序数。

通过添加正在运行的任务和待处理任务的总数，并将其除以每个执行程序的任务数来计算所需的最大执行程序数。此结果衡量满足当前负载所需的执行程序总数。

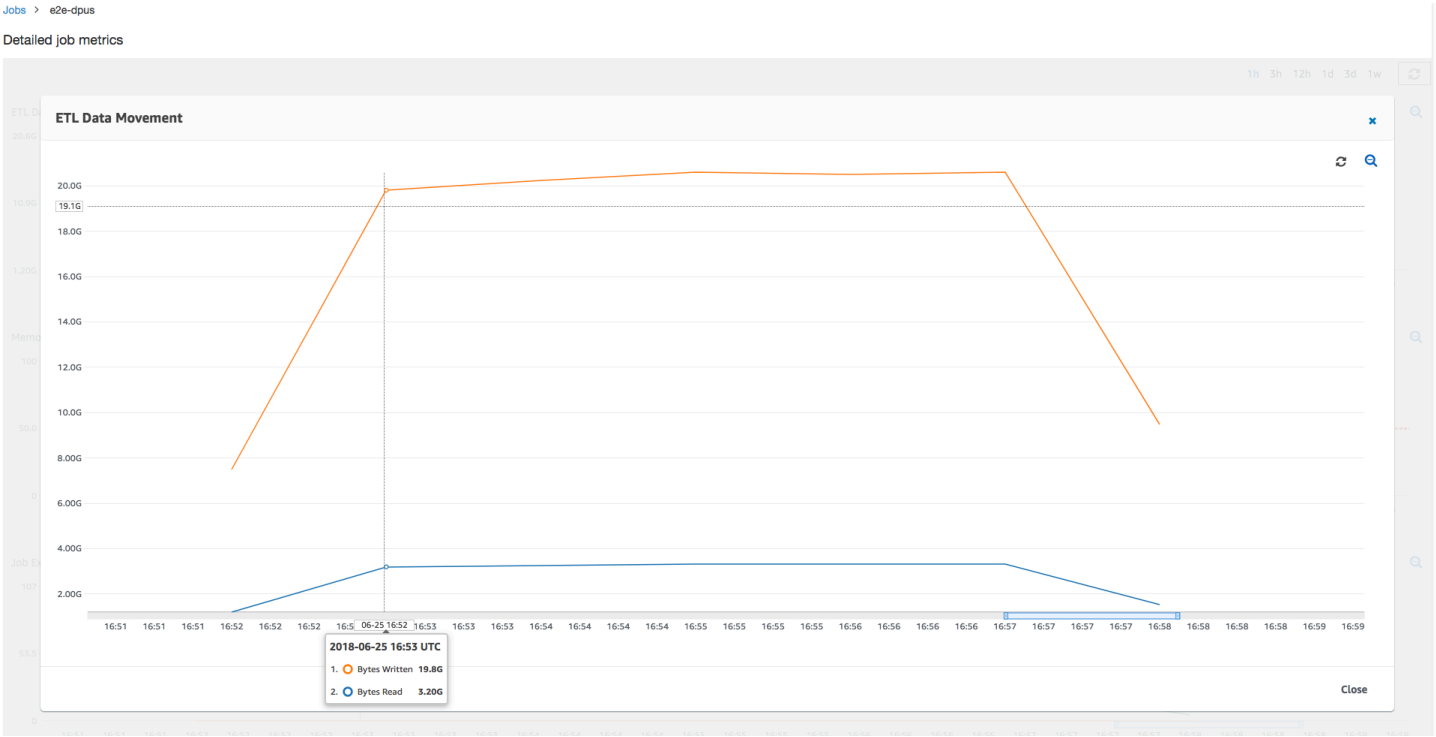
相反，主动运行的执行程序的数量衡量运行活动 Apache Spark 任务的执行程序数量。随着作业的运行，所需的最大执行程序数可能发生变化，并且通常会在待处理任务队列减少时向下移动到作业的末尾。

下图中的水平红线显示了最大已分配执行程序数，这取决于您为作业分配的 DPU 数。在此案例中，您为作业运行分配 10 个 DPU。为管理预留一个 DPU。其他九个 DPU 各自运行两个执行程序，并为 Spark 驱动程序保留一个执行程序。Spark 驱动程序在主应用程序内部运行。因此，最大已分配执行程序数为 2*9 - 1 = 17 个执行程序。



如图所示，所需的最大执行程序数在作业开始时从 107 开始，而活动执行程序数保持为 17。这与具有 10 个 DPU 的最大已分配执行程序数相同。所需的最大执行程序数和最大已分配执行程序数之间的比率（对于 Spark 驱动程序，两者都加 1）会为您提供预配置不足系数：108/18 = 6x。您可以预置 6（在配置比率下）* 9（当前 DPU 容量 - 1）+ 1 个 DPU = 55 个 DPU 来扩展作业，以最大并行度运行它并更快地完成。

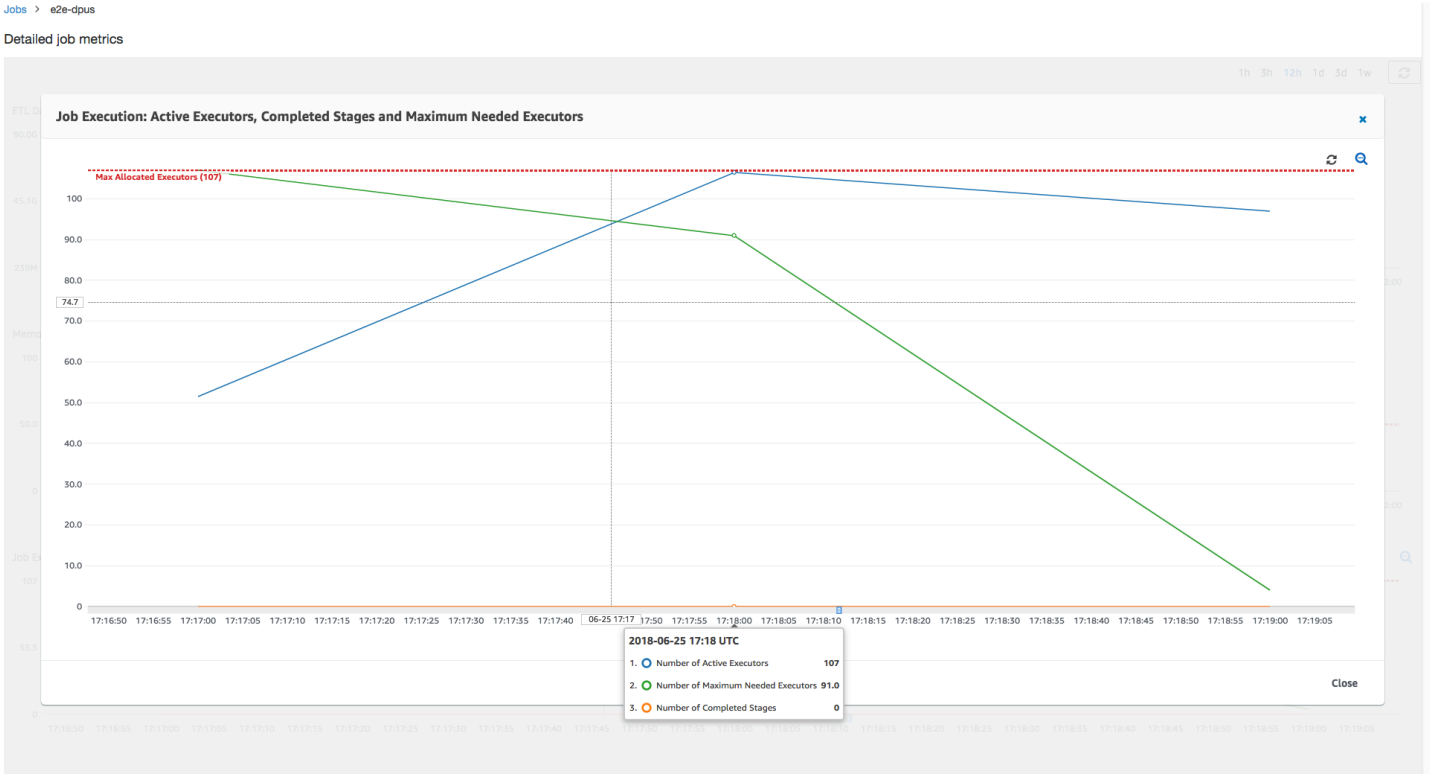
AWS Glue 控制台将详细的作业指标显示为表示原始最大已分配执行程序数的静态行。控制台从指标的作业定义计算最大已分配执行程序数。相比这下，对于详细的作业运行指标，控制台从作业运行配置计算最大已分配执行程序数，特别是为作业运行分配的 DPU。要查看各个作业运行的指标，请选择作业运行并选择 View run metrics (查看运行指标)。



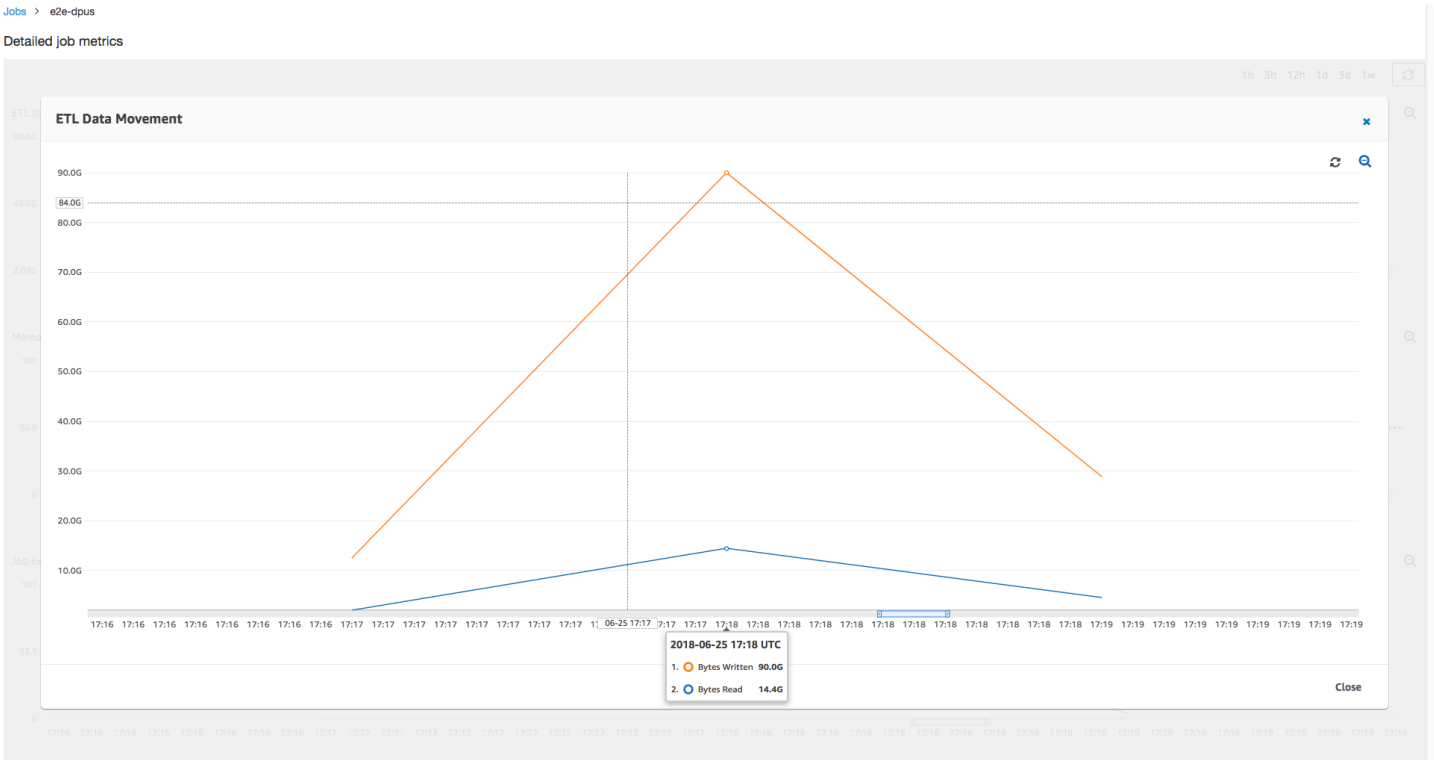
查看读取和写入的 Amazon S3 字节，请注意，该任务花费全部六分钟时间流式传输来自 Amazon S3 的数据并将其并行写出。已分配 DPU 上的所有核心都在读取和写入 Amazon S3。所需最大执行程序数为 107，同样与输入 Amazon S3 路径中的文件数 428 相匹配。每个执行程序可以启动四个 Spark 任务来处理四个输入文件（JSON gzip 类型）。

确定最佳 DPU 容量

根据先前作业运行的结果，您可以将已分配 DPU 总数增加到 55，并查看作业的执行情况。该任务在不到三分钟的时间内完成 – 是之前所需时间的一半。在这种情况下，作业扩展不是线性的，因为它是一个短时间运行作业。具有长期任务或大量任务（所需最大执行程序数）的作业受益于接近线性的 DPU 扩展性能加速。



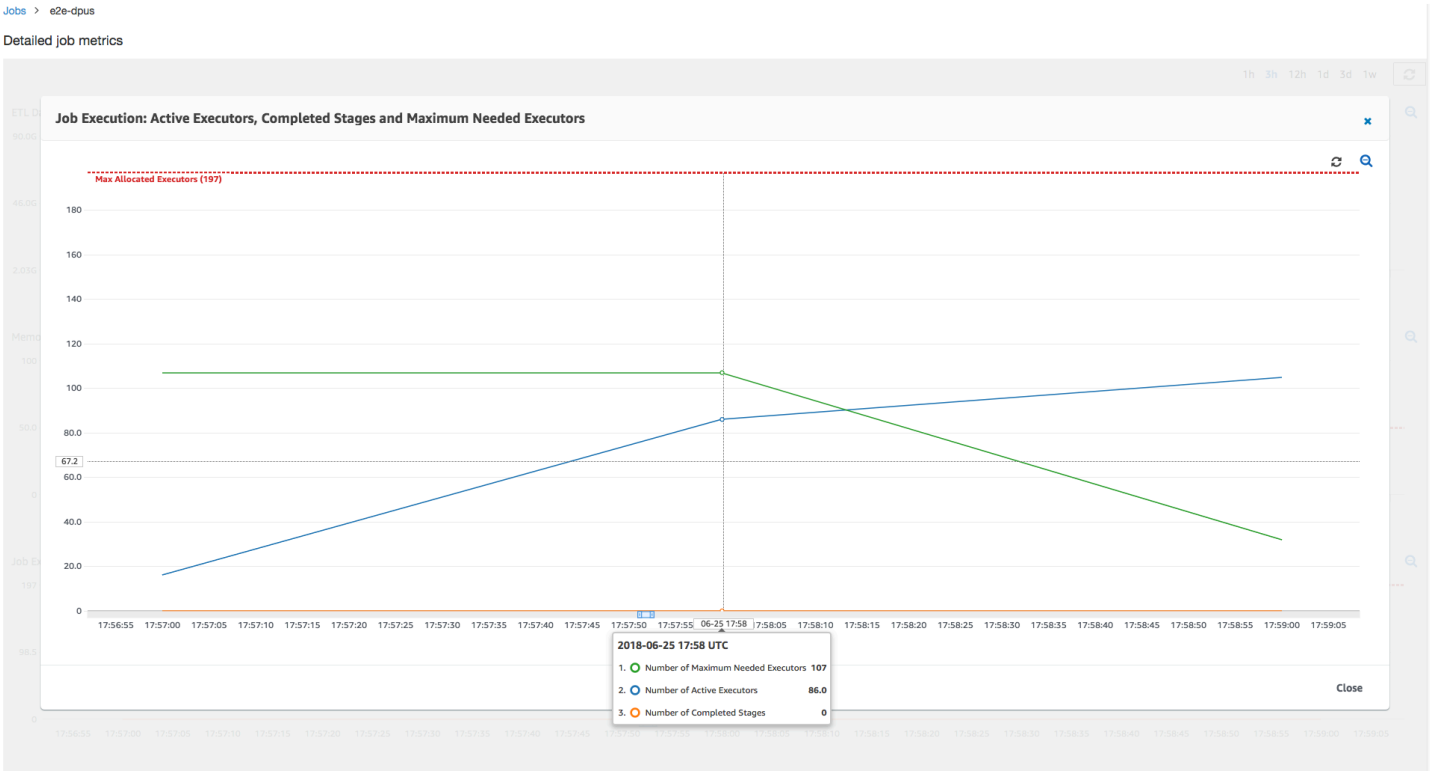
如上图所示，活动执行程序的总数达到最大分配数 – 107 个执行程序。同样，所需的最大执行程序数永远不会超过最大已分配执行程序数。所需的最大执行程序数根据主动运行和待处理任务计数计算，因此可能小于活动执行程序的数量。这是因为可能有执行程序在短时间内部分或完全空闲且尚未停用。



此任务运行使用 6 倍以上的执行程序来并行地在 Amazon S3 中读取和写入。因此，此任务运行使用更多 Amazon S3 带宽进行读取和写入，并且完成得更快。

识别过度预配置的 DPU

接下来，您可以确定使用 100 个 DPU（99 * 2 = 198 个执行程序）扩展作业是否有助于进一步扩展。如下图所示，该作业仍需要三分钟才能完成。同样，作业不会扩展超出 107 个执行程序（55 个 DPU 配置），其余 91 个执行程序被过度配置而根本未使用。这表明增加 DPU 的数量可能并不会始终提高性能，这从所需的最大执行程序数可以看出。



比较时间差异

下表中显示的三次作业运行汇总了 10 个 DPU、55 个 DPU 和 100 个 DPU 的作业执行时间。您可以使用通过监控第一次作业运行建立的估计值来查找 DPU 容量以缩短作业执行时间。

作业 ID	DPU 数量	执行时间
jr_c894524c8ef5048a4d9...	10	6 分钟
jr_1a466cf2575e7ffe6856...	55	3 分钟
jr_34fa1ed4c6aa9ff0a814...	100	3 分钟

在 AWS Glue 中流式处理 ETL 作业

您可以创建连续运行的串流提取、转换和负载（ETL）任务，使用来自 Amazon Kinesis Data Streams、Apache Kafka 和 Amazon Managed Streaming for Apache Kafka（Amazon MSK）等串流源的数据。这些任务会清理并转换数据，然后将结果加载到 Amazon S3 数据湖或 JDBC 数据存储中。

此外，您还可以为 Amazon Kinesis Data Streams 流生成数据。此功能仅在编写 AWS Glue 脚本时可用。有关更多信息，请参阅 the section called “Kinesis 连接”。

默认情况下，AWS Glue 在 100 秒的时段内处理和写出数据。这可以实现数据的高效处理，并允许对晚于预计时间到达的数据执行聚合。您可以修改此窗口时段的大小以提高及时性或聚合精度。AWS Glue 串流任务使用检查点而非任务书签来跟踪已读取的数据。

> **Note**
>
> 当串流 ETL 任务正在运行时，AWS Glue 会按小时计费。

该视频讨论了流媒体ETL的成本挑战以及节省成本的功能。 AWS Glue

创建串流 ETL 任务涉及以下步骤：

1. 对于 Apache Kafka 串流源，请创建与 Kafka 源或 Amazon MSK 集群的 AWS Glue 连接。
2. 手动为串流源创建数据目录表。
3. 为串流数据源创建 ETL 任务。定义特定于串流的任务属性，并提供您自己的脚本或（可选）修改生成的脚本。

有关更多信息，请参阅 AWS Glue 中的流式处理 ETL。

为 Amazon Kinesis Data Streams 创建串流 ETL 任务时，您无需创建 AWS Glue 连接。但是，如果有一个连接附加到以 Kinesis Data Streams 作为源的 AWS Glue 串流 ETL 任务，则需要提供到 Kinesis 的 Virtual Private Cloud（VPC）终端节点。有关更多信息，请参阅 Amazon VPC 用户指南中的创建接口端点 在另一个账户中指定 Amazon Kinesis Data Streams 串流时，您必须设置角色和策略从而允许跨账户访问。有关更多信息，请参阅示例：从不同账户的 Kinesis 串流中读取。

AWS Glue 流式传输 ETL 任务可以自动检测压缩数据，以透明方式解压流式传输数据，对输入源执行常见转换，并加载到输出存储。

如果是以下输入格式，则 AWS Glue 支持自动解压以下压缩类型：

压缩类型	Avro 文件	Avro 基准	JSON	CSV	Grok
BZIP2	支持	是	是	是	是
GZIP	否	是	是	是	是
SNAPPY	是（原始 Snappy）	是（framed Snappy）	是（framed Snappy）	是（framed Snappy）	是（framed Snappy）
XZ	支持	是	是	是	是
ZSTD	是	否	否	否	否
DEFLATE	支持	是	是	是	是

主题

- 为 Apache Kafka 数据流创建 AWS Glue 连接
- 为串流源创建数据目录表
- Avro 串流源的注释和限制
- 将 grok 模式应用于串流源
- 定义串流 ETL 作业的作业属性
- 串流 ETL 注释和限制

为 Apache Kafka 数据流创建 AWS Glue 连接

要从 Apache Kafka 流中进行读取，您必须创建 AWS Glue 连接。

为 Kafka 源创建 AWS Glue 连接（控制台）

1. 打开 AWS Glue 控制台，网址为 https://console.aws.amazon.com/glue/。
2. 在导航窗格的 Data catalog (数据目录) 下，选择 Connections (连接)。
3. 选择 Add connection (添加连接)，然后在 Set up your connection’s properties (设置连接的属性) 页面上，输入连接名称。

> **Note**
>
> 有关指定连接属性的更多信息，请参阅 AWS Glue 连接属性。。

4. 对于 Connection type (连接类型)，选择 Kafka。
5. 对于 Kafka bootstrap servers URLs (Kafka 引导服务器 URL)，输入您 Amazon MSK 集群或 Apache Kafka 集群引导代理的主机和端口编号。仅使用传输层安全性（TLS）端点建立到 Kafka 群集的初始连接。不支持 Plaintext 端点。

   以下是 Amazon MSK 集群的主机名和端口编号对的示例列表。

   ```
   myserver1.kafka.us-east-1.amazonaws.com:9094,myserver2.kafka.us-
   east-1.amazonaws.com:9094,
   myserver3.kafka.us-east-1.amazonaws.com:9094
   ```

   有关引导代理信息的更多信息，请参阅 Amazon Managed Streaming for Apache Kafka 开发人员指南中的获取 Amazon MSK 集群的引导代理。
6. 如果您希望与 Kafka 数据源建立安全连接，请选择 Require SSL connection (需要 SSL 连接)，并在 Kafka private CA certificate location (Kafka 私有 CA 证书位置) 中，输入自定义 SSL 证书的有效 Amazon S3 路径。

   对于与自我托管式 Kafka 的 SSL 连接，自定义证书是强制性的。对于 Amazon MSK 则是可选的。

   有关为 Kafka 指定自定义证书的更多信息，请参阅 the section called “SSL 连接属性”。
7. 使用 AWS Glue Studio 或 AWS CLI 指定 Kafka 客户端身份验证方法。要访问， AWS Glue Studio 请AWS Glue从左侧导航窗格的 ETL 菜单中选择。

   有关 Kafka 客户端身份认证方法的更多信息，请参阅适用于客户端身份认证的 AWS Glue Kafka 连接属性 。
8. （可选）输入描述，然后选择 Next (下一步)。
9. 对于 Amazon MSK 集群，请指定其 Virtual Private Cloud（VPC）、子网和安全组。对于自行托管式 Kafka，VPC 信息是可选的。
10. 选择 Next (下一步) 以查看所有连接属性，然后选择 Finish (结束)。

有关 AWS Glue 连接的更多信息，请参阅 连接到数据。

适用于客户端身份认证的 AWS Glue Kafka 连接属性

SASL/GSSAPI（Kerberos）身份认证

选择此身份认证方法将允许您指定 Kerberos 属性。

Kerberos Keytab

选择 keytab 文件的位置。keytab 可存储一个或多个主体的长期密钥。有关更多信息，请参阅 MIT Kerberos 文档：keytab。

Kerberos krb5.conf 文件

选择 krb5.conf 文件。它包含默认领域（一种类似于域的逻辑网络，用于定义同一 KDC 下的一组系统）和 KDC 服务器的位置。有关更多信息，请参阅 MIT Kerberos 文档：krb5.conf。

Kerberos 主体和 Kerberos 服务名称

输入 Kerberos 主体和服务名称 有关更多信息，请参阅 MIT Kerberos 文档：Kerberos 主体。

SASL/SCRAM-SHA-512 身份认证

选择此身份认证方法将允许您指定身份认证凭证。

AWS Secrets Manager

在搜索框中键入相应的名称或 ARN 以搜索令牌。

直接提供用户名和密码

在搜索框中键入相应的名称或 ARN 以搜索令牌。

SSL 客户端身份认证

选择此身份认证方法将允许您浏览 Amazon S3 以选择 Kafka 客户端密钥库的位置。或者，您可以输入 Kafka 客户端密钥库密码和 Kafka 客户端密钥密码。

IAM 身份验证

此身份验证方法不需要任何其他规范，仅在流媒体源为 MSK Kafka 时适用。

SASL/PLAIN 身份验证

选择此身份验证方法将让您能够指定身份验证凭证。

为串流源创建数据目录表

您可以为流式传输源手动创建数据目录表，以指定源数据流属性（包括数据 Schema）。此表用作串流 ETL 任务的数据源。

如果您不知道源数据流中数据的架构，则可以在不使用架构的情况下创建表。然后，当您创建串流 ETL 任务时，您可以打开 AWS Glue 架构检测函数。AWS Glue 会通过流数据来确定架构。

使用AWS Glue控制台、 AWS Command Line Interface (AWS CLI) 或 AWS Glue API 创建表。有关使用 AWS Glue 控制台手动创建表的信息，请参阅 the section called “创建表”。

> **Note**
>
> 您不能使用 AWS Lake Formation 控制台创建表；必须使用AWS Glue控制台。

另外，请考虑以下有关 Avro 格式的串流源或可以应用 Grok 模式的日志数据的信息。

- the section called “Avro 串流源的注释和限制”
- the section called “将 grok 模式应用于串流源”

主题

- Kinesis 数据源
- Kafka 数据源
- AWS Glue 架构注册表源

Kinesis 数据源

在创建表时，请设置以下串流 ETL 属性（控制台）。

源的类型

Kinesis

对于同一账户中的 Kinesis 源：

区域

Amazon Kinesis Data Streams 服务所在的地 AWS 区。“区域”和 Kinesis 流名称会一起转换为“流 ARN”。

示例：https://kinesis.us-east-1.amazonaws.com

Kinesis 流名称

流名称如《Amazon Kinesis Data Streams 开发人员指南》https://docs.aws.amazon.com/streams/latest/dev/kinesis-using-sdk-java-create-stream.html中的创建流所述。

有关其他账户中的 Kinesis 源，请参阅此示例来设置角色和策略以允许进行跨账户访问。配置以下设置：

流 ARN

使用者用于注册的 Kinesis Data Streams ARN。有关更多信息，请参阅中的 Amazon 资源名称 (ARN) 和 AWS 服务命名空间。AWS 一般参考

所担任角色的 ARN

担任角色的 Amazon Resource Name（ARN）。

会话名称（可选）

所担任角色会话的标识符。

在根据不同规则或因为不同原因担任相同角色时，使用角色会话名称对会话进行唯一标识。在跨账户方案中，角色会话名称对于拥有此角色的账户可见，并且可以由拥有该角色的账户记录。所担任角色规则的 ARN 中也使用角色会话名称。这意味着，后续使用临时安全证书的跨账户 API 请求将在其 AWS CloudTrail 日志中向外部账户公开角色会话名称。

为 Amazon Kinesis Data Streams 设置串流 ETL 属性（AWS Glue API 或 AWS CLI）

- 要为同一账户中的 Kinesis 源设置串流 ETL 属性，请在 `CreateTable` API 操作或 `create_table` CLI 命令的 `StorageDescriptor` 结构中指定 `streamName` 和 `endpointUrl` 参数。

```
"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"
 }
 ...
}
```

- 要为其他账户中的 Kinesis 源设置串流 ETL 属性，请在 `CreateTable` API 操作或 `create_table` CLI 命令的 `StorageDescriptor` 结构中指定 `streamARN`、`awsSTSRoleARN` 和 `awsSTSSessionName`（可选）参数。

```
"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 数据源

在创建表时，请设置以下串流 ETL 属性（控制台）。

源的类型

Kafka

对于 Kafka 源：

主题名称

Kafka 中指定的主题名称。

连接

一个引用 Kafka 源的 AWS Glue 连接，如 the section called “为 Kafka 数据流创建连接” 中所述。

AWS Glue 架构注册表源

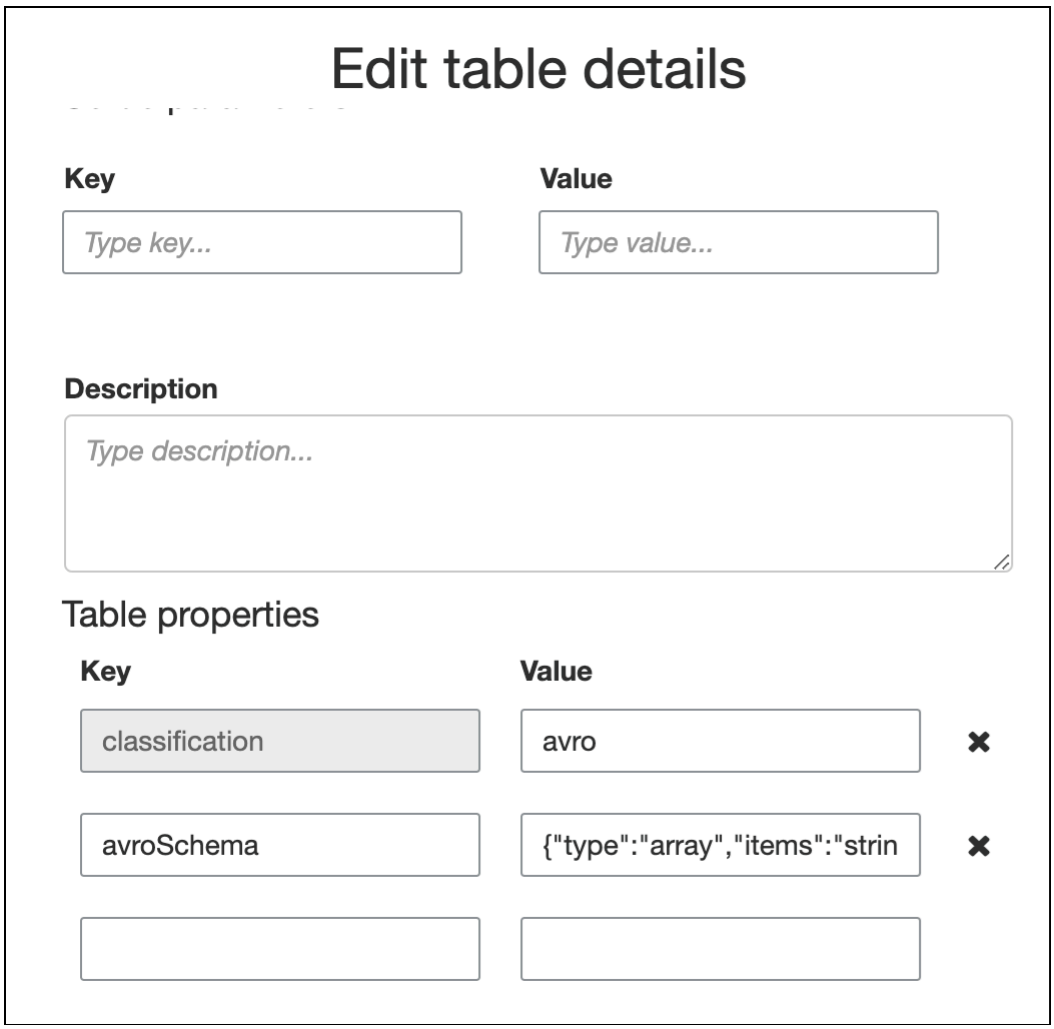
要将 AWS Glue 架构注册表用于串流任务，请按照位于 使用案例：AWS Glue Data Catalog 的说明创建或更新架构注册表。

目前，AWS Glue 串流仅支持架构推理推断设置为 `false` 的 Glue Schema Registry Avro 格式。

Avro 串流源的注释和限制

以下注释和限制适用于 Avro 格式的串流源：

- 启用架构检测后，Avro 架构必须包含在负载中。关闭时，负载应仅包含数据。
- 某些 Avro 数据类型在动态帧中不受支持。在使用 AWS Glue 控制台的创建表向导中的 Define a schema (定义架构) 页面时，您无法指定这些数据类型。在架构检测期间，Avro 架构中不受支持的类型将转换为受支持的类型，如下所示：
  - `EnumType => StringType`
  - `FixedType => BinaryType`
  - `UnionType => StructType`
- 如果使用控制台中的 Define a schema (定义架构) 页面，则架构的隐含根元素类型为 `record`。如果你想要一个除 `record` 以外的根元素类型，例如 `array` 或者 `map`，则不能使用 Define a schema (定义架构) 页面来指定架构。相反，您必须跳过该页并将架构指定为表属性或在 ETL 脚本中指定。
  - 要在表属性中指定架构，请完成创建表向导，编辑表详细信息，并在 Table properties (表属性) 下添加新的键值对。使用密钥 `avroSchema`，然后为值输入架构 JSON 对象，如以下屏幕截图所示。



- 要在 ETL 脚本中指定架构，请修改 `datasource0` 任务语句并将 `avroSchema` 密钥添加到 `additional_options` 参数，如以下 Python 和 Scala 示例所示。

  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()
```

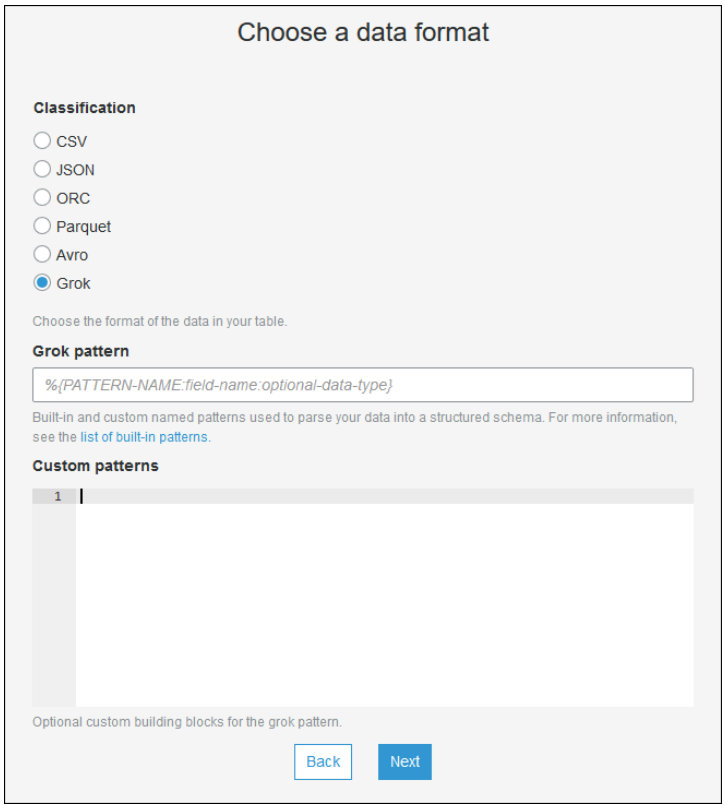
将 grok 模式应用于串流源

您可以为日志数据源创建串流 ETL 任务，并使用 Grok 模式将日志转换为结构化数据。然后，ETL 任务会将数据作为结构化数据源进行处理。在为串流源创建数据目录表时，可以指定要应用的 Grok 模式。

有关 Grok 模式和自定义模式字符串值的信息，请参阅 编写 grok 自定义分类器。

将 grok 模式添加到数据目录表（控制台）

- 使用创建表向导，并使用在 the section called “为串流源创建数据目录表” 中指定的参数创建表。将数据格式指定为 Grok，填写 Grok pattern (Grok 模式) 字段，并可选择在 Custom patterns (optional) (自定义模式（可选）) 下添加自定义模式。



  在每个自定义模式后按 Enter。

将 grok 模式添加到数据目录表（AWS Glue API 或 AWS CLI）

- 添加 `GrokPattern` 参数，并且可以选择将 `CustomPatterns` 参数添加到 `CreateTable` API 操作或 `create_table` CLI 命令。

```
 "Parameters": {
...
    "grokPattern": "string",
    "grokCustomPatterns": "string",
...
},
```

将 `grokCustomPatterns` 表达为字符串，并使用“\n”作为模式之间的分隔符。

以下是指定这些参数的示例。

Example

```
"parameters": {
...
    "grokPattern": "%{USERNAME:username} %{DIGIT:digit:int}",
    "grokCustomPatterns": "digit \d",
...
}
```

定义串流 ETL 作业的作业属性

在 AWS Glue 控制台上定义串流 ETL 任务时，请提供以下特定于流的属性。有关其他任务属性的说明，请参阅 定义 Spark 作业的作业属性。

IAM 角色

指定用于授权用于运行作业、访问流媒体源和访问目标数据存储的资源的 AWS Identity and Access Management (IAM) 角色。

要访问 Amazon Kinesis Data Streams，请将`AmazonKinesisFullAccess` AWS 托管策略附加到该角色，或者附加允许更精细访问权限的类似 IAM 策略。有关示例策略，请参阅使用 IAM 控制对 Amazon Kinesis Data Streams 资源的访问。

有关在 AWS Glue 中运行任务的权限的更多信息，请参阅 AWS Glue 的身份和访问管理。

类型

选择 Spark streaming (Spark 串流)。

AWS Glue 版本

AWS Glue 版本确定可用于任务的 Apache Spark、Python 或 Scala 版本。选择一个选项指定可供作业使用的 Python 或 Scala 版本。AWS Glue支持 Python 3 的 2.0 版是串流 ETL 任务的默认设置。

维护时段

指定可以重新启动流式处理作业的窗口。请参阅 the section called “维护时段”。

作业超时

（可选）输入持续时间（以分钟为单位）。默认值为空。

- 流式处理作业的超时值必须小于 7 天或 10080 分钟。
- 将该值留空时，如果您尚未设置维护时段，则任务将在 7 天后重新启动。如果您设置了维护时段，则该作业将在维护时段内在 7 天后重新启动。

数据来源

指定您在 the section called “为串流源创建数据目录表” 中创建的表。

数据目标

请执行以下操作之一：

- 选择 Create tables in your data target (在数据目标中创建表) 并指定以下数据目标属性。

  数据存储

  选择 Amazon S3 或 JDBC。

  格式

  选择任意格式。所有项都支持流式处理。
- 选择 Use tables in the data catalog and update your data target (使用数据目录中的表并更新数据目标)，然后选择用于 JDBC 数据存储的表。

输出架构定义

请执行以下操作之一：

- 选择 Automatically detect schema of each record (自动检测每条记录的架构) 以启动架构检测。AWS Glue 可以通过串流数据确定架构。

- 选择 Specify output schema for all records (指定所有记录的输出方案) 以使用 Apply Mapping（应用映射）转换来定义输出架构。

Script

（可选）提供您自己的脚本或修改生成的脚本以执行 Apache Spark Structured Streaming 引擎支持的操作。有关可用操作的信息，请参阅流式传输 DataFrames /Dataset 上的操作。

串流 ETL 注释和限制

请记住以下注释和限制：

- 自动解压 AWS Glue 流式传输 ETL 任务仅适用于受支持的压缩类型。另请注意以下几点：
  - Framed Snappy 是指适用于 Snappy 的帧格式。
  - Deflate 在 Glue 版本 3.0 而不是 Glue 版本 2.0 中受支持。
- 使用架构检测时，无法执行串流数据联接。
- AWS Glue 流式传输 ETL 任务不支持对具有 Avro 格式的 AWS Glue 架构注册表使用 Union 数据类型。
- 您的 ETL 脚本可以使用 AWS Glue 的内置转换和 Apache Spark Structured Streaming 的原生转换。有关更多信息，请参阅 Apache Spark 网站上的流媒体 DataFrames /数据集操作或。AWS Glue PySpark 变换参考
- AWS Glue 串流 ETL 任务使用检查点来跟踪已读取的数据。因此，停止并重新启动的任务将从流中停止的位置开始。如果要重新处理数据，您可以删除脚本中引用的检查点文件夹。
- 不支持任务书签。
- 要在作业中使用 Kinesis Data Streams 的增强型扇出功能，请参阅 the section called “在 Kinesis 流作业中使用增强型扇出功能”。
- 如果您使用在 AWS Glue 架构注册表中创建的数据目录表，则当新的架构版本可用时，要反映该新架构，您需要执行以下操作：
  1. 停止与该表关联的任务。
  2. 更新数据目录表的架构。
  3. 重新启动与该表关联的任务。

与 AWS Lake Formation FindMatches 匹配的记录

> **Note**
>
> 目前在 AWS Glue 控制台中的以下区域无法进行记录匹配：中东（阿联酋）、欧洲（西班牙）（eu-south-2）和欧洲（苏黎世）（eu-central-2）。

AWS Lake Formation 提供了机器学习功能，使您能够创建自定义转换来清理数据。当前有一个名为 FindMatches 的可用转换。通过 FindMatches 转换，您可以识别数据集中的重复或匹配记录，即使记录没有公共唯一标识符且没有完全匹配的字段也是如此。这不需要编写任何代码或了解机器学习的工作原理。FindMatches 可用于许多不同的问题，例如：

- 匹配客户：在不同的客户数据库中链接客户记录，即使许多客户字段与数据库不完全匹配（例如，不同的名称拼写、地址差异、缺失或不准确的数据等）。
- 匹配产品：将目录中的产品与其他产品来源相匹配，例如针对竞争对手目录的产品目录（其中条目的结构不同）。
- 改进欺诈检测：识别重复的客户账户，确定新创建的账户何时会（或可能会）与先前已知的欺诈用户匹配。
- 其他匹配问题：匹配地址、电影、零件列表等等。通常，如果有人员可以查看您的数据库行并确定它们是匹配的，那么 FindMatches 转换很可能会帮助到您。

您可以在创建作业时创建这些转换。您创建的转换基于源数据存储架构和您标记的源数据集中的示例数据（我们将此流程称为“指导”转换）。您标记的记录必须位于源数据集中。在此过程中，我们生成一个您标记的文件，然后上传回转换将以某种方式从中学习的文件。在您教授转换后，可以从基于 Spark 的 AWS Glue 任务（PySpark 或 Scala Spark）中调用它，并在具有兼容源数据存储的其他脚本中使用它。

在创建转换后，它存储在 AWS Glue 中。在 AWS Glue 控制台中，您可以管理您创建的转换。在导航窗格的数据集成和 ETL下，数据分类工具 > 记录匹配，您可以编辑并继续教机器学习转换。有关在控制台上管理转换的更多信息，请参阅 在 AWS Glue 控制台上使用机器学习转换。

> **Note**
>
> AWS Glue 版本 2.0 FindMatches 任务使用 Amazon S3 存储桶 `aws-glue-temp-<accountID>-<region>` 在转换处理数据时存储临时文件。您可以在运行完成后，以手动方式或通过设置 Amazon S3 生命周期规则来删除此数据。

机器学习转换的类型

您可以创建机器学习转换来清理您的数据。您可以从 ETL 脚本中调用这些转换。您的数据在一个称为 DynamicFrame 的数据结构中从转换传递到转换，该数据结构是 Apache Spark SQL `DataFrame` 的扩展。`DynamicFrame` 包含您的数据，并引用其架构来处理您的数据。

提供了以下类型的机器学习转换：

查找匹配项

在源数据中查找重复记录。您可以通过标记示例数据集来指示哪些行匹配，从而指导此机器学习转换。机器学习转换通过示例标记数据来了解哪些行应与您指导的内容更为匹配。根据您配置转换的方式，输出为下列项之一：

- 输入表的副本加上 `match_id` 列，其中填充了指示匹配记录集的值。`match_id` 列为任意标识符。任何具有相同 `match_id` 的记录都已被识别为彼此匹配。具有不同 `match_id` 的记录不匹配。
- 删除了重复行的输入表的副本。如果找到多个重复项，则保留带最小主键的记录。

查找递增匹配项

还可以将查找匹配项转换配置为在现有帧和递增帧中查找匹配项，然后作为输出返回一列，其中包含每个匹配组一个唯一的 ID。

有关更多信息，请参阅：查找递增匹配项。

使用 FindMatches 转换

您可以使用 `FindMatches` 转换查找源数据中的重复记录。生成或提供标签文件以帮助指导转换。

> **Note**
>
> 目前，在以下区域不支持使用自定义加密密钥的 `FindMatches` 转换：

- 亚太地区（大阪）- ap-northeast-3

要开始使用 FindMatches 转换，您可以按照以下步骤操作。有关更高级和更详细的示例，请参阅 AWS Big Data Blog: Harmonize data using AWS Glue and AWS Lake Formation FindMatches ML to build a customer 360 view。

开始使用查找匹配项转换

执行以下步骤来开始使用 FindMatches 转换：

1. 在 AWS Glue Data Catalog 中为要清理的源数据创建表。有关如何创建爬网程序的信息，请参阅在 AWS Glue 控制台上使用爬网程序。

   如果源数据是一个基于文本的文件（如逗号分隔值 (CSV) 文件），请考虑：

   - 将输入记录 CSV 文件和标签文件保存在单独的文件夹中。否则，AWS Glue 爬网程序可能会将其视为同一个表的多个部分，并在数据目录中错误地创建表。
   - 除非您的 CSV 文件仅包含 ASCII 字符，否则请确保将不含 BOM（字节顺序标记）的 UTF-8 编码用于 CSV 文件。Microsoft Excel 通常在 UTF-8 CSV 文件的开头添加 BOM。要删除它，请在文本编辑器中打开 CSV 文件，并将该文件重新保存为 UTF-8 without BOM (不含 BOM 的 UTF-8)。
2. 在 AWS Glue 控制台中，创建一个作业，然后选择 Find matches (查找匹配项) 转换类型。

   > **Important**
   >
   > 您为作业选择的数据源表不能具有 100 个以上的列。

3. 告诉 AWS Glue 生成标签文件，方法是选择 Generate labeling file (生成标签文件)。AWS Glue 在为每个 labeling_set_id 分组相似记录时进行第一次传递，以便您可以查看这些分组。您可以在 label 列中标记匹配项。
   - 如果您已有一个标签文件（即指示匹配行的记录示例），请将该文件上传到 Amazon Simple Storage Service（Amazon S3）。有关标签文件格式的信息，请参阅标签文件格式。继续执行步骤 4。
4. 下载标签文件并按照 标签 部分中的说明标记文件。
5. 上传更正后的标签文件。AWS Glue 运行任务以指导转换过程如何查找匹配项。

   在 Machine learning transforms (机器学习转换) 列表页面上，选择 History (历史记录) 选项卡。此页面指明 AWS Glue 何时执行以下任务：

- Import labels (导入标签)
- Export labels (导出标签)
- Generate labels (生成标签)
- Estimate quality (估计质量)

6. 要创建更好的转换，您可以迭代方式下载、标记和上传标签文件。在初始运行中，可能会有更多记录不匹配。但是，当您通过验证标签文件继续指导它时，AWS Glue 会学习。
7. 通过评估查找匹配项转换的性能和结果来评估和调整转换。有关更多信息，请参阅 在 AWS Glue 中优化机器学习转换。

标签

在 FindMatches 生成标签文件时，将从源表中选择记录。根据之前的训练，FindMatches 标识要从中学习的最有价值的记录。

贴标签行为是编辑标签文件（我们建议使用电子表格，例如 Microsoft Excel），并在标识匹配和不匹配记录的 label 列中添加标识符或标签。在源数据中对匹配项进行清晰一致的定义非常重要。FindMatches 了解您指定为匹配项（或不匹配项）的记录，并使用您的决定来学习如何查找重复记录。

当 FindMatches 生成标签文件时，会生成大约 100 条记录。这 100 条记录通常将分成 10 个标签集，其中每个标签集由 FindMatches 生成的一个唯一 labeling_set_id 标识。每个标签集应被视为独立于其他标签集的单独的贴标签任务。您的任务是标识每个标签集中的匹配记录和不匹配记录。

有关在电子表格中编辑标签文件的提示

在电子表格应用程序中编辑标签文件时，请考虑：

- 当列字段完全展开时，文件可能不会打开。您可能需要展开 labeling_set_id 和 label 列来查看这些单元格中的内容。
- 如果主键列是数字（例如 long 数据类型），则电子表格可能将它解释为数字并更改值。必须将此键值视为文本。要纠正此问题，请将主键列中的所有单元格格式化为 Text data (文本数据)。

标签文件格式

由 AWS Glue 生成的标签文件，用于指导 FindMatches 转换使用以下格式。如果您生成自己的 AWS Glue 文件，则它也必须采用此格式：

- 它是一个逗号分隔值（CSV）文件。

- 必须使用 UTF-8 对它进行编码。如果您使用 Microsoft Windows 编辑文件，则可使用 cp1252 对其进行编码。
- 它必须位于 Amazon S3 位置才能传递到 AWS Glue。
- 为每个标记任务使用中等数量的行。建议每个任务 10-20 行，但每个任务可以接受 2-30 行。建议不要执行具有 50 个以上的行的任务，否则可能会导致不良结果或系统故障。
- 如果已标记的数据由标记为“匹配”或“不匹配”的记录对组成，这种情况是可以接受的。这些标记对可表示为大小为 2 的标签集。在此情况下，如果两条记录匹配，则用字母“A”标记它们，但如果两条记录不匹配，则将一条记录标记为“A”，并将另一条记录标记为“B”。

> **Note**
>
> 由于它具有其他列，因此，标签文件具有与包含源数据的文件不同的架构。将标签文件放置到与任何转换输入 CSV 文件不同的文件夹中，以便 AWS Glue 爬网程序不会在数据目录中创建表时考虑它。否则，AWS Glue 爬网程序创建的表可能无法正确展示您的数据。

- AWS Glue 需要前 2 个列（`labeling_set_id`、`label`）。其余列必须与要处理的数据的架构匹配。
- 对于每个 `labeling_set_id`，您使用同一标签来标识所有匹配的记录。标签是 `label` 列中放置的唯一字符串。我们建议使用包含简单字符（例如 A、B、C 等）的标签。标签区分大小写，并且输入到 `label` 列中。
- 包含相同的 `labeling_set_id` 和相同的标签的行被视为将标记为匹配项。
- 包含相同的 `labeling_set_id` 和不同的标签的行被视为将标记为不匹配项
- 包含不同的 `labeling_set_id` 的行被视为不传达任何支持或反对匹配的信息。

下面的示例说明了如何标记数据：

labeling_set_id	label	first_name	last_name	生日
ABC123	A	John	Doe	04/01/1980
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

labeling_set_id	label	first_name	last_name	生日
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

- 在上面的示例中，我们将 John/Johnny/Jon Doe 标识为匹配项，并告知系统这些记录与 Jane Smith 不匹配。另外，我们告诉系统 Richard 和 Rich Jones 是同一个人，但是这些记录与 Sarah Jones/Jones-Walker 和 Richie Jones Jr 不匹配。
- 正如您所看到的，标签的范围限于 `labeling_set_id`。因此，标签不会跨 `labeling_set_id` 边界。例如，`labeling_set_id` 1 中的标签“A”与 `labeling_set_id` 2 中的标签“A”没有任何关系。
- 如果一条记录在标签集中没有任何匹配项，请为该记录分配一个唯一标签。例如，Jane Smith 与标签集 ABC123 中的任何记录都不匹配，因此它是该标签集中唯一带有标签 B 的记录。
- 标记集“GHI678”表明一个标签集可以仅包含两条记录，将为这两条记录分配同一标签来表明它们匹配。同样，“XYZABC”显示已分配不同标签的两条记录来表明它们不匹配。
- 请注意，有时标签集可能不包含匹配项（也就是说，您为标签集中的每条记录分配一个不同的标签），或者标签集中的所有记录可能都“相同”（您为它们分配了相同的标签）。只要您的标签集包含了根据您的标准是“相同的”和“不相同的”记录的示例，就可以了。

> **Important**
>
> 确认要传递到 AWS Glue 的 IAM 角色具有对包含标签文件的 Amazon S3 存储桶的访问权限。按照惯例，AWS Glue 策略向名称前缀为 aws-glue- 的 Amazon S3 存储桶或文件夹授予权限。如果您的标签文件位于其他位置，则在 IAM 角色中添加对该位置的权限。

在 AWS Glue 中优化机器学习转换

您可以在 AWS Glue 中优化机器学习转换来改进数据清理作业的结果，从而实现您的目标。要改进转换，您可以通过生成标记集，添加标签，然后重复这些步骤几次，直到获得所需结果来指导转换。您也可以通过更改一些机器学习参数来进行优化。

有关机器学习转换的更多信息，请参阅 与 AWS Lake Formation FindMatches 匹配的记录。

主题

机器学习测量值

要了解用于优化机器学习转换的测量值，您应熟悉以下术语：

真阳性 (TP)

转换正确找到的数据中的匹配项有时称作命中。

真阴性 (TN)

转换正确拒绝的数据中的不匹配项。

假阳性 (FP)

转换错误地分类为匹配项的数据中的不匹配项有时称作假警报。

假阴性 (FN)

转换未找到的数据中的匹配项有时称作未命中。

有关机器学习中使用的术语的更多信息，请参阅 Wikipedia 中的 混淆矩阵。

要优化您的机器学习转换，您可以在转换的 Advanced properties (高级属性) 中更改以下测量值。

- Precision (查准率) 衡量转换在其标识为阳性的记录总数（真阳性和假阳性）中找到真阳性的程度。有关更多信息，请参阅 Wikipedia 中的查准率和查全率。

- Recall (查全率) 衡量转换从源数据中的全部记录中找到真阳性的程度。有关更多信息，请参阅 Wikipedia 中的查准率和查全率。
- Accuracy (准确性) 衡量转换发现真阳性和真阴性的程度。提高准确性需要更多的机器资源和成本。但这也会导致查全率提高。有关更多信息，请参阅 Wikipedia 中的准确性和查准率。
- Cost (成本) 衡量运行转换所消耗的计算资源（从而产生资金）的数量。

在查准率和查全率之间做出决定

每个 FindMatches 转换均包含一个 precision-recall 参数。您可以使用此参数指定下列项之一：

- 如果您更关心转换错误地报告两个记录匹配，而实际上它们不匹配，则您应强调 precision (查准率)。
- 如果您更关心转换未能检测到真正匹配的记录，则您应强调 recall (查全率)。

您可以在 AWS Glue 控制台上或使用 AWS Glue 机器学习 API 操作进行此权衡。

何时倾向于查准率

如果您更关心 FindMatches 导致一对实际不匹配的记录进行匹配的风险，请倾向于查准率。要倾向于查准率，请选择 higher (较大) 查准率-查全率权衡值。对于较大的值，FindMatches 转换需要更多的证据来决定是否应匹配一对记录。将转换调整为偏向于表示记录不匹配。

例如，假设您使用 FindMatches 检测视频目录中的重复项，并且您向转换提供更大的查准率-查全率值。如果您的转换错误地检测到 Star Wars: A New Hope 与 Star Wars: The Empire Strikes Back 相同，则可能会为需要 A New Hope 的客户显示 The Empire Strikes Back。这将是一个糟糕的客户体验。

不过，如果转换无法检测到 Star Wars: A New Hope 和 Star Wars: Episode IV—A New Hope 是相同的项，则客户最初可能会感到困惑，不过可能最终会将其视为相同。这将是一个错误，但不像以前的情况那么糟糕。

何时倾向于查全率

如果您更关心 FindMatches 转换结果可能无法检测到实际匹配的一对记录的风险，请倾向于查全率。要倾向于查全率，请选择 lower (较小) 的查准率-查全率权衡值。对于较小的值，FindMatches 转换需要更少的证据来决定是否应匹配一对记录。将转换调整为偏向于表示记录匹配。

例如，这可能是安全组织的优先事项。假设您将客户与一系列已知的欺诈者进行匹配，并且确定客户是否为欺诈者非常重要。您使用 FindMatches 将欺诈者名单与客户名单进行匹配。每当 FindMatches 检测到两个名单之间的匹配项时，都会指派一名审计人员来验证该人员实际上是否为欺诈者。您的组织可能更愿意选择查全率而不是查准率。换句话说，当客户不是欺诈者时，您宁愿让审计人员手动审查并拒绝某些情况，而不是未能识别出客户实际上在欺诈者名单上。

如何倾向于查准率和查全率

提高查准率和查全率的最佳方法是标记更多数据。在标记更多数据时，FindMatches 转换的总体准确性将提高，从而提高查准率和查全率。然而，即使对于最准确的转换，也始终存在一个灰色区域，您需要在该区域中尝试倾向于查准率或查全率，或者在中间选择一个值。

在准确性和成本之间做出决定

每个 FindMatches 转换均包含一个 accuracy-cost 参数。您可以使用此参数指定下列项之一：

- 如果您更关心准确报告两个记录匹配的转换，则应强调准确性。
- 如果您更关心转换的运行成本或速度，则应强调更低成本。

您可以在 AWS Glue 控制台上或使用 AWS Glue 机器学习 API 操作进行此权衡。

何时倾向于准确性

如果您更关心 find matches 结果不包含匹配项的风险，则倾向于准确性。要倾向于准确性，请选择较大准确性-成本权衡值。对于较大的值，FindMatches 转换需要更多时间来更详细地搜索正确匹配的记录。请注意，此参数不会使错误地将不匹配记录对称为匹配项的可能性降低。转换将调整为倾向于花更多时间查找匹配项。

何时倾向于成本

如果您更关心运行 find matches 转换的成本，而不是找到多少匹配项，请倾向于成本。要倾向于成本，请选择较小 准确性-成本权衡值。对于较小的值，运行 FindMatches 转换所需的资源更少。转换将调整为倾向于查找更少的匹配项。如果在倾向于较低成本时结果是可接受的，请使用此设置。

如何倾向于准确性和较低成本

检查更多记录对以确定它们是否匹配需要更多的机器时间。如果您想减少成本而不降低质量，则可执行以下几个步骤：

- 消除数据源中您不关心匹配的记录。

- 从您的数据源中消除您确信在做出匹配/不匹配决策时没有用的列。确定这一点的一个好方法是消除您认为不会影响您就一组记录是否“相同”做出决定的列。

使用匹配项置信度分数估算匹配项质量

匹配项置信度分数可以提供对 FindMatches Match 找到的匹配项质量的估计值，以区分机器学习模型高度可信、不确定或不可能的匹配记录。匹配项置信度分数将介于 0 和 1 之间，其中分数越高，就意味着相似性越高。通过检查匹配项置信度分数，您可以区分系统高度信任的匹配项集群（您可能决定合并）、系统不确定的集群（您可能决定由人类审核），以及系统认为不可能的集群（您可能决定拒绝）。

如果您看到匹配项置信度分数很高，但又确定不存在匹配项，或者您看到分数很低，但又确定实际上存在匹配项，在此类情况下，您可能想调整训练数据。

当存在大型行业数据集时，审核每个 FindMatches 决策在此情况下并不可行，因此置信度分数特别有用。

在 AWS Glue 版本 2.0 或更高版本中提供了匹配项置信度分数。

生成匹配项置信度分数

您可以生成匹配项置信度分数，方法是在调用 `FindMatches` 或 `FindIncrementalMatches` API 时，将 `computeMatchConfidenceScores` 的布尔值设置为 True（真）。

AWS Glue 会将一个新的 `column match_confidence_score` 添加到输出。

匹配项评分示例

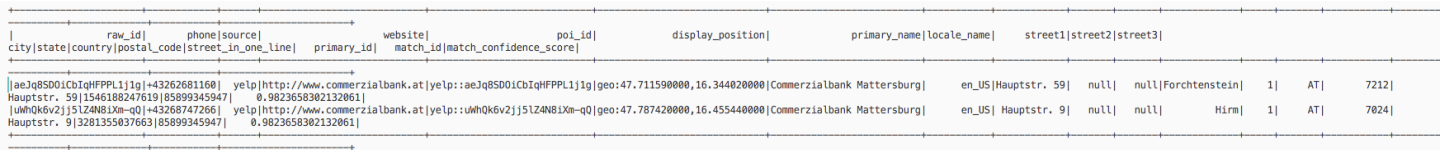
例如，考虑以下匹配记录：

分数 >= 0.9

匹配记录的摘要：

```
  primary_id  |   match_id  | match_confidence_score

3281355037663    85899345947   0.9823658302132061
1546188247619    85899345947   0.9823658302132061
```

详细信息:



```
+----------+----------+------+-----------------------+-----------------------+-----------------------+----------+-----------+-----------+-----------+-----------+-----+-------+-----------+-----------+-----------+-----------+-----------+-----------+
|raw_id    |phone     |source|website                |poi_id                 |display_position       |primary_name|locale_name|street1    |street2    |street3    |city |state|country|postal_code|street_in_one_line|primary_id|match_id|match_confidence_score|
+----------+----------+------+-----------------------+-----------------------+-----------------------+----------+-----------+-----------+-----------+-----------+-----+-------+-----------+-----------+-----------+-----------+-----------+-----------+
|aeJq8SDOiCbIqHFPPL1j1g|+43262681160|yelp|http://www.commerzialbank.at|yelp::aeJq8SDOiCbIqHFPPL1j1g|geo:47.711590000,16.344020000|Commerzialbank Mattersburg|en_US|Hauptstr. 59|null|null|Forchtenstein|1|AT|7212|Hauptstr. 59|154618824761 9|85899345947|0.9823658302132061|
|uWhQk6v2jj5lZ4N8iXm-qQ|+43268747266|yelp|http://www.commerzialbank.at|yelp::uWhQk6v2jj5lZ4N8iXm-qQ|geo:47.787420000,16.455440000|Commerzialbank Mattersburg|en_US|Hauptstr. 9|null|null|Hirm|1|AT|7024|Hauptstr. 9|32813550376 63|85899345947|0.9823658302132061|
+----------+----------+------+-----------------------+-----------------------+-----------------------+----------+-----------+-----------+-----------+-----------+-----+-------+-----------+-----------+-----------+-----------+-----------+-----------+
```

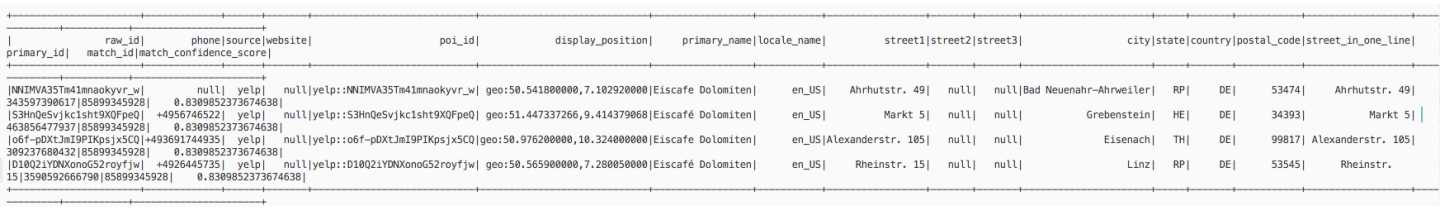
通过此示例，我们可以看到两条记录非常相似并且共享 display_position、primary_name 和 street name。

分数 >= 0.8 且分数 < 0.9

匹配记录的摘要：

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

详细信息:



```
+----------+----------+------+-----------------------+-----------------------+-----------------------+----------+-----------+-----------+-----------+-----------+-----+-------+-----------+-----------+-----------+-----------+-----------+-----------+
|raw_id    |phone     |source|website                |poi_id                 |display_position       |primary_name|locale_name|street1    |street2    |street3    |city |state|country|postal_code|street_in_one_line|primary_id|match_id|match_confidence_score|
+----------+----------+------+-----------------------+-----------------------+-----------------------+----------+-----------+-----------+-----------+-----------+-----+-------+-----------+-----------+-----------+-----------+-----------+-----------+
|NNIMVA35Tm41mnaokyvr_w|null|yelp|null|yelp::NNIMVA35Tm41mnaokyvr_w|geo:50.541800000,7.102920000|Eiscafe Dolomiten|en_US|Ahrhutstr. 49|null|null|Bad Neuenahr~Ahrweiler|RP|DE|53474|Ahrhutstr. 49|343597390617|85899345928|0.8309852373674638|
|S3HnQeSvjkc1sht9XQFpeQ|+4956746522|yelp|null|yelp::S3HnQeSvjkc1sht9XQFpeQ|geo:51.447337266,9.414379068|Eiscafé Dolomiten|en_US|Markt 5|null|null|Grebenstein|HE|DE|34393|Markt 5|463856477937|85899345928|0.8309852373674638|
|o6f-pDXtJmI9PIKpsjx5CQ|+4936917744935|yelp|null|yelp::o6f-pDXtJmI9PIKpsjx5CQ|geo:50.976200000,10.324000000|Eiscafe Dolomiten|en_US|Alexanderstr. 105|null|null|Eisenach|TH|DE|99817|Alexanderstr. 105|309237680432|85899345928|0.8309852373674638|
|D10Q2iYDNXonoG52royfjw|+4926445735|yelp|null|yelp::D10Q2iYDNXonoG52royfjw|geo:50.565900000,7.280050000|Eiscafé Dolomiten|en_US|Rheinstr. 15|null|null|Linz|RP|DE|53545|Rheinstr. 15|3590592666790|85899345928|0.8309852373674638|
+----------+----------+------+-----------------------+-----------------------+-----------------------+----------+-----------+-----------+-----------+-----------+-----+-------+-----------+-----------+-----------+-----------+-----------+-----------+
```

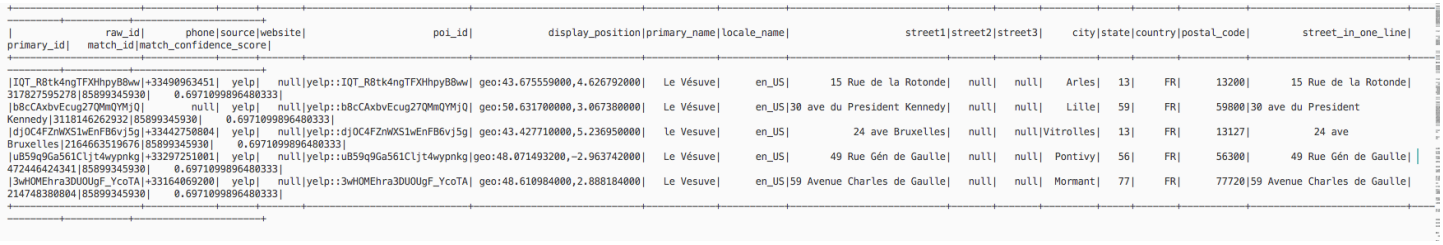
通过此示例，我们可以看到这些记录共享相同的 primary_name 和 country。

分数 >= 0.6 且分数 < 0.7

匹配记录的摘要：

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

详细信息:



```
+----------+--------------+--------+-------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+-----+-----+-----+-----+-----+--------------------+--------------------+
|    raw_id|         phone|  source|website|              poi_id|    display_position|primary_name|locale_name|             street1|             street2|             street3|         city|state|country|postal_code|  street_in_one_line|          primary_id|
|  match_id|match_confidence_score|
+----------+--------------+--------+-------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+-----+-----+-----+-----+-----+--------------------+--------------------+
|IQT_R8tk4ngTFXHhpyB8ww|+33490963451|    yelp|   null|yelp::IQT_R8tk4ngTFXHhpyB8ww|geo:43.675559000,4.626792000|   Le Vésuve|      en_US|       15 Rue de la Rotonde|        null|        null|       Arles|   13|     FR|      13200|       15 Rue de la Rotonde|
317827595278|85899345930|    0.6971099896480333|
|b8cCAxbvEcug27QMmQYMjQ|        null|    yelp|   null|yelp::b8cCAxbvEcug27QMmQYMjQ|geo:50.631700000,3.067380000|   Le Vésuve|      en_US|30 ave du President Kennedy|        null|        null|       Lille|   59|     FR|      59800|30 ave du President
Kennedy|3118146262932|85899345930|    0.6971099896480333|
|djOC4FZnWXS1wEnFB6vj5g|+33442750804|    yelp|   null|yelp::djOC4FZnWXS1wEnFB6vj5g|geo:43.427710000,5.236950000|   le Vesuve|      en_US|          24 ave Bruxelles|        null|        null|Vitrolles|   13|     FR|      13127|          24 ave
Bruxelles|2164663519676|85899345930|    0.6971099896480333|
|uB59q9Ga561Cljt4wypnkg|+33297251001|    yelp|   null|yelp::uB59q9Ga561Cljt4wypnkg|geo:48.071493200,-2.963742000|   Le Vésuve|      en_US|      49 Rue Gén de Gaulle|        null|        null|  Pontivy|   56|     FR|      56300|      49 Rue Gén de Gaulle|
472446424341|85899345930|    0.6971099896480333|
|3wHOMEhra3DUOUgF_YcoTA|+33164069200|    yelp|   null|yelp::3wHOMEhra3DUOUgF_YcoTA|geo:48.610984000,2.888184000|   Le Vesuve|      en_US|59 Avenue Charles de Gaulle|        null|        null|  Mormant|   77|     FR|      77720|59 Avenue Charles de Gaulle|
214748380804|85899345930|    0.6971099896480333|
+----------+--------------+--------+-------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+-----+-----+-----+-----+-----+--------------------+--------------------+
```

通过此示例，我们可以看到这些记录仅共享相同的 `primary_name`。

有关更多信息，请参阅：

- 步骤 5：使用机器学习转换添加和运行作业
- PySpark：FindMatches 类
- PySpark：FindIncrementalMatches 类
- Scala：FindMatches 类
- Scala：FindIncrementalMatches 类

指导查找匹配项转换

必须为每个 `FindMatches` 转换指导什么应被视为匹配项，什么不应被视为匹配项。您可以通过将标签添加到文件并将您的选择上传到 AWS Glue 来指导您的转换。

您可以在 AWS Glue 控制台上或使用 AWS Glue 机器学习 API 操作来编排此标签。

我应添加多少次标签？ 我需要多少个标签？

这些问题的答案主要取决于您。您必须评估 `FindMatches` 是否提供您所需的准确性，以及您是否认为额外的标签工作对您来说是值得的。决定这一点的最佳方法是查看“查准率”、“查全率”和“查准率-查全率曲线下的面积”指标，当您选择 AWS Glue 控制台上的 Estimate quality (估计质量) 时，可以生成这些指标。在标记多组任务后，请重新运行这些指标并验证它们是否已得到改进。如果在标记几组任务后，您没有看到您关注的指标有所改进，则转换质量可能已达到稳定水平。

为何需要真阳性和真阴性标签？

`FindMatches` 转换需要阳性和阴性示例来了解您认为的匹配项。如果标记 `FindMatches` 生成的训练数据（例如，使用 I do not have labels (我没有标签) 选项），则 `FindMatches` 会尝试为您生成一组“标签集 ID”。在每个任务中，您向一些记录添加相同的“标签”，向其他记录添加不同的“标签”。换

句话说，任务通常不是完全相同的，也不是完全不同的（但如果某个特定任务完全“相同”或完全“不相同”，也没关系）。

如果使用 Upload labels from S3 (从 S3 上载标签) 选项来设定 FindMatches 转换，请尝试同时包含匹配记录和不匹配记录的示例。仅具有一种类型是可以接受的。这些标签可帮助您构建更准确的 FindMatches 转换，但您仍需使用 Generate labeling file (生成标签文件) 选项来为生成的部分记录添加标签。

如何强制转换过程与我必须的内容完全匹配？

FindMatches 转换从您提供的标签中学习，因此它可能会生成不遵循提供的标签的记录对。要强制 FindMatches 转换遵守标签，请在 FindMatchesParameter 中选择 EnforceProvidedLabels。

当 ML 转换将项目标识为非真匹配项的匹配项时，您可以使用哪些方法？

您可以使用以下方法：

- 将 precisionRecallTradeoff 增至更大的值。这最终会导致找到较少的匹配项，但是当它达到足够高的值时，它也应分解您的大集群。
- 接受与错误结果对应的输出行并将其重新格式化为标签集（删除 match_id 列并添加 labeling_set_id 和 label 列）。如有必要，可拆分（细分）成多个标签集，以确保贴标签者在分配标签时能记住每个标签集。然后，正确标记匹配集，上传标签文件并将其附加到现有标签。这可能会让您的转换器充分了解它需要什么来理解模式。
- （高级）最后，查看该数据以了解是否存在可以检测到系统未注意到的模式。通过使用标准 AWS Glue 函数规范化数据来预处理数据。通过将重要程度不同的数据分离到它们自己的列中，突出显示您希望算法从中学到的内容。或者，从已知其数据相关的列构造组合列。

在 AWS Glue 控制台上使用机器学习转换

您可以使用创建 AWS Glue 可用于清理数据的自定义机器学习转换。在 AWS Glue 控制台上创建作业时，可以使用这些转换。

有关如何创建机器学习转换的信息，请参阅 与 AWS Lake Formation FindMatches 匹配的记录。

主题

转换属性

要查看现有的机器学习转换，请登录并打开AWS Glue控制台 AWS Management Console，网址为 https://console.aws.amazon.com/glue/。在导航窗格的数据集成和 ETL下，选择数据分类工具 > 记录匹配。

每个转换的属性：

转换名称

在创建转换时为其提供的唯一名称。

ID

转换的唯一标识符。

标签数

为帮助指导转换而提供的标签文件中的标签数。

Status

指示转换是否为 Ready (准备) 或 Needs training (需要培训)。要在任务中成功运行机器学习转换，则必须是 Ready (准备)。

创建时间

创建转换的日期。

已修改

上次更新转换的日期。

描述

为转换提供的描述（如果已提供）。

AWS Glue 版本

使用的 AWS Glue 版本。

运行 ID

在创建转换时为其提供的唯一名称。

任务类型

机器学习转换的类型；例如，Find matching records (查找匹配记录)。

Status

表示任务运行的状态。可能的状态包括：

- Starting
- Running
- Stopping
- Stopped
- 成功
- 失败
- 超时

错误

如果状态为“失败”，会显示一条错误消息，指出失败的原因。

添加和编辑机器学习转换

您可以在 AWS Glue 控制台上查看、删除、设置和指导或优化转换。选中列表中转换旁边的复选框，选择 Action (操作)，然后选择要采取的操作。

创建新的 ML 转换

要添加新的机器学习转换，请选择创建转换。按照添加作业向导中的说明操作。有关更多信息，请参阅与 AWS Lake Formation FindMatches 匹配的记录。

第 1 步。设置转换属性。

1. 输入名称和描述（可选）。
2. （可选）设置安全配置。请参阅 将数据加密与机器学习转换结合使用。
3. （可选）设置任务执行设置。任务执行设置允许您自定义任务的运行方式。选择工作线程类型、工作线程数量、任务超时（以分钟为单位）、重试次数和 AWS Glue 版本。
4. （可选）设置标签。标签是您可以分配给 AWS 资源的标签。每个标签都由一个键和一个可选值组成。标签可用于搜索和筛选您的资源或跟踪您的 AWS 成本。

第 2 步。选择表和主键。

1. 选择 AWS Glue Catalog 数据库和表。
2. 从所选表中选择一个主键。主键列通常包含数据来源中每条记录的唯一标识符。

第 3 步。选择调整选项。

1. 对于查全率与查准率，选择调整值以调整转换，使其偏向查全率或查准率。默认情况下，平衡处于选中状态，但您可以选择偏向查全率或偏向查准率，或者选择自定义并输入介于 0.0 和 1.0（含）之间的值。
2. 要获得更低的成本与查准率，请选择调整值以偏向降低成本或查准率，或者选择自定义并输入介于 0.0 和 1.0（含）之间的值。
3. 对于强制匹配，如果您想通过强制输出与使用的标签匹配来教授 ML 转换，请选择强制输出以匹配标签。

第 4 步。审核和创建

1. 查看步骤 1 - 3 的选项。
2. 对于需要修改的任何步骤，选择编辑。选择创建转换以完成创建转换向导。

将数据加密与机器学习转换结合使用

在将机器学习转换添加到 AWS Glue 时，您可以选择指定与数据源或数据目标关联的安全配置。如果用于存储数据的 Amazon S3 存储桶使用安全配置加密，请在创建转换时指定相同的安全配置。

您也可以选择使用服务器端加密 AWS KMS (SSE-KMS) 来加密模型和标签，以防止未经授权的人员对其进行检查。如果您选择此选项，则系统会提示您 AWS KMS key 按名称选择，或者您可以选择输入密钥 ARN。如果选择输入 KMS 密钥的 ARN，则会出现第二个字段，您可以在其中输入 KMS 密钥 ARN。

> **Note**
>
> 目前，在以下区域不支持使用自定义加密密钥的 ML 转换：
>
> - 亚太地区（大阪）- `ap-northeast-3`

查看转换详细信息

查看转换属性

转换属性页面包含转换的属性。它显示有关转换定义的详细信息，包括：

- Transform name (转换名称) 显示转换的名称。

- Type (类型) 列出了转换的类型。
- Status (状态) 显示是否可在脚本或作业中使用转换。
- Force output to match labels (强制输出与标签匹配) 显示转换是否强制输出与用户提供的标签匹配。
- Spark 版本与添加转换时在 Task run properties (任务运行属性) 中选择的 AWS Glue 版本有关。建议大多数用户使用 AWS Glue 1.0 和 Spark 2.4。有关更多信息，请参阅 AWS Glue 版本。

“历史记录”、“估算质量”和“标签”选项卡

转换详细信息包括您在创建转换时定义的信息。要查看某个转换的详细信息，请在 Machine learning transforms (机器学习转换) 列表中选择该转换，然后查看以下选项卡上的信息：

- 历史记录
- 估计质量
- 标签

历史记录

History (历史记录) 选项卡显示转换任务运行历史记录。运行几种类型的任务来指导转换。对于每个任务，运行指标包括：

- Run ID (运行 ID) 是 AWS Glue 为此任务的每次运行创建的标识符。
- Task type (任务类型) 显示任务运行的类型。
- Status (状态) 显示列出的每个任务的成功状态，其中最新运行位于顶部。
- Error (错误) 在运行不成功时显示错误消息的详细信息。
- Start time (开始时间) 显示任务的开始日期和时间（本地时间）。
- 结束时间显示任务结束的日期和时间（本地时间）。
- Logs (日志) 链接到写入此作业运行的 `stdout` 的日志。

  日志链接会将您带到 Amazon CloudWatch 日志。在这里，您可以查看有关在中创建的表的详细信息 AWS Glue Data Catalog 以及遇到的任何错误。您可以在 CloudWatch 控制台上管理日志保留期。默认日志保留为 `Never Expire`。有关如何更改保留期的更多信息，请参阅 Amazon Log CloudWatch s 用户指南中的更改 CloudWatch 日志数据保留期。
- 下载标签文件显示生成的标记文件的 Amazon S3 的链接。

估计质量

Estimate quality (估计质量) 选项卡显示用于测量转换质量的指标。通过使用标签数据的子集与您提供的标签比较变换匹配预测来计算估计值。这些估计值是近似值。可以从该选项卡调用 Estimate quality (估计质量) 任务运行。

Estimate quality (估计质量) 选项卡显示了上次 Estimate quality (估计质量) 运行的指标，包括以下属性：

- Area under the Precision-Recall curve (查准率-查全率曲线下的面积) 是估算变换整体质量上限的单个数字。它与查准率-查全率参数的选择无关。较高的值表明您具有更有吸引力的查准率-查全率权衡。
- Precision (查准率) 估算转换在预测匹配时正确的频率。
- Recall upper limit (查全率上限) 估算，对于实际匹配，转换预测匹配项的频率。
- F1 估算转换的准确性介于 0 和 1 之间，其中 1 是最大准确性。有关更多信息，请参阅 Wikipedia 中的 F1 分数。
- Column importance (列重要性) 表显示了每列的列名和重要性分数。列重要性通过识别记录中最常用于匹配的列，帮助您了解列对模型的贡献。此数据可以提示您添加或更改标签集以提高或降低列的重要性。

  重要性列提供每列的数值分数，作为不大于 1.0 的小数。

有关理解质量估计与实际质量的信息，请参阅 质量估计值与 end-to-end （真实）质量。

有关优化您的转换的更多信息，请参阅 在 AWS Glue 中优化机器学习转换。

质量估计值与 end-to-end （真实）质量

AWS Glue 通过向内部机器学习模型显示一些记录对来估计转换的质量，这些记录对您提供了匹配的标签，但是模型以前没有看到过。这些质量估计是机器学习模型质量的函数（受标记为“指导”转换的记录数量的影响）。或真实召回率（不是由自动计算的`ML transform`）还受到`ML transform`过滤机制的影响，该机制为机器学习的模型提出了各种可能的匹配项。 end-to-end

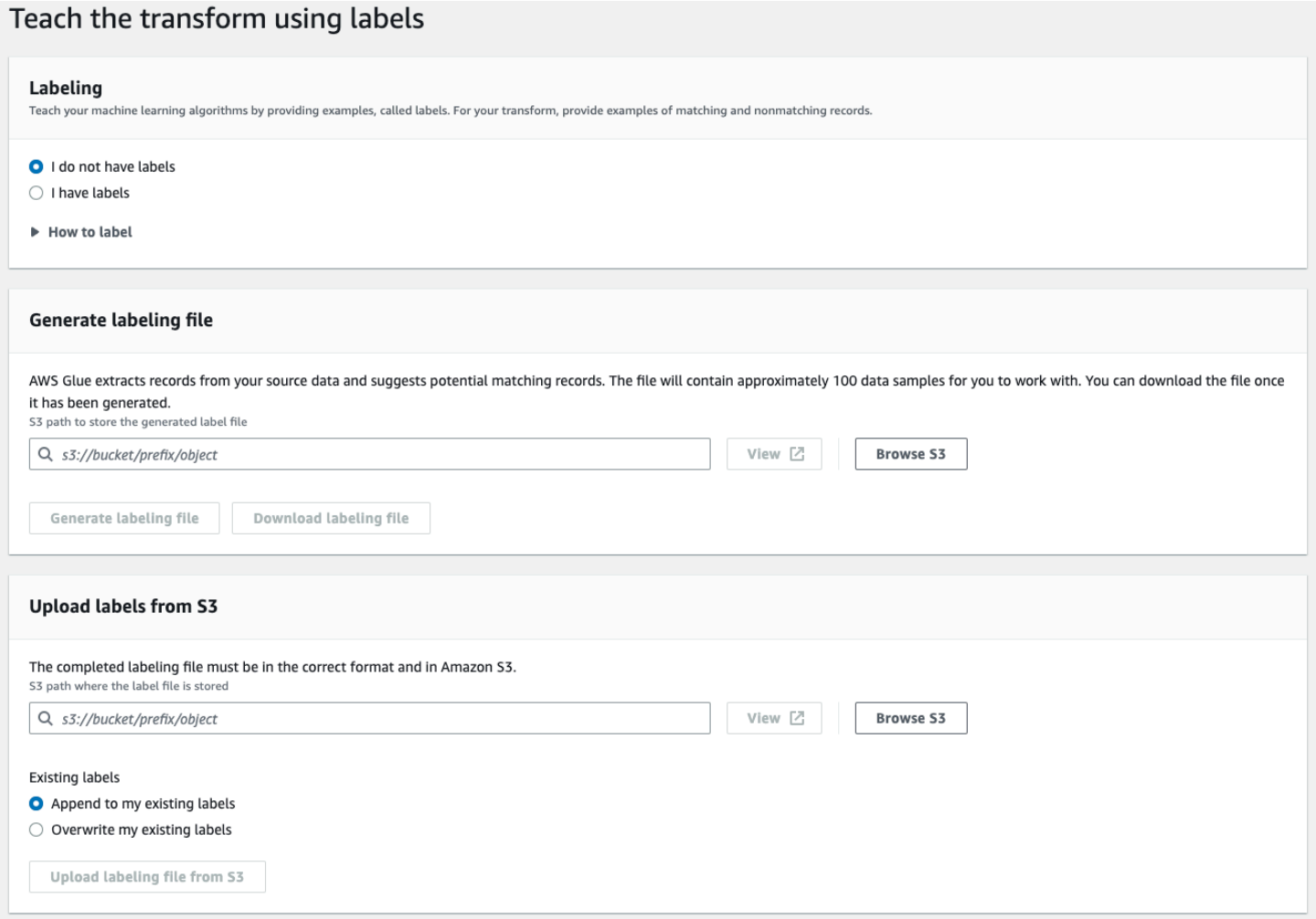
您可以主要通过指定低成本-准确性调整值，来调整此筛选方法。随着调整值更接近偏向准确性，系统对可能匹配的记录对进行更彻底和高成本的搜索。更多的记录会被输入到你的机器学习模型中，你的`ML transform`召回率 end-to-end或真实召回率会更接近估计的召回指标。因此，由于比赛成本/准确性权衡的变化而导致的比赛 end-to-end 质量变化通常不会反映在质量估算中。

标签

标签是您可以分配给 AWS 资源的标签。每个标签都由一个键和一个可选值组成。标签可用于搜索和筛选您的资源或跟踪您的 AWS 成本。

使用标签教授转换

您可以通过从 ML 转换详细信息页面选择教授转换来使用标签（示例）来教授您的 ML 转换。当您通过提供示例（称为标签）来教授机器学习算法时，您可以选择要使用的现有标签，也可以创建标签文件。



- 标签 - 如果您有标签，请选择我有标签。如果您没有标签，仍然可以继续执行生成标签文件的下一步。
- 生成标签文件 - AWS Glue 从源数据中提取记录并建议潜在的匹配记录。您可以选择 Amazon S3 存储桶来存储生成的标签文件。选择生成标签文件以启动该过程。完成后，选择下载标签文件。下载的文件将有一列用于标签，您可以在其中填写标签。

- 从 Amazon S3 上传标签 - 从存储标签文件的 Amazon S3 存储桶中选择已完成的标签文件。然后，选择将标签附加到现有标签或覆盖现有标签。选择从 Amazon S3 上传标签文件。

教程：使用 AWS Glue 创建机器学习转换

本教程指导您完成使用 AWS Glue 创建和管理机器学习 (ML) 转换的操作。在使用本教程之前，您应熟悉如何使用 AWS Glue 控制台添加爬网程序和作业以及编辑脚本。您还应熟悉如何在 Amazon Simple Storage Service（Amazon S3）控制台上查找和下载文件。

在本示例中，您将创建一个 `FindMatches` 转换以查找匹配的记录，指导它如何标识匹配和不匹配的记录，并在 AWS Glue 作业中使用它。AWS Glue 任务会编写一个带有名为 `match_id` 的附加列的新 Amazon S3 文件。

本教程使用的源数据是一个名为 `dblp_acm_records.csv` 的文件。此文件是可从原始 DBLP ACM 数据集获得的学术出版物（DBLP 和 ACM）的修改后版本。`dblp_acm_records.csv` 文件是一个 UTF-8 格式的逗号分隔值 (CSV) 文件，不带字节顺序标记 (BOM)。

另一个文件 `dblp_acm_labels.csv` 是示例标签文件，它包含在教程中用于指导转换的匹配记录和不匹配记录。

主题

步骤 1：爬取源数据

首先，对源 Amazon S3 CSV 文件进行爬网以在数据目录中创建相应的元数据表。

> **Important**
>
> 要指示爬网程序仅为 CSV 文件创建表，请将 CSV 元数据存储在与其他文件不同的 Amazon S3 文件夹中。

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格中，选择 Crawlers (爬网程序)、Add crawler (添加爬网程序)。
3. 按照向导使用数据库 demo-db-dblp-acm 的输出创建并运行名为 demo-crawl-dblp-acm 的爬网程序。在运行向导时，将创建数据库 demo-db-dblp-acm（如果该数据库不存在）。选择 Amazon S3 包含路径以在当前 AWS 区域中对数据进行采样。例如，对于 us-east-1，源文件的 Amazon S3 包含路径为 s3://ml-transforms-public-datasets-us-east-1/dblp-acm/records/dblp_acm_records.csv。

   如果成功，爬网程序会创建具有以下列的 dblp_acm_records_csv 表：ID、标题、作者、地点、年份和源。

步骤 2：添加机器学习转换

接下来，添加基于由名为 demo-crawl-dblp-acm 的爬网程序创建的数据源表的架构的机器学习转换。

1. 在 AWS Glue 控制台上，在导航窗格的数据集成和 ETL下，选择数据分类工具 > 记录匹配，再选择添加转换。按照向导使用以下属性创建 Find matches 转换。

   a. 对于 Transform name (转换名称)，输入 **demo-xform-dblp-acm**。这是用于在源数据中查找匹配项的转换的名称。
   b. 对于 IAM role (IAM 角色)，选择对 Amazon S3 源数据、标签文件和 AWS Glue API 操作具有权限的 IAM 角色。有关更多信息，请参阅 AWS Glue 开发人员指南中的为 AWS Glue 创建 IAM 角色。
   c. 对于 Data source (数据源)，请在数据库 demo-db-dblp-acm 中选择名为 dblp_acm_records_csv 的表。
   d. 对于 Primary key (主键)，选择表的主键列 id。
2. 在向导中，选择 Finish (完成) 并返回 ML transforms (ML 转换) 列表。

步骤 3：指导您的机器学习转换

接下来，您使用教程示例标签文件来指导您的机器学习转换。

您不能在提取、转换和加载 (ETL) 任务中使用机器语言转换，直到其状态为 Ready for use (可供使用)。要准备好转换，您必须通过提供匹配记录和不匹配记录的示例来指导它如何识别匹配记录和不匹配记录。要指导您的转换，您可以 Generate a label file (生成标签文件)，添加标签，然后 Upload label

file (上传标签文件)。在本教程中，您可以使用名为 dblp_acm_labels.csv 的示例标签文件。有关标记过程的更多信息，请参阅 标签。

1. 在 AWS Glue 控制台的导航窗格中，选择记录匹配。
2. 选择 demo-xform-dblp-acm 转换，然后选择 Action (操作)、Teach (指导)。按照向导指导您的 Find matches 转换。
3. 在转换属性页面上，选择 I have labels (我有标签)。选择当前 AWS 区域中的示例标签文件的 Amazon S3 路径。例如，对于 us-east-1，请从 Amazon S3 路径 s3://ml-transforms-public-datasets-us-east-1/dblp-acm/labels/dblp_acm_labels.csv 上传提供的标签文件，并选择 overwrite (覆盖) 现有标签。标签文件必须位于与 AWS Glue 控制台相同的区域中的 Amazon S3 中。

   上传标签文件时，会在 AWS Glue 中启动任务以添加或覆盖用于指导转换过程如何处理数据源的标签。
4. 在向导的最后一页上，选择 Finish (完成)，然后返回到 ML transforms (ML 转换) 列表。

步骤 4：估计您的机器学习转换的质量

接下来，您可以估计您的机器学习转换的质量。质量取决于您的标签数。有关估计质量的更多信息，请参阅 估计质量。

1. 在 AWS Glue 控制台上，在导航窗格的数据集成和 ETL下，选择数据分类工具 > 记录匹配。
2. 选择 demo-xform-dblp-acm 转换，然后选择 Estimate quality (估计质量) 选项卡。此选项卡显示转换的当前质量估计（如果有）。
3. 选择 Estimate quality (估计质量) 以开始一项任务来估计转换的质量。质量估计的准确性基于源数据的标记。
4. 导航到 History (历史记录) 选项卡。在此窗格中，列出转换的任务运行，包括 Estimating quality (估计质量) 任务。有关运行的更多详细信息，请选择 Logs (日志)。在运行完成时，检查其状态是否为 Succeeded (成功)。

步骤 5：使用机器学习转换添加和运行作业

在此步骤中，您将使用您的机器学习转换在 AWS Glue 中添加和运行作业。当转换 demo-xform-dblp-acm 为 Ready for use (可供使用) 时，您可以在 ETL 作业中使用它。

1. 在 AWS Glue 控制台的导航窗格中，选择 Jobs (任务)。

2. 选择 Add job (添加任务)，然后执行向导中的步骤以使用生成的脚本创建 ETL Spark 任务。为转换选择以下属性值：

   a. 对于 Name (名称)，选择此教程中的示例任务 demo-etl-dblp-acm。

   b. 对于 IAM role (IAM 角色)，选择对 Amazon S3 源数据、标签文件和 AWS Glue API 操作具有权限的 IAM 角色。有关更多信息，请参阅 AWS Glue 开发人员指南中的为 AWS Glue 创建 IAM 角色。

   c. 对于 ETL language (ETL 语言)，选择 Scala。这是 ETL 脚本中的编程语言。

   d. 对于 Script file name (脚本文件名称)，选择 demo-etl-dblp-acm。这是 Scala 脚本的文件名称（与作业名称相同）。

   e. 对于 Data source (数据源)，选择 dblp_acm_records_csv。您选择的数据源必须与机器学习转换数据源架构匹配。

   f. 对于 Transform type (转换类型)，选择 Find matching records (查找匹配记录) 以使用机器学习转换来创建任务。

   g. 清除 Remove duplicate records (删除重复记录)。您不想删除重复记录，因为写入的输出记录添加了额外的 `match_id` 字段。

   h. 对于 Transform (转换)，选择 demo-xform-dblp-acm（作业所使用的机器学习转换）。

   i. 对于 Create tables in your data target (在数据目标中创建表)，选择使用以下属性创建表：

      - Data store type (数据存储类型) – **`Amazon S3`**
      - Format (格式) – **`CSV`**
      - Compression type (压缩类型) – **`None`**
      - Target path (目标路径) – 在其中写入任务的输出的 Amazon S3 路径（在当前控制台 AWS 区域中）

3. 选择 Save job and edit script (保存任务和编辑脚本) 以显示脚本编辑器页。

4. 编辑脚本来添加语句，使 Target path (目标路径) 的任务输出写入单个分区文件中。在运行 `FindMatches` 转换的语句后添加此语句。此语句类似于以下内容。

   ```
   val single_partition = findmatches1.repartition(1)
   ```

   您必须修改 `.writeDynamicFrame(findmatches1)` 语句以将输出写入为 `.writeDynamicFrame(single_partion)`。

5. 在编辑脚本后，请选择 Save (保存)。修改后的脚本看上去类似于以下代码，但它已针对您的环境进行了自定义。

```
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()
  }
}
```

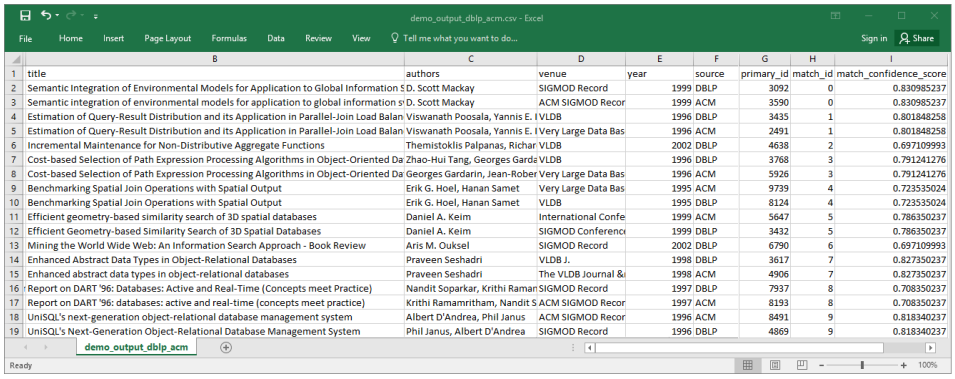
6. 选择 Run job (运行任务) 以开始作业运行。检查任务列表中的作业的状态。任务完成后，在 ML transform (ML 转换)、History (历史记录) 选项卡上，会添加一个 ETL job (ETL 任务) 类型的新 Run ID (运行 ID)。
7. 导航到 Jobs (任务)、History (历史记录) 选项卡。在此窗格中，将列出任务运行。有关运行的更多详细信息，请选择 Logs (日志)。在运行完成时，检查其状态是否为 Succeeded (成功)。

步骤 6：验证来自 Amazon S3 的输出数据

在这一步中，您将在添加任务时选择的 Amazon S3 存储桶中检查任务运行的输出。您可以将输出文件下载到本地计算机，并验证是否已标识匹配的记录。

1. 通过以下网址打开 Simple Storage Service（Amazon S3）控制台：https://console.aws.amazon.com/s3/。
2. 下载任务 `demo-etl-dblp-acm` 的目标输出文件。在电子表格应用程序中打开文件（您可能需要为文件添加文件扩展名 `.csv` 以使其正常打开）。

   下图通过 Microsoft Excel 显示了输出摘录。



	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

   数据源和目标文件都有 4911 条记录。不过，`Find matches` 转换会添加另一个名为 `match_id` 的列以标识输出中的匹配记录。具有相同 `match_id` 的行被视为匹配记录。`match_confidence_score` 是一个介于 0 与 1 之间的数字，它为 `Find matches` 找到的匹配项的质量提供一个估计值。

3. 按 match_id 对输出文件进行排序可轻松查看匹配的记录。比较其他列中的值以查看您是否同意 Find matches 转换的结果。如果您不同意，则可通过添加更多标签来继续指导转换。

   您也可以按其他字段（例如 title）对文件进行排序，以查看具有相似标题的记录的 match_id 是否相同。

查找递增匹配项

借助查找匹配项功能，您可以识别数据集中的重复或匹配记录，即使记录没有公共唯一标识符且没有完全匹配的字段也是如此。查找匹配项的初始版本将转换单个数据集中识别出来的匹配记录。在向数据集添加新数据时，必须将其与现有的干净数据集合并，然后对整个合并的数据集重新运行匹配。

递增匹配功能可使将递增记录与现有已匹配数据集进行匹配变得更加简单。假设您要将潜在客户数据与现有客户数据集进行匹配。递增匹配项功能通过将结果合并到单个数据库或表中，使您能够灵活地将成千上万的新潜在客户与潜在客户和客户的现有数据库进行匹配。通过仅匹配新数据集和现有数据集，查找递增匹配项优化可缩短计算时间，同时还可降低成本。

递增匹配的用法与查找匹配项类似，如 教程：使用 AWS Glue 创建机器学习转换 中所述。本主题仅确定了与递增匹配的差异。

有关更多信息，请参阅有关 递增数据匹配的博客文章。

运行增量匹配任务

对于以下过程，假设以下情况：

- 您已将现有数据集爬取到 first_records 表中。first_records 数据集必须是匹配的数据集，或者是匹配作业的输出。
- 您已使用 AWS Glue 2.0 版创建并训练了 Find matches（查找匹配项）转换。这是支持递增匹配项的唯一 AWS Glue 版本。
- ETL 语言是 Scala。请注意，还支持 Python。
- 已生成的模型名为 demo-xform。

1. 将递增数据集爬取到表 second_records。
2. 在 AWS Glue 控制台的导航窗格中，选择 Jobs (任务)。
3. 选择 Add job (添加任务)，然后执行向导中的步骤以使用生成的脚本创建 ETL Spark 任务。为转换选择以下属性值：

a. 对于 Name（名称），选择 demo-etl。

b. 对于 IAM role（IAM 角色），选择对 Amazon S3 源数据、标签文件和 AWS Glue API 操作拥有权限的 IAM 角色。

c. 对于 ETL language (ETL 语言)，选择 Scala。

d. 对于 Script file name（脚本文件名称），选择 demo-etl。这是 Scala 脚本的文件名称。

e. 对于 Data source（数据源），选择 first_records。您选择的数据源必须与机器学习转换数据源架构匹配。

f. 对于 Transform type (转换类型)，选择 Find matching records (查找匹配记录) 以使用机器学习转换来创建任务。

g. 选择递增匹配选项，对于 Data Source（数据源），选择名为 second_records 的表。

h. 对于 Transform（转换），选择 demo-xform（任务所使用的机器学习转换）。

i. 选择Create tables in your data target（在数据目标中创建表）或 Use tables in the data catalog and update your data target（使用数据目录中的表并更新您的数据目标）。

4. 选择 Save job and edit script (保存任务和编辑脚本) 以显示脚本编辑器页。

5. 选择 Run job (运行任务) 以开始任务运行。

在可视化作业中使用 FindMatches

要在 AWS Glue Studio 中使用 FindMatches 转换，您可以使用调用 FindMatches API 的自定义转换节点。有关如何使用自定义转换的更多信息，请参阅 Creating a custom transformation

> **Note**
>
> 目前，FindMatches API 仅适用于 `Glue 2.0`。要使用调用 FindMatches API 的自定义转换运行作业，请确保任务详细信息选项卡中的 AWS Glue 版本为 `Glue 2.0`。如果 AWS Glue 的版本不是 `Glue 2.0`，则作业将在运行时失败，并显示以下错误消息：“无法从 'awsglueml.transforms' 导入名称 'FindMatches'”。

先决条件

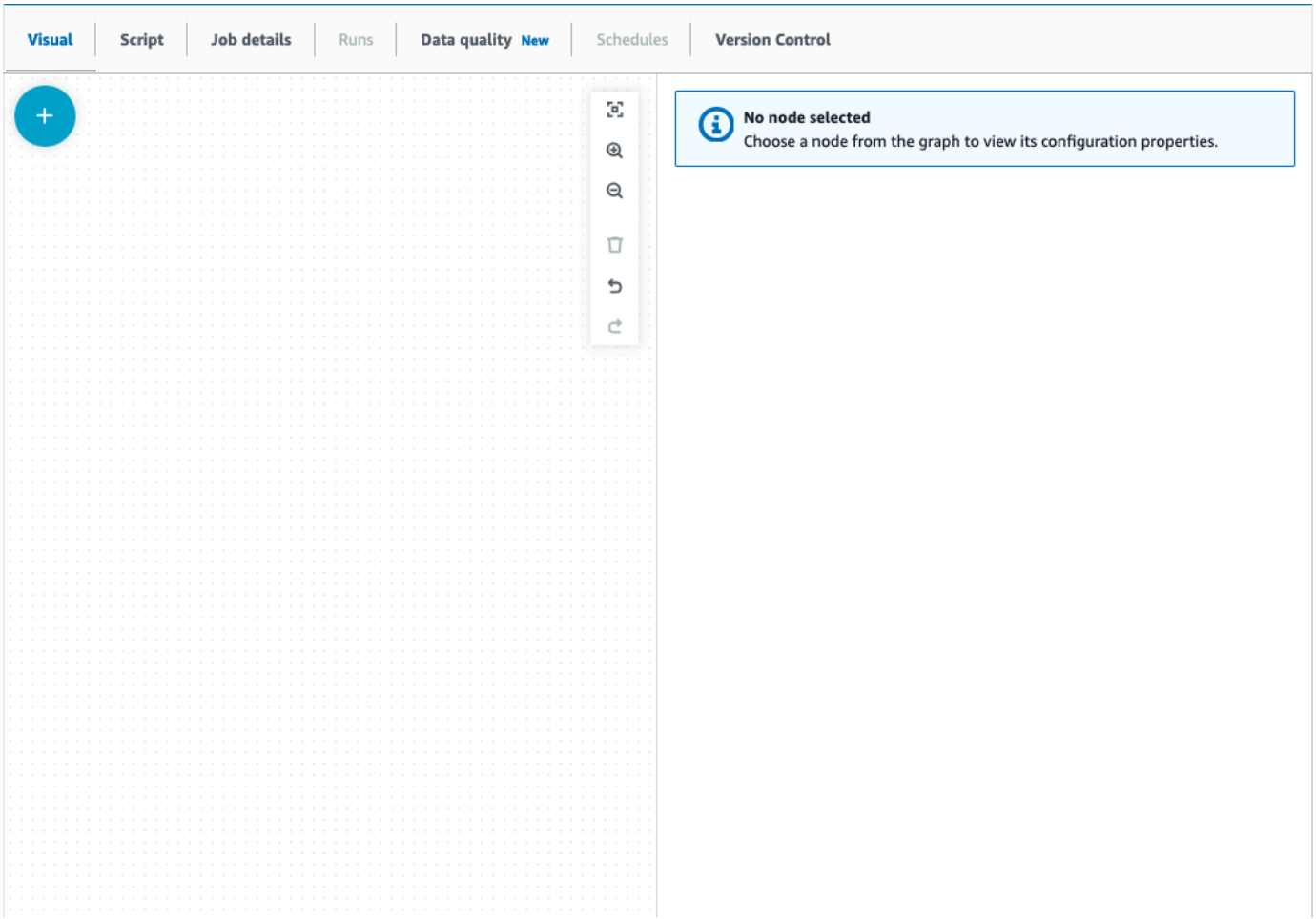
- 要使用 FindMatches 转换，请打开 AWS Glue Studio 控制台，网址为 https://console.aws.amazon.com/gluestudio/。

- 创建机器学习转换。创建后，会生成一个转换 ID。在执行以下步骤时，您将需要此 ID。有关如何创建机器学习转换的更多信息，请参阅 Adding and editing machine learning transforms。

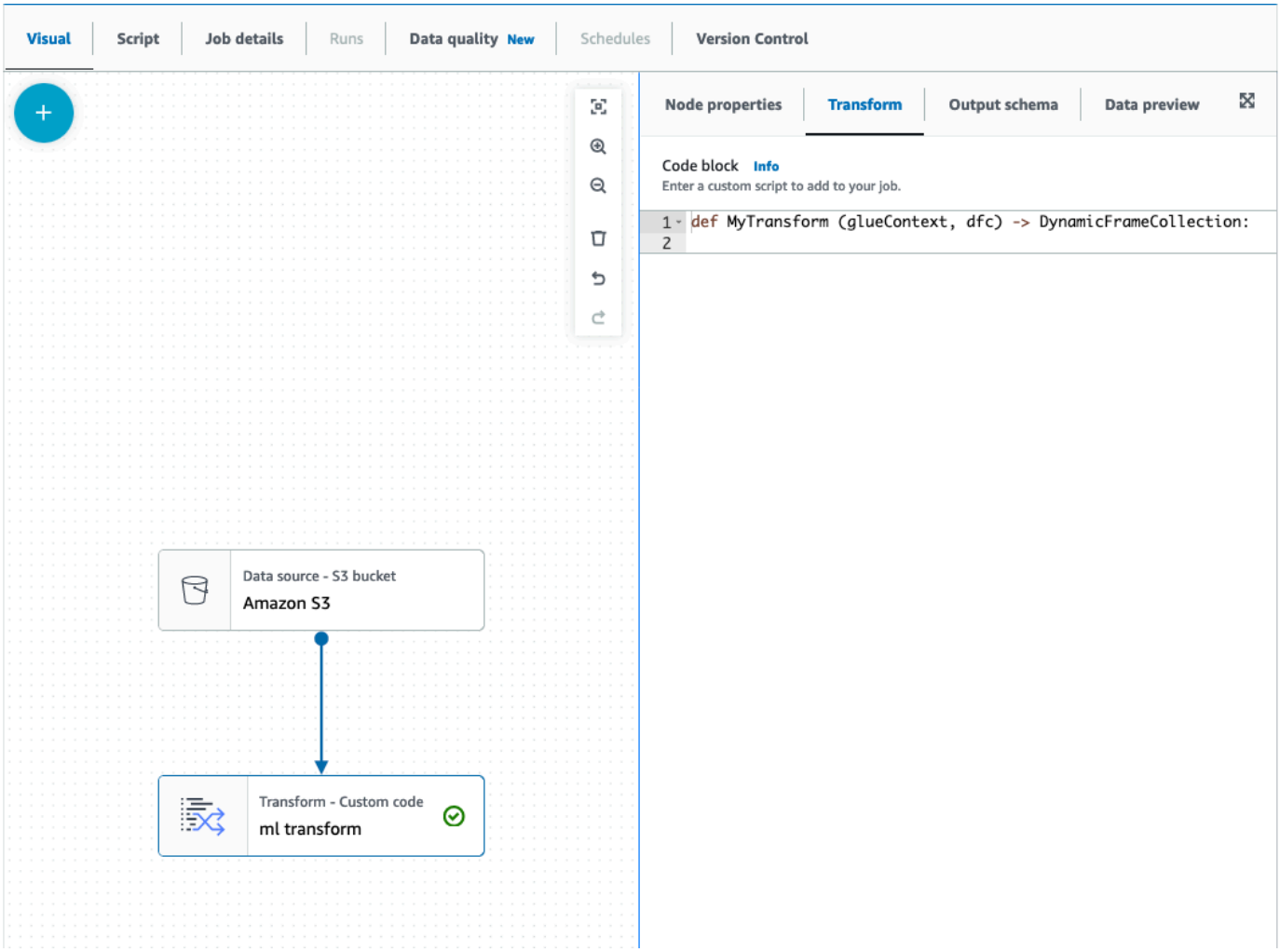
添加 FindMatches 转换

要添加 FindMatches 转换，请执行以下操作：

1. 在 AWS Glue Studio 作业编辑器中，通过单击可视作业图表左上角的十字符号打开“资源”面板，然后通过选择数据选项卡来选择数据来源。这是您要检查匹配项的数据来源。



2. 选择数据来源节点，然后通过单击可视化作业图左上角的十字符号打开“资源”面板，然后搜索“自定义转换”。选择自定义转换节点将其添加到图表中。自定义转换链接到数据来源节点。如果不是，则可以单击自定义转换节点并选择节点属性选项卡，然后在父节点下选择数据来源。
3. 在可视化图中单击自定义转换节点，然后选择节点属性选项卡并命名自定义转换。建议您重命名转换，以便在可视化图中可以轻松识别转换名称。
4. 选择转换选项卡，可以在其中编辑代码块。在这里，可以添加调用 FindMatches API 的代码。



代码块包含预先填充的代码，可帮助您入门。使用以下模板覆盖预先填充的代码。该模板有一个用于转换 ID 的占位符，您可以提供值。

```
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. 单击可视化图中的自定义转换节点，然后通过单击可视化作业图左上角的十字符号打开“资源”面板，然后搜索“从集合中选择”。由于集合中只有一个 DynamicFrame，因此无需更改默认选择。

6. 您可以继续添加转换或存储结果，现在，查找匹配的其他列丰富了结果。如果要在下游转换中引用这些新列，则需要将它们添加到转换输出架构中。最简单的方法是选择数据预览选项卡，然后在“架构”选项卡中选择“使用 datapreview 架构”。
7. 要自定义 FindMatches，您可以添加其他参数以传递给“apply”方法。请参阅 FindMatches class。

以增量方式添加 FindMatches 转换

对于增量匹配，其过程与添加 FindMatches 转换相同，但有以下区别：

- 您需要两个父节点，而不是自定义转换的父节点。
- 第一个父节点应该是数据集。
- 第二个父节点应该是增量数据集。

  将模板代码块中的 `transformId` 替换为您的 `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))
```

- 有关可选参数，请参阅 FindIncrementalMatches class。

将 Apache Spark 程序迁移到 AWS Glue

Apache Spark 是一个开源平台，用于在大型数据集上执行分布式计算工作负载。AWS Glue 利用 Spark 的功能为 ETL 提供优化的体验。可以将 Spark 程序迁移到 AWS Glue 以利用我们的功能。AWS Glue 提供了与 Amazon EMR 上的 Apache Spark 相同的性能增强。

运行 Spark 代码

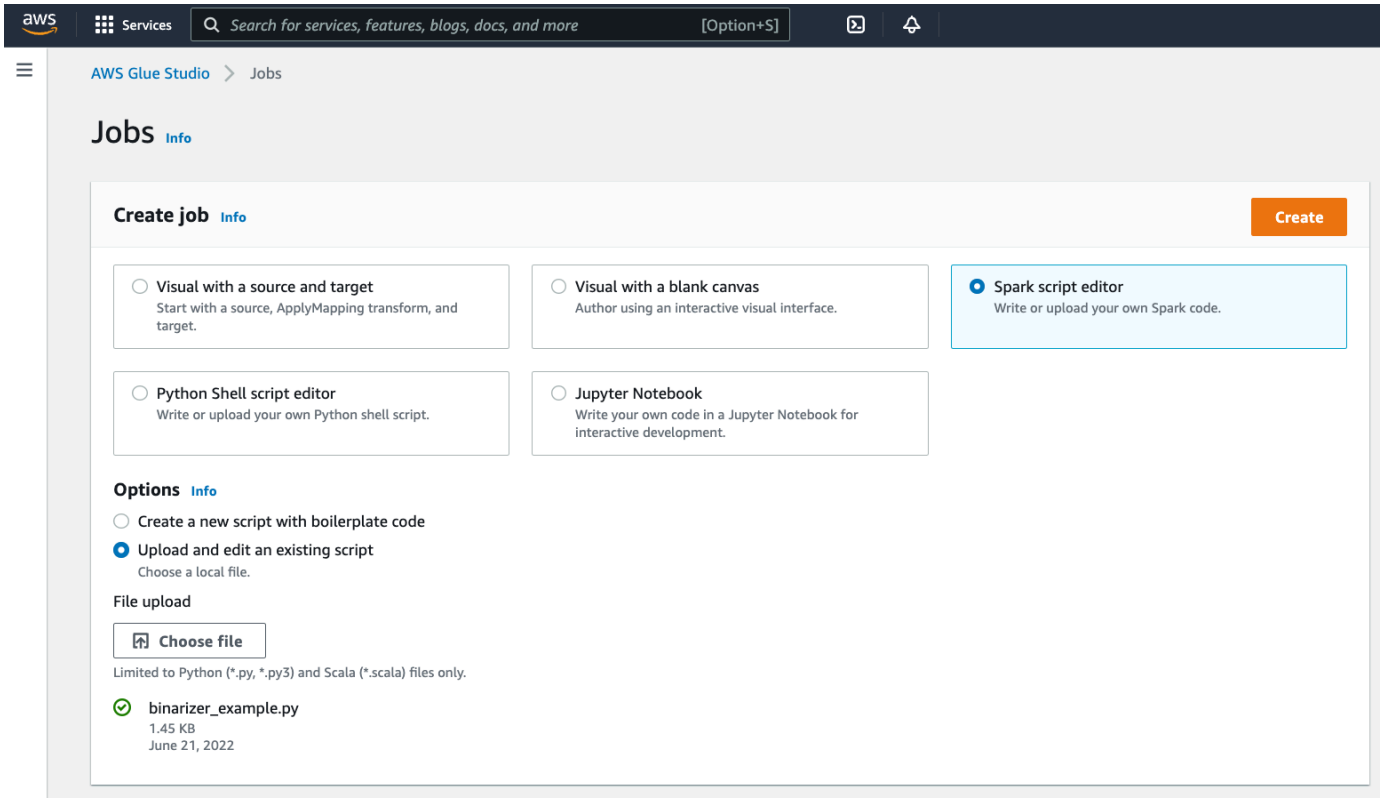
原生 Spark 代码可以在开箱即用的 AWS Glue 环境中运行。脚本通常是通过迭代更改一段代码来开发的，这是一种适合交互式会话的工作流。但是，现有的代码更适合在 AWS Glue 作业中运行，它允许您计划并持续获取每个脚本运行的日志和指标。可以通过控制台上传和编辑现有脚本。

1. 获取脚本的源。在此示例中，您将使用来自 Apache Spark 存储库的示例脚本。二值化程序示例
2. 在 AWS Glue 控制台中，展开左侧导航窗格并选择 ETL > Jobs（作业）

   在 Create job（创建作业）面板中，选择 Spark script editor（Spark 脚本编辑器）。Options（选项）部分将显示。在 Options（选项）下，选择 Upload and edit an existing script（上传和编辑现有脚本）。

   File upload（文件上传）部分将显示。在 File upload（文件上传）下，单击 Choose file（选择文件）。系统文件选择器将会出现。导航到保存 `binarizer_example.py` 的位置，将其选中，然后确认您的选择。

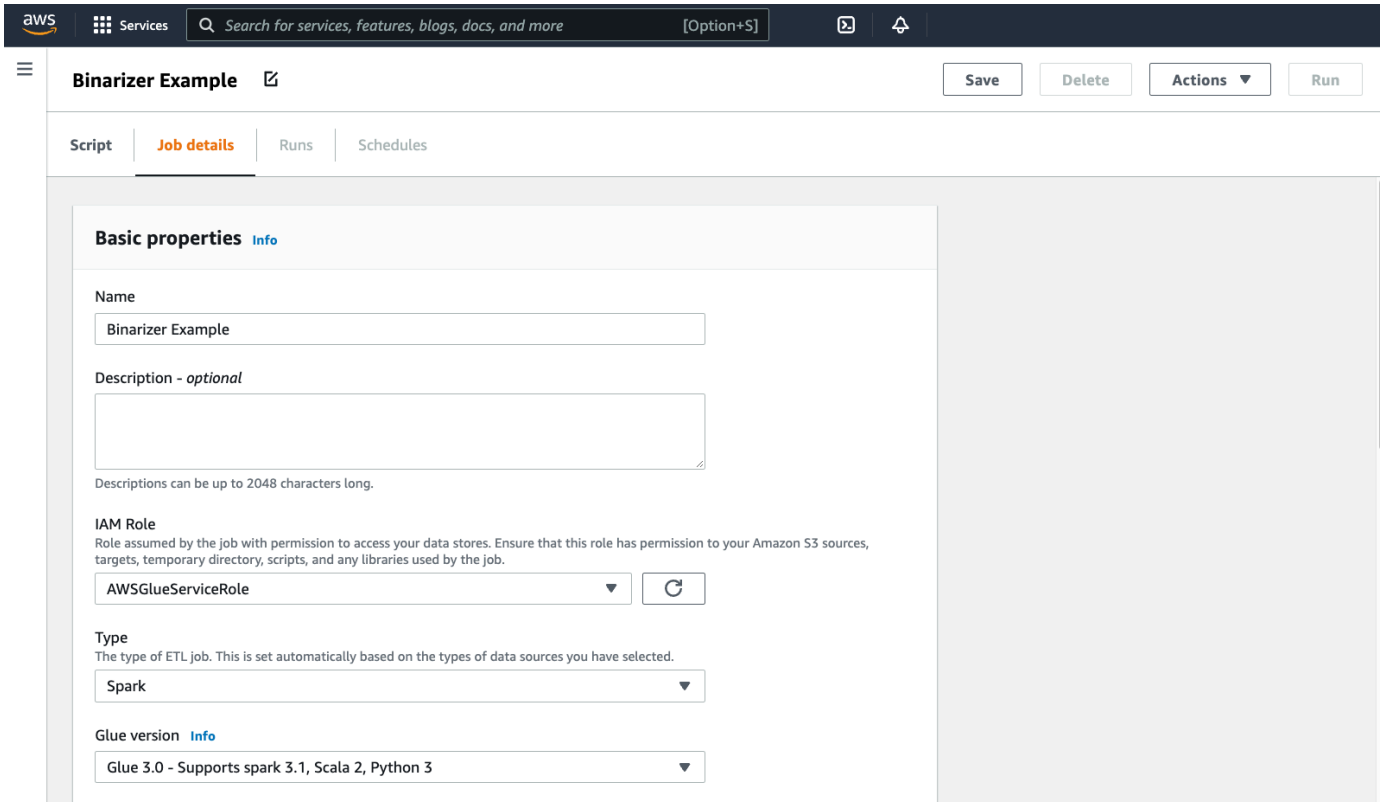
   Create（创建）按钮将出现在 Create job（创建作业）面板的标题上。单击它。



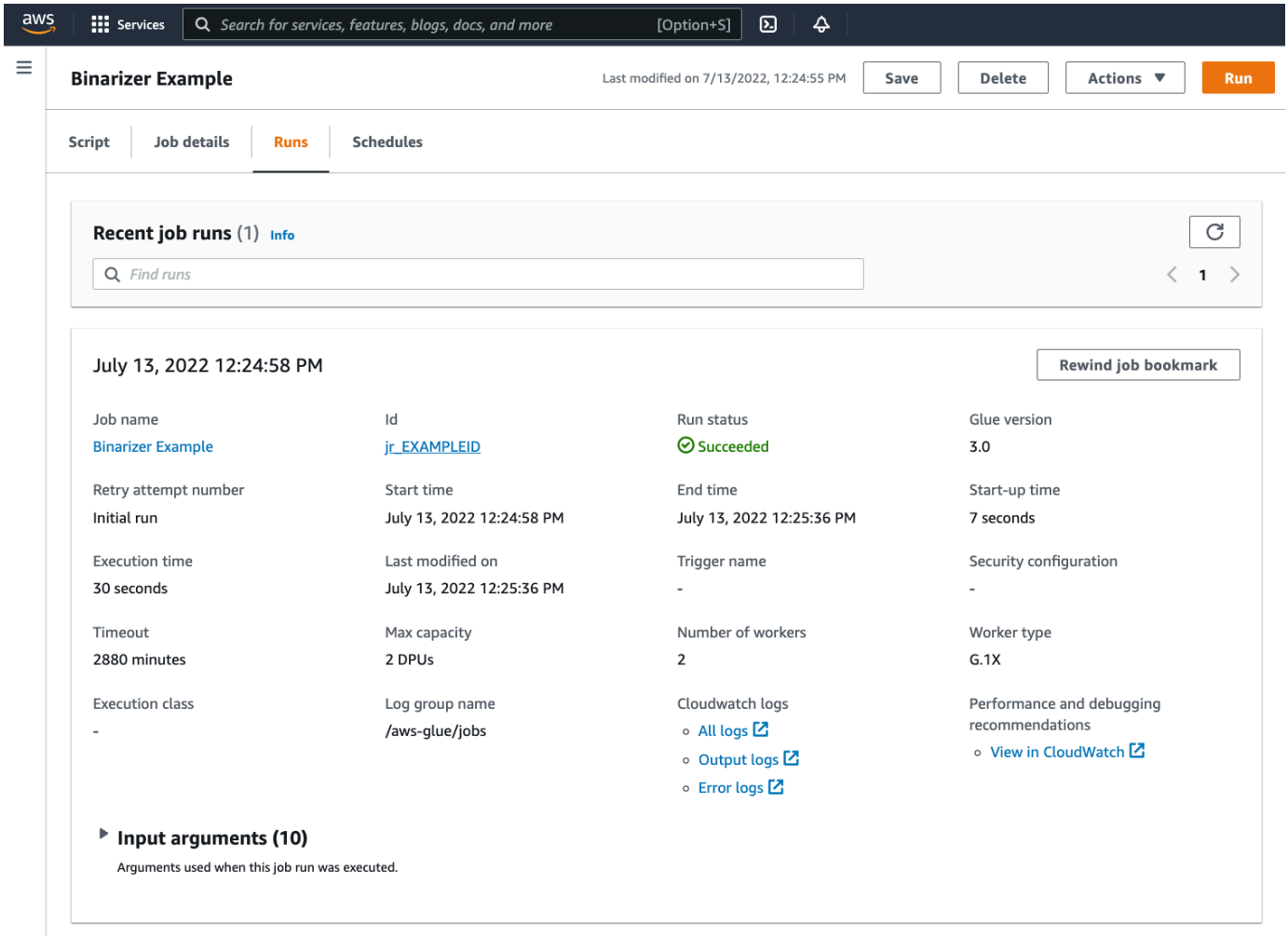
3. 您的浏览器将导航到脚本编辑器。在标题上，单击 Job details（作业详细信息）选项卡。设置名称和 IAM 角色。有关 AWS Glue IAM 角色的指导，请参阅 the section called “设置 IAM 权限”。

   可选 - 将 Requested number of workers（请求的工作线程数）设置为 2，Number of retries（重试次数）设置为 1。这些选项在运行生产作业时很有价值，但关闭这些选项将简化您在测试功能时的体验。

   在标题栏中，单击 Save（保存），然后单击 Run（运行）

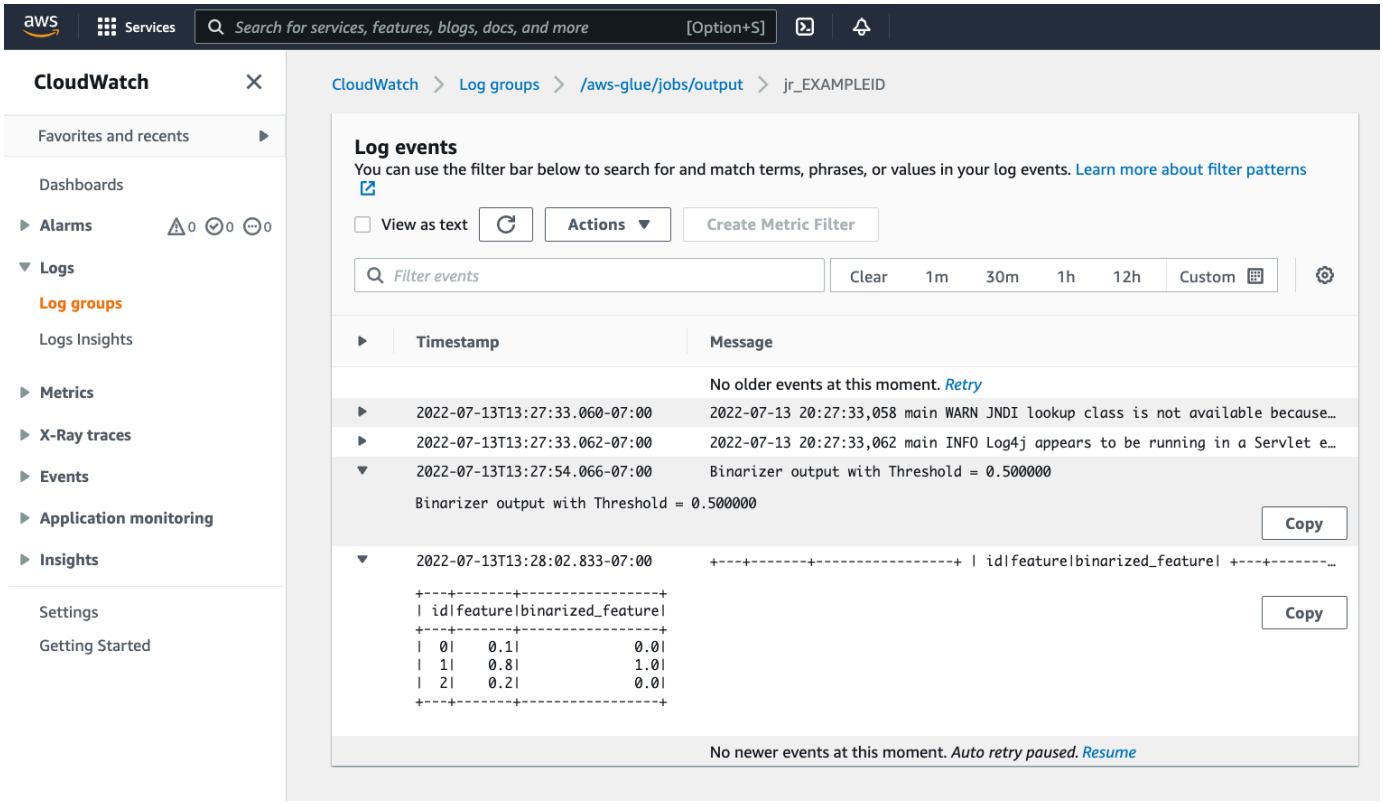


4. 导航到 Runs（运行）选项卡。您将看到一个与您的作业运行相对应的面板。等待几分钟，页面将自动刷新以在 Run status（运行状态）下显示 Succeeded（成功）。



5. 您需要检查输出，以确保 Spark 脚本按预期运行。此 Apache Spark 示例脚本应该会向输出流写入一个字符串。可以通过导航到面板中 Cloudwatch Logs 下的 Output logs（输出日志）找到成功的作业运行。请记下作业运行 ID，它是在 Id 标签下生成的以 `jr_` 开头的 ID。

   这将打开 CloudWatch 控制台，设置为可视化默认 AWS Glue 日志组 `/aws-glue/jobs/output` 的内容，筛选为作业运行 ID 的日志流的内容。每个工作线程都将生成一个日志流，显示在 Log streams（日志流）。应该有一个工作线程运行了请求的代码。您需要打开所有日志流来找到正确的工作线程。找到合适的工作线程后，您应该会看到脚本的输出，如下图所示：



迁移 Spark 程序所需的常用过程

评估 Spark 版本支持

AWS Glue 版本定义可用于 AWS Glue 作业的 Apache Spark 和 Python 版本。您可以在 the section called “AWS Glue 版本” 找到我们的 AWS Glue 版本及其支持的内容。您可能需要更新您的 Spark 程序，使其与较新版本的 Spark 兼容，才能访问某些 AWS Glue 功能。

包括第三方库

许多现有的 Spark 程序在私有和公共构件上都有依赖关系。AWS Glue 支持 Scala 作业的 JAR 样式依赖关系以及 Python 作业的 Wheel 和源纯 Python 依赖关系。

Python - 有关 Python 依赖关系的信息，请参阅 the section called “Python 库”

常见的 Python 依赖项在 AWS Glue 环境中提供，包括通常请求的 Pandas 库。这些依赖项包含在 AWS Glue 版本 2.0+ 中。有关提供的模块的更多信息，请参阅 the section called “AWS Glue 中已提供的 Python 模块”。如果需要为作业提供默认包含的不同版本的依赖项，则可以使用 `--additional-python-modules`。有关作业参数的信息，请参阅 the section called “任务参数”。

您可以为其他 Python 依赖关系提供 `--extra-py-files` 作业参数。如果要从 Spark 程序迁移作业，则此参数是一个不错的选择，因为它在功能上等同于 PySpark 中的 `--py-files` 标志，并且受到相同的限制。有关 `--extra-py-files` 参数的更多信息，请参阅 the section called “包括具有 PySpark 原生功能的 Python 文件”

对于新作业，您可以管理带 `--additional-python-modules` 作业参数的 Python 依赖关系。使用此参数可以获得更全面的依赖关系管理体验。此参数支持 Wheel 样式依赖项，包括具有与 Amazon Linux 2 兼容的原生代码绑定的依赖项。

Scala

您可以为其他 Scala 依赖关系提供 `--extra-jars` 作业参数。依赖关系必须托管在 Amazon S3 中，参数值应为逗号分隔的 Amazon S3 路径列表，且不含空格。您可能会发现，在托管和配置依赖关系之前，通过重新绑定依赖关系，可以更轻松地管理您的配置。AWS GlueJAR 依赖关系包含 Java 字节码，它可以从任何 JVM 语言生成。您可以使用其他 JVM 语言（如 Java）来编写自定义依赖关系。

管理数据源凭证

现有的 Spark 程序可能带有复杂或自定义配置，以便从其数据源中提取数据。AWS Glue 连接支持常见的数据源身份验证流程。有关 AWS Glue 连接的更多信息，请参阅 连接到数据。

AWS Glue 连接可通过两种主要方式帮助您将作业连接到各种类型的数据存储：通过方法调用我们的库和设置 AWS 控制台中的 Additional network connection（其他网络连接）。也可以从作业中调用 AWS SDK，以从连接中检索信息。

方法调用 - AWS Glue 连接与 AWS Glue 数据目录紧密集成，后者是一项服务，允许您整理有关数据集的信息以及可用来与反映该信息的 AWS Glue 连接交互的方法。如果您有想要重用的现有身份验证配置，对于 JDBC 连接，您可以通过 `GlueContext` 上的 `extract_jdbc_conf` 方法访问您的 AWS Glue 连接配置。有关更多信息，请参阅 the section called “extract_jdbc_conf”

控制台配置 - AWS Glue 作业使用关联 AWS Glue 连接来配置与 Amazon VPC 子网的连接。如果您直接管理自己的安全材料，则可能需要在 AWS 控制台上提供 `NETWORK` 类型的其他网络连接来配置路由。有关 AWS Glue 连接 API 的更多信息,请参阅 the section called “连接”

如果您的 Spark 程序具有自定义或不常见的身份验证流程，则可能需要亲自操作管理安全材料。如果 AWS Glue 连接似乎不太合适，您可以在 Secrets Manager 中安全地托管安全材料，然后通过作业中提供的 boto3 或 AWS SDK 来访问。

配置 Apache Spark

复杂的迁移通常会改变 Spark 配置以适应其工作负载。现代版本的 Apache Spark 允许使用 `SparkSession` 设置运行时配置。AWS Glue为 3.0+ 任务提供了 `SparkSession`，可以对其进行修

改以设置运行时配置。Apache Spark 配置。调整 Spark 很复杂，而且 AWS Glue 不保证支持设置所有 Spark 配置。如果您的迁移需要大量的 Spark 级别配置，请联系支持人员。

设置自定义配置

迁移的 Spark 程序可能设计为采用自定义配置。AWS Glue 允许通过作业参数在作业和作业运行级别设置配置。有关作业参数的信息，请参阅 the section called “任务参数”。您可以通过我们的库访问作业上下文中的作业参数。AWS Glue 提供了一个实用程序函数，用于在作业上设置的参数和在作业运行上设置的参数之间提供一致的视图。请参阅 Python 中的 the section called “getResolvedOptions” 和 Scala 中的 the section called “GlueArgParser”。

迁移 Java 代码

如 the section called “第三方库” 中解释，您的依赖关系可以包含由 JVM 语言（如 Java 或 Scala）生成的类。您的依赖关系可以包含 `main` 方法。您可以在依赖关系中使用 `main` 方法作为 AWS Glue Scala 作业的入口点。这允许您用 Java 编写 `main` 方法，或者重用您自己的库标准中打包的 `main` 方法。

要使用依赖关系中的 `main` 方法，请执行以下操作：清除提供默认 `GlueApp` 对象的编辑窗格的内容。在依赖关系中将类的完全限定名称作为带键 `--class` 的作业参数提供。然后您应该能够触发作业运行。

您无法配置参数 AWS Glue 传递到 `main` 方法的顺序或结构。如果您的现有代码需要读取 AWS Glue 中的配置集，这可能会导致与之前的代码不兼容。如果您使用 `getResolvedOptions`，也不会有调用此方法的好地方。请考虑直接利用 AWS Glue 生成的主方法来调用依赖关系。下方的 AWS Glue ETL 脚本显示了此操作的示例。

```
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 中处理 Ray 作业

此部分提供有关使用 AWS Glue for Ray 作业的信息。有关编写 AWS Glue for Ray 脚本的更多信息，请参阅 the section called “AWS Glue for Ray” 部分。

主题

- AWS Glue for Ray 入门
- 支持的 Ray 运行时环境
- Ray 作业的工作线程会计
- 在 Ray 作业中使用作业参数
- 使用指标监控 Ray 作业

AWS Glue for Ray 入门

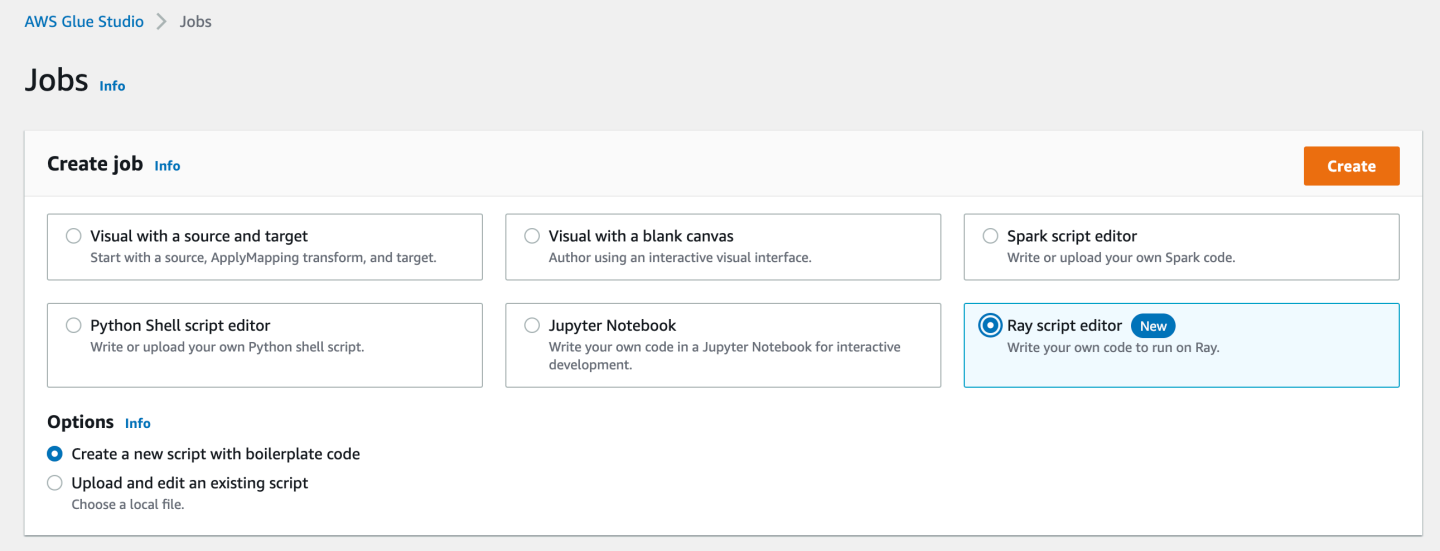
要使用 AWS Glue for Ray，您可以使用为 AWS Glue for Spark 使用的相同 AWS Glue 作业和交互式会话。AWS Glue 作业专为循环运行相同的脚本而设计，而交互式会话旨在让您针对相同的预置资源按顺序运行代码片段。

AWS Glue ETL 和 Ray 在底层是不同的，所以在您的脚本中，您可以访问不同的工具、功能和配置。作为一个由 AWS Glue 管理的新计算框架，Ray 具有不同的架构，并使用不同的词汇来描述它的作用。有关更多信息，请参阅 Ray 文档中的架构白皮书。

> **Note**
>
> AWS Glue for Ray 在美国东部（弗吉尼亚州北部）、美国东部（俄亥俄州）、美国西部（俄勒冈州）、亚太地区（东京）和欧洲地区（爱尔兰）发布。

AWS Glue Studio 控制台中的 Ray 作业

在 AWS Glue Studio—Ray 脚本编辑器中创建作业时，在 AWS Glue Studio 控制台的 作业页面上，可以选择一个新选项。选择此选项可在控制台中创建 Ray 作业。有关这些作业及其使用方法的详细信息，请参阅 使用 AWS Glue Studio 构建可视化 ETL 作业。



AWS CLI 和 SDK 中的 Ray 作业

AWS CLI 中的 Ray 作业与其他作业使用相同的 SDK 操作和参数。AWS Glue for Ray 为某些参数引入了新值。有关作业 API 的更多信息，请参阅 the section called “作业”。

支持的 Ray 运行时环境

在 Spark 作业中，`GlueVersion` 确定 AWS Glue for Spark 作业中可用的 Apache Spark 和 Python 版本。Python 版本指示了 Spark 类型的作业支持的版本。这不是 Ray 运行时环境的配置方式。

对于 Ray 作业，应将 `GlueVersion` 设置为 `4.0` 或更高。但是，Ray 作业中可用的 Ray、Python 和其他库的版本由作业定义中的 `Runtime` 字段决定。

`Ray2.4` 运行时环境将在发布后至少 6 个月内可用。随着 Ray 的快速发展，您将能够在未来的运行时环境版本中整合 Ray 的更新和改进功能。

有效值：`Ray2.4`

运行时值	Ray 和 Python 版本
`Ray2.4`（适用于 AWS Glue 4.0+ 版本）	Ray 2.4.0 Python 3.9

其他信息

- 有关 Ray 版本附带 AWS Glue 的发行说明，请参阅 the section called “AWS Glue 版本”。
- 有关在运行时环境中提供的 Python 库，请参阅 the section called “Ray 作业提供的模块”。

Ray 作业的工作线程会计

AWS Glue 在新的基于 Graviton 的 EC2 工作线程类型上运行 Ray 作业，这些类型仅适用于 Ray 作业。为了适当地为这些工作负载配置 Ray 所设计的工作负载，我们提供了与大多数工作线程不同的计算资源与内存资源的比例。为了考虑这些资源，我们使用内存优化数据处理单元（M-DPU），而不是标准数据处理单元（DPU）。

- 一个 M-DPU 相当于 4 个 vCPU 和 32 GB 内存。
- 一个 DPU 相当于 4 个 vCPU 和 16 GB 内存。DPU 用于核算 AWS Glue 中使用 Spark 作业和相应工作线程的资源。

Ray 作业目前可以访问一种工作者类型 `Z.2X`。`Z.2X` 工作线程映射到 2 个 M-DPU（8 个 vCPU，64 GB 内存），并拥有 128 GB 的磁盘空间。`Z.2X` 计算机提供 8 个 Ray 工作线程（每个 vCPU 一个）。

在一个账户中可以同时使用的 M-DPU 数量受服务限额的限制。有关 AWS Glue 账户限制的更多信息，请参阅AWS Glue端节点和配额。

您可以在作业定义中指定使用 `--number-of-workers` (`NumberOfWorkers`) 的 Ray 作业可用的工作节点数量。有关作业 API 中 Ray 值的更多信息，请参阅 the section called “作业”。

您可以使用 `--min-workers` 作业参数进一步指定 Ray 作业必须分配的最小工作线程数。有关任务参数的更多信息，请参阅 the section called “参考”。

在 Ray 作业中使用作业参数

为 AWS Glue Ray 任务设置参数的方式与为 AWS Glue Spark 作业设置参数的方法相同。有关 AWS Glue API 的更多信息，请参阅the section called “作业”。您可以使用本参考中列出的不同参数配置 AWS Glue Ray 作业。也可以提供自己的参数。

您可以在控制台的 Job details（作业详细信息）选项卡的 Job Parameters（作业参数）标题下配置作业。还可以设置作业上的 `DefaultArguments` 或作业运行上的 `Arguments` 来通过 AWS CLI 配置作业。默认参数和作业参数将在多次运行时保留在作业中。

例如，以下是运行任务的语法，使用 `--arguments` 设置特殊参数。

```
$ aws glue start-job-run --job-name "CSV to CSV" --arguments='--scriptLocation="s3://
my_glue/libraries/test_lib.py",--test-environment="true"'
```

设置参数后，您可以通过环境变量从 Ray 作业中访问作业参数。这为您提供了一种为每次运行配置作业的方法。环境变量的名称将是不带 `--` 前缀的作业参数名称。

例如，在上面的实例中，变量名将为 `scriptLocation` 和 `test-environment`。然后，您可以通过标准库中提供的方法检索参数：`test_environment = os.environ.get('test-environment')`。有关使用 Python 访问环境变量的更多信息，请参阅 Python 文档中的 os module。

配置 Ray 作业生成日志的方式

默认情况下，Ray 作业生成的日志和指标会发送到 CloudWatch 和 Amazon S3。您可以使用 `--logging_configuration` 参数更改日志的生成方式，目前还可以用它来阻止 Ray 作业生成各种类型的日志。该参数会获取一个 JSON 对象，其键对应于您要更改的日志/行为。它支持以下键：

- `CLOUDWATCH_METRICS` - 配置 CloudWatch 指标系列，这些指标可用于可视化作业运行状况。有关指标的更多信息，请参阅 the section called “Ray 作业指标”。
- `CLOUDWATCH_LOGS` - 配置 CloudWatch 日志，这些日志提供有关作业运行状态的 Ray 应用程序级别详细信息。有关日志的更多信息，请参阅the section called “Ray 错误故障排除”。
- `S3` - 配置 AWS Glue 写入 Amazon S3 的内容，这些内容主要是类似于 CloudWatch 日志的信息，但以文件而不是日志流的形式写入。

要禁用 Ray 日志记录行为，请提供值 `{\"IS_ENABLED\": \"False\"}`。例如，要禁用 CloudWatch 指标和 CloudWatch 日志，请提供以下配置：

```
"--logging_configuration": "{\"CLOUDWATCH_METRICS\": {\"IS_ENABLED\": \"False\"},
 \"CLOUDWATCH_LOGS\": {\"IS_ENABLED\": \"False\"}}"
```

参考

Ray 作业可识别以下参数名称，这些参数名称可用于设置 Ray 作业和作业运行的脚本环境：

- `--logging_configuration` - 用于停止生成 Ray 作业创建的各种日志。默认情况下，所有 Ray 作业都会生成这些日志。格式：字符串转义的 JSON 对象。有关更多信息，请参阅 the section called “配置 Ray 作业生成日志的方式”。

- `--min-workers` — 分配给 Ray 作业的 Worker 节点的最小数量。一个 Worker 节点可以运行多个副本，每个虚拟 CPU 一个副本。格式：整数。最小值：0。最大值：`--number-of-workers`（`NumberOfWorkers`）在作业定义中指定的值。有关 Worker 节点会计的更多信息，请参阅 the section called “Ray 作业的工作线程会计”。
- `--object_spilling_config` - AWS Glue for Ray 支持使用 Amazon S3 作为扩展 Ray 对象存储可用空间的一种方式。要启用此行为，您可以为 Ray 提供一个带有此参数的对象溢出 JSON 配置对象。有关 Ray 对象溢出配置的更多信息，请参阅 Ray 文档中的 Object Spilling。格式：JSON 对象。

  AWS Glue for Ray 仅支持一次性溢出到磁盘或溢出到 Amazon S3。您可以提供多个溢出地点，前提是它们遵守此限制。溢出到 Amazon S3 时，您还需要为该存储桶的作业添加 IAM 权限。

  在 CLI 中提供 JSON 对象作为配置时，必须将其作为字符串提供，并对 JSON 对象进行字符串转义。例如，溢出到一个 Amazon S3 路径的字符串值将如下所示：`"{\"type\": \"smart_open\", \"params\": {\"uri\":\"s3path\"}}"`。在 AWS Glue Studio 中，将此参数作为 JSON 对象提供，无需额外格式。
- `--object_store_memory_head` — 分配给 Ray 头节点上的 Plasma 对象存储的内存。此实例运行集群管理服务以及工作线程副本。该值表示热启动后实例上可用内存的百分比。使用此参数调整内存密集型工作负载，默认值对于大多数使用案例来说是可接受的。格式：正整数。最小值：1。最大值：100。

  有关 Plasma 的更多信息，请参阅 Ray 文档中的 Plasma 内存对象存储。
- `--object_store_memory_worker` — 分配给 Ray Worker 节点上的 Plasma 对象存储的内存。这些实例仅运行工作线程副本。该值表示热启动后实例上可用内存的百分比。此参数用于调整内存密集型工作负载，默认值对于大多数使用案例来说是可接受的。格式：正整数。最小值：1。最大值：100。

  有关 Plasma 的更多信息，请参阅 Ray 文档中的 Plasma 内存对象存储。
- `--pip-install` — 一组要安装的 Python 软件包。可以使用此参数从 PyPI 安装软件包。格式：以逗号分隔的列表。

  PyPI 软件包条目的格式为 `package==version`，带有目标软件包的 PyPI 名称和版本。条目使用 Python 版本匹配以匹配软件包和版本，例如 ==，而不是单一等于 =。还有其他版本匹配运算符。有关更多信息，请参阅 Python 网站上的 PEP 440。您还可以使用 `--s3-py-modules` 提供自定义模块。
- `--s3-py-modules` - 一组托管 Python 模块分发的 Amazon S3 路径。格式：以逗号分隔的列表。

您可以用它来将自己的模块分发到您的 Ray 作业中。您还可以使用 `--pip-install` 提供来自 PyPI 的模块。与 AWS Glue ETL 不同，自定义模块不是通过 pip 设置的，而是传递给 Ray 供分发。有关更多信息，请参阅 the section called “用于 Ray 作业的其他 Python 模块”。

- `--working-dir` - 托管在 Amazon S3 中的.zip 文件的路径，该文件包含要分发到运行 Ray 作业的所有节点的文件。格式：字符串。有关更多信息，请参阅 the section called “为您的 Ray 作业提供文件”。

使用指标监控 Ray 作业

您可以使用 AWS Glue Studio 和 Amazon CloudWatch 监控 Ray 作业。CloudWatch 可从 AWS Glue 收集和处理 Ray 中的原始指标，以便于分析。这些指标在 AWS Glue Studio 控制台中可视化，因此您可以在作业运行时对其进行监控。

有关如何监控 AWS Glue 的一般概述，请参阅the section called “使用 CloudWatch 指标”。有关如何使用 AWS Glue 发布的 CloudWatch 指标的总体概述，请参阅the section called “使用 CloudWatch 进行监控”。

在 AWS Glue 控制台中监控 Ray 作业

在作业运行的详细信息页面上的运行详细信息部分下面，您可以查看预先构建的聚合图，这些图可以可视化您的可用作业指标。AWS Glue Studio 将作业指标发送到 CloudWatch，用于每次作业运行。借助这些功能，您可以建立集群和任务的配置文件，也可以访问有关每个节点的详细信息。

有关可用指标图的更多信息，请参阅 the section called “查看 Ray 作业运行的 Amazon CloudWatch 指标”。

CloudWatch 中 Ray 作业指标概述

在 CloudWatch 中启用详细监控后，我们会发布 Ray 指标。指标已发布到 `Glue/Ray` CloudWatch 命名空间。

- 实例指标

  我们发布有关分配给作业的实例的 CPU、内存和磁盘利用率的指标。这些指标由 `ExecutorId`、`ExecutorType` 和 `host` 等功能标识。这些指标是标准 Linux CloudWatch 代理指标的子集。您可以在 CloudWatch 文档中找到有关指标名称和功能的信息。有关详细信息，请参阅 CloudWatch 代理收集的指标。
- Ray 集群指标

我们会将运行脚本的 Ray 进程的指标转发到这个命名空间，然后为您提供最关键的指标。可用指标可能因 Ray 版本而异。有关您的作业运行的 Ray 版本的更多信息，请参阅the section called “AWS Glue 版本”。

Ray 在实例级别收集指标。同时还提供任务和集群的指标。有关 Ray 基础指标策略的更多信息，请参阅 Ray 文档中的 Metrics。

> **Note**
>
> 我们不会将 Ray 指标发布到 `Glue/Job Metrics/` 命名空间，该命名空间仅用于 AWS Glue ETL 任务。

在 AWS Glue 中为 Python shell 作业配置作业属性

可以使用 Python shell 作业将 Python 脚本作为 AWS Glue 中的 shell 运行。利用 Python Shell 作业，您可以运行与 Python 3.6 或 Python 3.9 兼容的脚本。

主题

- 限制
- 定义 Python shell 作业的作业属性
- 适用于 Python shell 作业的支持的库
- 提供您自己的 Python 库
- 将 AWS CloudFormation 与 AWS Glue 中的 Python Shell 作业一起使用

限制

请注意 Python Shell 作业的以下限制：

- 您不能将作业书签用于 Python shell 作业。
- 在 Python 3.9+ 版本中，不能将任何 Python 库打包成 `.egg` 文件。请改用`.whl`。
- 由于 S3 数据的临时副本存在限制，因此无法使用 `--extra-files` 选项。

定义 Python shell 作业的作业属性

以下各节介绍如何在 AWS Glue Studio 中或使用 AWS CLI 定义任务属性。

AWS Glue Studio

当您在 AWS Glue Studio 中定义 Python shell 任务时，请提供以下一些属性：

IAM 角色

指定用于对运行任务和访问数据存储所用的资源进行授权的 AWS Identity and Access Management（IAM）角色。有关在 AWS Glue 中运行作业的权限的更多信息，请参阅 AWS Glue 的身份和访问管理。

类型

选择 Python shell 命令以使用名为 `pythonshell` 的作业运行 Python 脚本。

Python 版本

选择 Python 版本。默认为 Python 3.9。有效版本为 Python 3.6 和 Python 3.9。

加载常用分析库（推荐）

选择此选项可在 Python shell 中包含适用于 Python 3.9 的常用库。

如果您的库是自定义的或者与预安装的库冲突，则可以选择不安装常用库。但是，除了常用库之外，您还可以安装其他库。

如果您选择此选项，则 `library-set` 选项设置为 `analytics`。如果您取消选择此选项，则 `library-set` 选项设置为 `none`。

脚本文件名和脚本路径

脚本中的代码定义了作业的过程逻辑。您需要在 Amazon Simple Storage Service（Amazon S3）中提供脚本名称和位置。确认没有与路径中的脚本目录同名的文件。要了解有关使用脚本的更多信息，请参阅AWS Glue 编程指南。

Script

脚本中的代码定义了作业的过程逻辑。您可以在 Python 3.6 或 Python 3.9 中编写脚本代码。您可以在 AWS Glue Studio 中编辑脚本。

数据处理单元

此作业运行时可分配的 AWS Glue 数据处理单元 (DPU) 的最大数量。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅AWS Glue 定价。

您可以将该值设置为 0.0625 或 1。默认值为 0.0625。无论哪种情况，该实例的本地磁盘都将为 20GB。

CLI

您还可以使用 AWS CLI 创建 Python Shell 任务，如以下示例所示。

```
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**
>
> 您无需指定 AWS Glue 的版本，因为参数 `--glue-version` 不适用于 AWS Glue Shell 作业。指定的任何版本都将被忽略。

您使用 AWS CLI 创建的任务默认为 Python 3。有效的 Python 版本为 3（对应于 3.6）和 3.9。要指定 Python 3.6，请添加此元组到 `--command` 参数：`"PythonVersion":"3"`

要指定 Python 3.9，请添加此元组到 `--command` 参数：`"PythonVersion":"3.9"`

要设置 Python shell 作业使用的最大容量，请提供 `--max-capacity` 参数。对于 Python shell 作业，无法使用 `--allocated-capacity` 参数。

适用于 Python shell 作业的支持的库

在使用 Python 3.9 的 Python shell 中，您可以选择库集来使用预先打包的库集来满足您的需求。您可以使用 `library-set` 选项选择库集。有效值为 `analytics` 和 `none`。

用于运行 Python shell 作业的环境支持以下库：

Python 版本	Python 3.6	Python 3.9	
库集	不适用	分析	none
avro		1.11.0	

Python 版本	Python 3.6	Python 3.9	
awscli	116.242	1.23.5	1.23.5
awswrangler		2.15.1	
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	
请求	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	

可以在 Python shell 作业中使用 NumPy 库以进行科学计算。有关更多信息，请参阅 NumPy。以下示例显示了一个可在 Python shell 作业中使用的 NumPy 脚本。该脚本输出“Hello world”和多个数学计算的结果。

```
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)
```

提供您自己的 Python 库

使用 PIP

使用 Python 3.9 的 Python shell 允许您在任务层面提供其他 Python 模块或不同版本。您可以结合使用 `--additional-python-modules` 选项与一系列逗号分隔的 Python 模块，以添加新模块或更改现有模块的版本。使用 Python Shell 作业时，您不能为托管在 Amazon S3 上的自定义 Python 模块提供此参数。

例如，要更新或添加新的 `scikit-learn` 模块，请使用以下键/值：`"--additional-python-modules", "scikit-learn==0.21.3"`。

AWS Glue 使用 Python Package Installer（pip3）来安装其他模块。您可以在 `--additional-python-modules` 值中传递额外的 pip3 选项。例如，`"scikit-learn==0.21.3 -i https://pypi.python.org/simple/"`。pip3 的任何不兼容或限制都将适用。

> **Note**
>
> 为避免将来出现不兼容性，我们建议使用为 Python 3.9 构建的库。

使用 Egg 或 Whl 文件

您可能已将一个或多个 Python 库打包为一个 `.egg` 或 `.whl` 文件。如果是这样的话，您可以使用“`--extra-py-files`”标记下的 AWS Command Line Interface (AWS CLI) 将其指定到您的作业，如以下示例所示。

```
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"]}'
```

如果您不确定如何从 Python 库创建 `.egg` 或 `.whl` 文件，请使用以下步骤。此示例适用于 macOS、Linux 和 Windows Linux 子系统 (WSL)。

创建 Python .egg 或 .whl 文件

1. 在 Virtual Private Cloud（VPC）中创建 Amazon Redshift 集群，并且将一些数据添加到表。
2. 为您用于创建集群的 VPC-SecurityGroup-Subnet 组合创建 AWS Glue 连接。测试连接是否成功。
3. 创建一个名为 `redshift_example` 的目录，并且创建一个名为 `setup.py` 的文件。将以下代码粘贴到 `setup.py`。

   ```
   from setuptools import setup

   setup(
       name="redshift_module",
       version="0.1",
       packages=['redshift_module']
   )
   ```

4. 在 `redshift_example` 目录中，创建一个 `redshift_module` 目录。在 `redshift_module` 目录，创建文件 `__init__.py` 和 `pygresql_redshift_common.py`。

5. 将 `__init__.py` 文件留空。在 `pygresql_redshift_common.py` 中，粘贴以下代码。将 *port*、*db_name*、*user* 和 *password_for_user* 替换为特定于您的 Amazon Redshift 集群的详细信息。将 *table_name* 替换为 Amazon Redshift 中表的名称。

```
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. 如果您尚未在此处，请切换到 `redshift_example` 目录。
7. 请执行以下操作之一：

   - 要创建 `.egg` 文件，请运行以下命令。

     ```
     python setup.py bdist_egg
     ```

   - 要创建 `.whl` 文件，请运行以下命令。

     ```
     python setup.py bdist_wheel
     ```

8. 安装上述命令中所需的依赖项。
9. 此命令会在 `dist` 目录中创建一个文件：

   - 如果您创建了一个 egg 文件，则此文件的名称为 `redshift_module-0.1-py2.7.egg`。
   - 如果您创建了一个 wheel 文件，则此文件的名称为 `redshift_module-0.1-py2.7-none-any.whl`。

将此文件上载到 Amazon S3。

在本例中，上传的文件路径是 *s3://DOC-EXAMPLE-BUCKET/EGG-FILE* 或 *s3://DOC-EXAMPLE-BUCKET/WHEEL-FILE*。

10. 创建一个要用作 AWS Glue 作业的脚本的 Python 文件，并将以下代码添加到该文件。

```
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. 将上述文件上载到 Amazon S3。在本例中，上传的文件路径是 *s3://DOC-EXAMPLE-BUCKET/scriptname.py*。
12. 使用此脚本创建一个 Python shell 作业。在 AWS Glue 控制台上，使用 Job properties (任务属性) 页面上的 Python library path (Python 库路径) 框指定 `.egg/.whl` 文件的路径。如果您有多个 `.egg/.whl` 文件和 Python 文件，请在此框中提供逗号分隔的列表。

    修改或重命名 `.egg` 文件时，文件名必须使用由“python setup.py bdist_egg”命令生成的默认名称，或者必须遵守 Python 模块命名约定。有关更多信息，请参阅 Python 代码风格指南。

    借助 AWS CLI，使用命令创建作业，如以下示例中所示。

```
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"]}'
```

    任务运行时，脚本会打印在 Amazon Redshift 集群的 *table_name* 表中创建的行。

将 AWS CloudFormation 与 AWS Glue 中的 Python Shell 作业一起使用

您可以将 AWS CloudFormation 与 AWS Glue 中的 Python Shell 作业一起使用。以下是 示例：

```
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
```

用于 Amazon CloudWatch Logs 组的 Python shell 任务输出为 `/aws-glue/python-jobs/output`。有关错误，请参阅日志组 `/aws-glue/python-jobs/error`。

监控 AWS Glue

监控是保持 AWS Glue 和您的其他 AWS 解决方案的可靠性、可用性和性能的重要方面。AWS 提供了一些监控工具，您可以用来监控 AWS Glue、在出现错误时进行报告并适时自动采取措施：

您可以使用以下自动化监控工具来监控 AWS Glue 并在出现错误时报告：

- Amazon CloudWatch Events 提供几乎实时的系统事件流，这些事件描述 AWS 资源的更改。CloudWatch Events 支持自动事件驱动型计算。您可以编写规则，以监控某些事件和在这些事件发生时在其他 AWS 服务中触发自动操作。有关更多信息，请参阅 Amazon CloudWatch Events 用户指南。
- Amazon CloudWatch Logs 使您能够监控、存储和访问来自 Amazon EC2 实例、AWS CloudTrail 和其他来源的日志文件。CloudWatch Logs 可以监控日志文件中的信息，并在达到特定阈值时通知您。您还可以在高持久性存储中检索您的日志数据。有关更多信息，请参阅 Amazon CloudWatch Logs 用户指南。
- AWS CloudTrail 捕获由您的 AWS 账户或代表该账户发出的 API 调用和相关事件，并将日志文件传输到您指定的 Amazon S3 存储桶。您可以标识哪些用户和账户调用了 AWS、从中发出调用的源 IP 地址以及调用的发生时间。有关更多信息，请参阅《AWS CloudTrail 用户指南》。

此外，您还可以访问 AWS Glue 控制台中的以下见解，以帮助您调试和分析任务：

- Spark 任务 – 您可以查看所选 CloudWatch 指标系列的可视化效果，而较新的任务可以访问 Spark UI。有关更多信息，请参阅 the section called “监控 Spark 作业”。
- Ray 任务 – 您可以查看所选 CloudWatch 指标系列的可视化效果。有关更多信息，请参阅 the section called “Ray 作业指标”。

主题

AWS Glue 中的 AWS 标签

为了便于您管理 AWS Glue 资源，您可以选择将自己的标签分配给某些 AWS Glue 资源类型。标签是为AWS资源分配的标记。每个标签都包含定义的一个键 和一个可选值。您可以在 AWS Glue 中使用标签来组织和标识资源。标签可用于创建成本会计报告并限制对资源的访问。如果您使用的是 AWS Identity and Access Management，则可以控制 AWS 账户中的哪些用户拥有创建、编辑或删除标签的权限。除了调用标签相关的 API 的权限外，您还需要在连接上调用标记 API 的 `glue:GetConnection` 权限以及对数据库调用标记 API 的 `glue:GetDatabase` 权限。有关更多信息，请参阅 带有 Glue 的 ABA AWS C。

在 AWS Glue 中，您可以为以下资源添加标签：

- Connection
- 数据库
- 爬网程序
- 交互式会话
- 开发终端节点
- 任务
- 触发器
- 工作流
- 蓝图

- 机器学习转换
- 数据质量规则集
- 流架构
- 流架构注册表

> **Note**
>
> 作为最佳实践，要允许标记这些 AWS Glue 资源，请始终在您的策略中包含 `glue:TagResource` 操作。

在将标签用于 AWS Glue 时注意以下事项。

- 每个实体支持最多 50 个标签。
- 在 AWS Glue 中，您可以将标签指定为格式为 `{"string": "string" ...}` 的键值对列表
- 在对象上创建标签时，标签键是必需的，而标签值是可选的。
- 标签键和标签值区分大小写。
- 标签键和标签值不得包含前缀 aws。禁止对此类标签进行操作。
- 最大标签键长度为 128 个 Unicode 字符 (采用 UTF-8 格式)。标签键不得为空或为 null。
- 最大标签值长度为 256 个 Unicode 字符 (采用 UTF-8 格式)。标签值可以为空或为 null。

对 AWS Glue 连接的标记支持

您可以根据资源标签限制 `CreateConnection`、`UpdateConnection`、`GetConnection` 和 `DeleteConnection` 操作权限。这使您能够对需要从数据目录获取 JDBC 连接信息的 JDBC 数据源 AWS Glue 任务实施最低权限访问控制。

示例用法

创建带有标签 ["connection-category", "dev-test"] (“连接-类别”，“开发-测试”) 的 AWS Glue 连接。

为 IAM policy 中的 `GetConnection` 操作指定标签条件。

```
{
   "Effect": "Allow",
   "Action": [
     "glue:GetConnection"
```

```
    ],
   "Resource": "*",
   "Condition": {
     "ForAnyValue:StringEquals": {
       "aws:ResourceTag/tagKey": "dev-test"
     }
   }
 }
```

示例

以下示例创建一个具有分配的标签的作业。

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
```

有关更多信息，请参阅 AWS 标记策略。

有关如何使用标签控制访问的信息，请参阅 带有 Glue 的 ABA AWS C。

使用 CloudWatch Events 实现 AWS Glue 自动化

您可以使用 Amazon CloudWatch Events 自动执行您的 AWS 服务并自动响应系统事件，例如应用程序可用性问题或资源更改。AWS 服务中的事件近乎实时地传输到 CloudWatch Events。您可以编写简单规则来指示您关注的事件，并指示要在事件匹配规则时执行的自动化操作。可自动触发的操作包括：

- 调用 AWS Lambda 函数
- 调用 Amazon EC2 Run Command
- 将事件中继到 Amazon Kinesis Data Streams
- 激活 AWS Step Functions 状态机
- 通知 Amazon SNS 主题或 Amazon SQS 队列

一些将 CloudWatch Events 与 AWS Glue 结合使用的示例包括：

- 当 ETL 任务成功时激活 Lambda 函数
- 当 ETL 任务失败时通知 Amazon SNS 主题

以下 CloudWatch Events 由 AWS Glue 生成。

- 针对 SUCCEEDED、FAILED、TIMEOUT 和 STOPPED 生成的 "detail-type":"Glue Job State Change" 事件。
- 当超过作业延迟通知阈值时，则会针对 RUNNING、STARTING 和 STOPPING 作业运行生成 "detail-type":"Glue Job Run Status" 事件。必须设置作业延迟通知阈值属性才能接收这些事件。

  当超过作业延迟通知阈值时，每个作业运行状态仅生成一个事件。
- 针对 Started、Succeeded 和 Failed 生成 "detail-type":"Glue Crawler State Change" 事件。
- 针对 CreateDatabase、DeleteDatabase、CreateTable、DeleteTable 和 BatchDeleteTable 生成 "detail-type":"Glue Data Catalog Database State

Change" 的事件。例如，如果创建或删除表，系统会向 CloudWatch Events 发送一条通知。请注意，您不能编写取决于通知事件的顺序或存在的程序，因为它们可能是乱序或缺失的。尽最大努力发出事件。在通知的详细信息中：

- typeOfChange 包含 API 操作的名称。
- databaseName 包含受影响数据库的名称。
- changedTables 对于每个通知最多包含 100 个受影响的表的名称。当表名称过长时，可创建多个通知。

- 针对 UpdateTable、CreatePartition、BatchCreatePartition、UpdatePartition、DeleteParti 和 BatchDeletePartition 生成的 "detail-type":"Glue Data Catalog Table State Change" 事件。例如，如果更新表或分区，系统会向 CloudWatch Events 发送一条通知。请注意，您不能编写取决于通知事件的顺序或存在的程序，因为它们可能是乱序或缺失的。尽最大努力发出事件。在通知的详细信息中：
  - typeOfChange 包含 API 操作的名称。
  - databaseName 包含其中包含受影响资源的数据库的名称。
  - tableName 包含受影响的表的名称。
  - changedPartitions 在一条通知中最多指定 100 个受影响的分区。当分区名称过长时，可创建多个通知。

    例如，如果有两个分区键 Year 和 Month，则 "2018,01", "2018,02" 会修改分区（其中 "Year=2018" and "Month=01"）和分区（其中 "Year=2018" and "Month=02"）。

```
{
    "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"
        }
}
```

有关更多信息，请参阅 Amazon CloudWatch Events 用户指南。有关特定于 AWS Glue 的事件，请参阅 AWS Glue 事件。

AWS Glue 资源监控

AWS Glue 具有服务限制，可保护客户免受意外过度配置和旨在增加账单的恶意行为的侵害。这些限制还可以保护服务。登录 AWS Service Quota 控制台，客户可以查看其当前的资源限制并请求增加限额（如适用）。

AWS Glue 允许您在 Amazon CloudWatch 中以百分比形式查看服务的资源使用情况，并在其上配置 CloudWatch 警报以监控使用情况。Amazon CloudWatch 提供对在 Amazon 基础设施上运行的 AWS 资源和客户应用程序的监控。您可以免费使用这些指标。支持以下指标：

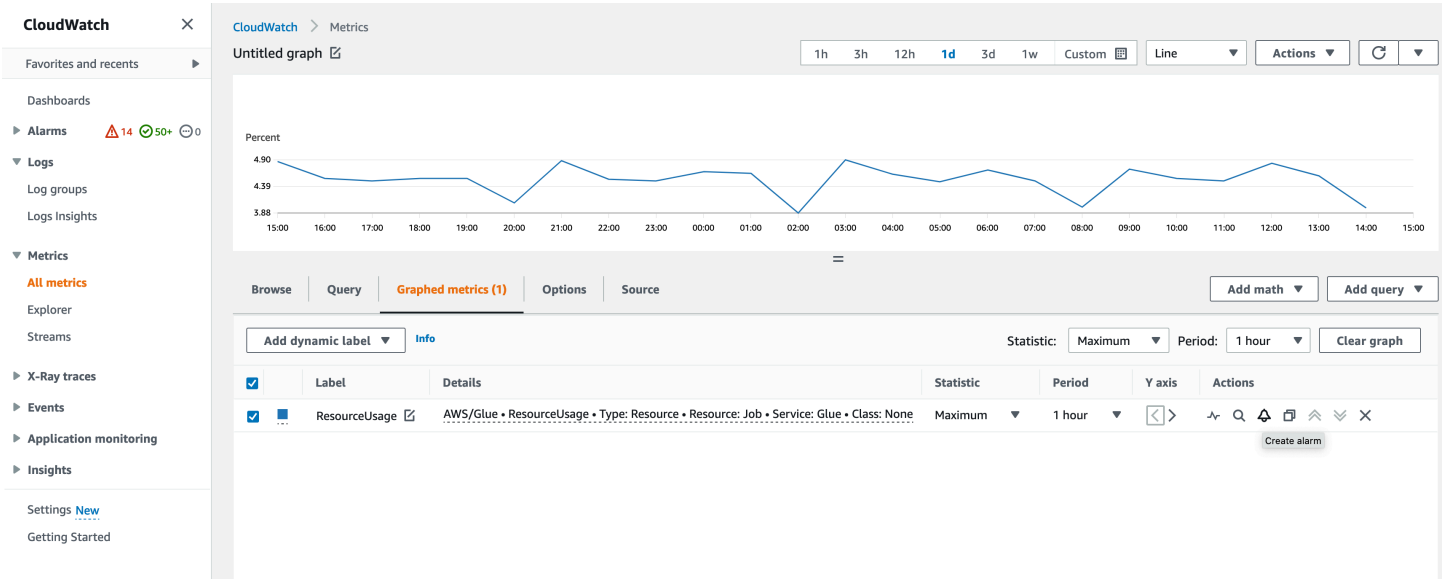
- 每个账户的工作流数量
- 每个账户的触发器数
- 每个账户的任务数
- 每个账户的并发作业运行数
- 每个账户的蓝图数量
- 每个账户的互动会话次数

配置和使用资源指标

要使用此功能，您可以前往 Amazon CloudWatch 控制台查看指标并配置警报。这些指标位于 AWS/Glue 命名空间下，是实际资源使用计数除以资源限额的百分比。CloudWatch 指标将发送到您的账户，这对您来说是免费的。例如，如果您创建了 10 个工作流程，并且您的服务限额允许最多拥有 200 个工作流程，则您的使用量为 10/200 = 5%，在图表中，您将看到一个百分比为 5 的数据点。更具体地说：

```
Namespace: AWS/Glue
Metric name: ResourceUsage
Type: Resource
Resource: Workflow (or Trigger, Job, JobRun, Blueprint, InteractiveSession)
```

```
Service: Glue
Class: None
```



在 CloudWatch 控制台中针对指标创建警报

1. 找到指标后，转到绘成图表的指标。
2. 单击操作下的创建警报。
3. 根据需要配置警报。

每当您的资源使用量发生变化（例如增加或减少）时，我们都会发布指标。但是，如果您的资源使用量没有变化，我们会每小时发布一次指标，这样您就可以获得连续的 CloudWatch 图表。为避免丢失数据点，我们建议您不要配置少于 1 小时的时间段。

您也可以使用 AWS CloudFormation 配置警报，如以下示例所示。在本例中，一旦工作流程资源使用量达到 80%，就会触发警报，向现有 SNS 主题发送消息，您可以订阅该主题以获取通知。

```
{
 "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 记录 AWS Glue API 调用

AWS Glue 与 AWS CloudTrail 集成，后者是在 AWS 中记录用户、角色或 AWS Glue 服务所执行操作的服务。CloudTrail 将 AWS Glue 的所有 API 调用作为事件捕获。捕获的调用包含来自 AWS Glue 控制台和代码的 AWS Glue API 操作调用。如果您创建跟踪，则可以使 CloudTrail 事件持续传送到 Amazon S3 存储桶（包括 AWS Glue 的事件）。如果您不配置跟踪记录，则仍可在 CloudTrail 控制台中的事件历史记录中查看最新事件。使用 CloudTrail 收集的信息，您可以确定向 AWS Glue 发出了什么请求、发出请求的 IP 地址、何人发出的请求、请求的发出时间以及其他详细信息。

要了解有关 CloudTrail 的更多信息，请参阅AWS CloudTrail用户指南。

CloudTrail 中的 AWS Glue 信息

在您创建 AWS 账户时，将在该账户上启用 CloudTrail。当 AWS Glue 中发生活动时，该活动将记录在 CloudTrail 事件中，并与其他 AWS 服务事件一同保存在事件历史记录 中。您可以在 AWS 账户中查看、搜索和下载最新事件。有关更多信息，请参阅使用 CloudTrail 事件历史记录查看事件。

要持续记录 AWS 账户中的事件（包括 AWS Glue 的事件），请创建跟踪。通过跟踪记录，CloudTrail 可将日志文件传送至 Amazon S3 存储桶。预设情况下，在控制台中创建跟踪时，此跟踪应用于所有 AWS 区域。此跟踪记录在 AWS 分区中记录所有区域中的事件，并将日志文件传送至您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，进一步分析在 CloudTrail 日志中收集的事件数据并采取行动。有关更多信息，请参阅下列内容：

- 为您的 AWS 账户创建跟踪
- CloudTrail supported services and integrations
- 为 CloudTrail 配置 Amazon SNS 通知
- 从多个区域接收 CloudTrail 日志文件和从多个账户接收 CloudTrail 日志文件

所有 AWS Glue 操作都由 CloudTrail 记录，并记录在 AWS Glue API 中。例如，对 `CreateDatabase`、`CreateTable` 和 `CreateScript` 操作的调用会在 CloudTrail 日志文件中生成条目。

每个事件或日志条目都包含有关生成请求的人员信息。身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 IAM 用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其它 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

但是，CloudTrail 不记录有关调用的所有信息。例如，它不会记录某些敏感信息 (如在连接请求中使用的 `ConnectionProperties`)，而且它记录 `null` 而不是以下 API 返回的响应：

```
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 日志文件条目

跟踪是一种配置，可用于将事件作为日志文件传送到您指定的 Amazon S3 存储桶。CloudTrail 日志文件包含一个或多个日记账条目。一个事件表示来自任何源的一个请求，包括有关所请求的操作、操作的日期和时间、请求参数等方面的信息。CloudTrail 日志文件不是公用 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序显示。

下面的示例显示了一个 CloudTrail 日志条目，该条目演示了 `DeleteCrawler` 操作。

```
{
  "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"
}
```

此示例显示了一个 CloudTrail 日志条目，该条目说明了 `CreateConnection` 操作。

```
{
  "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 作业运行状态

您可以查看 AWS Glue 提取、转换和加载（ETL）任务在运行时或停止后的状态。您可以使用 AWS Glue 控制台、AWS Command Line Interface（AWS CLI）或 AWS Glue API 中的 GetJobRun action (GetJobRun 操作) 查看状态。

可能的任务运行状态为
STARTING、RUNNING、STOPPING、STOPPED、SUCCEEDED、FAILED、ERROR、WAITING 和 TIMEOUT。

下表列出了指示异常任务终止的状态。

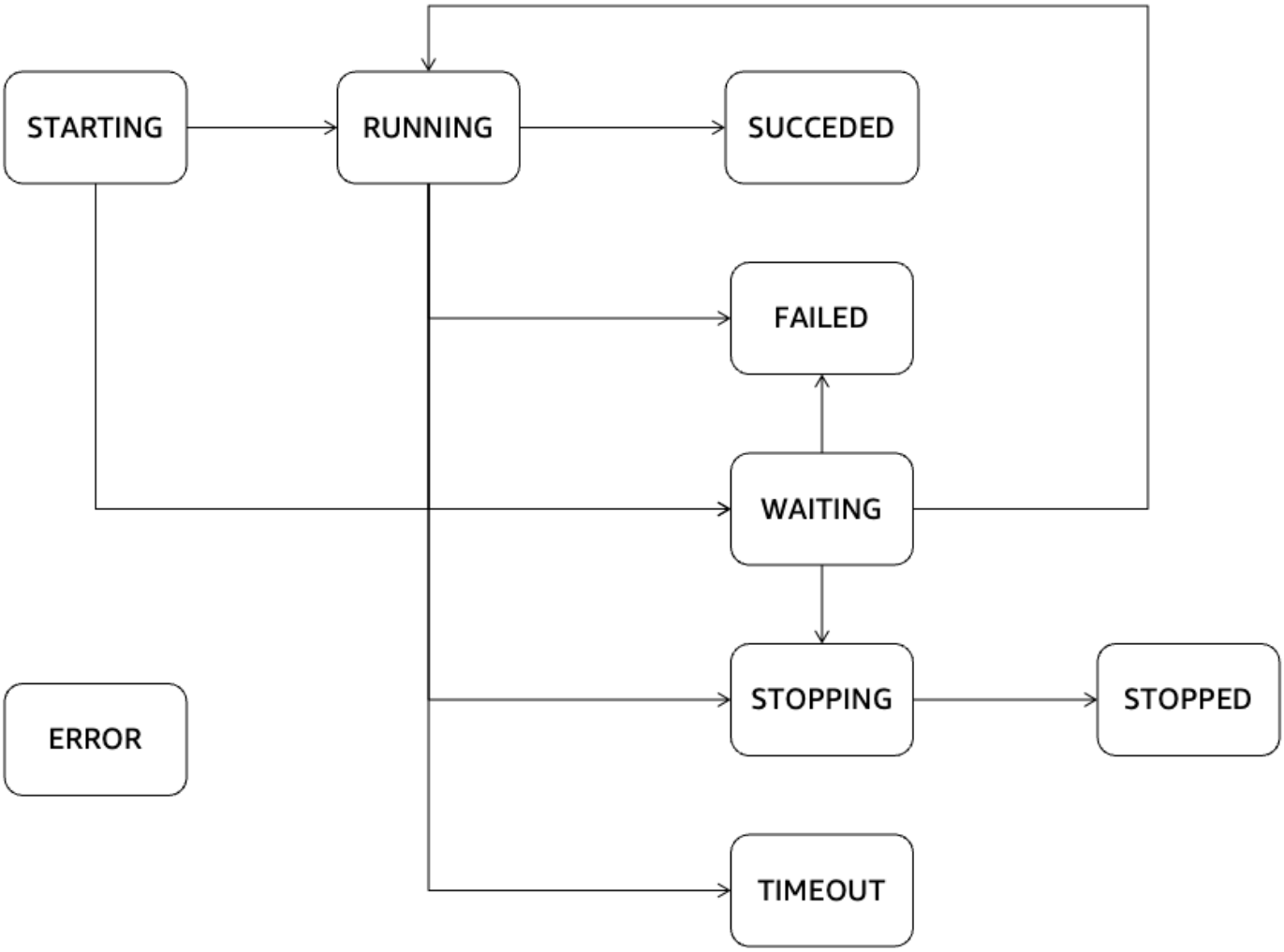
任务运行状态	描述
FAILED	任务超过了允许的最大并发运行数，或以未知的退出代码终止。
ERROR	工作流、计划触发器或事件触发器试图运行已删除的任务。
TIMEOUT	任务运行时间超过了其指定的超时值。

WAITING 状态表示作业运行正在等待资源。下表描述了不同作业类别的等待行为。

作业类型	行为
Spark 作业（标准）	尚未配置为根据您的 maxRetries 配置进行重试的作业可能会进入“等待”状态。如果服务无法获取足够的资源来启动运行，则新作业运行将处于“等待”状态。可能的原因包括您账户的服务限额或您所使用区域的容量限制出现了下列任何一种错误情况： • 超出了最大每账户并发作业运行数 • 超出了最大每作业并发作业运行数（包括账户级别服务限额以及您使用 MaxConcurrentRuns 指定的作业限制） • 超出了最大并发计算（DPU 使用量）限制 • 资源不可用 有关 AWS Glue 服务限额的更多信息，请参阅 AWS Glue 端点和限额。AWS Glue 等待资源的时间可能因具体情况而异。在尝试获取资

作业类型	行为
	源时，作业可能会在非终端状态之间切换。如果无法获取资源，作业最终将变为 FAILED 状态。AWSGlue 将最长重试 15 分钟或最多重试 10 次，以先到者为准。
Spark 作业（灵活）	如果服务无法获取足够的资源来启动运行，则新任务运行将处于 WAITING（等待）状态，这会延迟运行的开始。运行将最长 20 分钟处于 WAITING（等待）状态（超时由服务控制）。15 分钟后，该服务将尝试强制启动，根据可用容量，运行可能会开始或失败，并显示相应的错误消息。
Python shell 作业	行为与使用 Spark 的标准作业相同。

以下状态图概述了 AWS Glue 作业生命周期中预期的状态转换。此信息适用于所有作业类型。


STARTING
RUNNING
SUCCEDED
FAILED
WAITING
STOPPING
STOPPED
ERROR
TIMEOUT

AWS Glue 直播

AWS Glue 流媒体是其中的一个组成部分 AWS Glue，它使您能够近乎实时地高效处理流数据，从而使您能够执行关键任务，例如数据摄取、处理和机器学习。使用 Apache Spark Streaming 框架， AWS Glue Streaming 提供了一种可以大规模处理流数据的无服务器服务。 AWS Glue 在 Apache Spark 的基础上提供各种优化，例如无服务器基础架构、自动缩放、可视化作业开发、用于流式处理作业的即时启动笔记本以及其他性能改进。

流式处理用例

AWS Glue 直播的一些常见用例包括：

N ear-real-time 数据处理： AWS Glue 流媒体允许组织近乎实时地处理流数据，使他们能够根据最新信息获得见解并及时做出决策。

欺诈检测：您可以利用 AWS Glue 流媒体对流媒体数据进行实时分析，这对于检测欺诈活动（例如信用卡欺诈、网络入侵或在线诈骗）很有价值。通过持续处理和分析传入数据，您可以快速识别可疑模式或异常情况。

社交媒体分析： AWS Glue 流媒体可以处理实时社交媒体数据，例如推文、帖子或评论，使组织能够实时监控趋势、情绪分析和管理品牌声誉。

物联网 (IoT) 分析： AWS Glue 流媒体适用于处理和分析物联网设备、传感器和联网机器生成的高速数据流。它允许实时监控、异常检测、预测性维护和其他物联网分析用例。

点击流分析： AWS Glue 流媒体可以处理和分析来自网站或移动应用程序的实时点击流数据。这使企业能够深入了解用户行为，个性化用户体验，根据实时点击流数据优化营销活动。

日志监控和分析： AWS Glue 流媒体可以持续实时处理和分析来自服务器、应用程序或网络设备的日志数据。这有助于检测异常、排查问题、监控系统运行状况和性能。

推荐系统： AWS Glue 流媒体可以实时处理用户活动数据并动态更新推荐模型。这允许根据用户行为和偏好进行个性化和实时推荐。

以下是一些可以应用 AWS Glue 流媒体的不同用例的示例。它与 AWS 生态系统和托管服务的集成使其成为云端实时流处理和分析的便捷选择。

使用 AWS Glue 直播有什么好处？

使用流 AWS Glue 媒体的好处如下：

- 无服务器： AWS Glue 流媒体是无服务器的，因此无需管理基础架构。这减少了运营开销，使用户可以专注于数据处理和分析任务，而不是基础设施管理。
- 自动扩展： AWS Glue 流媒体提供自动缩放功能，可根据工作负载动态调整处理能力。它会自动扩展或缩减以处理数据量的波动，从而确保最佳性能和资源利用率。
- 视觉开发：流媒体作业开发可能很复杂。 AWS Glue 流媒体通过提供可视化创作工具 AWS Glue Studio 来应对这一挑战。 AWS Glue Studio 简化了创建流媒体工作流程的过程，使开发人员能够直观地设计和管理流媒体应用程序，从而缩短学习曲线并提高工作效率。
- 经济高效：作为一项无服务器服务， AWS Glue 流媒体无需配置和维护基础架构，从而提高了成本效益。用户根据流式处理作业执行期间消耗的资源付费，从而根据实际使用量进行成本优化和扩缩。
- 处理复杂的工作负载： AWS Glue 流媒体专为处理复杂的流媒体工作负载而设计。它可以处理和分析大量实时数据，支持高级转换，并与其他 AWS 服务集成，从而实现复杂的流数据管道和分析工作流程。
- 无锁定： AWS Glue 流媒体提供灵活性，避免供应商锁定。用户可以利用 AWS Glue 流媒体作为更广泛的 AWS 生态系统的一部分，将其与其他 AWS 服务无缝集成。这样就可以与现有的数据来源、应用程序和服务轻松集成，而不必受制于特定的技术或平台。

何时使用 AWS Glue 流媒体？

关于流式处理用例，有很多选择。我们建议在以下情况下进行 AWS Glue 直播。

1. 如果您已经在使用 AWS Glue 或 Spark 进行批处理，那么 AWS Glue 流媒体是您的理想选择。它可以无缝过渡到构建流式处理作业，而无需学习新的语言或框架。利用您现有的知识和基础架构， AWS Glue Streaming 简化了任务开发流程，并允许您轻松地将数据处理能力扩展到实时流式传输场景。
2. 如果您需要统一的服务或产品来处理批处理、流式处理和事件驱动的工作负载，那么 AWS Glue 流媒体就是您的理想之选。借 AWS Glue 助 Streaming，您可以将数据处理需求整合到一个框架中，从而消除管理多个系统的复杂性。这样就能高效开发和维护各种数据工作流，同时确保不同工作负载类型之间的一致性和兼容性。
3. AWS Glue 流式传输非常适合涉及超大流数据量和复杂转换（例如流或关系数据库之间的联接）的场景。它可以高效处理和分析大量数据流，使您能够轻松处理要求苛刻的工作负载。无论是高速数据摄取还是复杂的数据操作， AWS Glue Streaming 的可扩展性和高级处理能力都能确保最佳性能和准确的结果。
4. 如果您更喜欢使用可视化方法来构建流媒体作业，可以 AWS Glue 提供 AWS Glue Studio，您可以使用它直观地设计和管理流媒体应用程序，从而简化开发过程。这种直观的界面使开发人员能够使用可视化界面创建、配置和监控流式处理工作流，从而缩短学习曲线并提高工作效率。

5. AWS Glue 对于严格的 SLA（服务级别协议）超过 10 秒的 near-real-time 用例，流媒体是绝佳的选择。
6. 如果您正在使用 Apache Iceberg、Apache Hudi 或 Delta Lake 构建交易数据湖，则 AWS Glue 流媒体为这些开放表格式提供原生支持。这种无缝集成使您能够直接处理来自这些事务数据湖的流式处理数据，从而确保数据一致性、完整性和兼容性。
7. 当需要为各种数据目标摄取流数据时： AWS Glue 流式传输为各种数据目标提供原生目标，例如亚马逊 Redshift、Amazon RDS、Amazon Aurora、Oracle、SQL Server 和其他目标。

支持的数据来源

AWS Glue 流式传输支持以下数据源：

- Amazon Kinesis
- Amazon MSK（Managed Streaming for Apache Kafka）
- 自行管理的 Apache Kafka

支持的数据目标

AWS Glue 流媒体支持多种数据目标，例如：

- 数据目录支持 AWS Glue 的数据目标
- Amazon S3
- Amazon Redshift
- MySQL
- PostgreSQL
- Oracle
- Microsoft SQL Server
- Snowflake
- 任何可以使用 JDBC 连接的数据库
- Apache Iceberg、Delta 和 Apache Hudi
- AWS Glue Marketplac

教程：使用 AWS Glue Studio 构建第一个流式处理工作负载

在本教程中，您将了解如何使用 AWS Glue Studio 创建流式处理作业。AWS GlueStudio 是一个用于创建 AWS Glue 作业的可视化界面。

您可以创建连续运行的流式处理提取、转换、加载（ETL）作业，该作业使用来自 Amazon Kinesis Data Streams、Apache Kafka 和 Amazon Managed Streaming for Apache Kafka（Amazon MSK）等流式处理数据源的数据。

先决条件

要学习本教程，您需要一个具有 AWS 控制台权限的用户以使用 AWS Glue、Amazon Kinesis、Amazon S3、Amazon Athena、AWS CloudFormation、AWS Lambda 和 Amazon Cognito。

使用来自 Amazon Kinesis 的流式处理数据

主题

- 使用 Kinesis Data Generator 生成模拟数据
- 使用 AWS Glue Studio 创建 AWS Glue 流式处理作业
- 执行转换并将转换后的结果存储在 Amazon S3 中

使用 Kinesis Data Generator 生成模拟数据

您可以使用 Kinesis Data Generator（KDG）生成 JSON 格式的合成示例数据。您可以在工具文档中找到完整的说明和详细信息。

1. 首先，单击

   Launch Stack

   以在您的 AWS 环境中运行 AWS CloudFormation 模板。

   > **Note**
   >
   > 您可能会遇到 CloudFormation 模板故障，因为您的 AWS 账户中已存在某些资源，例如 Kinesis Data Generator 的 Amazon Cognito 用户。这可能是因为您已经从其他教程或博客中进行了设置。要解决此问题，您可以在一个新的 AWS 账户中重新尝试此模板，也可以尝试使用其他 AWS 区域。通过这些选项，您可以在不与现有资源冲突的情况下运行此教程。

该模板已经预置了一个 Kinesis 数据流和一个 Kinesis Data Generator 账户。它还会创建一个 Amazon S3 存储桶来保存数据，并创建一个具有本教程所需权限的 Glue 服务角色。

2. 输入用户名和密码，以便 KDG 进行身份验证。记下用户名和密码以备将来使用。
3. 选择下一步，一直到最后一步。确认 IAM 资源的创建。检查屏幕上方是否有任何错误，比如密码不符合最低要求，然后部署模板。
4. 导航到堆栈的输出选项卡。模板部署完成后，将显示生成的属性 KinesisDataGeneratorUrl。单击该 URL。
5. 输入您记下的用户名和密码。
6. 选择您正在使用的区域，然后选择 Kinesis Stream `GlueStreamTest-{AWS::AccountId}`
7. 输入以下模板：

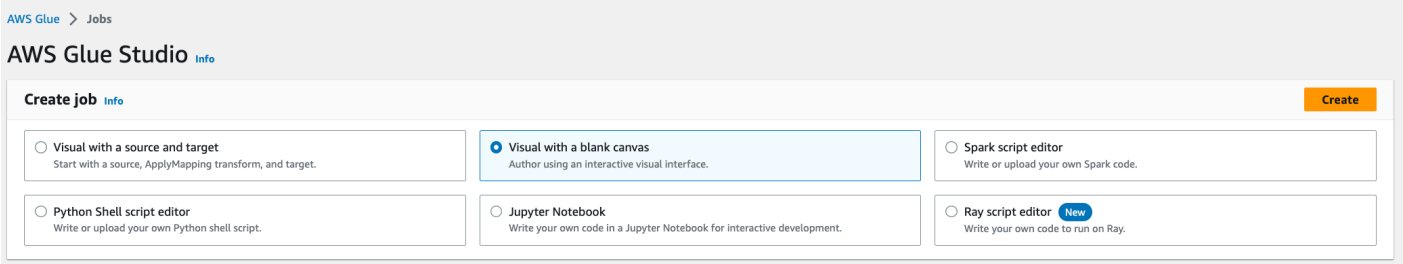
```
{
    "ventilatorid": {{random.number(100)}},
    "eventtime": "{{date.now("YYYY-MM-DD HH:mm:ss")}}",
    "serialnumber": "{{random.uuid}}",
    "pressurecontrol": {{random.number(
        {
            "min":5,
            "max":30
        }
    )}},
    "o2stats": {{random.number(
        {
            "min":92,
            "max":98
        }
    )}},
    "minutevolume": {{random.number(
        {
            "min":5,
            "max":8
        }
    )}},
    "manufacturer": "{{random.arrayElement(
        ["3M", "GE","Vyaire", "Getinge"]
    )}}"
}
```

现在，您可以使用测试模板查看模拟数据，并使用发送数据将模拟数据摄取到 Kinesis。

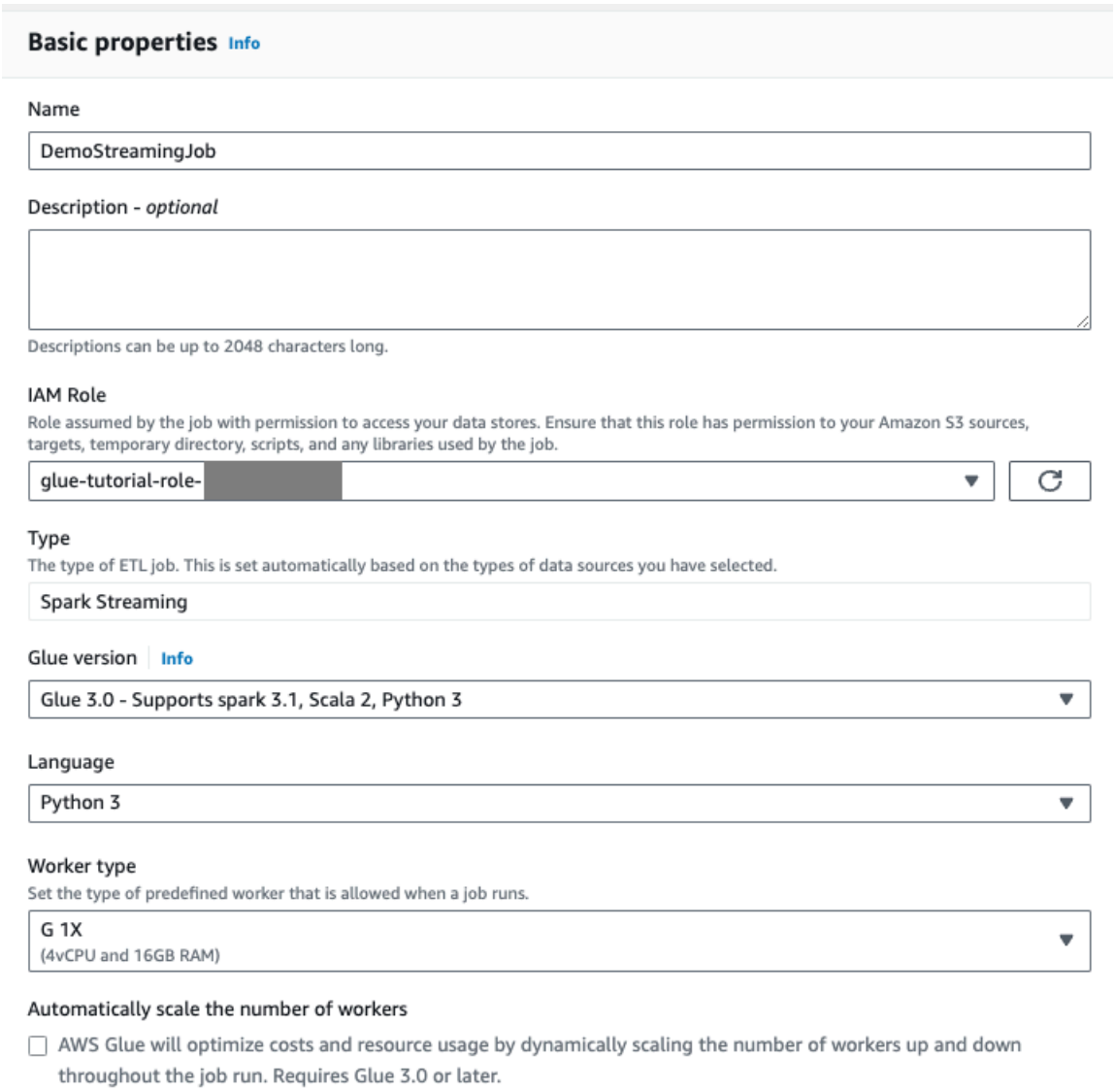
8. 单击发送数据，并向 Kinesis 生成 0.5-1 万条记录。

使用 AWS Glue Studio 创建 AWS Glue 流式处理作业

1. 在同一区域的控制台中导航到 AWS Glue。
2. 在左侧导航栏的数据集成和 ETL 下选择 ETL 作业。
3. 使用空白画布通过 Visual 创建 AWS Glue 作业。

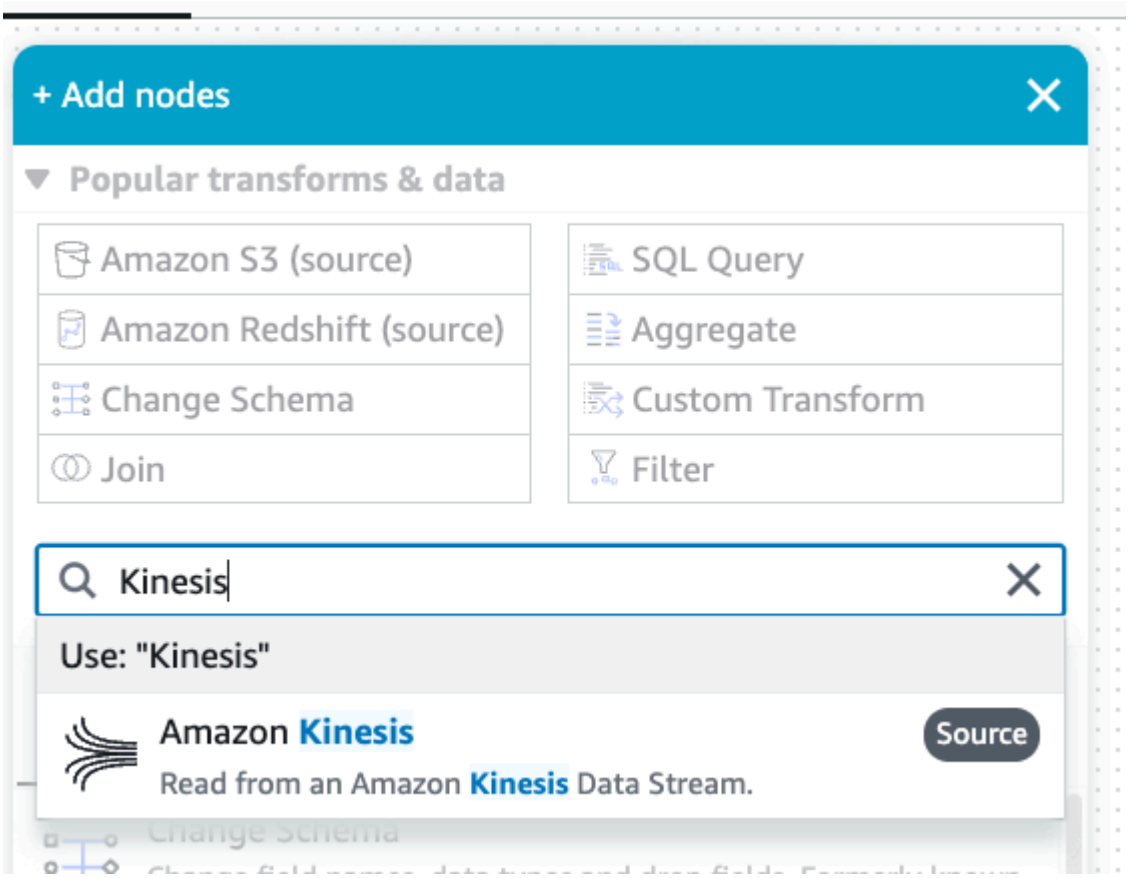


4. 导航到作业详细信息选项卡。
5. 对于 AWS Glue 作业名称，输入 `DemoStreamingJob`。
6. 对于 IAM 角色，选择 CloudFormation 模板预置的角色 `glue-tutorial-role-${AWS::AccountId}`。
7. 对于 Glue 版本，选择 Glue 3.0。将所有其他选项保留为默认。



8. 导航到 Visual 选项卡。

9. 单击加号图标。在搜索栏中输入 Kinesis。选择 Amazon Kinesis 数据来源。



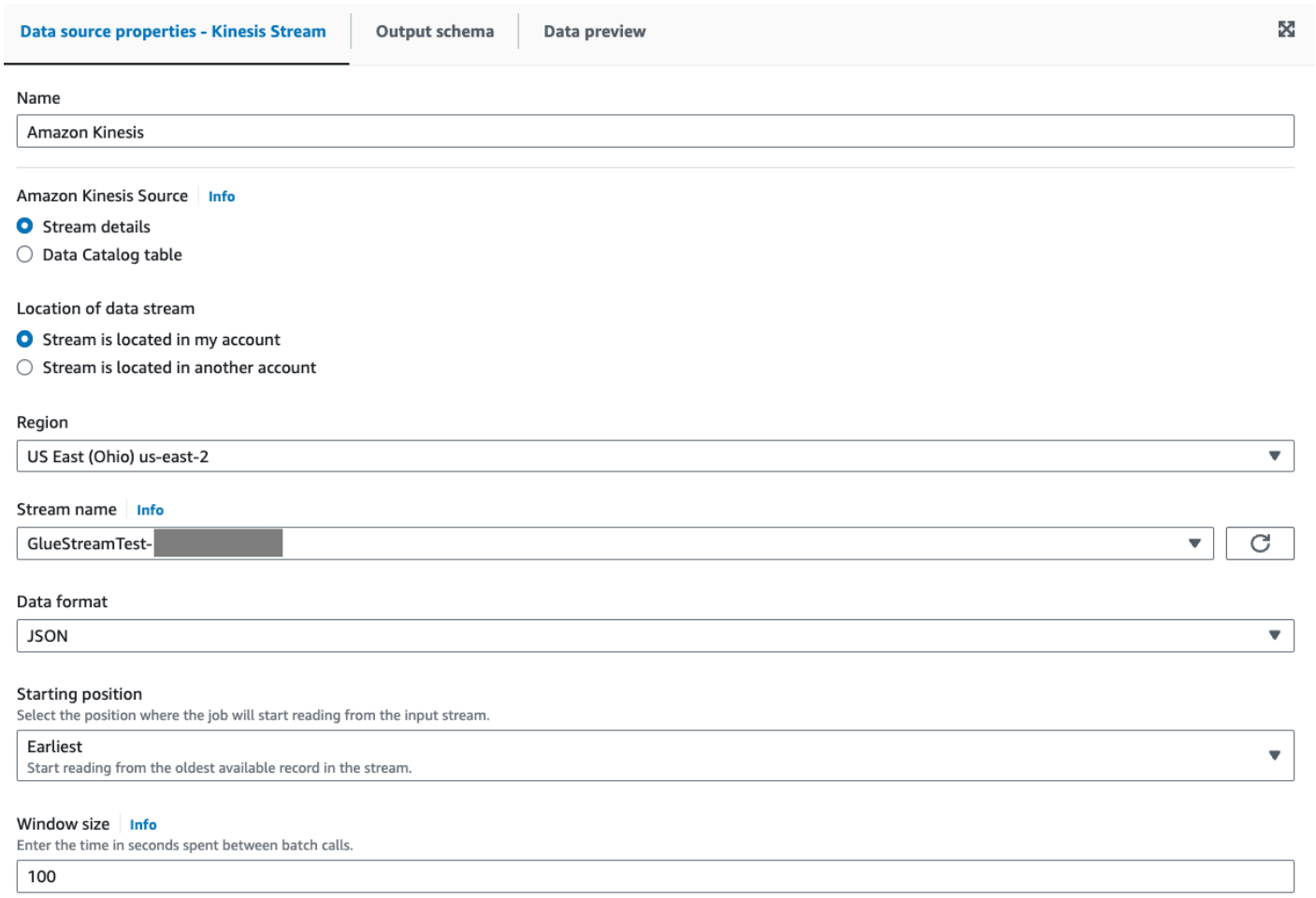
10.在数据来源属性 - Kinesis Stream 选项卡下，为 Amazon Kinesis Source 选择流详细信息。

11.对于数据流位置，选择流位于我的账户中。

12.选择您使用的区域。

13.选择 `GlueStreamTest-{AWS::AccountId}` 流。

14.将所有其他设置保留为默认。

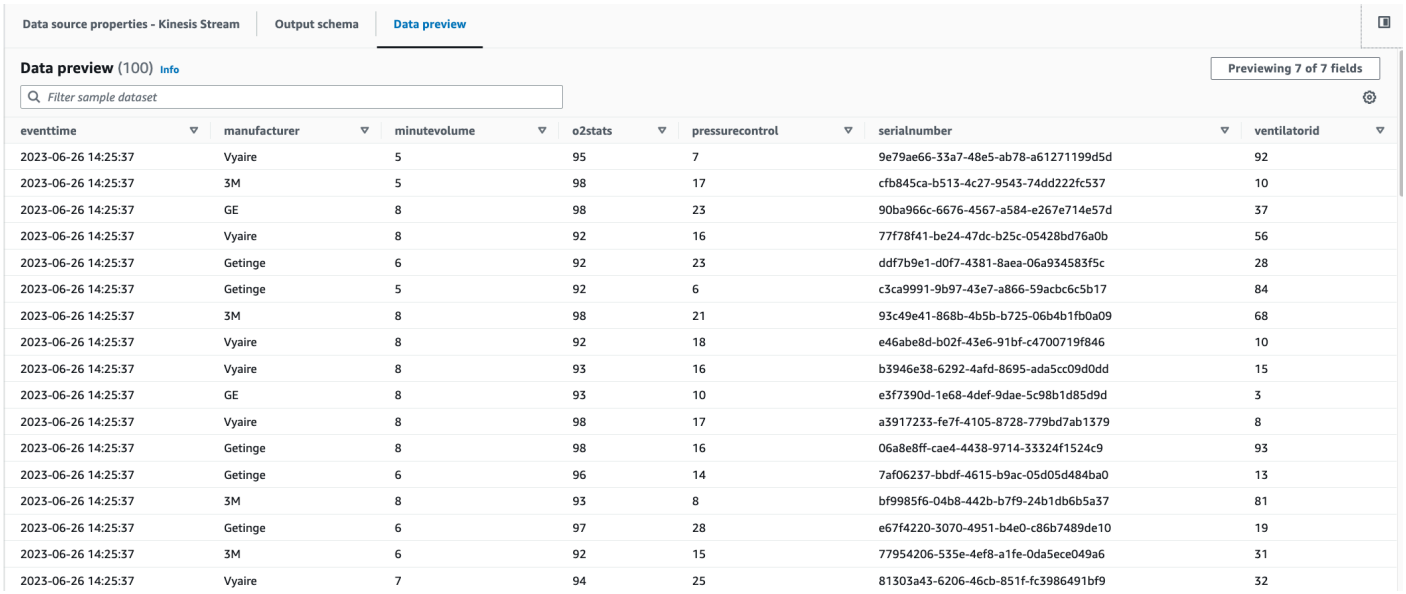


15.导航到数据预览选项卡。

16.单击启动数据预览会话，预览 KDG 生成的模拟数据。选择您之前为 AWS Glue 流式处理作业创建的 Glue 服务角色。

预览数据需要 30-60 秒才能显示。如果显示没有可显示的数据，请单击齿轮图标并将要采样的行数更改为 100。

您可以看到如下示例数据：



Data source properties - Kinesis Stream | Output schema | Data preview

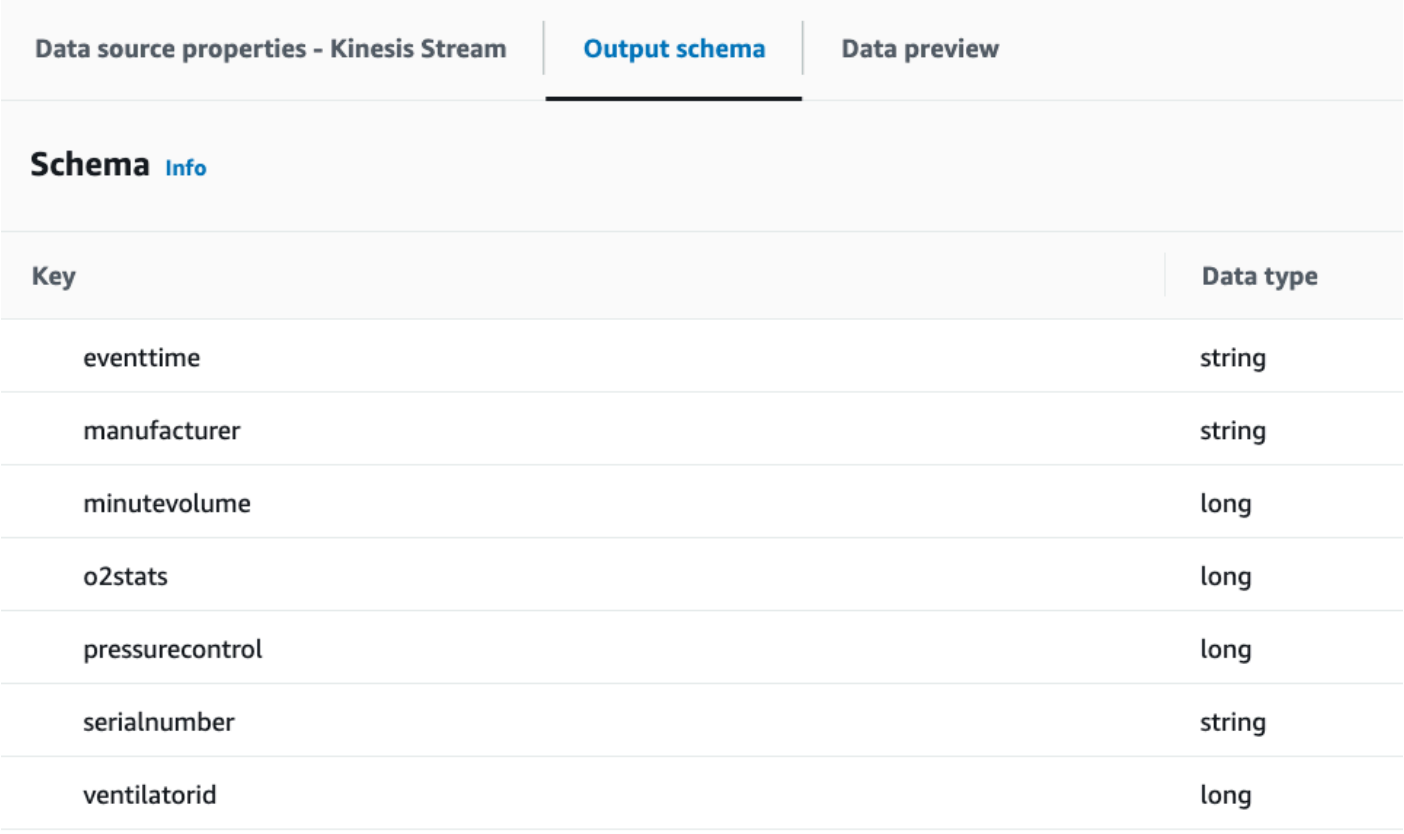
Data preview (100) Info

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

您还可以在输出架构选项卡中查看推断的架构。



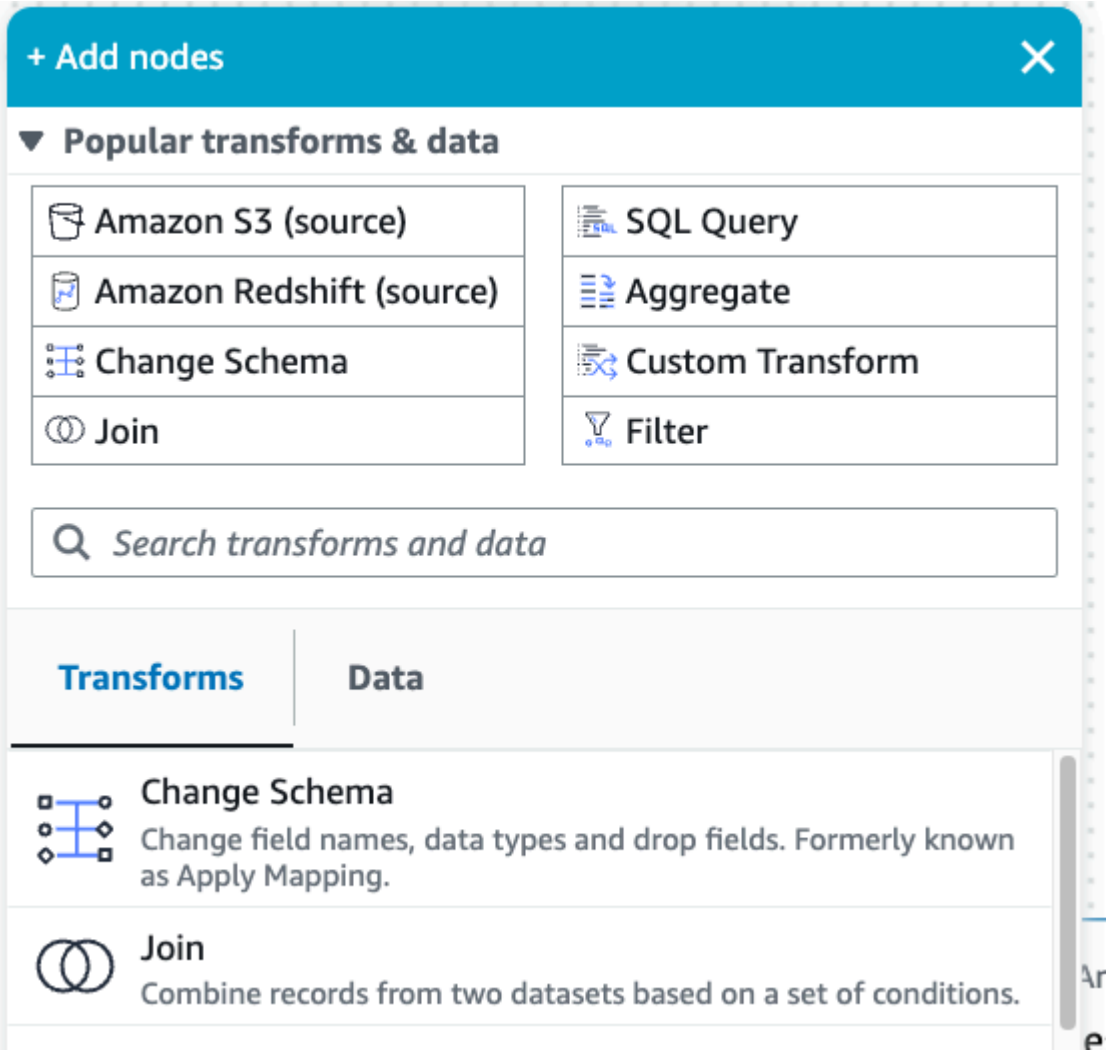
Data source properties - Kinesis Stream | Output schema | Data preview

Schema Info

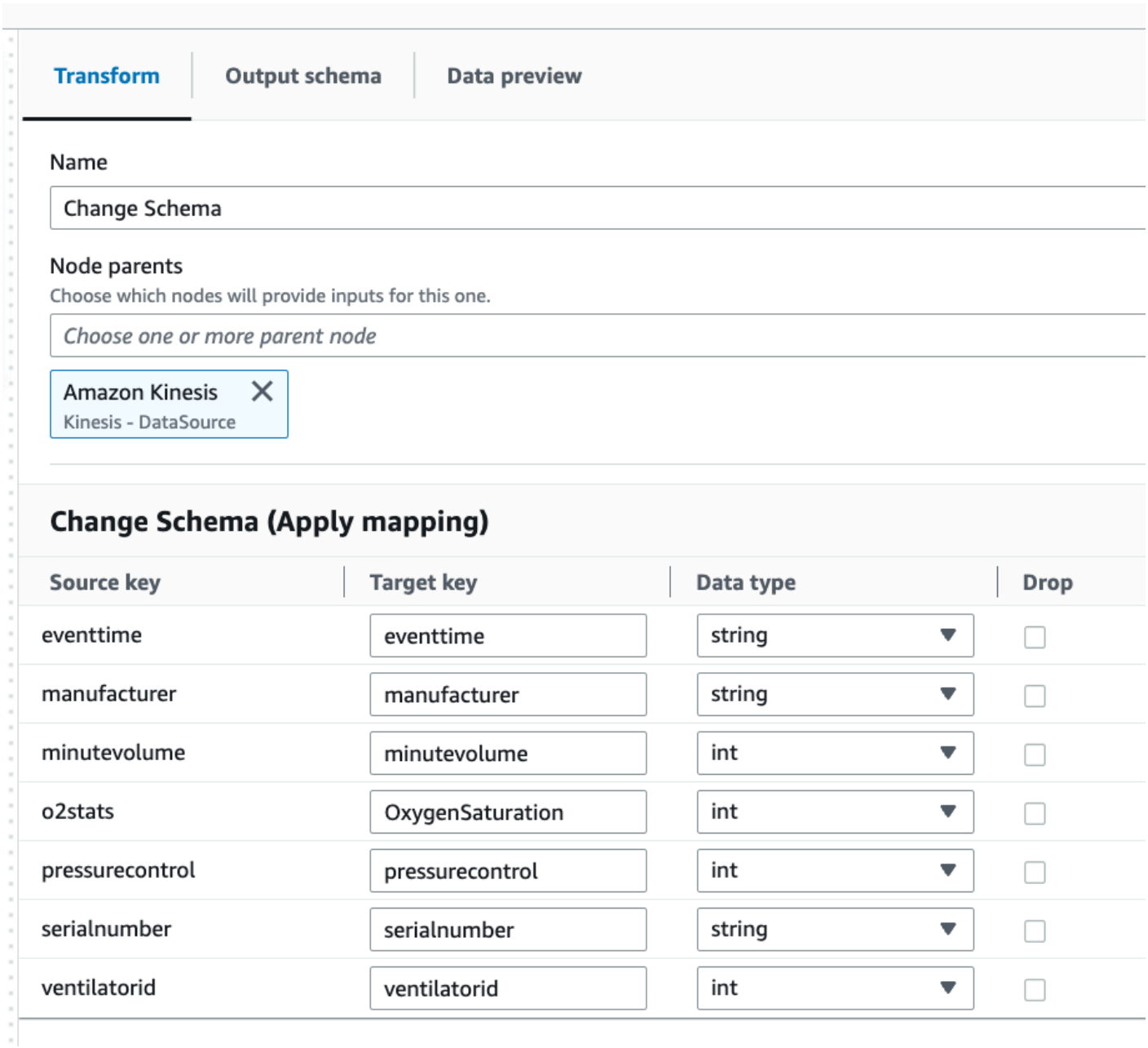
Key	Data type
eventtime	string
manufacturer	string
minutevolume	long
o2stats	long
pressurecontrol	long
serialnumber	string
ventilatorid	long

执行转换并将转换后的结果存储在 Amazon S3 中

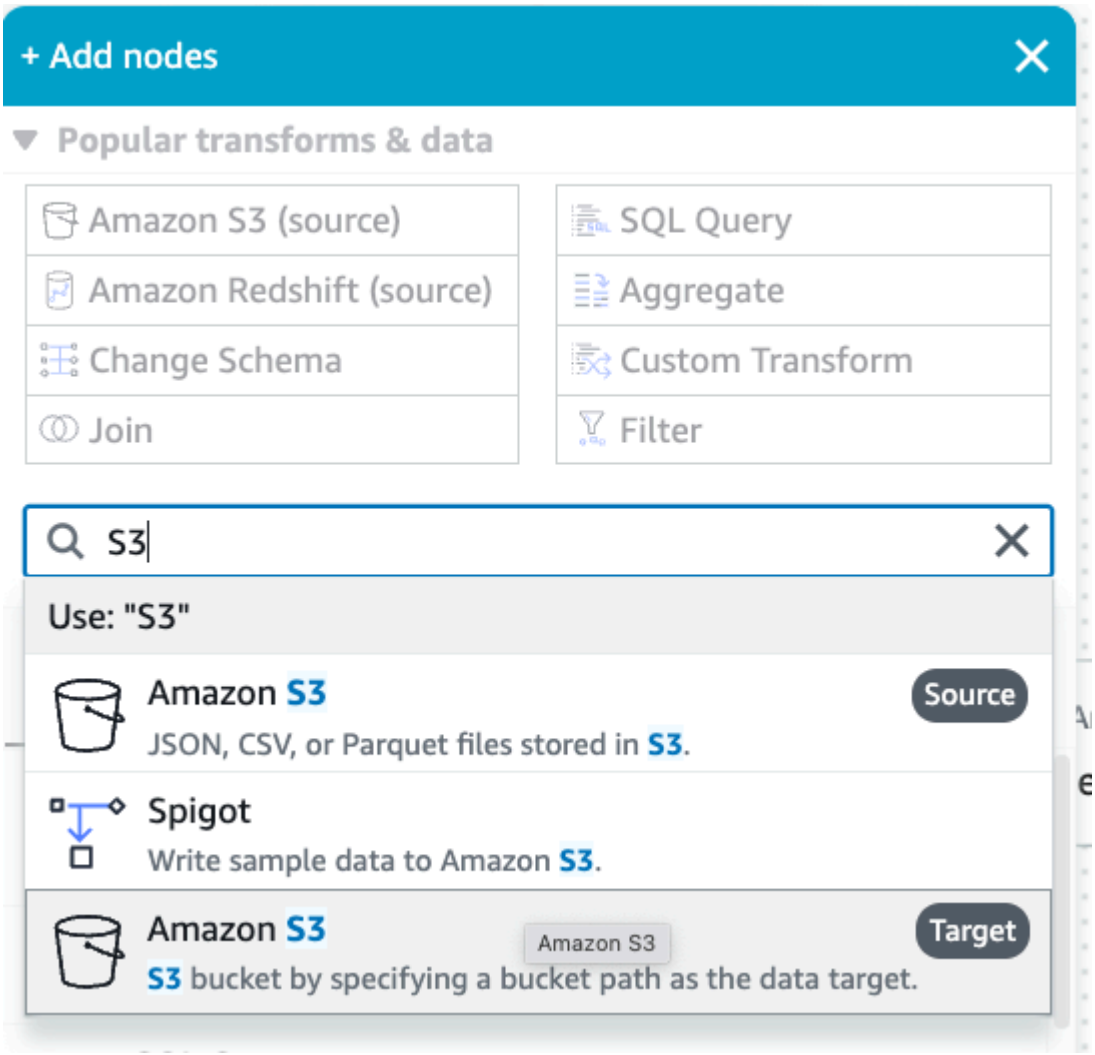
1. 选中源节点后，单击左上角的加号图标添加一个转换步骤。
2. 选择更改架构步骤。



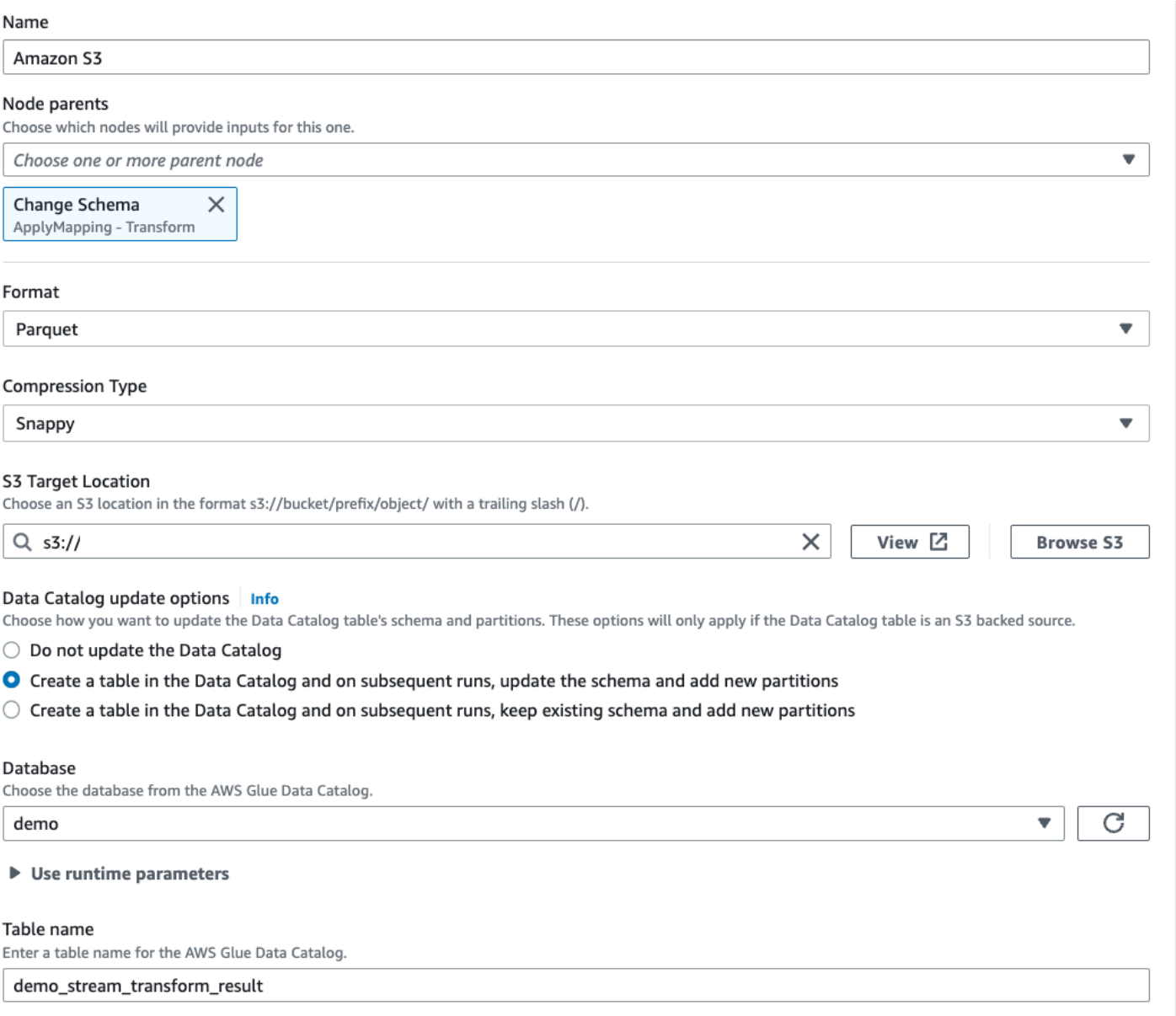
3. 在此步骤中，您可以重命名字段并转换字段的数据类型。将 `o2stats` 列重命名为 `OxygenSaturation`，并将所有 `long` 数据类型转换为 `int`。



4. 单击加号图标添加一个 Amazon S3 目标。在搜索框中输入 S3，然后选择 Amazon S3 – 目标转换步骤。



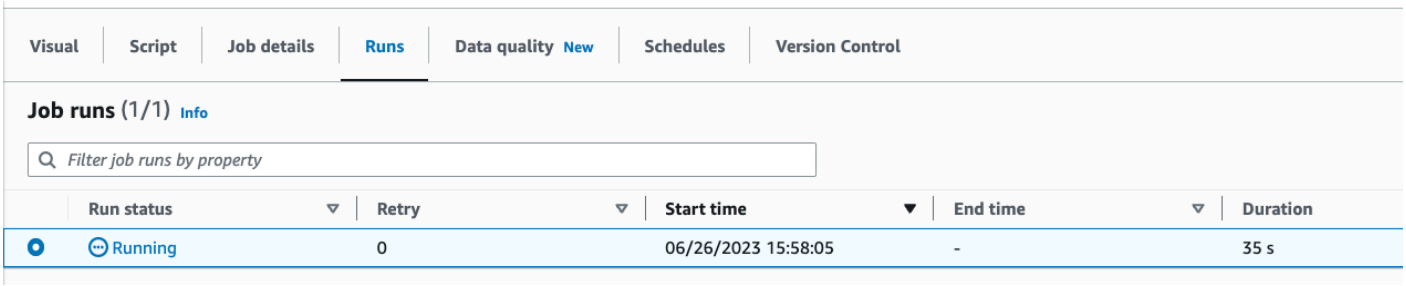
5. 选择 Parquet 作为目标文件格式。
6. 选择 Snappy 作为压缩类型。
7. 输入 CloudFormation 模板 `streaming-tutorial-s3-target-{AWS::AccountId}` 创建的 S3 目标位置。
8. 选择在数据目录中创建表，在后续运行中更新架构并添加新分区。
9. 输入目标数据库和表名称以存储 Amazon S3 目标表的架构。



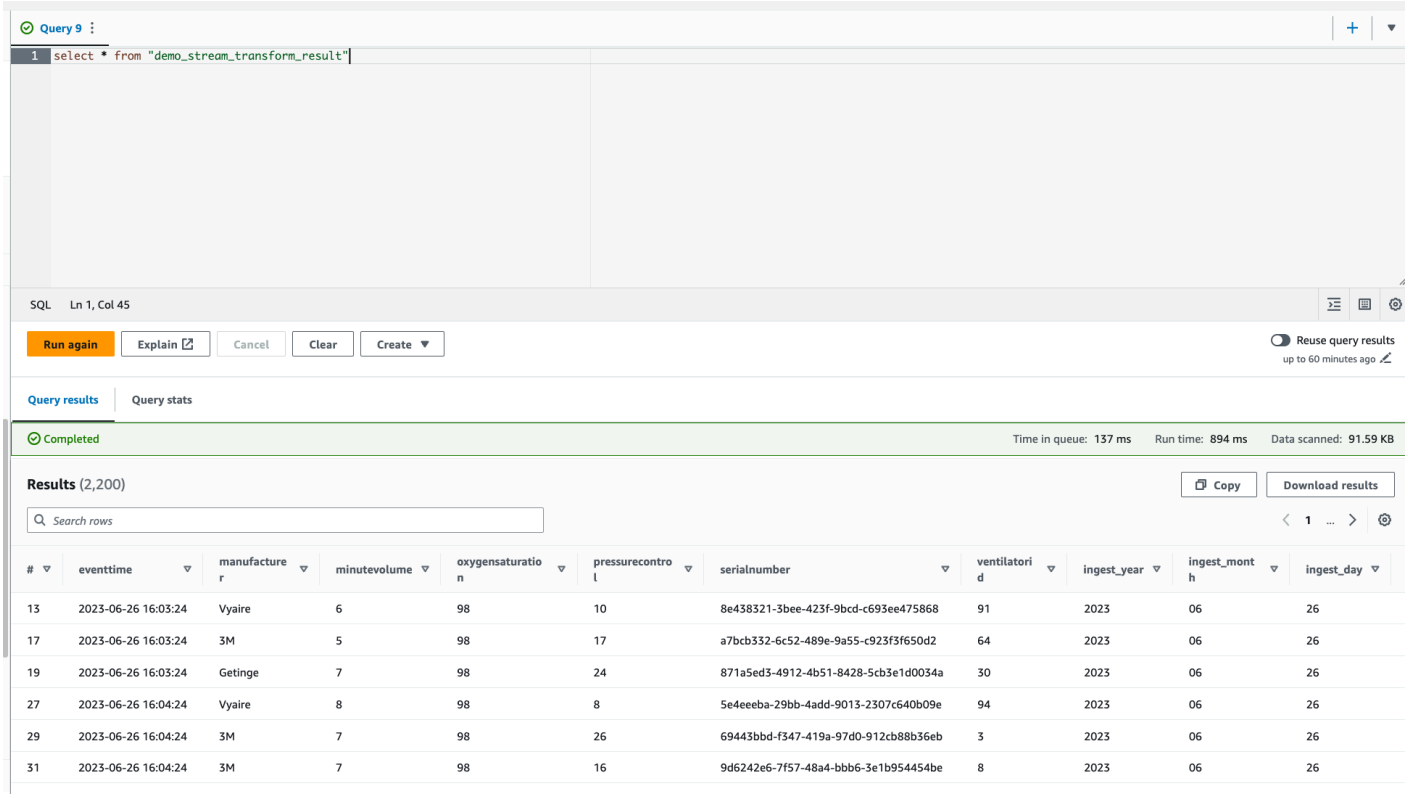
10.单击脚本选项卡查看生成的代码。

11.单击右上角的保存以保存 ETL 代码，然后单击运行以启动 AWS Glue 流式处理作业。

您可以在运行选项卡中找到运行状态。让作业运行 3-5 分钟，然后停止作业。



12验证在 Amazon Athena 中创建的新表。



教程：使用 AWS Glue Studio 笔记本构建第一个流工作负载

在本教程中，您将探索如何利用 AWS Glue Studio 笔记本以交互方式构建和优化 ETL 作业，以实现近乎实时的数据处理。无论您是刚接触 AWS Glue，还是想提高自己的技能，本指南都将引导您完成整个过程，让您能够充分利用 AWS Glue 交互式会议笔记本的潜力。

有了 AWS Glue 流式处理，您可以创建连续运行的流提取、转换、加载（ETL）作业，该作业使用来自 Amazon Kinesis Data Streams、Apache Kafka 和 Amazon Managed Streaming for Apache Kafka（Amazon MSK）等流式处理数据源的数据。

先决条件

要学习本教程，您需要一个具有 AWS 控制台权限的用户以使用 AWS Glue、Amazon Kinesis、Amazon S3、Amazon Athena、AWS CloudFormation、AWS Lambda 和 Amazon Cognito。

使用来自 Amazon Kinesis 的流式处理数据

主题

- 使用 Kinesis Data Generator 生成模拟数据
- 使用 AWS Glue Studio 创建 AWS Glue 流式处理作业
- 清理
- 结论

使用 Kinesis Data Generator 生成模拟数据

> **Note**
>
> 如果您已经完成了之前的 教程：使用 AWS Glue Studio 构建第一个流式处理工作负载，您的账户中已经安装了 Kinesis Data Generator，则可以跳过下面的步骤 1-8，继续阅读第 使用 AWS Glue Studio 创建 AWS Glue 流式处理作业 节。

您可以使用 Kinesis Data Generator（KDG）生成 JSON 格式的合成示例数据。您可以在工具文档中找到完整的说明和详细信息。

1. 首先，单击

   以在您的 AWS 环境中运行 AWS CloudFormation 模板。

   > **Note**
   >
   > 您可能会遇到 CloudFormation 模板故障，因为您的 AWS 账户中已存在某些资源，例如 Kinesis Data Generator 的 Amazon Cognito 用户。这可能是因为您已经从其他教程或博客中进行了设置。要解决此问题，您可以在一个新的 AWS 账户中重新尝试此模板，也可以尝试使用其他 AWS 区域。通过这些选项，您可以在不与现有资源冲突的情况下运行此教程。

   该模板已经预置了一个 Kinesis 数据流和一个 Kinesis Data Generator 账户。
2. 输入用户名和密码，以便 KDG 进行身份验证。记下用户名和密码以备将来使用。
3. 选择下一步，一直到最后一步。确认 IAM 资源的创建。检查屏幕上方是否有任何错误，比如密码不符合最低要求，然后部署模板。
4. 导航到堆栈的输出选项卡。模板部署完成后，将显示生成的属性 KinesisDataGeneratorUrl。单击该 URL。

5. 输入您记下的用户名和密码。

6. 选择您正在使用的区域，然后选择 Kinesis Stream `GlueStreamTest-{AWS::AccountId}`

7. 输入以下模板：

```
{
    "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"]
    )}}"
}
```

现在，您可以使用测试模板查看模拟数据，并使用发送数据将模拟数据摄取到 Kinesis。

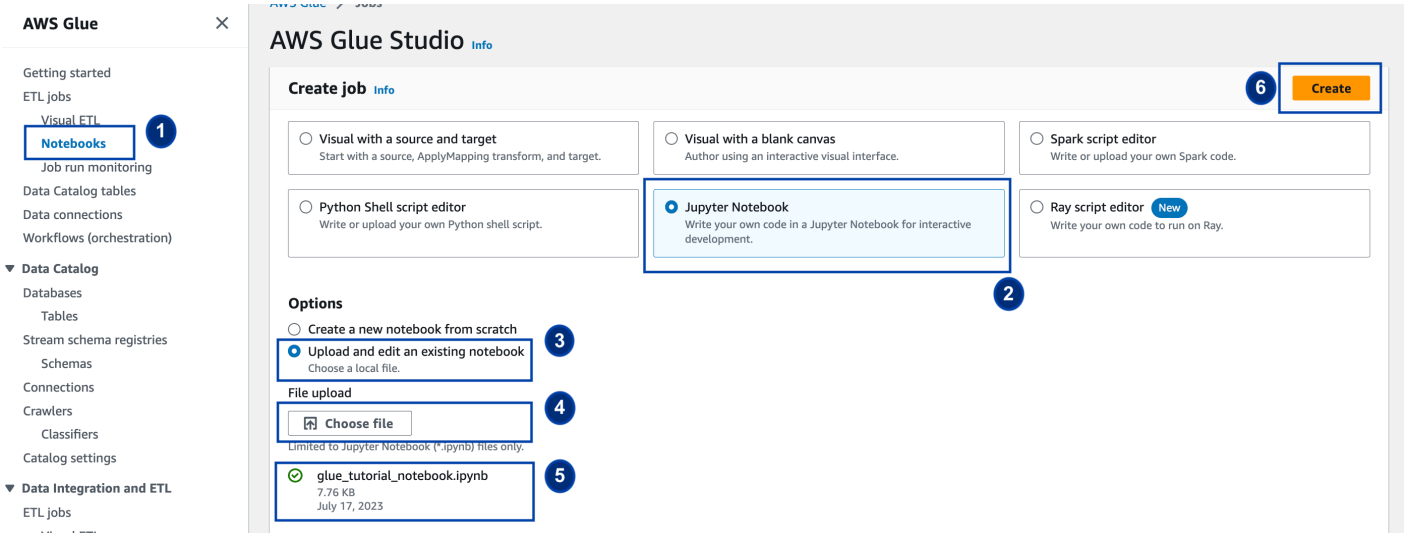
8. 单击发送数据，并向 Kinesis 生成 0.5-1 万条记录。

使用 AWS Glue Studio 创建 AWS Glue 流式处理作业

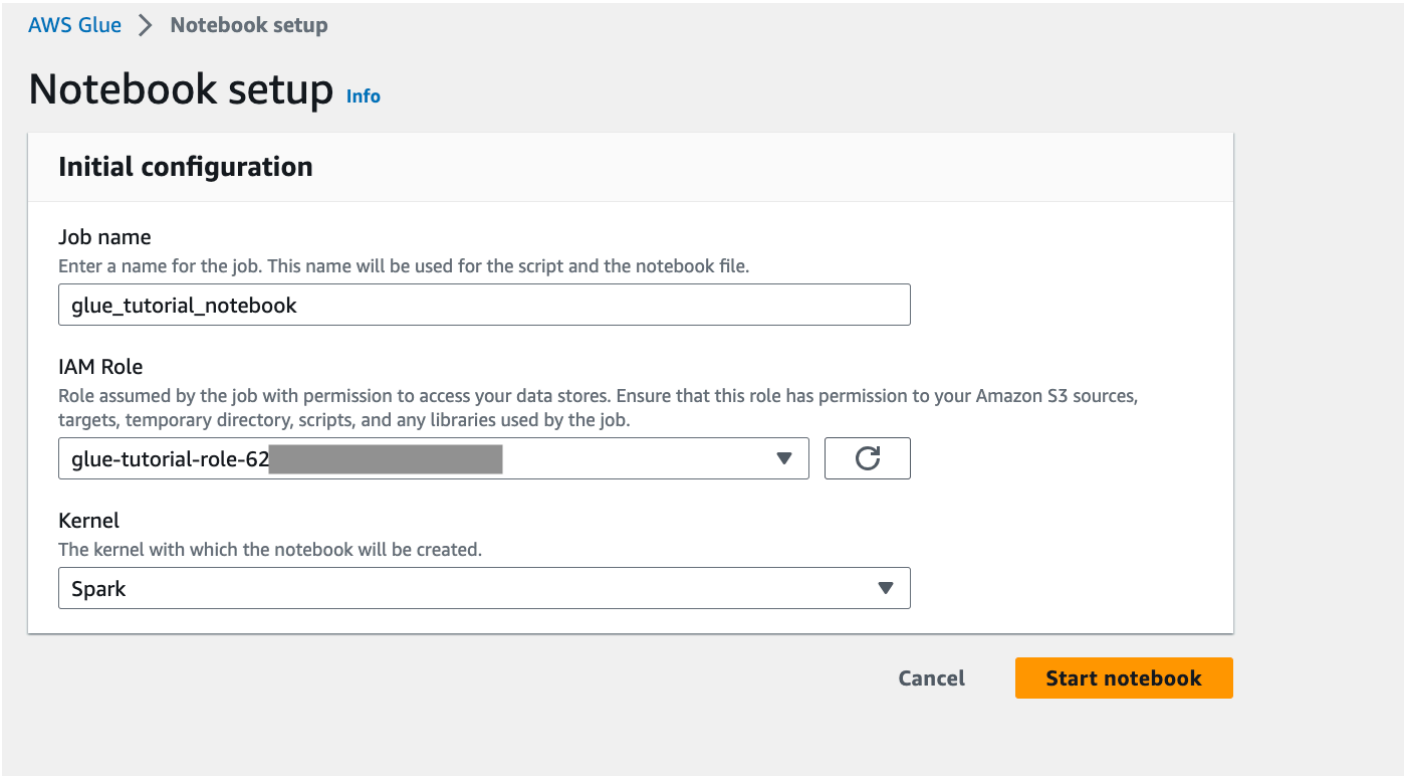
AWS Glue Studio 是一个可视化界面，可简化设计、编排和监控数据集成管道的过程。它使用户无需编写大量代码即可构建数据转换管道。除了可视化作业创作体验外，AWS Glue Studio 还提供了一个由 AWS Glue 交互式会话支持的 Jupyter notebook，您将在本教程的剩余部分中使用它。

设置 AWS Glue 流式处理交互式会话作业

1. 下载提供的笔记本文件并保存到本地目录
2. 打开 AWS Glue 控制台，在左窗格中单击笔记本 > Jupyter Notebook > 上传并编辑现有笔记本。上传上一步中的笔记本，然后单击创建。



3. 为作业提供名称和角色并选择默认的 Spark 内核。然后单击启动笔记本。对于 IAM 角色，选择 CloudFormation 模板预置的角色。您可以在 CloudFormation 的输出选项卡中看到它。

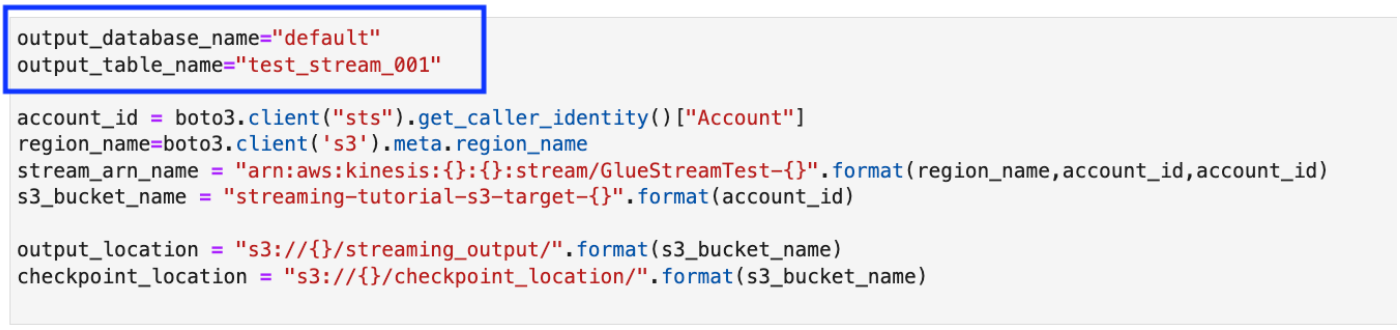


笔记本上有继续本教程的所有必要说明。您可以在笔记本上运行说明，也可以按照本教程继续进行作业开发。

运行笔记本单元格

1. （可选）第一个代码单元格 %help 列出了所有可用的笔记本魔术命令。您可以暂时跳过这个单元格，但可以随意探索它。
2. 从下一个代码块 %streaming 开始。这个魔法命令将作业类型设置为流式处理，以便您可以开发、调试和部署 AWS Glue 流式处理 ETL 作业。
3. 运行下一个单元格以创建 AWS Glue 交互式会话。输出单元格中有一条消息，用于确认会话创建。

4. 下一个单元格定义了变量。将值替换为适合您作业的值，然后运行单元格。例如：



5. 由于数据已流送到 Kinesis Data Streams，您的下一个单元将使用来自该流的结果。运行下一个单元格。由于没有打印语句，该单元格没有预期的输出。

6. 在下面的单元格中，您可以通过采集样本集探索传入流，并打印其架构和实际数据。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. 接下来，定义实际的数据转换逻辑。该单元由在每个微批次期间触发的 `processBatch` 方法组成。运行单元格。总的来说，我们对传入流执行以下操作：

   a. 选择输入列的子集。

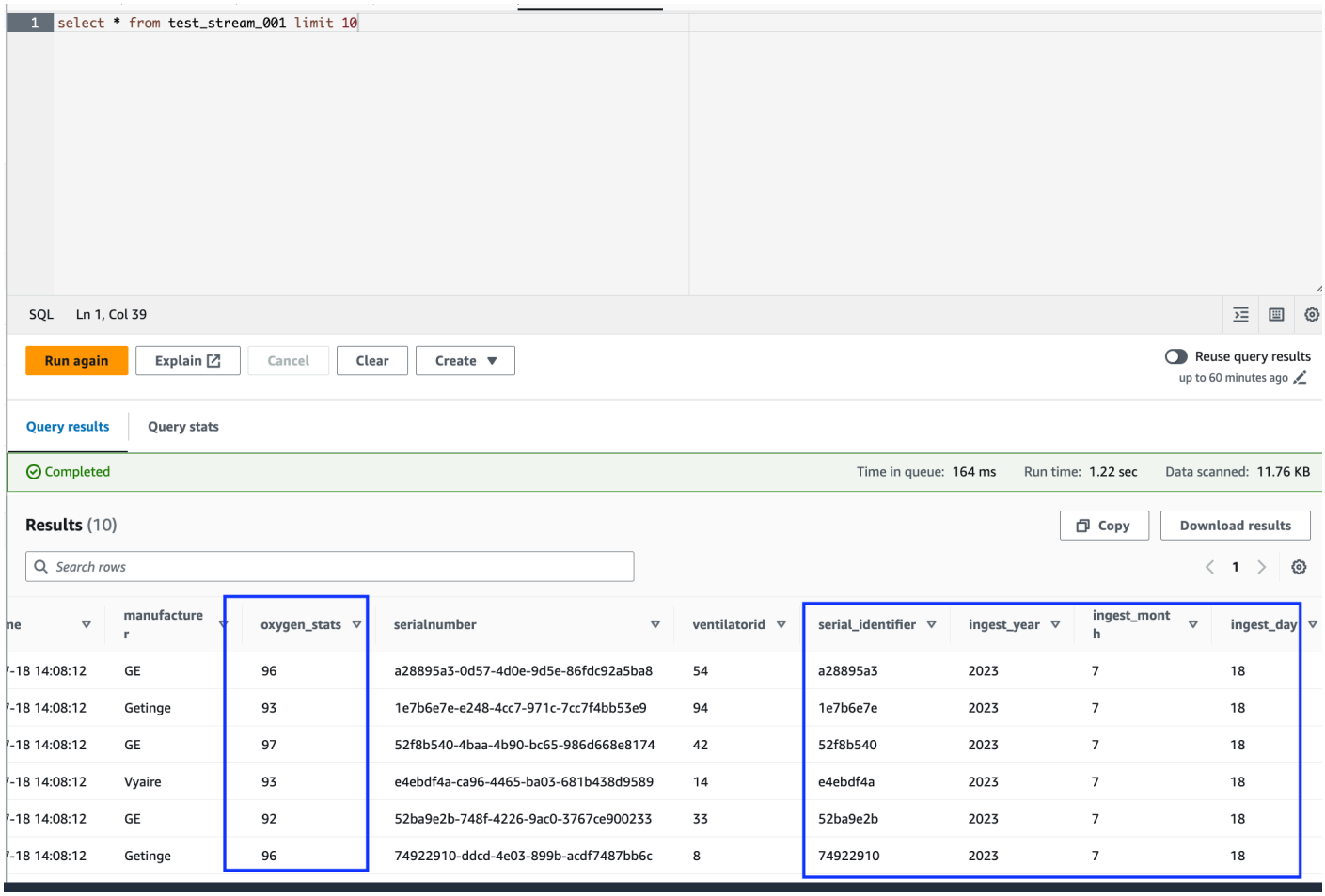
   b. 重命名列（将 o2stats 重命名为 oxygen_stats）。

   c. 衍生新列（serial_identifier、ingest_year、ingest_month 和 ingest_day）。

   d. 将结果存储到 Amazon S3 存储桶中，并创建一个分区的 AWS Glue 目录表

8. 在最后一个单元格中，每 10 秒触发一次批处理。运行单元格，等待约 30 秒，让它填充 Amazon S3 存储桶和 AWS Glue 目录表。

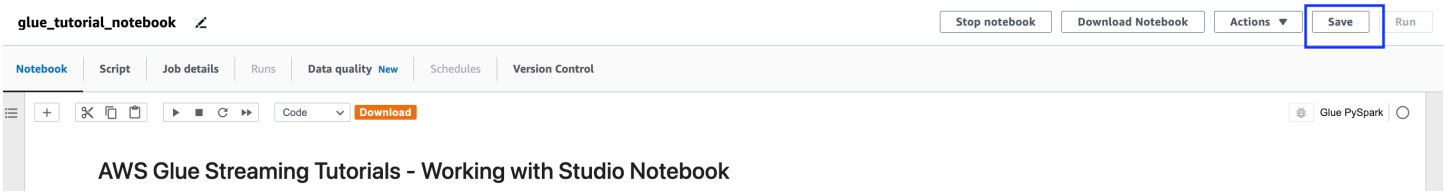
9. 最后，使用 Amazon Athena 查询编辑器浏览存储的数据。您可以看到重命名的列和新分区。



笔记本上有继续本教程的所有必要说明。您可以在笔记本上运行说明，也可以按照本教程继续进行作业开发。

保存并运行 AWS Glue 作业

使用交互式会话笔记本完成应用程序的开发和测试后，单击笔记本界面顶部的保存。保存后，您还可以将应用程序作为作业运行。



清理

为了避免您的账户产生额外费用，请停止按照说明启动的流式处理作业。您可以停止笔记本完成此操作，这将结束会话。清空 Amazon S3 存储桶并删除您之前预置的 AWS CloudFormation 堆栈。

结论

在本教程中，我们演示了如何使用 AWS Glue Studio 笔记本执行以下操作

- 使用笔记本创作流式处理 ETL 作业
- 预览传入数据流
- 无需发布 AWS Glue 作业即可编写代码和修复问题
- 查看端到端工作代码，删除所有调试，并从笔记本中打印语句或单元格
- 将代码发布为 AWS Glue 作业

本教程旨在让您亲身体验如何使用 AWS Glue 流式处理和交互式会话。我们建议您在您个人的 AWS Glue 流式处理用例中参考本教程中的内容。有关更多信息，请参阅 开始使用 AWS Glue 交互式会话。

AWS Glue 流式处理概念

以下部分介绍了 AWS Glue 流式处理概念。

主题

AWS Glue 流式处理作业剖析

AWS Glue 流式处理作业以 Spark 流式处理范式运行，并利用 Spark 框架的结构化流。流式处理作业以特定的时间间隔不断轮询流式处理数据源，以微批次的形式获取记录。以下各节探讨了 AWS Glue 流式处理作业的不同部分。

forEachBatch

`forEachBatch` 方法是 AWS Glue 流式处理作业运行的入口点。AWS Glue 流式处理作业使用 `forEachBatch` 方法轮询数据，其功能类似于迭代器，在流式处理作业的生命周期中保持活动状态，定期轮询流式处理数据源以获取新数据，并以微批次处理最新数据。

```
glueContext.forEachBatch(
    frame=dataFrame_AmazonKinesis_node1696872487972,
    batch_function=processBatch,
    options={
        "windowSize": "100 seconds",
        "checkpointLocation": args["TempDir"] + "/" + args["JOB_NAME"] + "/
checkpoint/",
    },
)
```

配置 forEachBatch 的 frame 属性以指定流式处理数据源。在此示例中，在创建作业期间在空白画布中创建的源节点将填充作业的默认 DataFrame。将 batch_function 属性设置为您决定为每个微批次操作调用的 function。必须定义一个函数来处理传入数据的批量转换。

来源

在 processBatch 函数的第一步中，程序会验证您定义为 forEachBatch 帧属性的 DataFrame 记录数。该程序会在非空 DataFrame 上附加摄取时间戳。data_frame.count()>0 子句决定了最新的微批次是否为空，是否可以进一步处理。

```
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",
        )
```

Mapping

该程序的下一部分是应用映射。通过 spark DataFrame 的 Mapping.apply 方法，围绕数据元素定义转换规则。通常，您可以对源数据列进行重命名、更改数据类型或应用自定义函数，然后将其映射到目标列。

```
    #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",
    )
        )
```

sink

在这一部分，来自流式处理数据源的传入数据集存储在目标位置。在本示例中，我们将数据写入 Amazon S3 位置。AmazonS3_node_path 属性详细信息是根据您在画布上创建作业期间使用的设置预先填充的。您可以根据自己的用例设置 updateBehavior，然后决定不更新数据目录表，或者创建数据目录并在后续运行中更新数据目录架构，或者创建目录表但在后续运行时不更新架构定义。

partitionKeys 属性定义了存储分区选项。默认行为是根据源部分中提供的 ingestion_time_columns 对数据进行分区。compression 属性允许您设置在目标写入期间应用的压缩算法。您可以选择将 Snappy、LZO 或 GZIP 设置为压缩技术。enableUpdateCatalog 属性控制是否需要更新 AWS Glue 目录表。此属性的可用选项为 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 目录接收器

作业的这一部分控制 AWS Glue 目录表更新行为。根据您的 AWS Glue 目录数据库名称和与您设计的 AWS Glue 作业关联的表名称设置 catalogDatabase 和 catalogTableName 属性。您可以通过 setFormat 属性定义目标数据的文件格式。在本示例中，我们将以 parquet 格式存储数据。

参考本教程设置并运行 AWS Glue 流式处理作业后，在 Amazon Kinesis Data Streams 生成的流式处理数据将以 parquet 格式存储在 Amazon S3 位置，并快速压缩。成功运行流式处理作业后，您将能够通过 Amazon Athena 查询数据。

```
    AmazonS3_node1696872743449 = setCatalogInfo(
        catalogDatabase =  "demo", catalogTableName = "demo_stream_transform_result"
    )
    AmazonS3_node1696872743449.setFormat("glueparquet")
    AmazonS3_node1696872743449.writeFormat("ChangeSchema_node16986872679326")
    )
```

Kafka 连接

指定与 Kafka 集群或 Amazon Managed Streaming for Apache Kafka 集群的连接。

您可以使用存储在 Data Catalog 表中的信息或通过提供信息直接访问数据流来从 Kafka 数据流读取或向其中写入数据。你可以从 Kafka 读取信息到 Spark 中 DataFrame，然后将其转换为 AWS Glue。DynamicFrame你可以用 JSON 格式写入 DynamicFrames Kafka。如果直接访问数据流，请使用这些选项提供有关如何访问数据流的信息。

如果您通过 `getCatalogSource` 或 `create_data_frame_from_catalog` 来使用来自 Kafka 流式处理源的记录，或通过 `getCatalogSink` 或 `write_dynamic_frame_from_catalog` 将记录写入 Kafka，则作业具有 Data Catalog 数据库和表名称信息，并且可以使用该信息来获取一些基本参数，以便从 Kafka 流式处理源读取数据。如果使用 `getSource`、`getCatalogSink`、`getSourceWithFormat`、`getSinkWithFormat`、`createDataFrame` 或 `write_dynamic_frame_from_catalog`，则必须使用此处描述的连接选项指定这些基本参数。

您可以使用 `GlueContext` 类中指定方法的以下参数为 Kafka 指定连接选项。

- Scala
  - `connectionOptions`：与 `getSource`、`createDataFrameFromOptions`、`getSink` 结合使用
  - `additionalOptions`：与 `getCatalogSource`、`getCatalogSink` 结合使用
  - `options`：与 `getSourceWithFormat`、`getSinkWithFormat` 结合使用
- Python
  - `connection_options`：与 `create_data_frame_from_options`、`write_dynamic_frame_from_options` 结合使用
  - `additional_options`：与 `create_data_frame_from_catalog`、`write_dynamic_frame_from_catalog` 结合使用

- `options`：与 `getSource`、`getSink` 结合使用

有关流式处理 ETL 作业的注意事项和限制，请参阅 the section called “串流 ETL 注释和限制”。

配置 Kafka

连接可通过互联网访问的 Kafka 直播没有任何 AWS 先决条件。

您可以创建 AWS Glue Kafka 连接来管理您的连接凭证。有关更多信息，请参阅 the section called “为 Kafka 数据流创建连接”。在你的 AWS Glue 作业配置中，提供 c *onnectionName* 作为附加网络连接，然后在方法调用中为参数提供 c *onnectionName*。connectionName

在某些情况下，您需要配置其他先决条件：

- 如果使用 Amazon Managed Streaming for Apache Kafka 搭配 IAM 身份验证，则需要适当的 IAM 配置。
- 如果在 Amazon VPC 内使用 Amazon Managed Streaming for Apache Kafka，则需要适当的 Amazon VPC 配置。您需要创建一个提供 Amazon VPC 连接信息的 AWS Glue 连接。您需要在作业配置中将 Glue AWS 连接作为附加网络连接包括在内。

有关流式处理 ETL 作业先决条件的更多信息，请参阅 the section called “流式处理 ETL 作业”。

示例：从 Kafka 流读取

与 the section called “forEachBatch” 结合使用。

Kafka 流式处理源示例：

```
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)
```

示例：写入 Kafka 流

写入 Kafka 的示例：

getSink 方法示例：

```
data_frame_datasource0 =
glueContext.getSink(
 connectionType="kafka",
 connectionOptions={
  JsonOptions("""{
   "connectionName": "ConfluentKafka",
   "classification": "json",
   "topic": "kafka-auth-topic",
   "typeOfData": "kafka"}
  """)},
 transformationContext="dataframe_ApacheKafka_node1711729173428")
 .getDataFrame()
```

write_dynamic_frame.from_options 方法示例：

```
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 连接选项参考

在读取时，可以使用以下连接选项和 "connectionType": "kafka"：

- "bootstrap.servers"（必需）引导服务器 URL 的列表，例如，作为 b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094。此选项必须在 API 调用中指定，或在数据目录的表元数据中定义。
- "security.protocol"（必填）用于与代理通信的协议。可能的值为 "SSL" 或 "PLAINTEXT"。
- "topicName"（必填）要订阅的以逗号分隔的主题列表。您必须指定"topicName"、"assign" 或 "subscribePattern" 中的其中一个，且只能指定一个。
- "assign"：（必填）用于指定要使用的 TopicPartitions 的 JSON 字符串。您必须指定"topicName"、"assign" 或 "subscribePattern" 中的其中一个，且只能指定一个。

例如：“{"topicA":[0,1],"topicB":[2,4]}”

- "subscribePattern"：（必需）标识要订阅的主题列表的 Java 正则表达式字符串。您必须指定"topicName"、"assign" 或 "subscribePattern" 中的其中一个，且只能指定一个。

  示例：“topic.*”

- "classification"（必需）记录中数据使用的文件格式。除非 Data Catalog 提供，否则为必需。
- "delimiter"（可选）当 classification 为 CSV 时使用的值分隔符。默认为“,”。
- "startingOffsets"：（可选）Kafka 主题中数据读取的起始位置。可能的值为 "earliest" 或 "latest"。默认值为 "latest"。
- "startingTimestamp": (可选，仅支持 AWS Glue 4.0 或更高版本）Kafka 主题中要从中读取数据的记录的时间戳。可能的值是以模式 yyyy-mm-ddTHH:MM:SSZ 采用 UTC 格式的时间戳字符串（其中 Z 表示带有 +/-的 UTC 时区偏移量。例如：“2023-04-04T08:00:00-04:00”）。

  注意：GI AWS ue 直播脚本的 “连接选项” 列表中只能出现 “startingOffsets” 或 “startingTimeStamp” 中的一个，包括这两个属性都会导致任务失败。

- "endingOffsets"：（可选）批处理查询结束时的终点。可能值为 "latest"，或者为每个 TopicPartition 指定结束偏移的 JSON 字符串。

  对于 JSON 字符串，格式为 {"topicA":{"0":23,"1":-1},"topicB":{"0":-1}}。偏移值 -1 表示 "latest"。

- "pollTimeoutMs"：（可选）Spark 任务执行程序中，从 Kafka 轮询数据的超时时间（以毫秒为单位）。默认值为 512。
- "numRetries"：（可选）无法获取 Kafka 偏移时的重试次数。默认值为 3。
- "retryIntervalMs"：（可选）重试获取 Kafka 偏移时的等待时间（以毫秒为单位）。默认值为 10。
- "maxOffsetsPerTrigger"：（可选）每个触发间隔处理的最大偏移数的速率限制。指定的总偏移数跨不同卷的 topicPartitions 按比例分割。默认值为 null，这意味着使用者读取所有偏移，直到已知的最新偏移。
- "minPartitions"：（可选）从 Kafka 读取数据的必需最小分区数。默认值为 null，这意味着 Spark 分区数等于 Kafka 分区数。
- "includeHeaders"：（可选）是否包含 Kafka 标头。当选项设置为“true”时，数据输出将包含一个名为“glue_streaming_kafka_headers”的附加列，类型为 Array[Struct(key: String, value: String)]。默认值为“false”。此选项仅适用于 AWS Glue 版本 3.0 或更高版本。

- "schema"：（当 inferSchema 设为 false 时为必填）用于处理有效工作负载的架构。如果分类为 avro，则提供的架构必须采用 Avro 架构格式。如果分类不是 avro，则提供的架构必须采用 DDL 架构格式。

  以下是一些架构示例。

  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"：（可选）默认值为“false”。如果设置为“true”，则会在运行时检测到 foreachbatch 内的有效工作负载中的架构。
- "avroSchema"：（已弃用）用于指定 Avro 数据架构（使用 Avro 格式时）的参数。此参数现已被弃用。使用 schema 参数。

- "addRecordTimestamp"：（可选）当选项设置为 'true' 时，数据输出将包含一个名为 "__src_timestamp" 的附加列，表示主题收到相应记录的时间。默认值为‘false’。4.0 或更高 AWS Glue 版本支持此选项。
- "emitConsumerLagMetrics": (可选）当该选项设置为 “true” 时，对于每个批次，它将发布从主题收到的最旧记录到该记录到达的时间之间的持续时间内的AWS Glue指标。 CloudWatch该指标的名字是 “glue.driver.streaming”。 maxConsumerLagInMs”。默认值为‘false’。4.0 或更高 AWS Glue 版本支持此选项。

在写入时，可以使用以下连接选项和 "connectionType": "kafka"：

- "connectionName"（必填）用于连接到 Kafka 集群的 AWS Glue 连接的名称（类似于 Kafka 源代码）。
- "topic"（必填项）如果存在主题列，则在将给定行写入 Kafka 时，除非设置了主题配置选项，否则其值将用作主题。也就是说，topic 配置选项会覆盖主题列。
- "partition"（可选）如果指定了有效的分区号，则 partition 将在发送记录时使用该分区号。

  如果未指定分区但存在 key，则将使用该键的哈希值来选择分区。

  如果 key 和 partition 都不存在，则将根据在向分区至少生成了 batch.size 字节的数据时，对这些更改进行粘性分区来选择分区。
- "key"（可选）如果 partition 为空，则用于分区。
- "classification"（可选）记录中数据使用的文件格式。我们只支持 JSON、CSV 和 Avro。

  对于 Avro 格式，我们可以提供自定义 avroSchema 来进行序列化，但请注意，还需要在源中提供该格式以进行反序列化。否则，默认情况下它使用 Apache AvroSchema 进行序列化。

此外，您可以在需要时通过更新 Kafka 生产者配置参数来微调 Kafka 接收器。请注意，不存在有关连接选项的允许列表，所有键值对都按原样保存在接收器上。

虽然存在一个很小的选项拒绝列表，不过不会生效。有关更多信息，请参阅 Kafka specific configurations。

Kinesis 连接

您可以使用存储在 Data Catalog 表中的信息或通过提供信息直接访问数据流来读取 Amazon Kinesis Data Streams 或向其中写入数据。你可以从 Kinesis 读取信息到 Spark 中 DataFrame，然后将其转换

为 G AWS lue。 DynamicFrame你可以用 JSON 格式写入 DynamicFrames Kinesis。如果直接访问数据流，请使用这些选项提供有关如何访问数据流的信息。

如果您使用 `getCatalogSource` 或 `create_data_frame_from_catalog` 使用来自 Kinesis 流式处理源的记录，则任务具有数据目录数据库和表名称信息，将其用于获取一些基本参数，以便从 Kinesis 流式处理源读取数据。如果使用 `getSource`、`getSourceWithFormat`、`createDataFrameFromOptions` 或 `create_data_frame_from_options`，则必须使用此处描述的连接选项指定这些基本参数。

您可以使用 `GlueContext` 类中指定方法的以下参数为 Kinesis 指定连接选项。

- Scala
  - `connectionOptions`：与 `getSource`、`createDataFrameFromOptions`、`getSink` 结合使用
  - `additionalOptions`：与 `getCatalogSource`、`getCatalogSink` 结合使用
  - `options`：与 `getSourceWithFormat`、`getSinkWithFormat` 结合使用
- Python
  - `connection_options`：与 `create_data_frame_from_options`、`write_dynamic_frame_from_options` 结合使用
  - `additional_options`：与 `create_data_frame_from_catalog`、`write_dynamic_frame_from_catalog` 结合使用
  - `options`：与 `getSource`、`getSink` 结合使用

有关流式处理 ETL 作业的注意事项和限制，请参阅 the section called “串流 ETL 注释和限制”。

配置 Kinesis

要在 G AWS lue Spark 作业中连接到 Kinesis 数据流，你需要一些先决条件：

- 如果读取， AWS Glue 作业必须具有对 Kinesis 数据流的读取访问权限级别 IAM 权限。
- 如果要写入，则 AWS Glue 作业必须对 Kinesis 数据流具有写入访问级别 IAM 权限。

在某些情况下，您需要配置其他先决条件：

- 如果您的 AWS Glue 任务配置了其他网络连接（通常用于连接到其他数据集），并且其中一个连接提供了 Amazon VPC 网络选项，则这将引导您的任务通过 Amazon VPC 进行通信。在这种情况下，您还需要将 Kinesis 数据流配置为通过 Amazon VPC 进行通信。您可以通过在 Amazon VPC

和 Kinesis 数据流之间创建接口 VPC 端点实现此目的。有关更多信息，请参阅 Using Kinesis Data Streams with Interface VPC Endpoints。

- 在另一个账户中指定 Amazon Kinesis Data Streams 时，您必须设置角色和策略，从而允许跨账户访问。有关更多信息，请参阅示例：从不同账户的 Kinesis 串流中读取。

有关流式处理 ETL 作业先决条件的更多信息，请参阅 the section called “流式处理 ETL 作业”。

从 Kinesis 读取

示例：从 Kinesis 流读取

与 the section called “forEachBatch” 结合使用。

Amazon Kinesis 流式处理源示例：

```
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)
```

写入 Kinesis

示例：写入 Kinesis 流

与 the section called “forEachBatch” 结合使用。 DynamicFrame 您将以 JSON 格式写入直播中。如果作业在多次重试后仍无法写入，则会失败。默认情况下，每 DynamicFrame 条记录都将单独发送到 Kinesis 直播中。您可以使用 `aggregationEnabled` 和相关参数配置此行为。

从流式处理作业写入 Amazon Kinesis 的示例：

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 连接参数

为 Amazon Kinesis Data Streams 指定连接选项。

为 Kinesis 流式处理数据源使用以下连接选项：

- "streamARN"（必填）用于读/写。Kinesis 数据流的 ARN。
- "classification"（读取所必填）用于读取。记录中数据使用的文件格式。除非 Data Catalog 提供，否则为必需。
- "streamName" –（可选）用于读取。要从中读取的 Kinesis 数据流的名称。与 endpointUrl 一起使用。
- "endpointUrl" –（可选）用于读取。默认：“https://kinesis.us-east-1.amazonaws.com”。Kinesis 直播的 AWS 端点。除非要连接到特殊区域，否则您无需对此进行更改。
- "partitionKey" –（可选）用于写入。生成记录时所使用的 Kinesis 分区键。
- "delimiter"（可选）用于读取。当 classification 为 CSV 时使用的值分隔符。默认为“,”。
- "startingPosition"：（可选）用于读取。要从中读取数据的 Kinesis 数据流中的起始位置。可能的值是 "latest"、"trim_horizon"、"earliest" 或以模式 yyyy-mm-ddTHH:MM:SSZ 采用 UTC 格式的时间戳字符串（其中 Z 表示带有 +/-的 UTC 时区偏移量。例如：“2023-04-04T08:00:00-04:00”）。默认值为 "latest"。注意：只有 G "startingPosition" AWS lue 版本 4.0 或更高版本支持 UTC 格式的时间戳字符串。

- "failOnDataLoss"：（可选）如果有任何活动分片丢失或已过期，则作业失败。默认值为 "false"。
- "awsSTSRoleARN"：（可选）用于读取/写入。使用 () 代入角色的亚马逊资源名称 AWS Security Token Service (AWS STS ARN)。此角色必须拥有针对 Kinesis 数据流执行描述或读取记录操作的权限。在访问其他账户中的数据流时，必须使用此参数。与 "awsSTSSessionName" 结合使用。
- "awsSTSSessionName"：（可选）用于读取/写入。使用 AWS STS代入角色的会话的标识符。在访问其他账户中的数据流时，必须使用此参数。与 "awsSTSRoleARN" 结合使用。
- "awsSTSEndpoint": (可选）使用代入角色连接到 Kinesis 时要使用的 AWS STS 端点。这允许在 VPC 中使用区域 AWS STS 终端节点，而默认的全局终端节点是不可能的。
- "maxFetchTimeInMs"：（可选）用于读取。作业执行程序从 Kinesis 数据流中读取当前批处理记录所花费的最长时间，以毫秒为单位指定。在这段时间内可以进行多次 GetRecords API 调用。默认值为 1000。
- "maxFetchRecordsPerShard"：（可选）用于读取。每个微批次将从 Kinesis 数据流中的每个分片获取的最大记录数。注意：如果流式传输作业已经从 Kinesis 读取了额外的记录（在同一个 get-records 调用中），则客户端可以超过此限制。如果 maxFetchRecordsPerShard 需要严格，则必须是 maxRecordPerRead 的整数倍。默认值为 100000。
- "maxRecordPerRead"：（可选）用于读取。每项 getRecords 操作中要从 Kinesis 数据流获取的最大记录数。默认值为 10000。
- "addIdleTimeBetweenReads"：（可选）用于读取。在两项连续 getRecords 操作之间添加时间延迟。默认值为 "False"。此选项仅适用于 Glue 版本 2.0 及更高版本。
- "idleTimeBetweenReadsInMs"：（可选）用于读取。两项连续 getRecords 操作之间的最短时间延迟，以毫秒为单位。默认值为 1000。此选项仅适用于 Glue 版本 2.0 及更高版本。
- "describeShardInterval"：（可选）用于读取。两个 ListShards API 调用之间的最短时间间隔，供您的脚本考虑重新分区。有关更多信息，请参阅《Amazon Kinesis Data Streams 开发人员指南》中的重新分区策略。默认值为 1s。
- "numRetries"：（可选）用于读取。Kinesis Data Streams API 请求的最大重试次数。默认值为 3。
- "retryIntervalMs"：（可选）用于读取。重试 Kinesis Data Streams API 调用之前的冷却时间（以毫秒为单位指定）。默认值为 1000。
- "maxRetryIntervalMs"：（可选）用于读取。Kinesis Data Streams API 调用的两次重试之间的最长冷却时间（以毫秒为单位指定）。默认值为 10000。
- "avoidEmptyBatches"：（可选）用于读取。在批处理开始之前检查 Kinesis 数据流中是否有未读数据，避免创建空白微批处理任务。默认值为 "False"。

- "schema"：（当 inferSchema 设为 false 时为必填）用于读取。用于处理有效负载的架构。如果分类为 avro，则提供的架构必须采用 Avro 架构格式。如果分类不是 avro，则提供的架构必须采用 DDL 架构格式。

  以下是一些架构示例。

  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"：（可选）用于读取。默认值为‘false’。如果设置为“true”，则会在运行时检测到 foreachbatch 内的有效工作负载中的架构。
- "avroSchema"：（已弃用）用于读取。用于指定 Avro 数据架构（使用 Avro 格式时）的参数。此参数现已被弃用。使用 schema 参数。

- "addRecordTimestamp"：（可选）用于读取。当选项设置为 'true' 时，数据输出将包含一个名为 "__src_timestamp" 的附加列，表示数据流收到相应记录的时间。默认值为‘false’。4.0 或更高 AWS Glue 版本支持此选项。
- "emitConsumerLagMetrics"：（可选）用于读取。当该选项设置为 “true” 时，对于每个批次，它将发出从直播收到的最旧记录到它到达的时间之间的持续时间内的AWS Glue指标。 CloudWatch 该指标的名字是 “glue.driver.streaming”。 maxConsumerLagInMs”。默认值为‘false’。4.0 或更高 AWS Glue 版本支持此选项。
- "fanoutConsumerARN"：（可选）用于读取。streamARN 中指定的流的 Kinesis 流用户的 ARN。用于为您的 Kinesis 连接启用增强型扇出功能模式。有关使用增强型扇出功能的 Kinesis 流的更多信息，请参阅 the section called “在 Kinesis 流作业中使用增强型扇出功能”。
- "recordMaxBufferedTime" –（可选）用于写入。默认值：1000（ms）。记录在等待写入时缓冲的最长时间。
- "aggregationEnabled" –（可选）用于写入。默认值：真。指定是否应在将记录发送到 Kinesis 之前对其进行汇总。
- "aggregationMaxSize" –（可选）用于写入。默认值：51200（字节）。如果记录超过了此限制，则它将绕过聚合器。注意：Kinesis 将记录大小限制为 50KB。如果您将其设置为 50KB 以上，Kinesis 将拒绝过大的记录。
- "aggregationMaxCount" –（可选）用于写入。默认值：4294967295。要打包至汇总记录的最大项目数量。
- "producerRateLimit" –（可选）用于写入。默认值：150（%）。限制从单个生产者（例如您的作业）发送的每个分片的吞吐量，以占后端限制的百分比表示。
- "collectionMaxCount" –（可选）用于写入。默认值：500。一个 PutRecords 请求中要打包的最大物品数量。
- "collectionMaxSize" –（可选）用于写入。默认值：5242880（字节）。与 PutRecords 请求一起发送的最大数据量。

AWS Glue 直播选项

指定与 Kafka 集群或 Amazon Managed Streaming for Apache Kafka 集群的连接。

您可以使用存储在 Data Catalog 表中的信息或通过提供信息直接访问数据流来从 Kafka 数据流读取或向其中写入数据。你可以从 Kafka 读取信息到 Spark 中 DataFrame，然后将其转换为 AWS Glue。DynamicFrame你可以用 JSON 格式写入 DynamicFrames Kafka。如果直接访问数据流，请使用这些选项提供有关如何访问数据流的信息。

如果您通过 getCatalogSource 或 create_data_frame_from_catalog 来使用来自 Kafka 流式处理源的记录，或通过 getCatalogSink 或 write_dynamic_frame_from_catalog 将记录写入 Kafka，则作业具有 Data Catalog 数据库和表名称信息，并且可以使用该信息来获取一些基本参数，以便从 Kafka 流式处理源读取数据。如果使用 getSource、getCatalogSink、getSourceWithFormat、getSinkWithFormat、createDataFrame 或 write_dynamic_frame_from_catalog，则必须使用此处描述的连接选项指定这些基本参数。

您可以使用 GlueContext 类中指定方法的以下参数为 Kafka 指定连接选项。

- Scala
  - connectionOptions：与 getSource、createDataFrameFromOptions、getSink 结合使用
  - additionalOptions：与 getCatalogSource、getCatalogSink 结合使用
  - options：与 getSourceWithFormat、getSinkWithFormat 结合使用
- Python
  - connection_options：与 create_data_frame_from_options、write_dynamic_frame_from_options 结合使用
  - additional_options：与 create_data_frame_from_catalog、write_dynamic_frame_from_catalog 结合使用
  - options：与 getSource、getSink 结合使用

有关流式处理 ETL 作业的注意事项和限制，请参阅 the section called “串流 ETL 注释和限制”。

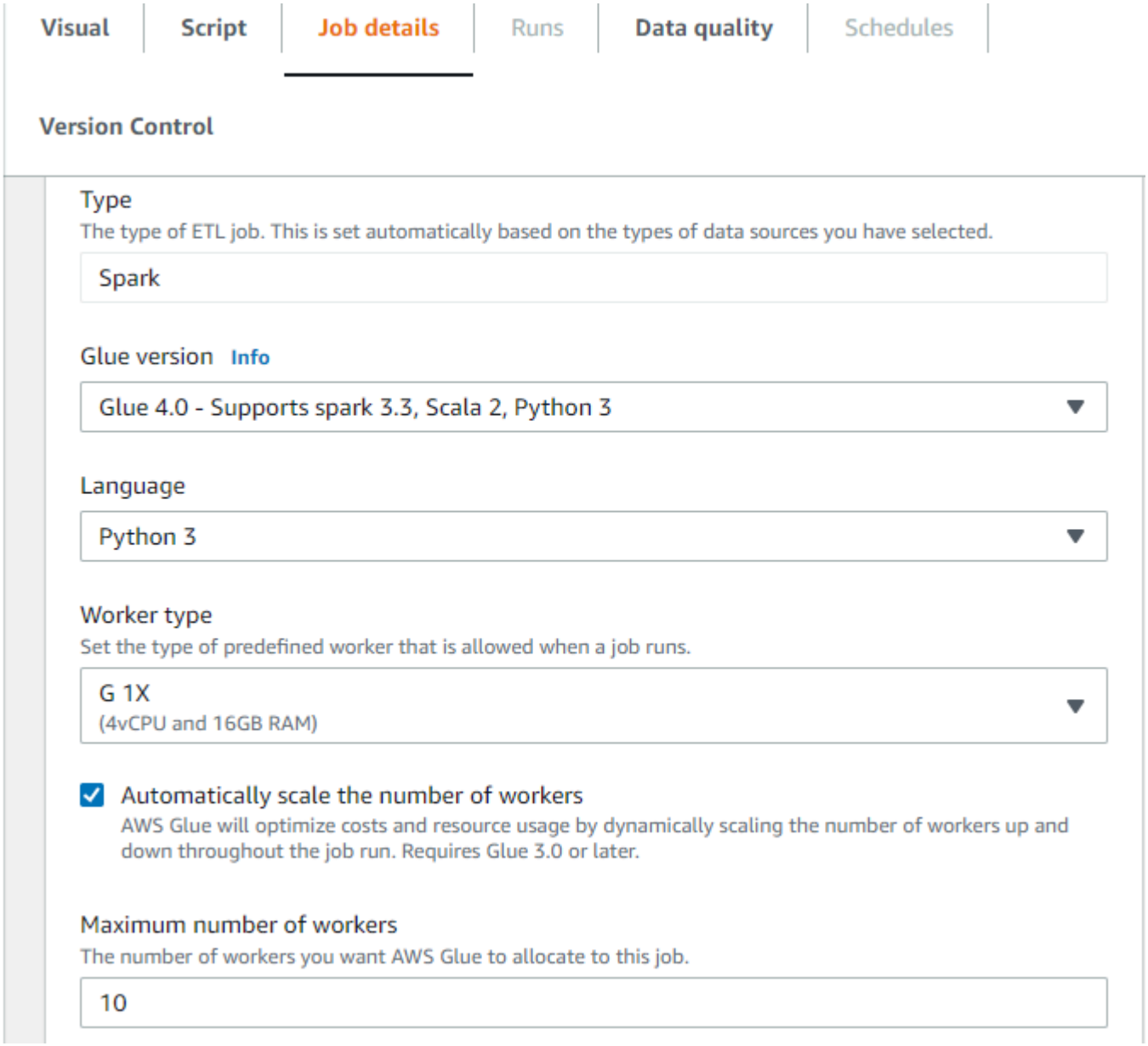
AWS Glue 流式处理自动扩缩

以下部分介绍了有关 AWS Glue 流式处理自动扩缩的信息

在 AWS Glue Studio 中启用自动扩缩

在 AWS Glue Studio 中的 Job details（任务详细信息）选项卡上，将类型选择为 Spark 或 Spark Streaming，并将 Glue version（Glue 版本）选择为 **Glue 3.0** 或 **Glue 4.0**。随后将在 Worker type（工件类型）下方出现一个复选框。

- 选择 Automatically scale the number of workers（自动扩展工件数量）选项。
- 设置 Maximum number of workers（最大工件数量）以定义可提供给任务运行的最大工件数量。



使用 AWS CLI 或开发工具包启用自动扩缩

要通过 AWS CLI 为任务运行启用自动扩缩，使用以下配置运行 `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 任务运行完成后，您还可以调用 `get-job-run` 以检查 DPU-seconds 内任务运行的实际资源使用情况。请注意：为 AWS Glue 3.0 或更高版本上运行的批处理任务启用自动扩缩时，才会显示新字段 DPUSeconds。流式处理任务不支持此字段。

```
$ 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
    }
}
```

您还可以使用具有相同配置的 AWS Glue 开发工具包配置启用自动扩缩的任务运行。

工作方式

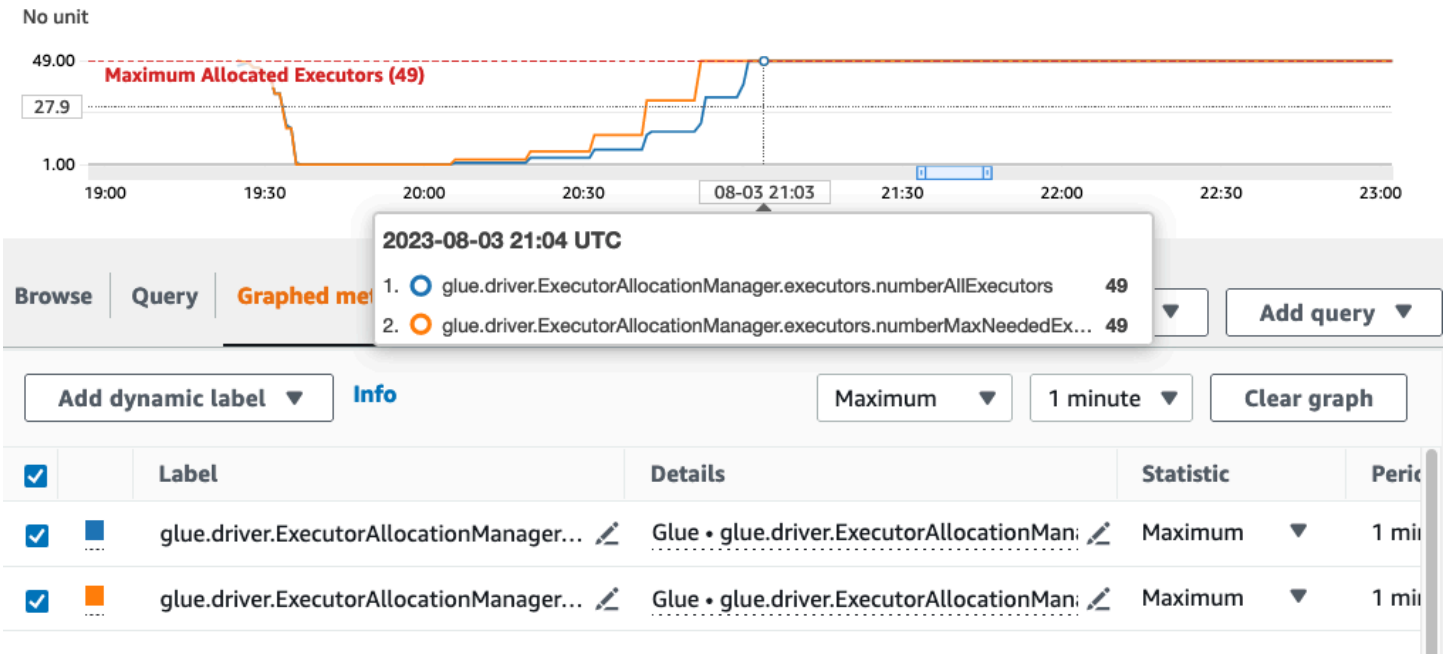
跨微批次扩缩

以下示例描述了自动扩缩的工作原理。

- 您有一个 AWS Glue 作业，该作业起初为 50 个 DPU。
- 自动扩缩已启用。

在本示例中，AWS Glue 将查看几个微批次的“batchProcessingTimeInMs”指标，并确定您的作业是否会在设定的窗口大小内完成。如果您的作业完成得较早，根据完成的速度，AWS Glue 可能会缩减。此指标使用“numberAllExecutors”进行绘制，并可在 Amazon CloudWatch 中进行监控，以了解自动扩缩是如何工作的。

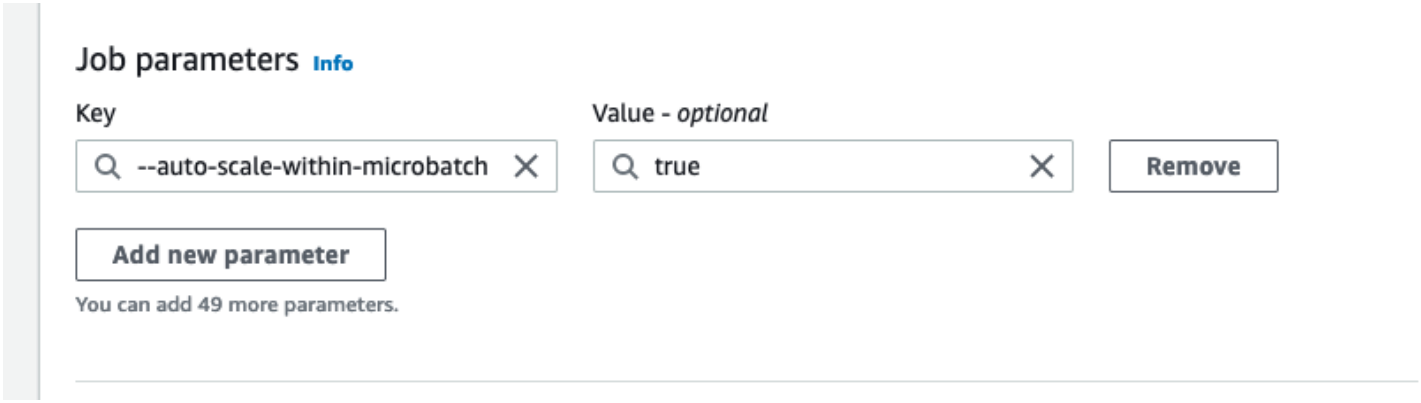
只有在每个微批次完成后，执行程序的数量才会呈指数级扩缩。从 Amazon CloudWatch 监控日志中可以看到，AWS Glue 查看了所需的执行程序数量（橙色线），并自动根据该数量扩缩了执行程序（蓝色线）。



一旦 AWS Glue 缩减了执行程序的数量并观察到数据量增加，从而增加了微批次处理时间，AWS Glue 就会扩展到 50 个 DPU，即指定的上限。

微批次内扩缩

在上面的示例中，系统会监控几个已完成的微批次，以决定是扩大还是缩小规模。如果窗口较长，则需要在微批次内更快地响应，而不是等待几个微批次。对于此类情况，您可以使用一个额外配置 `--auto-scaIe-within-microbatch` 并将其设置为 `true`。您可以将其添加到 AWS Glue Studio 的 AWS Glue 作业属性中，如下所示。



AWS Glue 直播的维护窗口

AWS Glue 定期执行维护活动。在这些维护时段内， AWS Glue 需要重新启动您的直播作业。您可以通过指定维护时段来控制何时重新启动作业。在本节中，我们概述了可以在何处设置维护时段以及应考虑的特定行为。

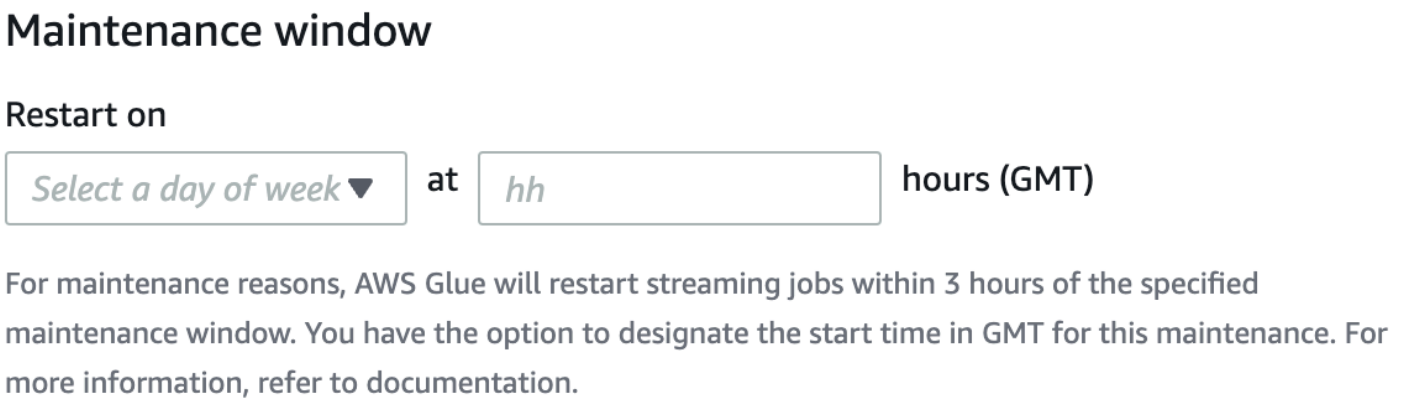
主题

设置维护时段

您可以使用 AWS Glue Studio 或 API 设置维护时段。

在 AWS Glue Studio 中设置维护窗口

您可以在 AWS Glue 流式处理作业的 “任务详情” 页面中指定维护时段。您可以以 GMT 为单位指定日期和时间。 AWS Glue 将在指定的时间窗口内重新启动您的作业。



在 API 中设置维护时段

您也可以在 Create Job API 中设置维护时段。以下是通过 API 配置维护时段的示例。

```
aws glue create-job –name jobName –role roleArnForTheJob –command
 Name=gluestreaming,ScriptLocation=s3-path-to-the-script --maintenance-window="Sun:10"
```

命令示例如下：

```
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
```

维护时段行为

AWS Glue 通过一系列步骤来决定何时重启作业：

1. 启动新的流式处理任务时， AWS Glue 首先检查任务运行是否存在超时。超时允许您配置作业的结束时间。如果超时时间少于 7 天，则作业将不会重新启动。
2. 如果超时时间超过 7 天，则 AWS Glue 检查是否为该作业配置了维护时段。如果是，则会选择该窗口，并将该窗口分配给作业运行。 AWS Glue 将在指定维护时段后的 3 小时内重新启动作业。例如，如果您将维护时段设置为星期一格林威治标准时间上午 10:00，则您的作业将在格林威治标准时间上午 10:00 至下午 1:00 之间重新启动。
3. 如果未配置维护时段，则 AWS Glue 会自动将重启时间设置为作业运行启动时间后 7 天。例如，如果您在格林威治标准时间 2024 年 7 月 1 日上午 12:00 启动任务，但未指定维护时段，则您的任务将设置为格林威治标准时间 2024 年 7 月 8 日上午 12:00 重启。

   > **Note**
   >
   > 如果您已经在运行流媒体作业，那么从 2024 年 7 月 1 日起，这一变化将对您产生影响。在 6 月 30 日之前，您将有时间配置维护时段。7 月 1 日之后，根据本文档，您启动的所有流媒体作业都将重新启动。如果您需要任何其他支持，可以联系 Suppor AWS t。

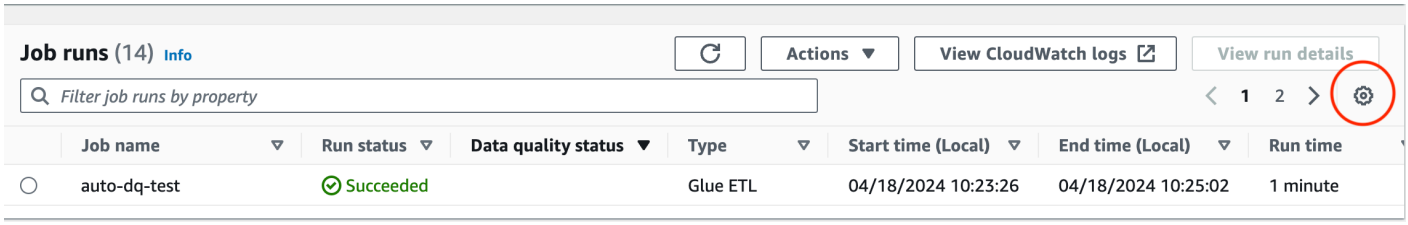
4. 有时， AWS Glue 可能无法重新启动作业，尤其是在未处理正在进行的微批处理时。在这些情况下，作业不会中断。在这些情况下， AWS Glue 将在 14 天后重新启动作业，在这种情况下，维护时段将不被遵守。

Job 监控

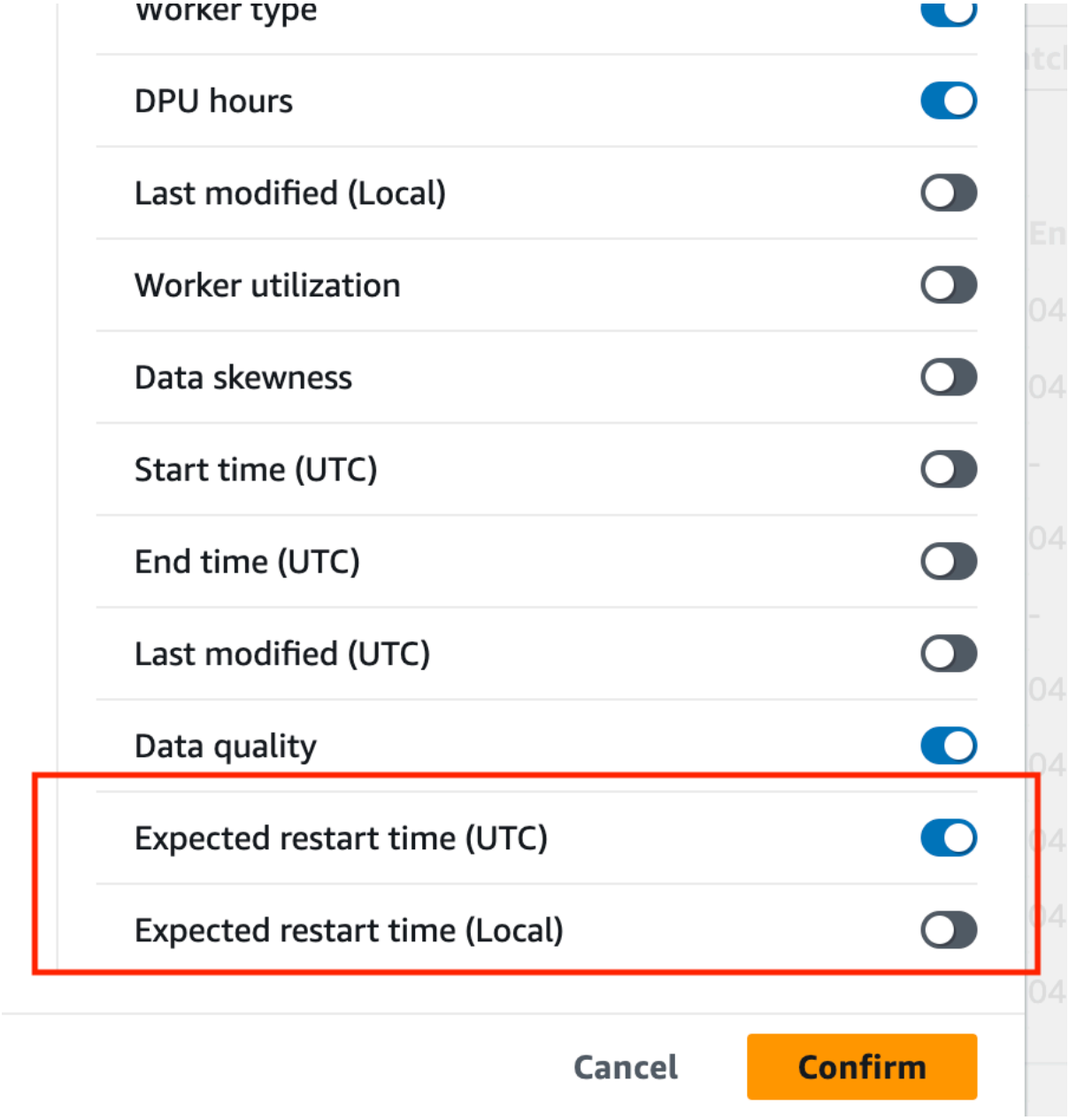
您可以在 AWS Glue Studio 监控页面中监控作业。

要查看流式处理作业的预计下次重启时间，请在 “监控” 页面的 “任务运行” 表中显示相应列。

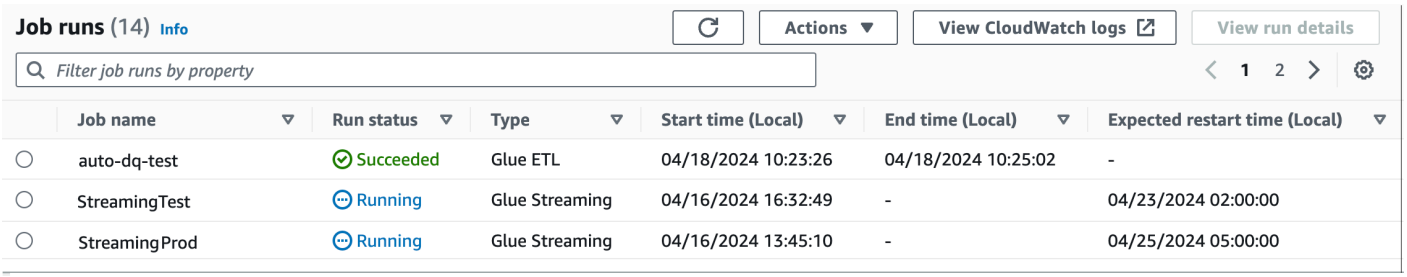
1. 点击表格右上角的齿轮图标。



2. 向下滚动，然后打开 “预计重启时间” 列。UTC 和本地时间选项均可用。



3. 然后，您可以查看表中的列。



原始作业的状态将为 “已过期”，而新的作业实例的状态将为 “正在运行”。重新启动的新作业运行将有一个任务运行 ID，由初始作业运行 ID 加上代表重启计数的前缀 “restart_” 的串联组成。例如，如果初始作业运行 ID 为`jr_1234`，则重新启动的作业运行将具有第一次重启`jr1234_restart_1`的 ID。第二次重启将`jr1234_restart_2`用于第二次重启，依此类推。

您的重试尝试不会因为重启而受到影响。如果运行失败并且由于自动重试而开始了新的运行，则重启计数器将再次从 1 开始。例如，如果运行失败`jr_1234_attempt_3_restart_5`，则自动重试将开始新的运行，ID 为:`jr_id1_attempt_4`，当此尝试在 7 天后重新启动时，新的运行 ID 将为。`jr_id1_attempt_4_restart_1`

数据丢失处理

在维护重启期间， AWS Glue Streaming 会遵循一个确保数据完整性和上一次作业运行与重新启动的作业运行之间一致性的流程。请注意，这并 AWS Glue 不能保证两次任务重启之间的数据完整性和一致性，我们建议在处理流式作业中的重复数据时考虑架构问题。

1. 检测维护重启条件： AWS Glue 流式传输监视指明何时应触发维护重启的情况，例如 7 天后达到维护窗口或 14 天后需要硬重启的情况。
2. 调用优雅终止：当满足维护重启条件时， AWS Glue Streaming 会为当前正在运行的作业启动一个优雅的终止流程。此过程包括以下步骤：
   a. 停止摄取新数据：流式传输作业停止使用来自输入源（例如 Kafka 主题、Kinesis 直播或文件）的新数据。
   b. 处理待处理数据：作业继续处理其内部缓冲区或队列中已存在的任何数据。
   c. 提交偏移量和检查点：任务将最新的偏移量或检查点提交到外部系统（例如 Kafka、Kinesis 或 Amazon S3），以确保重新启动的任务可以从上一个任务中断的地方继续运行。
3. 重新启动作业：正常终止过程完成后， AWS Glue Streaming 将使用保留的状态和检查点重新启动作业。重新启动的作业从上次提交的偏移量或检查点开始处理，确保没有数据丢失或重复。
4. 恢复数据处理：重新启动的作业将从上一个作业中断的地方恢复数据处理。它继续从输入源提取新数据，从上次提交的偏移量或检查点开始，并根据定义的 ETL 逻辑处理数据。

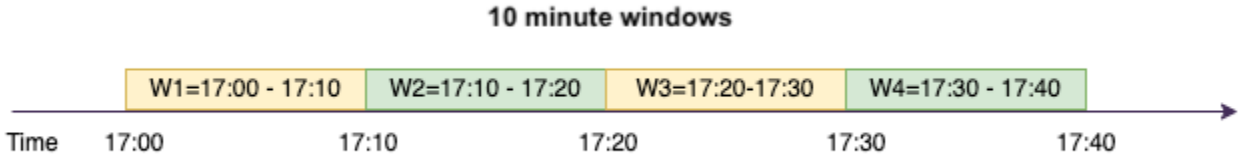
高级 AWS Glue 流式处理概念

在当代数据驱动的应用中，数据的重要性会随着时间的推移而减弱，其价值也从预测性转变为被动性。因此，客户希望实时处理数据，以便更快地做出决策。在处理实时数据源时，比如来自 IoT 传感器的数据，由于网络延迟和摄取过程中其他源方面的故障，数据可能会无序到达或在处理过程中出现延迟。作为 AWS Glue 平台的一部分，AWS Glue 流式处理基于这些功能构建，提供可扩展的无服务器流 ETL，由 Apache Spark 结构化流提供支持，使用户能够进行实时数据处理。

在本主题中，我们将探讨 AWS Glue 流式处理的高级流概念和功能。

处理流时的时间注意事项

处理流时有四个时间概念：



- 事件时间 - 事件发生的时间。在大多数情况下，该字段在源处嵌入事件数据本身。
- E vent-time-window-两个事件时间之间的时间范围。如上图所示，W1 的范围是 17:00 event-time-window 到 17:10。每个事件 event-time-window 都是由多个事件组成的分组。
- 触发时间 - 触发时间控制数据处理和结果更新的频率。这是微批次开始的时间。
- 摄取时间 - 将流式处理数据摄取到流服务中的时间。如果事件时间未嵌入事件本身，则在某些情况下，此时间可用于窗口化。

窗口化

窗口化是一种按以下方式对多个事件进行分组和聚合的 event-time-window技术。我们将在以下示例中探讨窗口化的好处以及何时使用。

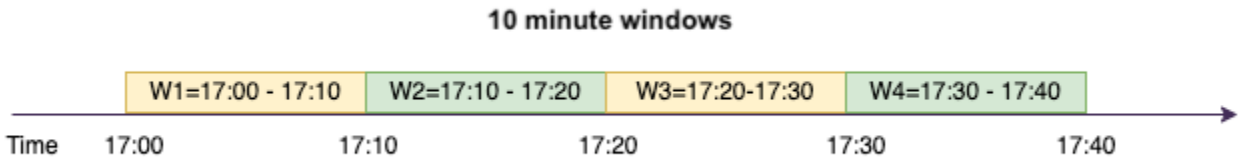
根据业务用例，Spark 支持三种类型的时间窗口。

- 翻滚窗口 — 一系列不重叠的固定大小，您可以聚合 event-time-windows 在上面。
- 滑动窗口 - 从“固定大小”的角度来看，它类似于滚动窗口，但只要滑动的持续时间小于窗口本身的持续时间，窗口就可以重叠或滑动。

- 会话窗口 - 从输入数据事件开始，只要在间隙或非活动时间内收到输入，它就会继续自行扩展。会话窗口的窗口长度可以是静态的，也可以是动态的，具体取决于输入。

滚动窗口

翻滚窗口是一系列不重叠的固定大小 event-time-windows ，您可以聚合这些窗口。为便于理解，我们来看一个真实的例子。



ABC 汽车公司想为一款新型跑车做一个营销活动。他们想挑选一个拥有庞大跑车迷群体的城市。为此，他们在网站上投放了一段 15 秒钟的广告短片，介绍这款汽车。所有 “点击” 和相应的 “城市” 都被记录下来并流式传输到 Amazon Kinesis Data Streams。我们想计算 10 分钟内的点击次数，然后按城市分组，看看哪个城市的需求量最大。以下是聚合的输出。

window_start_time	window_end_time	city	total_clicks
2023-07-10 17:00:00	2023-07-10 17:10:00	达拉斯	75
2023-07-10 17:00:00	2023-07-10 17:10:00	芝加哥	10
2023-07-10 17:20:00	2023-07-10 17:30:00	达拉斯	20
2023-07-10 17:20:00	2023-07-10 17:30:00	芝加哥	50

如上所述， event-time-windows 这些间隔不同于触发时间间隔。例如，即使您的触发时间是每分钟，输出结果也只会显示 10 分钟不重叠的聚合窗口。为了进行优化，最好将触发间隔与 event-time-window。

在上表中，达拉斯在 17:00-17:10 窗口内有 75 次点击，而芝加哥有 10 次点击。此外，17:10-17:20 窗口内没有任何城市的数据，因此忽略此窗口。

现在，您可以使用下游分析应用程序对这些数据进行进一步分析，来确定最适合开展营销活动的城市。

在 AWS Glue 中使用滚动窗口

1. 创建 Amazon Kinesis Data Streams DataFrame 并从中读取。例如：

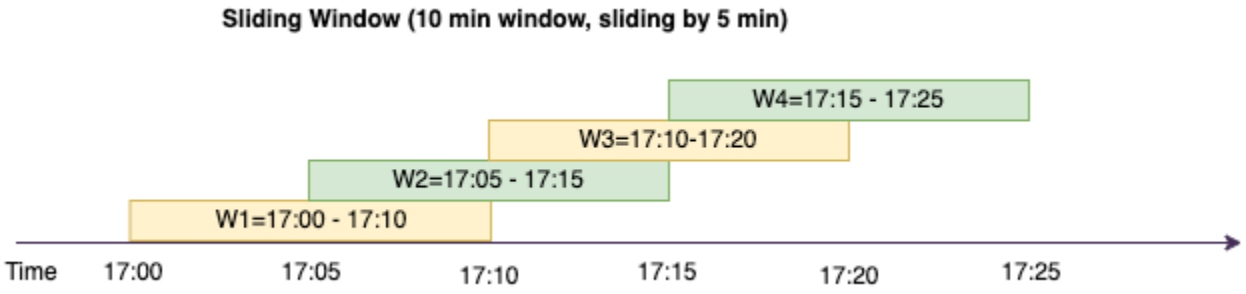
```
parsed_df = kinesis_raw_df \
                .selectExpr('CAST(data AS STRING)') \
                .select(from_json("data", ticker_schema).alias("data")) \
                .select('data.event_time','data.ticker','data.trade','data.volume',
 'data.price')
```

2. 在滚动窗口中处理数据。在下面的示例中，根据 10 分钟滚动窗口中的输入字段“event_time”对数据进行分组，并将输出写入 Amazon S3 数据湖。

```
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()
```

滑动窗口

从“固定大小”的角度来看，滑动窗口类似于滚动窗口，但只要滑动的持续时间小于窗口本身的持续时间，窗口就可以重叠或滑动。由于滑动的性质，一个输入可以绑定到多个窗口。



为了更好地理解，我们以一家银行为例，该银行想要检测潜在的信用卡欺诈行为。流式处理应用程序可以监控连续的信用卡交易流。这些交易可以聚合到持续时间为 10 分钟的窗口中，每隔 5 分钟，窗口就会向前滑动，删除最早 5 分钟的旧数据，添加最近 5 分钟的新数据。在每个窗口内，可按国家/地区对交易进行分组，检查可疑模式，例如在美国进行一笔交易后，紧接着在澳大利亚进行另一笔交易。为简单起见，当交易总额超过 100 美元时，我们将此类交易归类为欺诈。如果检测到这种模式，则表明存在潜在的欺诈行为，信用卡可能会被冻结。

信用卡处理系统会将每张卡片 ID 以及国家/地区的一系列交易事件发送到 Kinesis。 AWS Glue 作业运行分析并生成以下聚合输出。

window_start_time	window_end_time	card_last_four	country	total_amount
2023-07-10 17:00:00	2023-07-10 17:10:00	6544	美国	85
2023-07-10 17:00:00	2023-07-10 17:10:00	6544	澳大利亚	10
2023-07-10 17:05:45	2023-07-10 17:15:45	6544	美国	50
2023-07-10 17:10:45	2023-07-10 17:20:45	6544	美国	50

window_start_time	window_end_time	card_last_four	country	total_amount
2023-07-10 17:10:45	2023-07-10 17:20:45	6544	澳大利亚	150

根据上述聚合数据，可以看到 10 分钟窗口每 5 分钟滑动一次，交易金额相加。在 17:10 - 17:20 的窗口中检测到异常，其中有一个异常值，这是在澳大利亚的一笔 150 美元的交易。AWS Glue 可以检测到这种异常，然后使用 boto3 将带有违规键的警报事件推送到 SNS 主题。此外，Lambda 函数可以订阅此主题并采取行动。

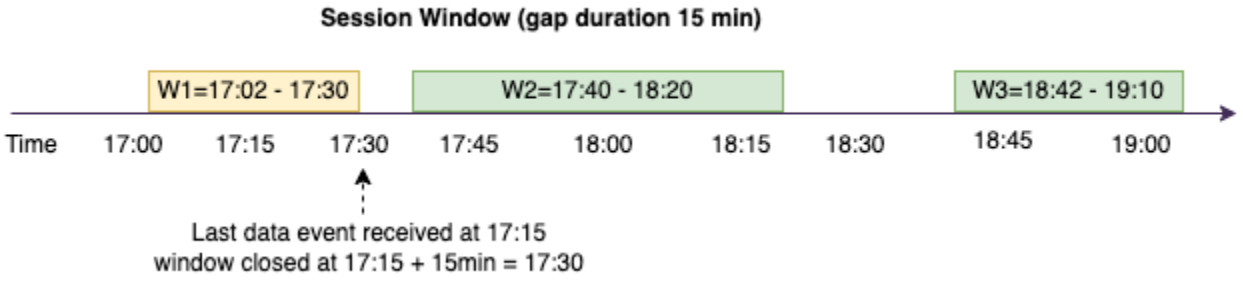
在滑动窗口中处理数据

group-by 子句和窗口函数用于实现滑动窗口，如下所示。

```
grouped_df = parsed_df \
            .groupBy(window(col("event_time"), "10 minute", "5 min"), "country",
 "card_last_four") \
            .agg(sum("tx_amount").alias("total_amount"))
```

会话窗口

与上面两个固定大小的窗口不同，会话窗口的窗口长度可以是静态的，也可以是动态的，具体取决于输入。会话窗口从输入数据事件开始，只要在间隙或非活动时间内收到输入，它就会继续自行扩展。



举个例子。ABC 酒店公司希望了解一周中最繁忙的时间是什么时候，从而为客人提供更好的服务。访客签入后，会话窗口就会启动，并且 spark 会通过聚合来保持该会话窗口的状态。 event-time-window 每当房客办理入住手续时，都会生成一个事件并将其发送到 Amazon Kinesis Data Streams。酒店决定，如果在15分钟内没有办理入住手续，则 event-time-window 可以关闭。当有新的办理登机手续时，下一次 event-time-window 将重新开始。输出如下所示。

window_start_time	window_end_time	city	total_checkins
2023-07-10 17:02:00	2023-07-10 17:30:00	达拉斯	50
2023-07-10 17:02:00	2023-07-10 17:30:00	芝加哥	25
2023-07-10 17:40:00	2023-07-10 18:20:00	达拉斯	75
2023-07-10 18:50:45	2023-07-10 19:15:45	达拉斯	20

第一次入住发生在 event_time=17:02。聚合 event-time-window 将从 17:02 开始。只要我们在 15 分钟内收到事件，聚合就会继续。在上面的示例中，我们收到的最后一个事件是在 17:15，然后接下来的 15 分钟内没有事件。结果，Spark event-time-window 在 17:15 +15min = 17:30 将其关闭，并将其设置为 17:02-17:30。它 event-time-window 在 17:47 开始了一个新的签到数据事件，当时它收到了一个新的签到数据事件。

在会话窗口中处理数据

`group-by` 子句和窗口函数用于实现滑动窗口。

```
grouped_df = parsed_df \
            .groupBy(session_window(col("event_time"), "10 minute"), "city") \
            .agg(count("check_in").alias("total_checkins"))
```

输出模式

输出模式是将来自无界表的结果写入外部接收器的模式。共有三种模式。在下面的示例中，在每个微批次中流送和处理数据行时，系统将会计算词的出现次数。

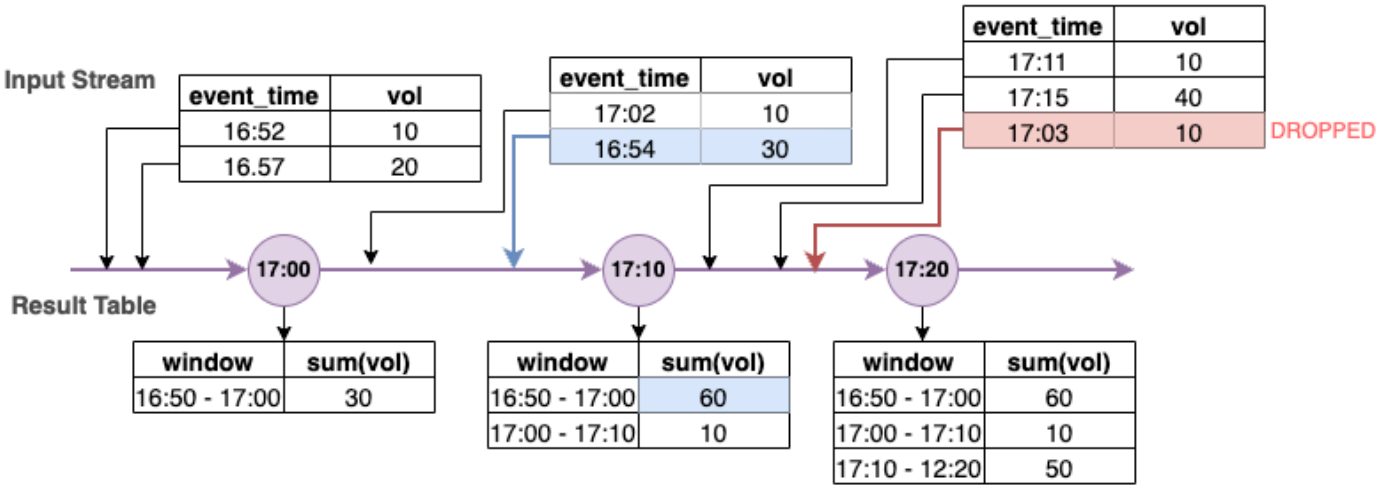
- 完成模式 — 即使当前字数未更新，每次微批处理后，整个结果表仍将写入接收器 event-time-window。
- 追加模式 - 这是默认模式，只有自上次触发以来添加到结果表中的新词和/或行才会写入接收器。这种模式适用于无状态流式处理查询，比如 map、flatMap、filter 等。
- 更新模式 - 只有自上次触发以来更新或添加的结果表中的词和/或行才会写入接收器。

> **Note**
>
> 会话窗口不支持输出模式 = "update"。

处理延迟数据和水印

在处理实时数据时，由于网络延迟和上游故障，数据的到达可能会延迟，我们需要一种机制来重新对错过的数据进行聚合 event-time-window。但要做到这一点，就需要维护状态。同时，需要清理旧数据以限制状态的大小。Spark 2.1 版新增了对水印功能的支持，该功能可维护状态并允许用户指定延迟数据的阈值。

参考上面的股票代码示例，我们考虑一下允许的延迟数据阈值不超过 10 分钟的情况。为了简单起见，我们假设采用滚动窗口，股票代码为 AMZ，交易为买入。



在上图中，我们计算的是 10 分钟滚动窗口内的总交易量。触发时间分别为 17:00、17:10 和 17:20。时间轴箭头上方是输入数据流，下方是无界结果表。

在第一个 10 分钟滚动窗口中，我们根据 event_time 进行聚合，计算的 total_volume 为 30。在第二个事件中 event-time-window，spark 以 event_time= 17:02 的形式获得了第一个数据事件。由于这是 Spark 迄今为止看到的最大 event_time，因此将水印阈值设置为 10 分钟前（即 watermark_event_time=16:52）。event_time 在 16:52 之后的任何数据事件都将被考虑进行时间限制聚合，而在此之前的任何数据事件都将被丢弃。这样，Spark 就能将中间状态多维持 10 分钟，以容纳延迟数据。时钟时间 17:08 左右，Spark 收到一个 event_time=16:54 的事件，该事件在阈值内。因此，Spark 重新计算了 “16:50-17:00” event-time-window ，总音量从 30 更新为 60。

但在触发时间 17:20，当 Spark 接收到 event_time = 17:15 的事件时，它会设置 watermark_event_time=17:05。因此，event_time=17:03 的延迟数据事件被认为“太迟”而被忽略。

```
Watermark Boundary = Max(Event Time) - Watermark Threshold
```

在 AWS Glue 中使用水印

在越过水印边界之前，Spark 不会向外部接收器发出或写入数据。要在 AWS Glue 中实现水印，请参阅以下示例。

```
grouped_df = parsed_df \
                .withWatermark("event_time", "10 minutes") \
                .groupBy(window("event_time", "5 minutes"), "ticker") \
                .agg(sum("volume").alias("total_volume"))
```

监控 AWS Glue 流式处理作业

监控流式处理作业是构建 ETL 管道的关键部分。除了使用 Spark UI 之外，您还可以使用 Amazon CloudWatch 来监控各项指标。以下是 AWS Glue 框架发出的流式处理指标列表。有关所有 AWS Glue 指标的完整列表，请参阅使用 Amazon CloudWatch 指标监控 AWS Glue。

AWS Glue 使用结构化流式处理框架处理输入事件。您可以直接在代码中使用 Spark API，也可以利用 `GlueContext` 提供的 `ForEachBatch` 发布这些指标。要了解这些指标，我们需要先了解 `windowSize`。

windowSize：`windowSize` 是您提供的微批次间隔。如果指定的窗口大小为 60 秒，则 AWS Glue 流式处理作业将等待 60 秒（如果届时上一批次还未完成，则会等待更长时间），然后才会从流式处理数据源批量读取数据，并应用 `ForEachBatch` 中提供的转换。这也称为触发间隔。

我们详细看一下这些指标，以了解运行状况和性能特征。

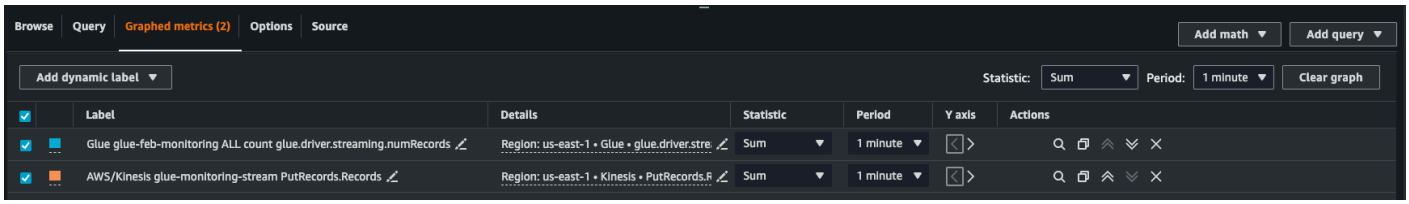
> **Note**
>
> 这些指标每 30 秒发出一次。如果 `windowSize` 少于 30 秒，则报告的指标就是一个聚合。例如，假设 `windowSize` 为 10 秒，并且对于每个微批次，您稳定处理 20 条记录。在这种情况下，为 numRecords 发出的指标值为 60。
>
> 如果没有可用的数据，则不会发出指标。此外，对于使用者滞后指标，必须启用该功能才能获取相关指标。

可视化指标

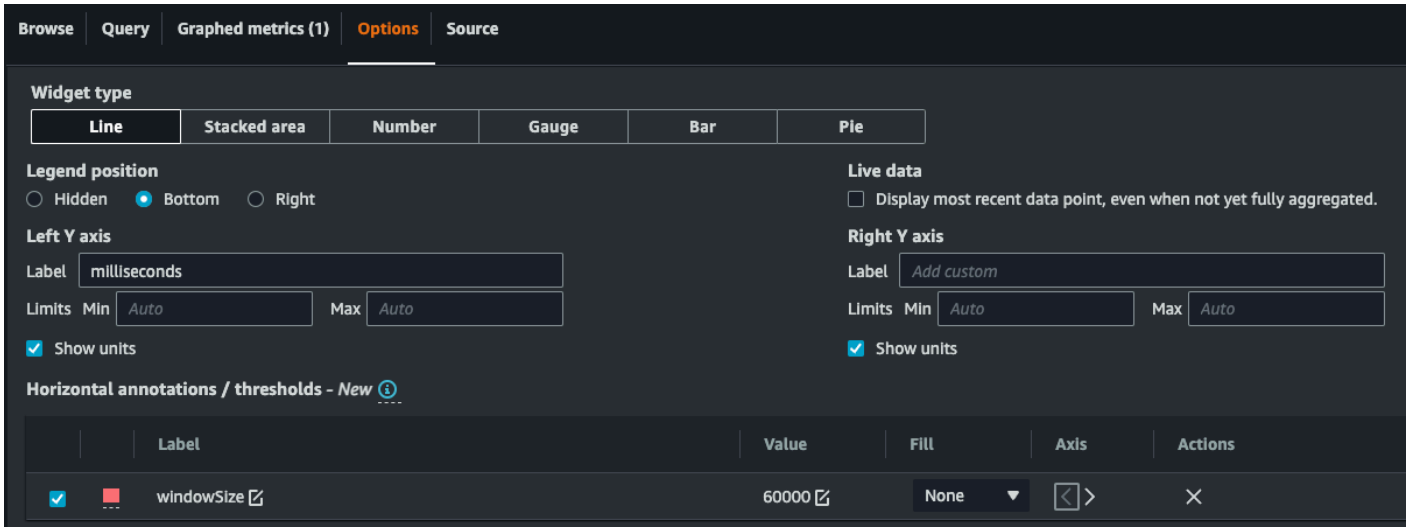
要绘制视觉指标：

1. 转到 Amazon CloudWatch 控制台中的指标，然后选择浏览选项卡。然后在“自定义命名空间”下选择 Glue。

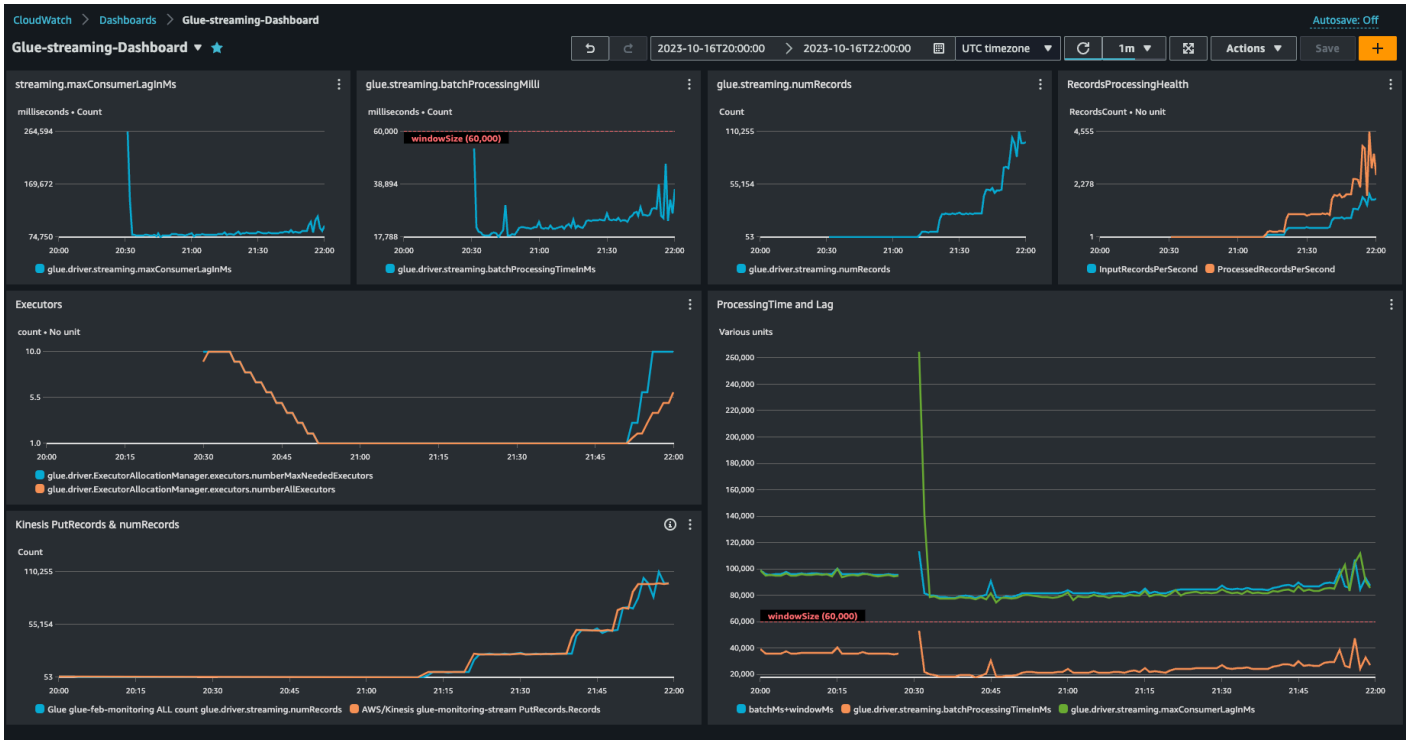
2. 选择作业指标以显示所有作业的指标。
3. 依次按 JobName=glue-feb-monitoring 和 JobRunId=ALL 筛选指标。您可以单击“+”号（如下图所示），将其添加到搜索筛选条件。
4. 选中您感兴趣的指标对应的复选框。在下图中，我们选择了 `numberAllExecutors` 和 `numberMaxNeededExecutors`。



5. 选择这些指标后，您可以转到绘成图表的指标选项卡，应用您的统计数据。
6. 由于指标每分钟发出一次，因此可以对 `batchProcessingTimeInMs` 和 `maxConsumerLagInMs` 应用一分钟的“平均值”。对于 `numRecords`，您可以应用每分钟的“总和”值。
7. 您可以使用选项选项卡为图表添加水平 `windowSize` 注释。



8. 选择指标后，创建控制面板并添加。下面是一个示例控制面板。

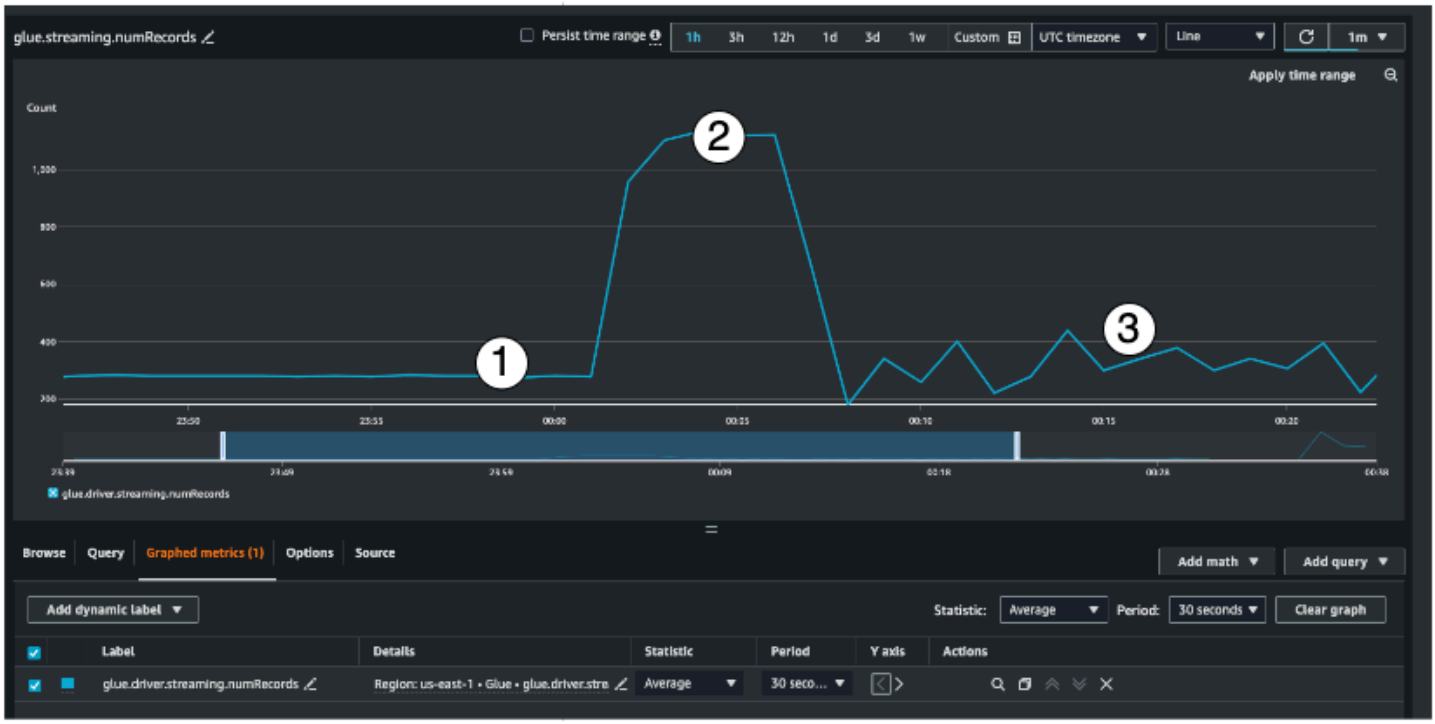


指标详解

该部分介绍了每个指标以及它们之间的相互关系。

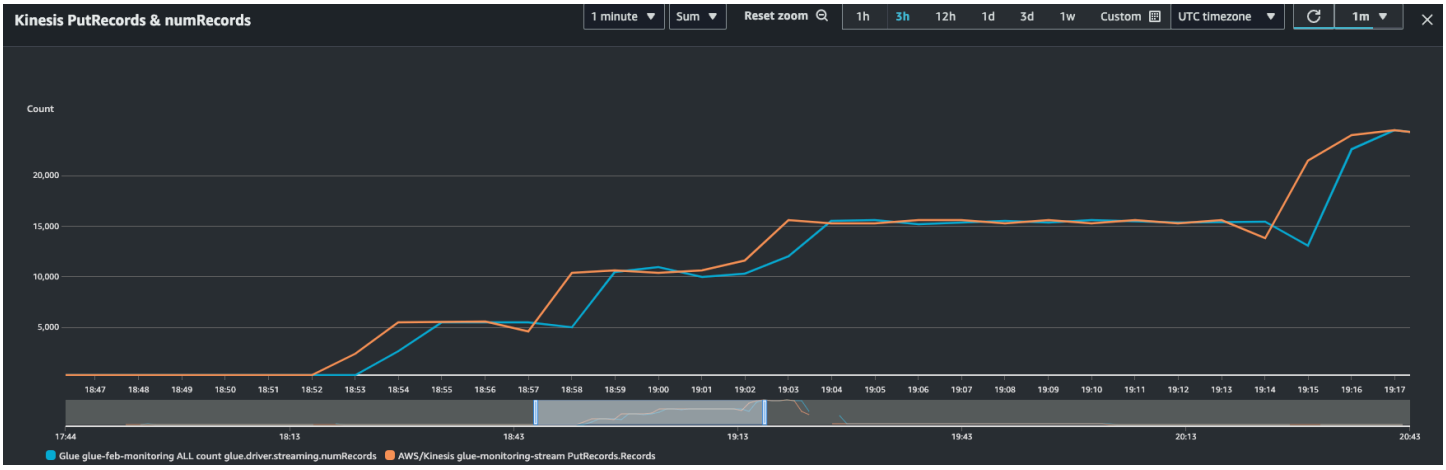
记录数（指标：streaming.numRecords）

该指标指示正在处理的记录数。



该流式处理指标让您可以了解窗口中正在处理的记录数。除了正在处理的记录数，它还可以帮助您了解输入流量的行为。

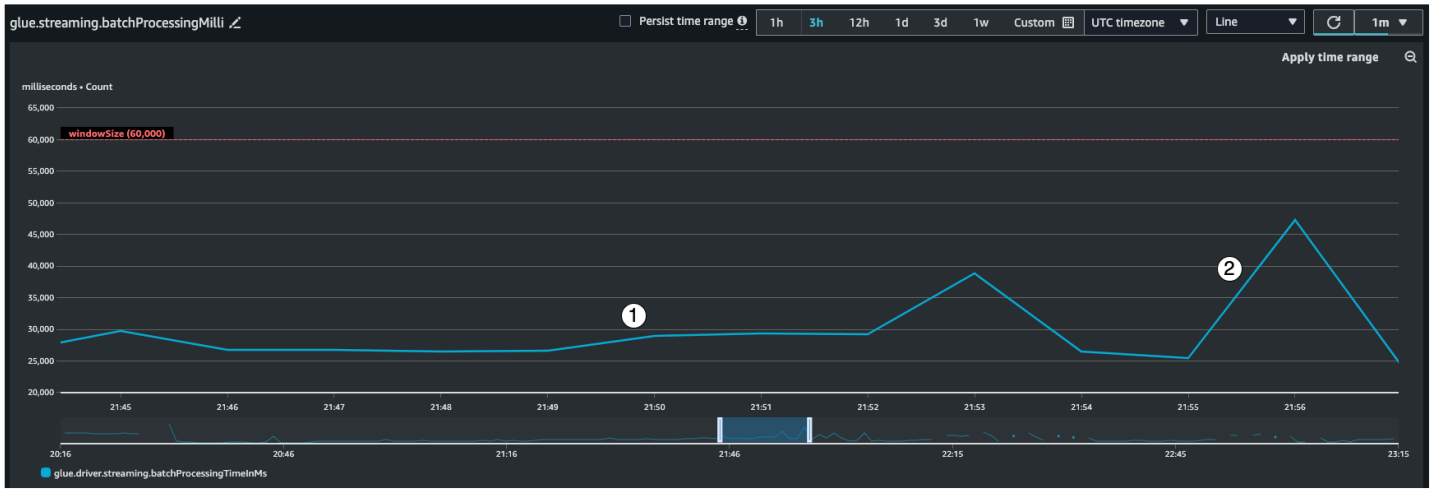
- 指标 #1 显示了一个流量稳定、无流量突发的例子。通常，类似于 IoT 传感器这样的应用程序，它们会定期收集数据并将其发送到流式处理数据源。
- 指标 #2 显示了一个在原本稳定的负载下流量突发的例子。这可能发生在点击流应用中，比如黑色星期五这样的营销活动，点击量激增
- 指标 #3 显示了一个流量不可预测的例子。不可预测的流量确实意味着存在问题。这是输入数据的性质决定的。回到 IoT 传感器的例子，您可以想象有数百个传感器将天气变化事件发送到流式处理数据源。天气变化不可预测，因此数据也不可预测。了解流量模式是调整执行程序数量的关键。如果输入量很大，可以考虑使用自动扩缩（稍后会详细介绍）。



您可以将此指标与 Kinesis PutRecords 指标相结合，确保摄取的事件数和读取的记录数几乎相同。当您试图理解滞后时，这特别有用。随着摄取速率的提高，AWS Glue 的 numRecords 读取量也随之增加。

批处理时间（指标：streaming.batchProcessingTimeInMs）

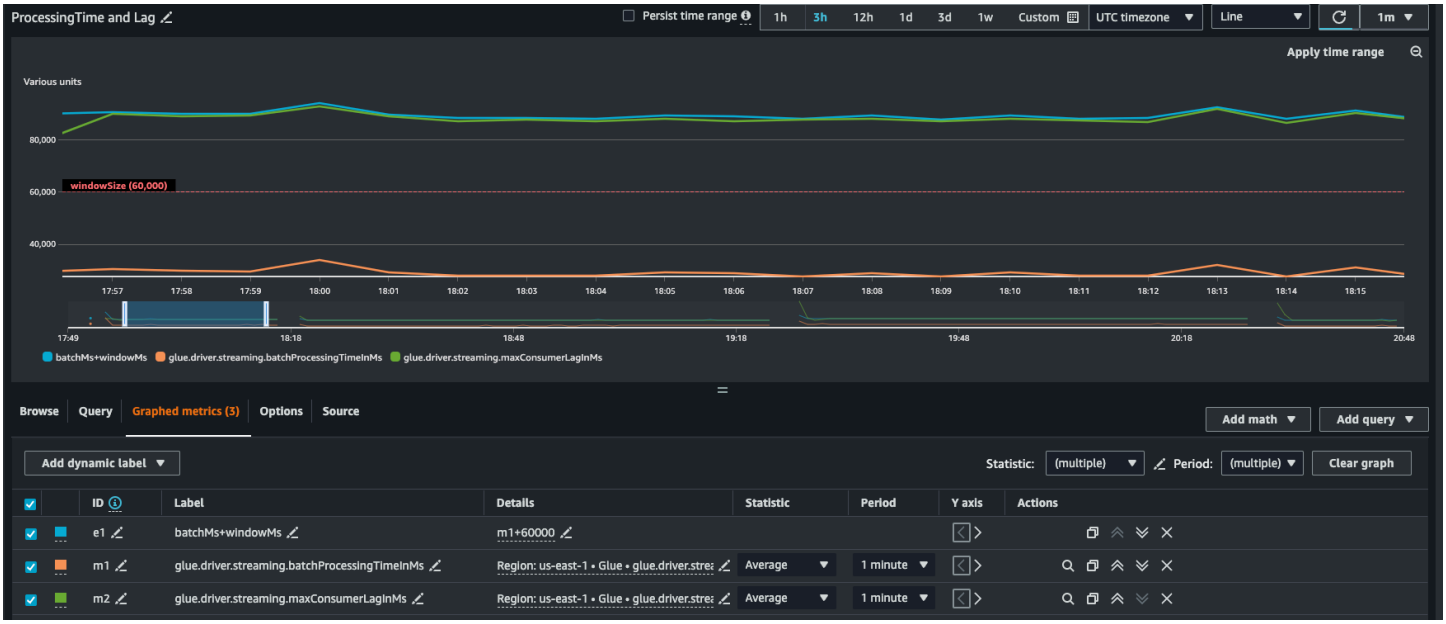
批处理时间指标有助于确定集群是预置不足还是预置过度。



该指标指示处理每个微批次记录所用的毫秒数。其主要目标是监控这段时间，确保其小于 windowSize 间隔。只要在下一个窗口间隔内恢复，batchProcessingTimeInMs 暂时超时也没关系。指标 #1 显示了处理作业所需的或多或少的稳定时间。但如果输入记录数在增加，处理作业所需的时间就会增加，如指标 #2 所示。如果 numRecords 没有增加，但处理时间却在增加，则需要深入了解执行程序的作业处理情况。最好设置阈值和警报，确保 batchProcessingTimeInMs 在 120% 以上的时间不会超过 10 分钟。有关设置警报的更多信息，请参阅使用 Amazon CloudWatch 警报。

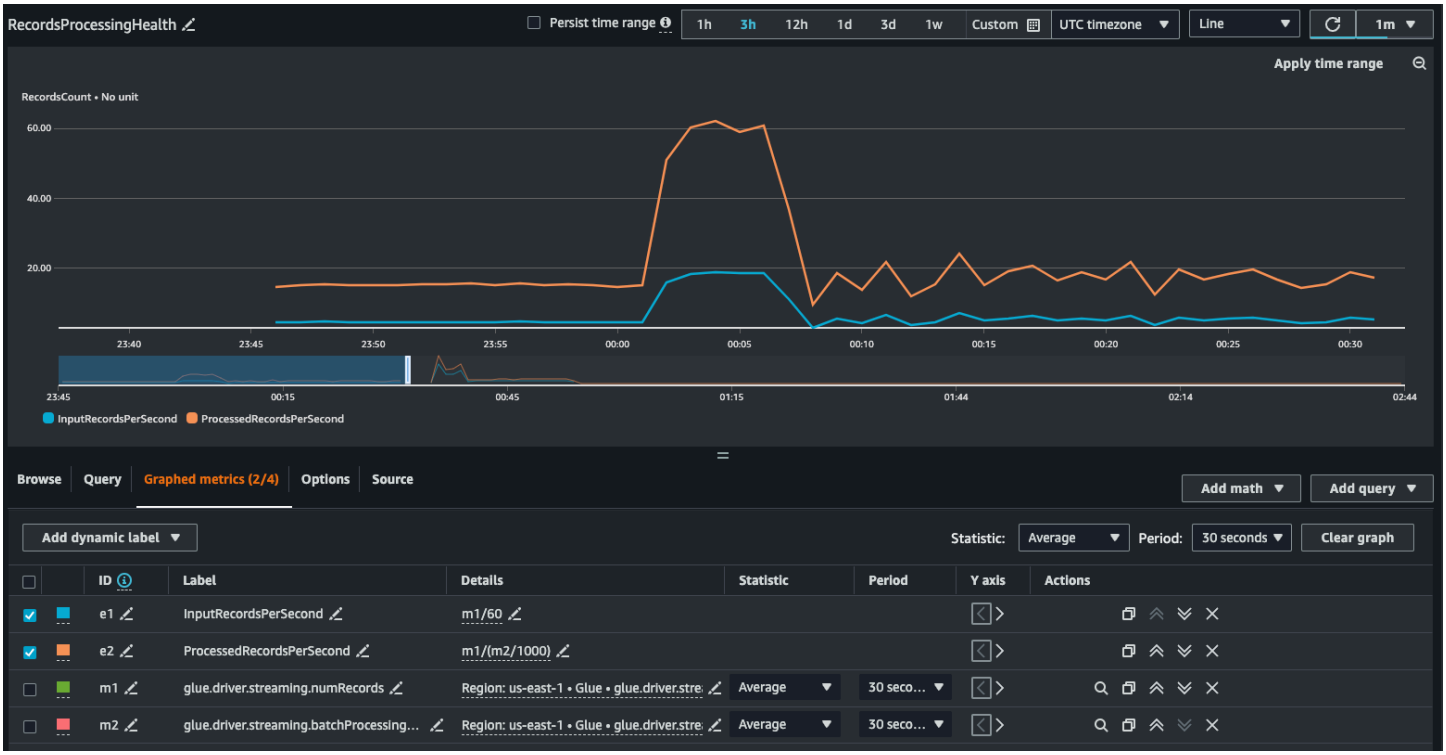
使用者滞后（指标：streaming.maxConsumerLagInMs）

使用者滞后指标有助于您了解在处理事件时是否存在滞后。如果滞后太高，那么即使 windowSize 是正确的，也可能会错过业务所依赖的处理 SLA。您必须使用 `emitConsumerLagMetrics` 连接选项明确启用此指标。有关更多信息，请参阅 KinesisStreamingSourceOptions。



派生指标

为了获得更深入的见解，您可以创建衍生指标，以进一步了解 Amazon CloudWatch 中的流式处理作业。



您可以使用衍生指标构建图表，来决定是否需要使用更多 DPU。虽然自动扩缩可以帮助您自动完成此操作，但您可以使用衍生指标来确定自动扩缩是否有效。

- InputRecordsPerSecond 指示获取输入记录的速率。其衍生如下：输入记录数（glue.driver.streaming.numRecords）/WindowSize。
- ProcessingRecordsPerSecond 指示处理记录的速率。其衍生如下：输入记录数（glue.driver.streaming.numRecords）/batchProcessingTimeInMs。

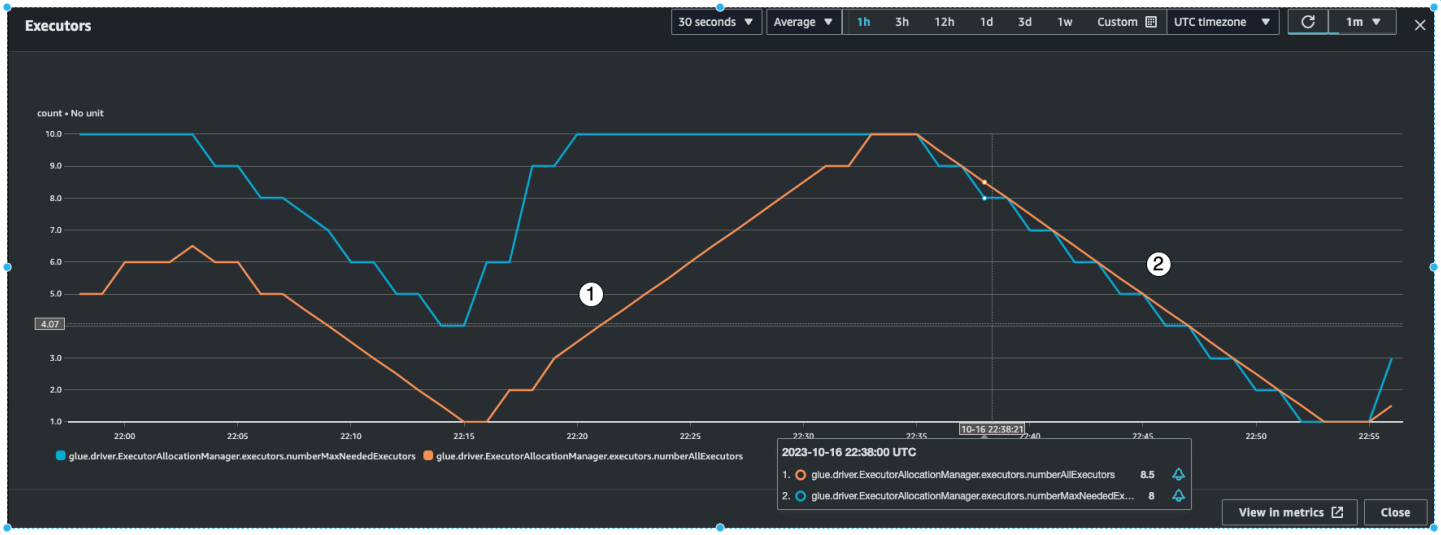
如果输入速率高于处理速率，则可能需要添加更多容量来处理作业或提高并行度。

自动扩缩指标

当输入流量激增时，应考虑启用自动扩缩并指定最大工作线程数量。这样，您就可以获得两个额外的指标：`numberAllExecutors` 和 `numberMaxNeededExecutors`。

- numberAllExecutors 是主动运行的作业执行程序数量
- numberMaxNeededExecutors 是为满足当前负载所需的最大（主动运行和待处理）作业执行程序数量。

这两个指标有助于您了解自动扩缩是否正常运行。



AWS Glue 将在几个微批次中监控 batchProcessingTimeInMs 指标，然后执行以下两个操作之一。如果 batchProcessingTimeInMs 接近于 windowSize，则会扩展执行程序；如果 batchProcessingTimeInMs 相对低于 windowSize，则会缩减执行程序。此外，还将使用一种算法来逐步扩缩执行程序。

- 指标 #1 显示了活动执行程序如何扩展，以满足处理负载所需的最大执行程序数量。
- 指标 #2 显示了自 batchProcessingTimeInMs 较低以来活动执行程序是如何缩减的。

您可以使用这些指标来监控当前执行程序级别的并行度，并相应地调整自动扩缩配置中的最大工作线程数量。

如何获得最佳性能

Spark 会尝试为每个分片创建一个任务，以便从 Amazon Kinesis 流中读取数据。每个分片中的数据成为一个分区。然后，它将根据每个工作线程上的内核数量（每个工作线程的内核数量取决于您选择的工作线程类型 G.025X、G.1X 等），在执行程序/工作线程中分配这些任务。但任务的分配方式是不确定的。所有任务均在各自的内核上并行执行。如果分片数量多于可用的执行程序内核数量，任务就会排队。

您可以使用上述指标和分片数量的组合，为执行程序配置稳定的负载，并留出一定的突发空间。建议您运行几次作业迭代，以确定工作线程的大致数量。对于不稳定/突发的工作负载，您也可以通过设置自动扩缩和最大工作线程来实现。

根据业务的 SLA 要求设置 windowSize。例如，如果您的业务要求处理后的数据不能滞后超过 120 秒，那么将 windowSize 设置为至少 60 秒，这样使用者的平均滞后就会少于 120 秒（参阅上文有

关使用者滞后的部分）。从那里开始，根据分片的 numRecords 和数量，规划 DPU 中的容量，确保 batchProcessingTimeInMs 在大部分时间里都低于 70% 的 windowSize。

> **Note**
>
> 热分片会导致数据倾斜，这意味着某些分片/分区比其他分片/分区大得多。这可能会导致一些并行运行的任务耗时更长，造成任务滞后。因此，在上一批任务全部完成之前，下一批任务无法启动，这将影响 batchProcessingTimeInMillis 和最大滞后。

AWS Glue 数据质量

AWS Glue 数据质量允许您衡量和监控数据的质量，从而做出正确的业务决策。 AWS Glue Data Quality 建立在开源 DeeQu 框架之上，可提供托管的无服务器体验。 AWS Glue 数据质量与数据质量定义语言 (DQDL) 配合使用，这是一种用于定义数据质量规则的特定领域语言。要了解有关 DQDL 和支持的规则类型的更多信息，请参阅 数据质量定义语言（DQDL）引用。

有关产品详细信息和定价，请参阅 AWS Glue Data Quality 服务页面。

优点和主要功能

AWS Glue 数据质量的优势和主要特点包括：

- 无服务器 — 无需安装、修补或维护。
- 快速入门 — AWS Glue 数据质量可快速分析您的数据并为您创建数据质量规则。只需点击两下即可开始：“创建数据质量规则 → 推荐规则”。
- 检测数据质量问题-使用机器学习 (ML) 检测异常和 hard-to-detect 数据质量问题。
- 即兴制定规则 — 从 25 多条 out-of-the-box DQ 规则开始，您可以创建适合自己特定需求的规则。
- 评估质量并做出自信的业务决策 — 评估规则后，您将获得一个数据质量分数，该分数可以概述数据的运行状况。使用数据质量分数做出自信的业务决策。
- 聚焦不良数据 — AWS Glue 数据质量可帮助您识别导致质量分数下降的确切记录。轻松识别它们，对其进行隔离和修复。
- 即用@@ 即付 — 无需年度许可证即可使用 AWS Glue Data Quality。
- 无锁定 — AWS Glue 数据质量建立在开源之上 DeeQu，允许您保留以开放语言编写的规则。
- 数据质量检查 — AWS Glue 数据质量您可以对 ETL 管道Data Catalog和 AWS Glue ETL 管道进行数据质量检查，从而管理静态和传输中的数据质量。
- 基于 ML 的数据质量检测-使用机器学习 (ML) 检测异常和 hard-to-detect 数据质量问题。

工作方式

AWS Glue 数据质量有两个切入点： AWS Glue Data Catalog 和 AWS Glue ETL 作业。本节概述了每个入口点支持的用例和 AWS Glue 功能。

的数据质量 AWS Glue Data Catalog

AWS Glue 数据质量评估存储在中的对象。 AWS Glue Data Catalog 它为非编码人员提供了一种设置数据质量规则的简便方法。这些角色包括数据管理员和业务分析师。

您可以为以下用例选择此选项：

- 您想对已在 AWS Glue Data Catalog中编目的数据集执行数据质量任务。
- 您从事数据治理工作，需要持续识别或评估数据湖中的数据质量问题。

您可以使用以下界面管理 Data Catalog 的数据质量：

- AWS Glue 管理控制台
- AWS Glue API

要开始使用 “ AWS Glue 数据质量”，请参 AWS Glue Data Catalog 阅AWS Glue Data Quality for the Data Catalog 入门。

AWS Glue ETL 作业的数据质量

AWS Glue AWS Glue ETL 作业的数据质量允许您主动执行数据质量任务。主动任务可帮助您在将数据集加载到数据湖之前识别并筛选出不良数据。

视频：介绍 ETL 管道 AWS Glue 的数据质量

您可以针对以下用例为 ETL 作业选择数据质量：

- 您想将数据质量任务整合到您的 ETL 作业中
- 您想编写在 ETL 脚本中定义数据质量任务的代码
- 您想管理可视化数据管道中流出的数据的质量

您可以使用以下界面管理适用于 ETL 作业的数据质量：

- AWS Glue Studio、 AWS Glue Studio 笔记本和 AWS Glue 交互式会话
- AWS Glue 用于 ETL 脚本的库
- AWS Glue API

要开始了解适用于 ETL 作业的数据质量，请参阅《AWS Glue Studio 用户指南》中的 Tutorial: Getting started with Data Quality。

将 Data Catalog 的数据质量与适用于 ETL 作业的数据质量进行比较

下表概述了每个 AWS Glue 数据质量入口点支持的功能。

功能	Data Catalog 的数据质量	ETL 作业的数据质量
数据来源	Amazon S3、Amazon Redshift、与 Data Catalog 兼容的 JDBC 源以及交易数据湖格式，例如 Apache Iceberg、Apache Hudi 和 Delta Lake。请注意，如果 AWS Lake Formation 管理表，则不支持 Iceberg、Delta 和 HUDI 表。 Amazon Athena 不支持编入目录 AWS Glue Data Catalog 的视图。	支持的所有数据源 AWS Glue，包括自定义连接器和第三方连接器。
数据质量规则建议	支持	不支持
编写并运行 DQDL 规则	支持	支持
自动扩缩	不支持	支持
AWS Glue Flex 支持	不支持	支持
调度	在评估 Data Quality 规则时和通过 Step Functions 时支持。	使用 Step Functions 和工作流程时支持。
识别未通过数据质量检查的记录	不支持	支持
与 Amazon EventBridge 集成	支持	支持
与 AWS 云监视集成	支持	支持

功能	Data Catalog 的数据质量	ETL 作业的数据质量
将数据质量结果写入 Amazon S3	支持	支持
增量数据质量	通过下推谓词支持	通过 AWS Glue 书签支持
AWS CloudFormation 支持	支持	支持
基于 ML 的异常检测	不支持	预览
动态规则	不支持	支持

注意事项

在使用 “ AWS Glue 数据质量” 之前，请考虑以下事项：

- 数据质量规则无法评估嵌套或列表类型的数据源。请参阅 扁平化嵌套结构。

术语

以下列表定义了与 AWS Glue 数据质量相关的术语。

数据质量定义语言（DQDL）

一种特定于域的语言，可用于编写 AWS Glue 数据质量规则。

要了解有关 DQDL 的更多信息，请参阅数据质量定义语言（DQDL）引用指南。

数据质量

描述数据集如何发挥其特定用途。 AWS Glue 数据质量根据数据集评估规则以衡量数据质量。每条规则都检查特定特征，例如数据新鲜度或完整性。要量化数据质量，可以使用数据质量分数。

数据质量分数

在评估具有数据质量的规则集时，通过（结果为真） AWS Glue 的数据质量规则的百分比。

规则

DQDL 表达式，用于检查您的数据是否存在特定特征并返回布尔值。有关更多信息，请参阅 规则结构。

分析器

用于收集数据统计信息的 DQDL 表达式。分析器收集数据统计信息，机器学习算法可以使用这些统计数据来检测一段时间内的异常和 hard-to-detect 数据质量问题。

规则集

一种包含一组数据质量规则的 AWS Glue 资源。规则集必须与 AWS Glue Data Catalog中的一个表格关联。保存规则集时， AWS Glue 会为规则集分配一个 Amazon 资源名称(ARN)。

数据质量分数

当您使用 AWS Glue 数据质量评估规则集时，通过（结果为真）的数据质量规则的百分比。

观测值

AWS Glue 通过分析一段时间内从规则和分析器收集的数据统计信息得出的未经证实的见解。

限制

AWS Glue 数据质量服务限制：

- 一个规则集中可以有 2000 条规则。如果您的规则集较大，我们建议将其拆分为多个规则集。
- 规则集的大小为 65KB。如果您的规则集较大，我们建议将其拆分为多个规则集。

AWS Glue 数据质量发布说明

本主题介绍了《 AWS Glue 数据质量》中引入的功能。

正式上市：新功能

随着 “ AWS Glue 数据质量” 的正式上市，将提供以下新功能：

- 现在支持识别哪些记录未通过数据质量检查的功能 AWS Glue Studio
- 新的数据质量规则类型，例如验证两个数据集之间数据的引用完整性、比较两个数据集之间的数据以及数据类型检查
- 改善了中的用户体验 AWS Glue Data Catalog
- 支持 Apache Iceberg、Apache Hudi 和 Delta Lake
- 支持 Amazon Redshift

- 通过 Amazon 简化通知 EventBridge
- AWS CloudFormation 支持创建规则集
- 性能改进：ETL 中的缓存选项以及 AWS Glue Studio 评估数据质量时更快的性能

2023 年 11 月 27 日（预览版）

- 由 ML 提供支持的异常检测功能现已在 AWS Glue ETL 和 AWS Glue Studio 中提供。有了这个，您现在可以检测异常和 hard-to-detect 数据质量问题。
- 动态规则允许您提供动态阈值（例如：`RowCount> avg(last(10))`）。

2024 年 3 月 12 日

- DQDL 改进
  - 支持 NULL、BLANKS、WHITESPACES_ONLY 等关键字
  - 用于指定 AWS Glue 数据质量必须如何处理复合规则的选项
  - ColumnValues 规则类型不允许在比较期间传递 NULL 值
  - 在 DQDL 中支持 NOT 运算符

2024年6月26日

- DQDL 改进
  - DQDL 现在支持 where 子句，这样你就可以在应用 DQ 规则之前筛选数据

AWS Glue 数据质量自动监测功能中的异常检测

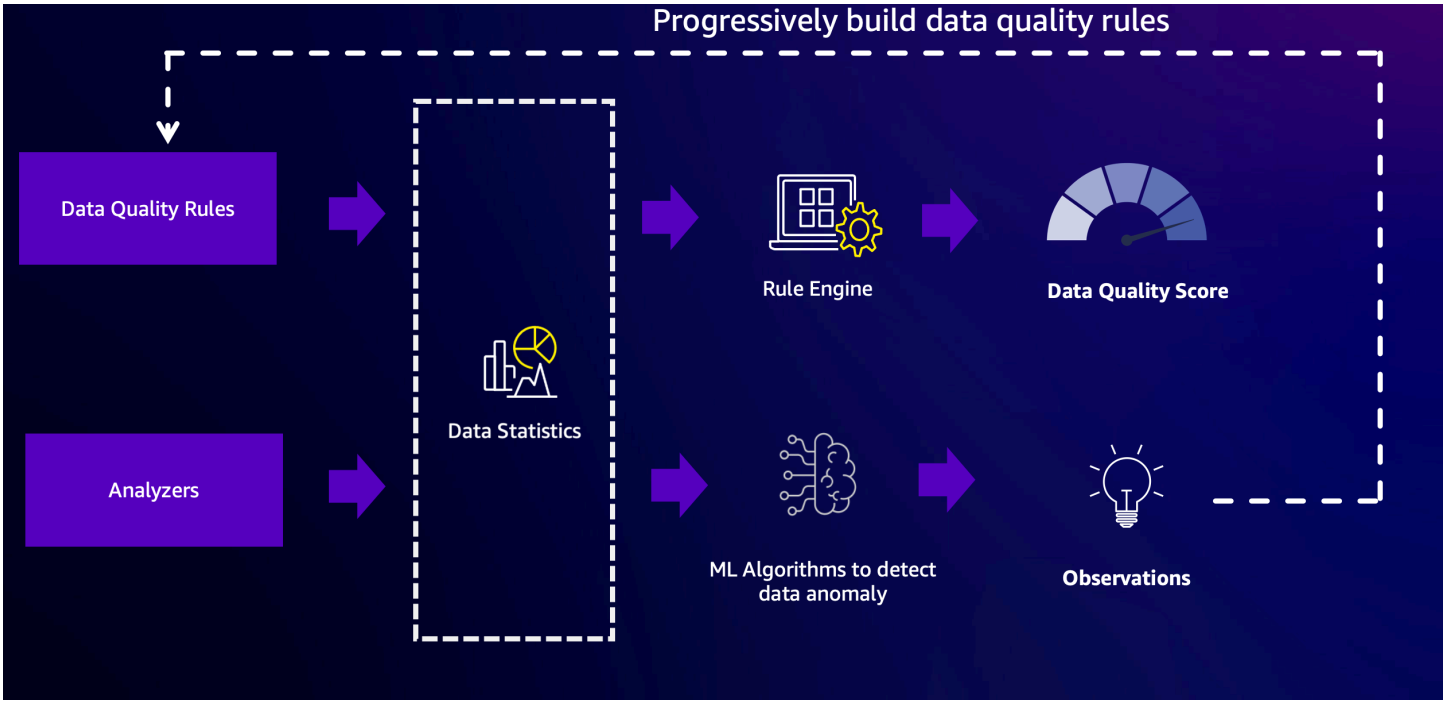
> **Note**
>
> 在以下区域中提供 AWS Glue Data Quality 预览版：
>
> - 美国东部（俄亥俄州、北弗吉尼亚州）
> - 美国西部（俄勒冈）
> - 亚太地区（东京）
> - 欧洲地区（爱尔兰）

AWS Glue 数据质量异常检测将机器学习（ML）算法应用于一段时间内的数据统计信息，以检测难以通过规则检测到的异常模式和隐藏的数据质量问题。目前，异常检测仅适用于 AWS Glue 4.0。此功能目前仅在 AWS Glue Studio Visual ETL 和 AWS Glue ETL 中提供。此功能不适用于 AWS Glue Studio 笔记本、AWS Glue Data Catalog、AWS Glue 交互式会话和 AWS Glue 数据预览。

工作方式

在评估数据质量规则时，AWS Glue 会捕获确定数据是否符合规则所需的数据统计信息。例如，数据质量自动监测功能将计算数据集中不同值的数量，然后将该值与预期值进行比较。

数据质量规则引擎将统计值与定义的阈值进行比较，然后评估您的质量要求。由于这些统计信息是随着时间的推移而收集的，因此您可以在 ETL 管道上启用异常检测，让 AWS Glue 从过去的统计信息中学习，并将隐藏的模式报告为观测值。观测值是 AWS Glue ML 算法所识别的未经证实的见解。其附带了推荐的数据质量规则，您可以将这些规则应用于自己的规则集，以监测发现的模式。我们建议定期运行作业（例如，每小时和每天）。不规律的运行可能会产生糟糕的见解。



使用分析器检查您的数据

有时，您可能没有时间制定数据质量规则。这就是分析器派上用场的地方。分析器是您规则集的一部分，配置非常简单。例如，您可以在自己的规则集中编写以下内容：

```
Analyzers = [
      RowCount,
```

```
    Completeness “AllColumns”
]
```

这将收集以下统计信息：

- 整个数据集的行数
- 数据集中每列的完整性

我们建议使用分析器，因为这样您就不必担心阈值问题。您可以运行数据管道，运行三次后，AWS Glue 数据质量自动监测功能将在发现任何异常时开始生成观测值和规则建议。您可以查看观测值和相关统计信息，并可以轻松地将规则建议纳入规则集。要了解其用法，请参阅 配置异常检测并生成见解 。请注意，分析器不会影响数据质量分数。其生成的统计信息可以随着时间的推移进行分析以生成观测值。

使用 DetectAnomaly 规则

有时，您希望您的作业在检测到异常时失败。要强制执行约束，必须配置一条规则。分析器无法阻止作业。相反，其将收集统计信息并分析数据。在规则集的规则部分配置 `DetectAnomaly` 规则将确认 DQ 扫描报告作业未能通过扫描中的所有规则。

异常检测的好处和用例

工程师可以在任何给定时间，管理数百个数据管道。每个管道都可以从不同的来源提取数据，并将其加载到数据湖中。由于每个管道都可能从不同的来源提取数据，并将其加载到数据湖中，因此很难立即获得有关数据的反馈：无论是其形状发生了重大变化，还是与现有趋势背道而驰。

过去，上游数据源在没有向数据工程团队发出警告的情况下发生了变化，从而在这一过程中引入了 hard-to-track “数据错误”。通过向作业添加数据质量节点，这使生活变得更加轻松，因为当发现问题时，作业就会失败。但是，这并不能消除数据团队担心的所有故障模式，这为其他数据错误的出现敞开了大门。

一种故障模式是围绕数据量。由于公司的数据存储随着时间的推移而增长，数据管道生成的记录数量可能会呈指数级增长。每周，数据团队可能需要手动更新 ETL 作业，以增加每条设置采集行数限制的数据质量规则。

另一种故障模式是，某些数据质量规则的限制非常宽泛，以适应交易量因一周中的某一天而异的事实。周末几乎没有交易，而周一的交易量约为其他工作日的三倍。数据团队有两种选择：实施逻辑以根据日期动态更改规则集，或设置非常广泛的期望。

最后，数据团队还关注定义不太明确的数据错误。模型已经根据具有特定特征的数据进行了训练，如果这些数据开始以意想不到的方式出现偏差，团队希望得到通知。例如，2 月份，一家公司可能会扩展到蒙大拿州，因此开头包含“MT”代码的交易会更频繁地出现。这可能会破 ML 推断，因此，模型错误地预测了蒙大拿州的每笔交易都是欺诈性的。

这就是数据质量异常检测可以帮助解决这些问题的地方。数据质量异常检测的一些好处包括：

- 按计划、事件驱动或手动扫描数据。
- 检测可能表明意外事件、季节性或统计异常的异常。
- 提供规则建议，以便对数据质量异常检测发现的观测值采取措施。

这在以下情况下时很有用：

- 您想要自动检测数据中的异常情况，而无需写入数据质量规则。
- 您想要捕捉数据中仅靠数据质量规则无法发现的潜在问题。
- 您想要自动执行一些随时间推移而演变的任务，例如限制为监测数据质量而提取的行数。

配置 AWS Glue Data Quality的 IAM 权限

本主题提供的信息可帮助您了解 IAM 管理员了解可以在 AWS Glue Data Quality 的 AWS Identity and Access Management（IAM）policy中使用的操作和资源。它还包括 IAM policy 示例，其中包含在 AWS Glue Data Catalog 中使用 AWS Glue 数据质量所需的最低权限。

有关 AWS Glue 的其他安全信息，请参阅 AWS Glue 中的安全性。

AWS Glue 数据质量的 IAM 权限

下表列出了用户执行特定 AWS Glue Data Quality 操作所需的权限。要为 AWS Glue Data Quality 设置精细授权，您可以在 IAM policy 语句的 `Action` 元素中指定这些操作。

AWS Glue Data Quality 操作

操作	描述	资源类型
`glue:CreateDataQualityRuleset`	授予权限以创建数据质量规则集。	`::dataQualityRuleset/<name>`

操作	描述	资源类型
glue:DeleteDataQualityRuleset	授予权限以删除数据质量规则集。	::dataQualityRuleset/<name>
glue:GetDataQualityRuleset	授予权限以检索数据质量规则集。	::dataQualityRuleset/<name>
glue:ListDataQualityRulesets	授予权限以检索所有数据质量规则集。	::dataQualityRuleset/*
glue:UpdateDataQualityRuleset	授予权限以更新数据质量规则集。	::dataQualityRuleset/<name>
glue:GetDataQualityResult	授予权限以检索数据质量任务运行结果。	::dataQualityRuleset/<name>
glue:ListDataQualityResults	授予权限以检索所有数据质量任务运行结果。	::dataQualityRuleset/*
glue:CancelDataQualityRuleRecommendationRun	授予权限以停止正在运行的数据质量建议任务运行。	::dataQualityRuleset/*
glue:GetDataQualityRuleRecommendationRun	授予权限以检索数据质量建议任务运行。	::dataQualityRuleset/*
glue:ListDataQualityRuleRecommendationRuns	授予权限以检索所有数据质量建议任务运行。	::dataQualityRuleset/*
glue:StartDataQualityRuleRecommendationRun	授予权限以开始数据质量规则建议任务运行。	::dataQualityRuleset/*
glue:CancelDataQualityRulesetEvaluationRun	授予权限以停止正在运行的数据质量任务运行。	::dataQualityRuleset/*

操作	描述	资源类型
glue:GetDataQualityRulesetEvaluationRun	授予权限以检索数据质量任务运行结果。	::dataQualityRuleset/*
glue:ListDataQualityRulesetEvaluationRuns	授予权限以检索所有数据质量任务运行。	::dataQualityRuleset/*
glue:StartDataQualityRulesetEvaluationRun	授予权限以开始数据质量规则任务运行。	::dataQualityRuleset/<name>
glue:PublishDataQuality	授予权限以发布数据质量结果。	::dataQualityRuleset/<name>

计划评估运行需要 IAM 设置

IAM 权限

要运行计划的数据质量评估运行，必须将 IAM:PassRole 操作添加到权限策略中。

AWS EventBridge 调度器所需的权限

操作	描述	资源类型
iam:PassRole	向 IAM 授予权限，允许用户传递已批准的角色。	用于调用 StartDataQualityRulesetEvaluationRun 的角色的 ARN

如果没有这些权限，则会发生以下错误：

```
"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 可信实体

AWS Glue and AWS EventBridge 调度器服务需要在可信实体中列出，才能创建和运行计划的 `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"
        }
    ]
}
```

示例 IAM policies

AWS Glue 数据质量的 IAM 角色需要以下类型的权限：

- AWS Glue Data Quality 操作权限，以便您获得推荐的数据质量规则，并针对 AWS Glue Data Catalog 中的表运行数据质量任务。本节中的 IAM policy 示例包括 AWS Glue 数据质量操作所需的最低权限。
- 授予访问 Data Catalog 表和基础数据的权限。这些权限因使用案例而异。例如，对于在 Amazon S3 中编录的数据，权限应包括对 Amazon S3 的访问。

> **Note**
>
> 除了本节中描述的权限之外，还必须配置 Amazon S3 权限。

获取推荐数据质量规则的最低权限

此示例策略包括生成推荐的数据质量规则所需的权限。

```
{
  "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": "*"
    },
  ]
}
```

运行数据质量任务所需的最低权限。

此示例策略包括运行数据质量评估任务所需的权限。

根据使用案例可选择以下策略声明：

- `AllowCloudWatchPutMetricDataToPublishTaskMetrics`：如果要将数据质量运行指标发布到 Amazon CloudWatch，必须选择此项。
- `AllowS3PutObjectToWriteTaskResults`：如果要将数据质量运行结果写入 Amazon S3，必须选择此项。

```
{
  "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>/*"
    }
  ]
}
```

AWS Glue Data Quality for the Data Catalog 入门

此 设置开始部分提供的说明有助于您在 AWS Glue 控制台上开始使用 AWS Glue Data Quality。您将学习如何完成基本任务，例如生成数据质量规则建议和根据您的数据评估规则集。

主题

先决条件

在 AWS Glue Data Quality 使用之前，您应熟悉在 AWS Glue 中使用 Data Catalog 和爬网程序。使用 AWS Glue Data Quality，您可以评估 Data Catalog 数据库中表的质量。您需要以下项目：

- Data Catalog 中的一个表，用于评估您的数据质量规则集。
- AWS Glue 的 IAM 角色，在生成规则建议或运行数据质量任务时提供。此角色必须具有运行各种 AWS Glue Data Quality 进程所需的资源的访问权限。这些资源包括 AWS Glue、Amazon S3 和 CloudWatch。要查看包含 AWS Glue Data Quality 最低权限的策略示例，请参阅 示例 IAM policies。

  要了解有关 AWS Glue 的 IAM 角色的更多信息，请参阅 Create an IAM policy for the AWS Glue service 和 Create an IAM role for the AWS Glue service。您还可以查看 Authorization for AWS Glue Data Quality actions 上特定于数据质量的所有 AWS Glue 权限的列表。
- 至少有一个包含各种数据的表的数据库。本教程中使用的表名为 `yyz-tickets`，包含表 `tickets`。该数据是多伦多市政府提供的用于停车收费的公开信息的集合。如果您创建自己的表，请确保表中填充了各种有效数据，以获得最佳的推荐规则集。

分步示例

有关示例数据集的分步示例，请参阅 AWSGlue Data Quality 博客文章。

生成规则建议

规则建议使您无需编写代码即可轻松开始提高数据质量。使用 AWS Glue Data Quality，您可以分析数据、确定规则并创建规则集，您可以在数据质量任务中对其进行评估。建议运行在 90 天后被自动删除。

生成数据质量规则建议

1. 打开 AWS Glue 控制台，网址为 https://console.aws.amazon.com/glue/。
2. 在导航窗格中，选择 Tables（表）。然后选择要为其生成数据质量规则建议的表。
3. 在表格详细信息页面上，选择数据质量选项卡以访问表的 AWS Glue Data Quality 规则和设置。
4. 在数据质量选项卡上，选择添加规则并监控数据质量。
5. 在规则集生成器页面上，如果没有规则建议运行，页面顶部的警报将提示您启动建议任务。
6. 选择推荐规则以打开模态并输入建议任务的参数。
7. 选择一个有权访问 AWS Glue 的 IAM 角色。此角色必须具有运行各种 AWS Glue Data Quality 进程所需的资源的访问权限。
8. 根据您的偏好填写字段后，选择推荐规则以开始运行建议任务。如果建议运行正在进行或已完成，则可以在此警报中管理您的运行。您可能需要刷新警报才能查看状态更改。已完成和正在进行的建议任务运行显示在运行历史记录页面中，该页面列出了过去 90 天的所有建议运行。

推荐的规则是什么意思

AWS Glue Data Quality 根据输入表中每列的数据生成规则。它使用这些规则来确定可以筛选数据以保持质量要求的潜在界限。以下生成的规则列表包括一些示例，这些示例有助于理解这些规则的含义以及它们在应用于您的数据时可能做什么。

有关生成的数据质量定义语言（DQDL）规则类型的完整列表，请参阅 DQDL rule type reference。

- `IsComplete "SET_FINE_AMOUNT"` - `IsComplete` 规则验证是否为任何给定行填写了该列。使用此规则在数据中将列标记为非可选列。
- `Uniqueness "TICKET_NUMBER" > 0.95` - `Uniqueness` 规则验证列中的数据是否满足某个唯一性阈值。在本例中，确定为 `"TICKET_NUMBER"` 填充任何给定行的数据的内容与所有其他行的内容最多 95% 相同，这表明了这一规则。
- `ColumnValues "PROVINCE" in ["ON", "QC", "AB", "NY",...]` - `ColumnValues` 规则根据现有列内容定义列的有效值。在本例中，每行的数据是州或省的 2 个字母的牌照。

- `ColumnLength "INFRACTION_DESCRIPTION" between 15 and 31` - `ColumnLength` 规则对列的数据强制执行长度限制。此规则是根据样本数据生成的，该数据基于一列字符串的最小和最大记录长度。

监控规则建议

运行数据质量规则建议时，添加规则并监控数据质量页面会在顶部栏中显示您可以执行的信息和其他操作。

当规则建议正在进行时，您可以在建议任务完成之前选择停止运行。任务正在进行时，您将看到运行的状态进行中以及运行开始的日期和时间。

规则建议完成后，规则建议栏会显示建议的规则数量、上次建议运行的状态以及完成的日期和时间戳。

您可以通过选择插入规则建议来添加推荐的规则。要查看之前推荐的规则，请选择一个特定的日期。要运行新的建议，请选择更多操作，然后选择推荐的规则。

通过选择管理用户设置来设置默认设置。您可以为 Amazon S3 设置默认路径，来存储规则集或设置运行 Data Catalog 的默认角色。

编辑推荐的规则集

由于 AWS Glue Data Quality 会根据您现有的可用数据生成规则，因此您可能会在自动建议中看到一些意想不到或不想要的规则。为了充分利用推荐的规则集，您需要对其进行评估和修改。在本教程的这一步中，您将采用上一步中生成的规则并对其进行调整，以对某些数据强制执行更严格的质量。您还可以放宽其他规则，以确保以后可以添加正确、唯一的数据。

编辑建议的规则集

1. 在 AWS Glue 控制台中的导航窗格中，选择 Data Catalog，然后选择数据库表。选择 `tickets` 表。
2. 在表格详细信息页面上，选择数据质量选项卡以访问表的 AWS Glue Data Quality 规则和设置。
3. 在规则集部分，选择在 生成规则建议 中生成的规则集。
4. 选择操作，然后在控制台窗口中选择编辑。规则集编辑器加载到控制台中。它包括规则的编辑窗格和 DQDL 的快速参考。
5. 删除脚本中的 2 行。这放宽了将数据库大小限制在一定行数之内的要求。编辑完成后，您的文件应在第 1-3 行包含以下内容：

```
Rules = [
```

```
    IsComplete "TAG_NUMBER_MASKED",
    ColumnLength "TAG_NUMBER_MASKED" between 6 and 9,
```

6. 删除脚本中的 25 行。这放宽了记录在案的省份中占 96% 的要求为 0N。编辑完成后，您的文件应包含从行 24 到规则集末尾的以下内容：

```
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. 将行14 更改为以下内容：

```
IsComplete "TIME_OF_INFRACTION",
```

通过将数据库限制为仅包含记录的违规时间的工单，这加强了对列的要求。在此数据集中，您应始终将没有记录违规时间的工单视为无效数据。这与分区或转换可能更适合进一步使用或检查数据以确定质量规则的情况不同。

8. 选择控制台页面底部的更新规则集。

创建新规则集

规则集是一组数据质量规则，您可以根据数据进行评估。在 AWS Glue 控制台中，您可以使用数据质量定义语言（DQDL）创作自定义规则集。

创建数据质量规则集

1. 在 AWS Glue 控制台中的导航窗格中，选择 Data Catalog，选择数据库，然后选择表。选择表 `tickets`。
2. 打开 Data quality（数据质量）选项卡。
3. 在规则集部分中，选择创建规则集。DQDL 编辑器将在控制台中启动。它有一个用于直接编辑的文本区域，还有用于 DQDL 规则和表架构的快速参考。
4. 开始向 DQDL 编辑器的文本区域添加规则。您可以直接从本教程中编写规则，也可以使用数据质量规则编辑器的 DQDL 规则生成器功能。

> **Note**
>
> 如何使用 DQDL 规则生成器
>
> 1. 从列表中选择规则类型，然后选择加号将示例语法插入编辑器窗格。
> 2. 将占位符列名与您自己的列名交换。表中的列名可在架构选项卡中找到。
> 3. 根据需要更新表达式参数。有关 DQDL 支持的表达式的完整列表，请参阅Expressions。

例如，以下规则是对 `tickets` 表中 `ticket_number` 列进行数据验证的约束。要添加以下规则，请使用 DQDL 规则生成器或直接编辑您的规则集：

```
IsComplete "ticket_number",
IsUnique "ticket_number",
ColumnValues "ticket_number" > 9000000000
```

5. 在规则集名称字段中为您的新规则集提供一个名称。
6. 选择保存规则集。

评估多个数据集的数据质量

您可以使用 ReferentialIntegrity 和 DatasetMatch 规则集跨多个数据集设置数据质量规则。ReferentialIntegrity 会检查主数据集中的数据是否存在于其他数据集中。

要添加引用数据集，请选择架构选项卡，然后选择更新引用表。系统将提示您选择数据库和表。您可以添加表，然后设置数据质量规则。AggregateMatch、RowCountMatch、ReferentialIntegrity、SchemaMatch 和 DatasetMatch 等规则类型支持对多个数据集执行数据质量检查的功能。

运行规则集以评估数据质量

当您运行数据质量任务时，AWS Glue Data Quality 会根据您的数据评估规则集并计算数据质量分数。此分数代表通过输入的数据质量规则的百分比。

运行数据质量任务

1. 在 AWS Glue 控制台中的导航窗格中，选择 Data Catalog，选择数据库，然后选择表。选择表 `tickets`。
2. 选择数据质量选项卡。
3. 在规则集列表中，选择要对照表评估的规则集。在此步骤中，我们建议使用您已经编写或修改过的规则集，而不是生成的规则。选择运行。
4. 在模态中，选择您的 IAM 角色。此角色必须具有运行各种 AWS Glue Data Quality 进程所需的资源的访问权限。您可以将 IAM 角色保存为默认角色，也可以前往默认设置页面对其进行修改。
5. 在 Data quality actions（数据质量操作）下，选择是否要 Publish metrics to Amazon CloudWatch（将指标发布到 Amazon CloudWatch）。选择此选项后，AWS Glue Data Quality 会发布指标，指示通过的规则数量和失败的规则数量。要对以这种方式存储的指标采取行动，您可以使用 CloudWatch 警报。系统还会将关键指标发布到 Amazon EventBridge 供您设置警报。有关更多信息，请参阅 Setting up alerts, deployments, and scheduling。
6. 在运行频率中，选择按需运行或计划规则集。计划规则集时，系统会提示您输入任务名称。计划将在 Amazon EventBridge 中创建。您可以在 Amazon EventBridge 中编辑计划。
7. 要在 Amazon S3 中保存数据质量结果，请选择数据质量结果位置。您之前为此任务选择的 IAM 角色必须具有您选择的位置的写入权限。
8. 在其他配置下，输入您希望 AWS Glue 为数据质量任务分配的请求的工作线程数。
9. 您可以选择在数据来源上设置筛选器。这可以帮助减少要读取的数据。您还可以使用筛选器来运行增量验证，方法是选择分区信息并通过 API 调用将其作为参数传递。为了提高性能，您可以提供分区谓词。
10. 选择运行。您应该在 Data quality task runs（数据质量任务运行）列表中看到您的新任务。当任务的运行状态列显示为已完成时，您可以查看质量分数结果。您可能需要刷新控制台窗口才能正确更新状态。
11. 要查看数据质量结果详细信息的列，请选择“+”图标展开规则集。结果显示了在评估中通过和失败的规则，以及触发规则失败的原因。

查看数据质量分数和结果

查看所有已创建的规则集的最新运行情况

1. 在 AWS Glue 控制台中，选择 Tables（表）。然后选择要为其运行数据质量任务的表。
2. 选择数据质量选项卡。

3. 数据质量快照显示了一段时间内运行的总体趋势。默认情况下，系统会显示所有规则集的最近 10 次运行。要按规则集进行筛选，请从下拉列表中选择所需的规则集。如果运行次数少于 10 次，则会显示所有已完成的可用运行。
4. 在数据质量表中，系统显示了每个规则集及其最新运行次数（如果有的话）以及分数。展开规则集会显示该规则集中的规则以及该运行的规则结果。

查看特定规则集的最新运行情况

1. 在 AWS Glue 控制台中，选择 Tables（表）。然后选择要为其运行数据质量任务的表。
2. 选择数据质量选项卡。
3. 在数据质量表中，选择特定的规则集。
4. 在规则集详细信息页面上，选择运行历史记录选项卡。

   该特定规则集的所有评估运行都列在该选项卡的表格中。您可以查看分数历史记录和运行状态。
5. 要查看有关特定运行的更多信息，请选择运行 ID 以转到评估运行详细信息页面。在此页面上，您可以看到有关运行的详细信息以及有关各个规则结果状态的更多详细信息。

相关 主题

- DL 规则类型引用
- 数据质量定义语言（DQDL）引用

使用 AWS Glue Studio 评估数据质量

AWS Glue 数据质量可根据您定义的规则评估和监控您的数据质量。这样可以轻松识别需要操作的数据。在 AWS Glue Studio 中，您可以向可视化作业中添加数据质量节点，以便为数据目录中的表创建数据质量规则。然后，您可以监控和评估数据集随着时间的推移而发生的变化。有关如何在 AWS Glue 中使用 AWS Glue Studio Data Quality 的概述，请观看以下视频。

以下是有关如何操作 AWS Glue 数据质量的总体步骤：

1. Create data quality rules（创建数据质量规则）— 通过选择您配置的内置规则集，使用 DQDL 生成器构建一组数据质量规则。
2. Configure a data quality job（配置数据质量作业）— 根据数据质量结果和输出选项定义操作。

3. 保存并运行数据质量作业 — 创建和运行作业。保存作业将保存您为该作业创建的规则集。
4. Monitor and review the data quality results（监控和查看数据质量结果）— 在作业运行完成后查看数据质量结果。（可选）将作业安排在未来的某个日期运行。

优势

数据分析师、数据工程师和数据科学家可以使用 AWS Glue Studio 中的评估数据质量节点来分析、配置、监控和提高可视化作业编辑器中的数据质量。使用数据质量节点的好处包括：

- 您可以检测数据质量问题 — 您可以通过创建一些规则来检查数据集特征来查看是否存在问题。
- 轻松上手 — 您可以从预先构建的规则和操作开始。
- 紧密集成 — 您可以使用 AWS Glue Studio 中的数据质量节点，因为 AWS Glue Data Quality 是基于 AWS Glue Data Quality 运行的。

评估 AWS Glue Studio 中 ETL 作业的数据质量

在本教程中，您将开始使用 AWS Glue Studio 中的 AWS Glue 数据质量。您将了解如何执行以下操作：

- 使用数据质量定义语言（DQDL）规则构建器创建规则集。
- 指定数据质量操作、要输出的数据以及数据质量结果的输出位置。
- 查看数据质量结果。

要通过示例进行练习，请查看博客文章适用于 ETL 管道的 AWS Glue Data Quality 的入门。

步骤 1：将“评估数据质量”转换节点添加到可视化作业

在此步骤中，将“评估数据质量”节点添加到可视作业编辑器中。

添加数据质量节点

1. 在 AWS Glue Studio 控制台中，从创建作业部分选择源和目标可视化，然后选择创建。
2. 选择要应用数据质量转换的节点。通常，这将是转换节点或数据来源。
3. 选择“+”图标打开左侧的资源面板。然后在搜索栏中搜索评估数据质量，然后从搜索结果中选择评估数据质量。

4. 可视化作业编辑器将显示来自所选节点的评估数据质量转换节点分支。在控制台的右侧，Transform（转换）选项卡自动打开。如果您需要更改父节点，请选择节点属性选项卡，然后从下拉菜单中选择父节点。

   当您选择新的父节点时，将在父节点和 Evaluate Data Quality（评估数据质量）节点之间建立新的连接。移除所有不需要的父节点。只能将一个父节点连接到一个 Evaluate Data Quality（评估数据质量）节点。

5. 评估数据质量转换支持多个父数据集，因此您可以跨多个数据集验证数据质量规则。支持多个数据集的规则包括 ReferentialIntegrity、DatasetMatch、SchemaMatch、RowCountMatch 和 AggregateMatch。

   向“评估数据质量”转换添加多个输入时，需要选择“主要”输入。您的主要输入是要验证数据质量的数据集。所有其他节点或输入都被视为引用。

   您可以使用“评估数据质量”转换来标识未通过数据质量检查的特定记录。我们建议您选择主数据集，因为标记不良记录的新列会添加到主数据集中。

6. 您可以为输入数据来源指定别名。在使用 ReferentialIntegrity 规则时，别名提供了另一种引用输入源的方式。由于只能将一个数据来源指定为主要源，因此您添加的每个额外数据来源都需要一个别名。

   在以下示例中，ReferentialIntegrity 规则通过别名指定输入数据来源，并与主数据来源进行一对一比较。

   ```
   Rules = [
    ReferentialIntegrity “Aliasname.name” = 1
   ]
   ```

步骤 2：使用 DQDL 创建规则

在本步骤中，您将使用 DQDL 来创建规则。在本教程中，您使用完整性规则类型创建单个规则。此规则类型检查列中完整（非空）值与给定表达式的百分比。有关使用 DQDL 的更多信息，请参阅 DQDL。

1. 在转换选项卡中，选择插入按钮添加规则类型。这会将规则类型添加到规则编辑器中，您可以在其中输入规则的参数。

> **Note**
>
> 编辑规则时，请确保规则在方括号内，并确保规则用逗号分隔。例如，完整的规则表达式将如下所示：
>
> ```
> Rules= [
>     Completeness "year">0.8, Completeness "month">0.8
> ]
> ```
>
> 此示例为名为“年”和“月”的列指定了完整性参数。为了让规则通过，这些列的“完成”值必须大于 80%，或者每个相应列的实例中的数据必须超过 80%。

在此示例中，搜索并插入 Completeness（完整性）规则类型。这会将规则类型添加到规则编辑器中。此规则类型具有以下语法：`Completeness <COL_NAME> <EXPRESSION>`。

大多数规则类型都要求您提供表达式作为参数才能创建布尔响应。有关支持的 DQDL 表达式的更多信息，请参阅 DQDL expressions。接下来，您将添加列名。

2. 在 DQDL 规则生成器中，选择架构选项卡。使用搜索栏在输入架构中查找列名。输入架构显示列名和数据类型。
3. 在规则编辑器中，单击规则类型的右侧，在要插入列的位置插入光标。或者，您也可以在规则中输入列的名称。

   例如，在输入架构列表的列列表中，选择该列旁边的插入按钮（在本示例中为年份）。这会将该列添加到规则中。
4. 然后，在规则编辑器中添加一个表达式来评估规则。由于完整性规则类型会根据给定表达式检查列中完整（非空）值的百分比，因此请输入一个表达式（例如 `> 0.8`）。该规则检查该列的完整值（非空值）是否大于 80%。

步骤 3：配置数据质量输出

创建数据质量规则后，您可以选择其他选项来指定数据质量节点输出：

1. 在 Data quality transform output（数据质量转换输出）中，从以下选项中选择：

- 原始数据 — 选择输出原始输入数据。当您选择此选项时，作业中会添加一个新的子节点“rowLevelOutcomes”。该架构与作为输入传递给转换的主数据集的架构相匹配。如果您只想在出现质量问题时传递数据并使作业失败，则此选项非常有用。

  另一个用例是您想要检测未通过数据质量检查的不良记录。要检测不良记录，请选择添加新列以指示数据质量错误选项。此操作将在“rowLevelOutcomes”转换的架构中添加四个新列。

  - DataQualityRulesPass（字符串数组）— 提供一组通过数据质量检查的规则。
  - DataQualityRulesFail（字符串数组）— 提供一组未通过数据质量检查的规则。
  - DataQualityRulesSkip（字符串数组）— 提供已跳过的规则数组。以下规则无法识别错误记录，因为它们是在数据集级别应用的。
    - AggregateMatch
    - ColumnCount
    - ColumnExists
    - ColumnNamesMatchPattern
    - CustomSql
    - RowCount
    - RowCountMatch
    - StandardDeviation
    - 平均值
    - ColumnCorrelation
  - DataQualityEvaluationResult — 在行级别提供“通过”或“失败”状态。请注意：您的总体结果可能为“失败”，但某项记录可能会通过。例如，RowCount 规则可能已失败，但所有其他规则可能都已成功。在这种情况下，此字段状态为“已通过”。

2. 数据质量结果 — 选择输出配置的规则及其通过或失败状态。如果要将结果写入 Amazon S3 或其他数据库，此选项非常有用。
3. 数据质量输出设置（可选）— 选择数据质量输出设置以显示数据质量结果位置字段。然后，选择浏览搜索要设置为数据质量输出目标的 Amazon S3 位置。

第 4 步。配置数据质量操作

您可以使用操作将指标发布到 CloudWatch 或根据特定条件停止作业。只有在创建规则后，操作才可用。当您选择此选项时，同样的指标也会发布到 Amazon EventBridge。您可以使用这些选项来创建通知警报。

- 规则集失败时 — 您可以选择在作业运行时，如果规则集失败该怎么做。如果您希望在数据质量失败时作业失败，请通过选择以下选项之一来选择作业何时失败。默认情况下，此操作处于未选中状态，即使数据质量规则失败，作业也将完成运行。
  - 无 — 如果选择无（默认值），则作业不会失败，即使规则集失败，作业仍会继续运行。
  - 将数据加载到目标后作业失败 — 作业失败且未保存任何数据。要保存结果，请选择用于保存数据质量结果的 Amazon S3 位置。
  - 在不加载到目标数据的情况下作业失败 — 当发生数据质量错误时，此选项会立即使作业失败。它不加载任何数据目标，包括数据质量转换的结果。

步骤 5：查看数据质量结果

运行作业后，选择数据质量选项卡查看数据质量结果。

1. 对于每个运行作业，查看数据质量结果。每个节点显示数据质量状态和状态详细信息。选择节点可查看所有规则和每条规则的状态。
2. 选择下载结果可下载包含有关作业运行和数据质量结果信息的 CSV 文件。
3. 如果您运行了多个带有数据质量结果的作业，则可以按日期和时间范围筛选结果。选择按日期和时间范围筛选以展开筛选窗口。
4. 您可以选择 Relative range（相对范围）或者 Absolute range（绝对范围）。对于绝对范围，请使用日历选择日期并输入开始时间和结束时间的值。完成后，选择应用。

数据质量规则生成器

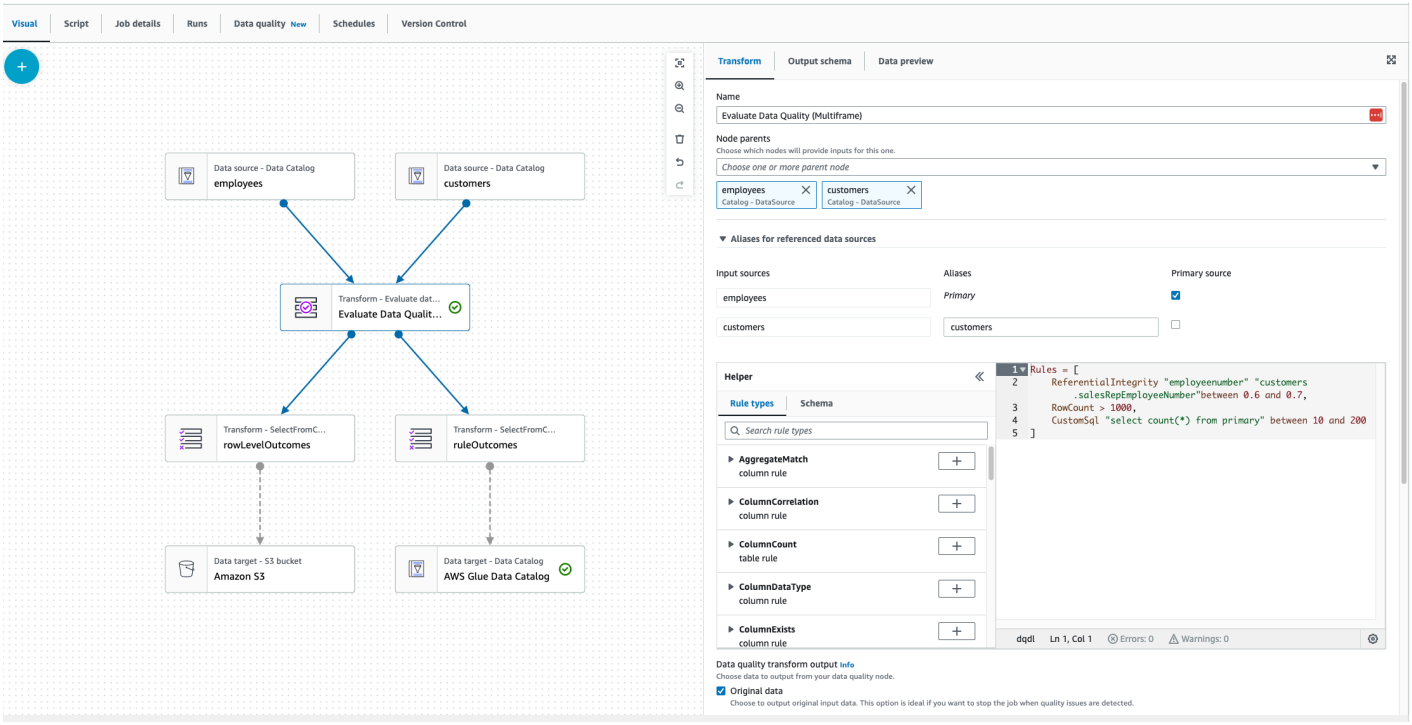
使用数据质量定义语言（DQDL）规则生成器，您可以创建数据质量规则来评估数据。首先选择规则类型，然后在规则编辑器中指定参数。在您创建规则时，规则编辑器还会向您显示任何错误和警告。

DQDL 指南提供了有关如何使用 DQDL 语法、内置规则类型和示例构造规则的综合性文档。

Evaluate Data Quality（评估数据质量）节点

使用评估数据质量转换节点和 DQDL 规则生成器时，可以扩展工作空间。

- 要展开转换选项卡以填满整个屏幕，请选择节点详细信息面板右上角的展开图标。
- 要展开 DQDL 规则编辑器，请选择 << 图标展开规则编辑器并折叠规则类型和架构选项卡。



组件

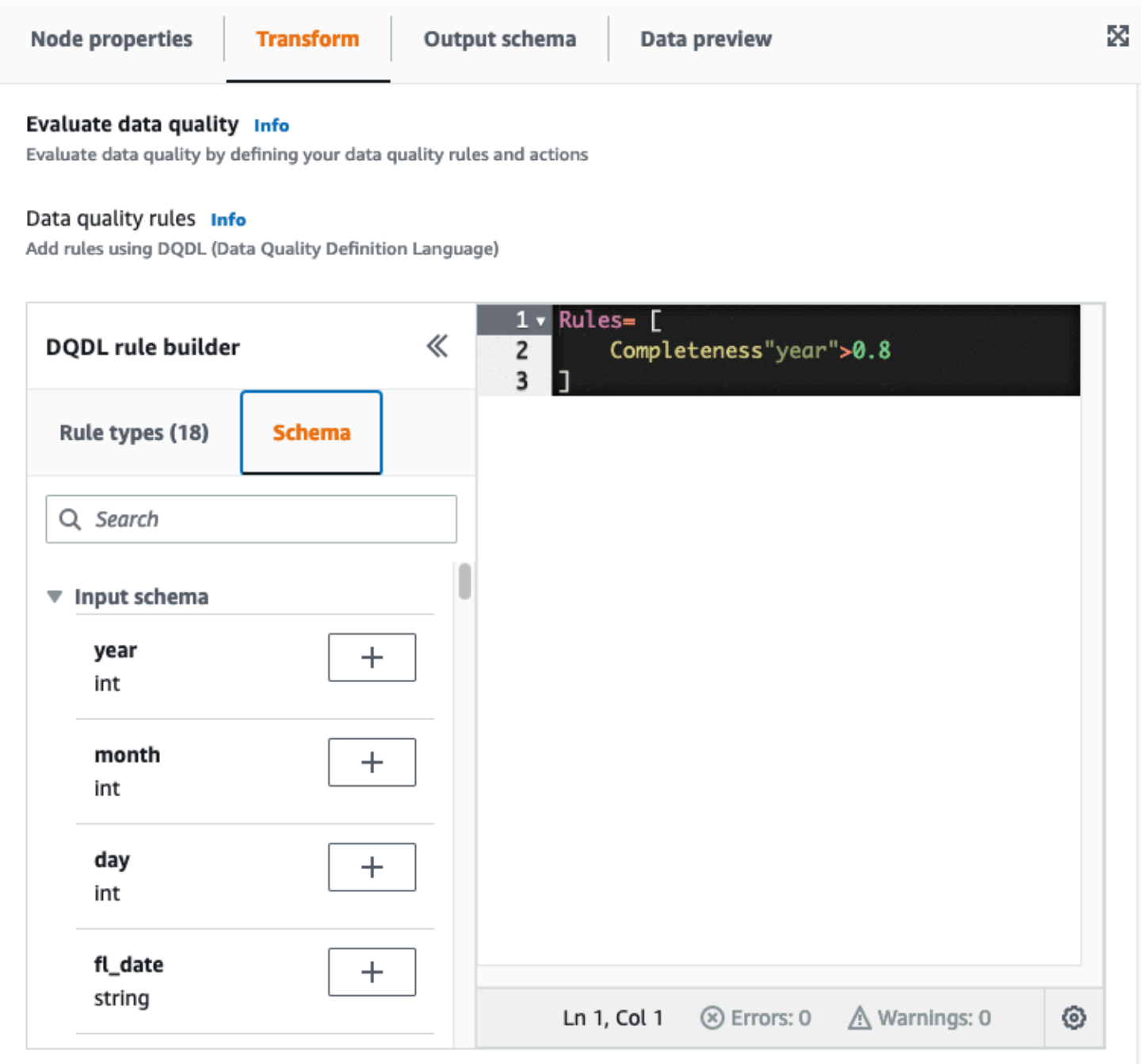
AWS Glue Studio 内置了 26 种规则类型。每种规则类型都有一个描述以及若干个用法示例。

数据质量规则类型

AWS Glue Studio 提供了一些内置规则类型，以便于创建规则。有关规则类型的更多信息，请参阅 DQDL 规则类型参考。

架构

Schema（架构）选项卡显示来自父节点的列名和数据类型。显示来自多个节点的架构。您可以查看输入架构、按列名搜索并将列插入规则编辑器。



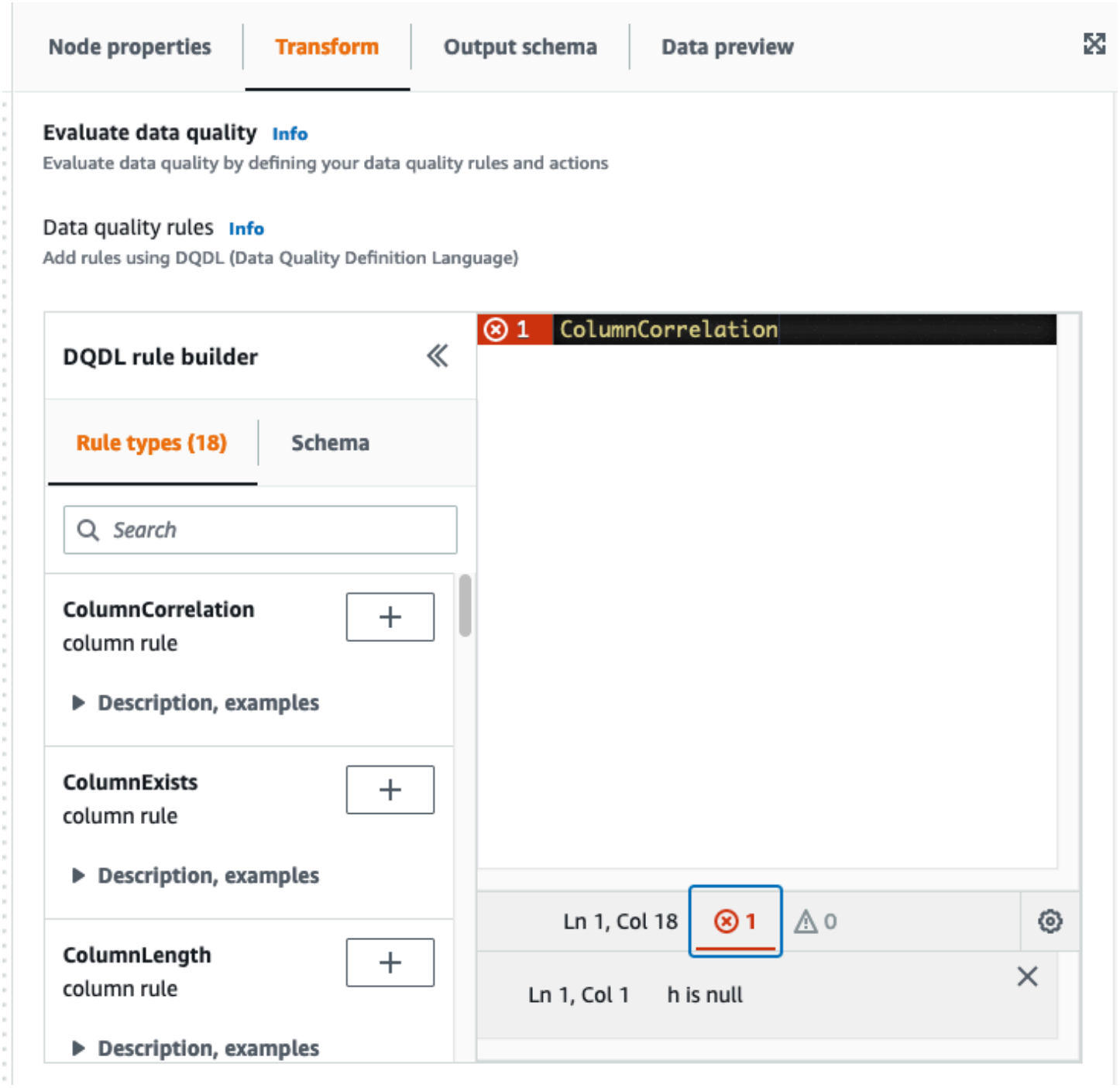
规则编辑器

规则编辑器是一个文本编辑器，您可以在其中编写和编辑规则。如果您从 DQDL 规则生成器中选择规则类型，则该规则类型将添加到规则编辑器中。然后，您可以根据需要通过修改文本来指定参数、添加规则和编辑规则。AWS Glue Studio 验证规则编辑器中的规则并显示错误和警告（如果存在）。

Errors and warnings（错误和警告）

如果规则不遵循 DQDL 规则语法，则规则编辑器将显示几个可视指示符指示存在错误：

- 规则编辑器在出现错误的行上显示错误图标和红色。
- 规则编辑器在红色错误图标旁边显示错误数。
- 当您选择有错误的行时，错误描述和位置（行和列）将显示在规则编辑器的底部。



数据质量操作

默认情况下，此操作处于未选中状态，即使数据质量规则失败，作业也将完成运行。

在以下操作之间进行选择。您可以使用操作将结果发布到 CloudWatch 或根据特定条件停止作业。只有在创建规则后，操作才可用。

- 将结果发布到 CloudWatch — 当您运行作业时，将结果添加到 CloudWatch。
- 数据质量失败时作业失败 — 如果数据质量规则失败，作业也将因此失败。

数据质量转换输出

- 原始数据 — 选择输出原始输入数据。如果您想在检测到质量问题时停止作业，则此选项非常理想。
- 数据质量指标 — 选择输出配置的规则及其通过或失败状态。如果您想执行自定义操作，此选项很有用。

数据质量输出设置

通过将 Amazon S3 位置指定为数据质量输出目标来设置数据质量结果位置。

配置异常检测并生成见解

AWS Glue 数据质量自动监测功能（DQ）根据您编写的数据质量规则评估您的数据，并提供有关数据随时间变化的见解和观测值，以便您可以立即采取行动。由于 DQ 会扫描您的数据，因此 DQ 会计算统计指标，例如行数、最大值或最小值，然后将其与阈值表达式进行比较。

数据质量异常检测的一些好处包括：

- 持续自动扫描数据
- 检测可能表明意外事件或统计异常的异常
- 提供规则建议，以便对数据质量异常检测发现的观测值采取措施

这在以下情况下时很有用：

- 您想要自动检测数据中的异常情况，而无需写入数据质量
- 您想要分析自己的数据并查看数据外观的直观表现
- 您想要跟踪自己的数据如何随着时间的推移而变化

我可以查看有关我的数据的哪些观测值？

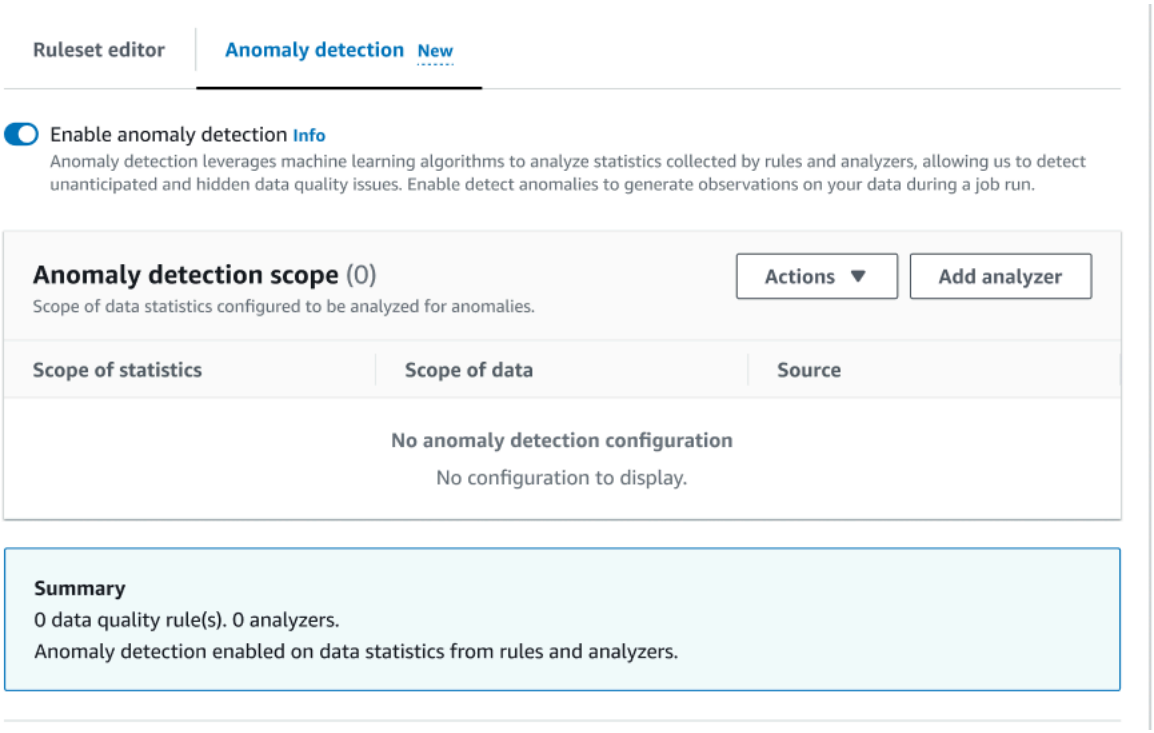
DQ 可识别收集的数据统计信息中的异常值、数据格式变化、数据漂移和架构更改。根据观察，DQ 推荐了用户可以轻松操作的数据质量规则。统计数据包括完整性、唯一性、均值、总和 StandardDeviation、熵和。 DistinctValuesCount UniqueValueRatio

在 AWS Glue Studio 中启用异常检测

要启用异常检测，您可以打开 AWS Glue Studio 并开启“启用异常检测”。启用此功能后，您可以分析一段时间内的数据，并提供有关数据和观测值的数据统计信息，来对您的数据进行异常检测。

在 AWS Glue Studio 中启用异常检测：

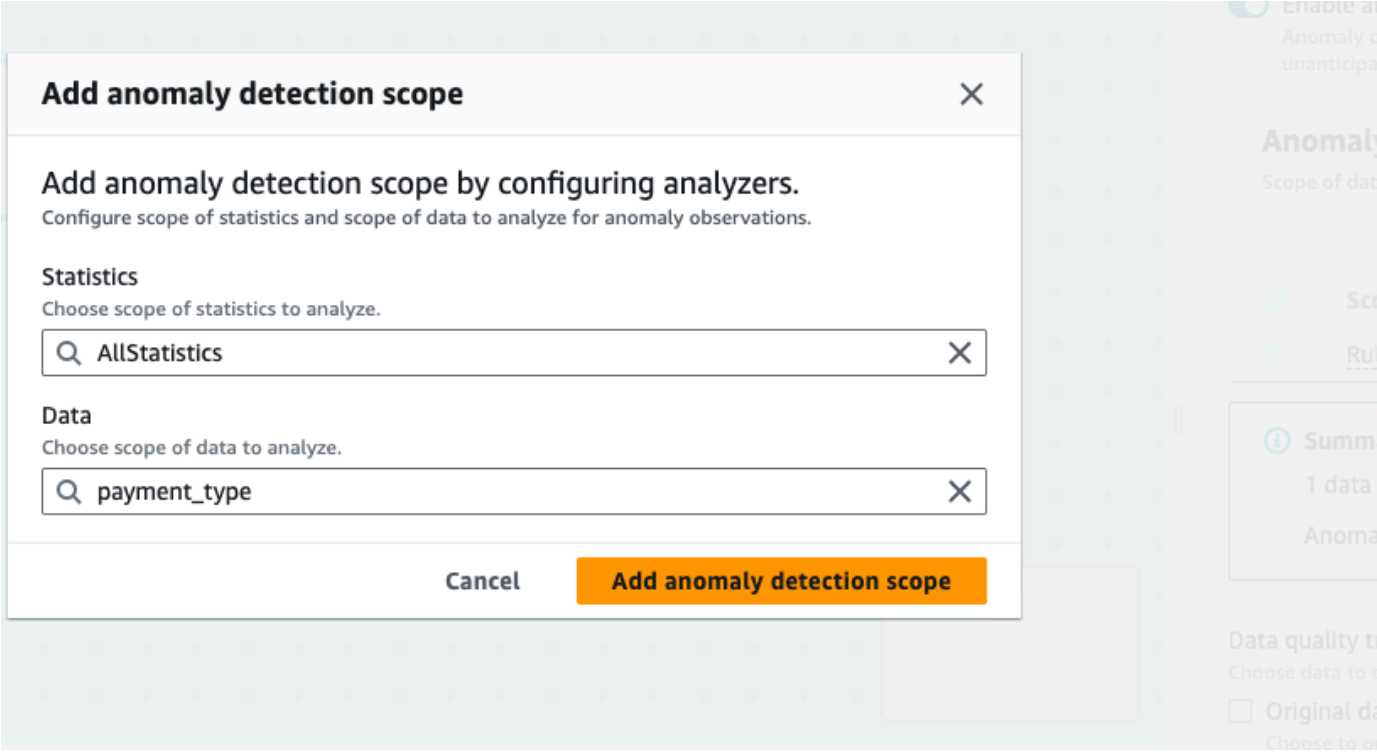
1. 在作业中选择数据质量节点，然后选择异常检测选项卡。开启“启用异常检测”。



2. 通过选择添加分析器来定义要监测异常情况的数据。您可以填充两个字段：“统计信息”和“数据”。

   统计信息是有关数据形状和其他属性的信息。您可以一次选择一个或多个统计信息，也可以选择所有统计信息。统计数据包括：完整性、唯一性、均值、总和 StandardDeviation、熵和。 DistinctValuesCount UniqueValueRatio

   数据是数据集中的列。您可以选择所有列或单个列。



3. 选择添加异常检测范围，以保存您的更改。创建分析器后，可在异常检测范围部分中看到它们。

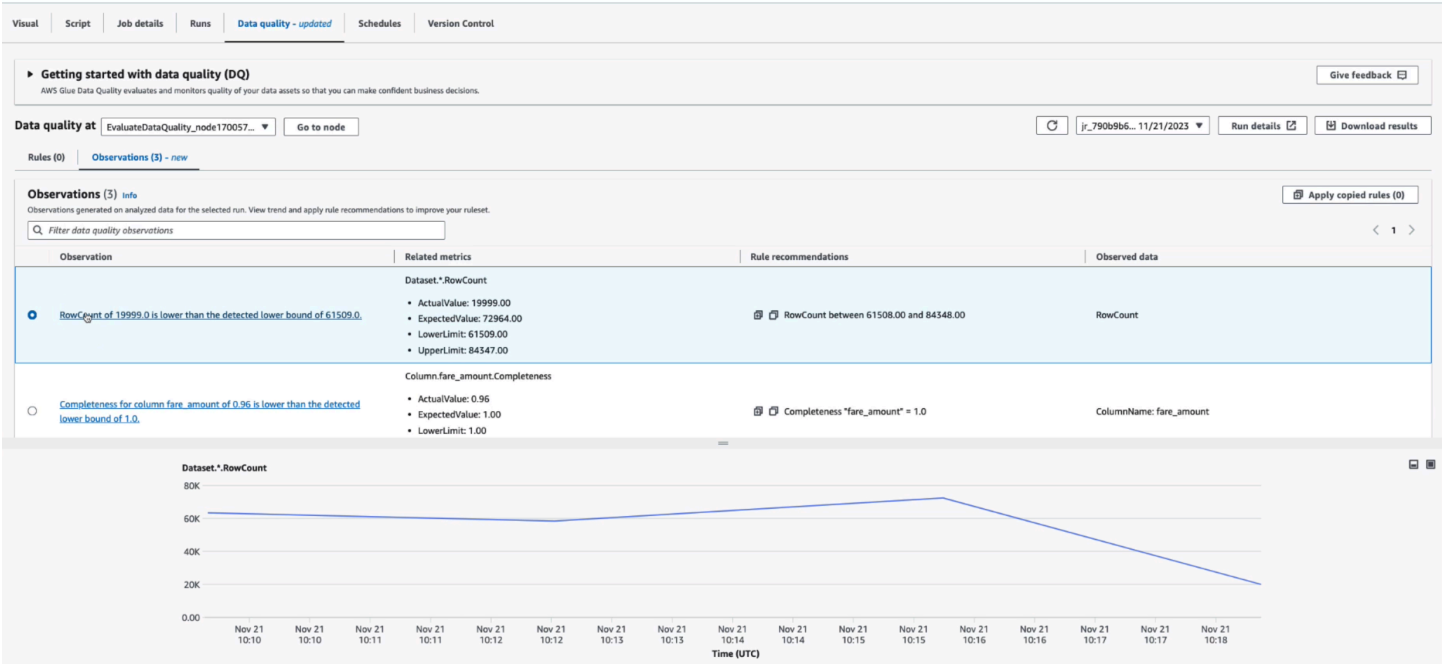
   您也可以使用操作菜单编辑分析器，或者选择规则集编辑器选项卡，直接在规则集编辑器记事本中编辑分析器。您将在自己创建的所有规则下方看到您保存的分析器。

```
Rules = [
]

Analyzers = [
    Completeness “id”
]
```

   借助更新的规则集和分析器，数据质量自动监测功能可以持续监测传入的数据，并根据您的设置通过警报或任务停止发出异常信号。

> **Note**
>
> 当在数据集中观察到每个数据统计信息至少有三个值时，就会生成观测值。如果没有显示观测值，则数据质量自动监测功能没有足够的数据来生成观测值。经过几次作业运行后，数据质量自动监测功能便可提供对您的数据的见解，并在“观测值”部分中显示这些见解。

分析器通过检测数据中的异常生成观测值，并为您提供逐步构建规则的建议。您可以通过选择“数据质量”选项卡查看观测值。观测值特定于每个作业运行。您可以在“观测值”部分的顶部查看特定的数据质量节点和作业运行。选择新的节点或作业运行以查看特定于该节点和作业的观测值。



观测值 - 每个见解都基于由您指定的规则集和分析器配置的特定作业运行。

相关指标 - 生成观测值时，“相关指标”列会显示规则、实际值和预期值以及下限和上限。

规则建议 – AWS Glue 随后还会推荐解决这个问题的规则。通过单击复制图标可以复制每条推荐的规则。您可以通过单击每条规则旁边的复制图标，然后单击应用复制的规则来复制所有推荐的规则。

监测的数据 -“监测的数据”列提供已监测并触发观测值的列或行。

将推荐规则应用于您的数据质量节点

生成观测值并提供推荐规则后，您可以将该规则应用于您的数据质量节点。要实现此目的，应按照以下步骤进行：

1. 单击每条规则建议旁边的复制图标。这会将规则建议添加到记事本中，可供您稍后检索。
2. 单击应用规则建议。此操作将打开记事本，您可以在其中查看之前复制的规则。
3. 选择复制规则。
4. 选择应用于规则集编辑器。此操作将打开规则集编辑器，您可在其中粘贴复制的规则。
5. 将复制的规则粘贴到规则集编辑器中。

Glue Studio 笔记本中 AWS ETL 作业的数据质量

在本教程中，您将学习如何使用 AWS Glue Data Quality 来处理 AWS Glue Studio 笔记本中的提取、转换、加载（ETL）作业。

您可以在 AWS Glue Studio 中使用笔记本编辑作业脚本并查看输出，而不必运行完整作业。您还可以添加 Markdown 并将笔记本另存为 `.ipynb` 文件和作业脚本。请注意：您可以在无需本地安装软件或管理服务器的情况下开启笔记本。当您对自己的代码感到满意时，可以使用 AWS Glue Studio 轻松地将笔记本转换为 AWS Glue 作业。

此示例使用的数据集包括从两个 Data.CMS.gov 数据集下载的医疗保健提供商付款数据："Inpatient Prospective Payment System Provider Summary for the Top 100 Diagnosis-Related Groups - FY2011" 和 "Inpatient Charge Data FY 2011"。

下载该数据后，我们修改了数据集，以在文件末尾引入了几个错误的记录。这个修改过的文件位于 `s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv` 上的公有 Amazon S3 存储桶。

先决条件

- 具有 Amazon S3 权限的 AWS Glue 角色可以写入您的目标 Amazon S3 存储桶
- 一个新的笔记本（请参阅 Getting started with notebooks in AWS Glue Studio）

在 AWS Glue Studio 中创建 ETL 作业

创建 ETL 作业

1. 将会话版本更改为 AWS Glue 3.0。

   为此，请使用以下魔术命令删除所有样板代码单元格，然后运行该单元格。请注意，创建新笔记本时，此样板代码会自动在第一个单元格中提供。

```
%glue_version 3.0
```

2. 将以下代码复制并粘贴到单元格中。

```
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. 在下一个单元格中，导入评估 AWS Glue Data Quality 的 `EvaluateDataQuality` 类。

```
from awsgluedq.transforms import EvaluateDataQuality
```

4. 在下一个单元格中，使用存储在公共 Amazon S3 存储桶中的 .csv 文件读入源数据。

```
medicare = spark.read.format(
"csv").option(
"header", "true").option(
"inferSchema", "true").load(
's3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv')
medicare.printSchema()
```

5. 将数据转换为 AWS Glue DynamicFrame.

```
from awsglue.dynamicframe import DynamicFrame
medicare_dyf = DynamicFrame.fromDF(medicare,glueContext,"medicare_dyf")
```

6. 使用数据质量定义语言（DQDL）创建规则集。

```
EvaluateDataQuality_ruleset = """
    Rules = [
```

```
        ColumnExists "Provider Id",
        IsComplete "Provider Id",
        ColumnValues  " Total Discharges " > 15
]
    ]
"""
```

7. 根据规则集验证数据集。

```
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. 查看结果。

```
ruleOutcomes = SelectFromCollection.apply(
    dfc=EvaluateDataQualityMultiframe,
    key="ruleOutcomes",
    transformation_ctx="ruleOutcomes",
)

ruleOutcomes.toDF().show(truncate=False)
```

输出：

```
-----------------------------------+-------
+-----------------------------------------------------
+-----------------------------------------+
|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. 筛选通过的行并查看数据质量行级结果中的失败行。

```
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
```

输出：

```
+----------------------------------------+-----------
+-------------------------------------+--------------------------+-------------
+--------------+-----------------+-----------------------------------
+------------------+-------------------------+-----------------------
+--------------------------+-------------------------
+-----------------------------------------+---------------------------
+---------------------------+
|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
```

请注意，AWS Glue 数据质量增加了四个新列（DataQualityRulesPass DataQualityRulesFail、DataQualityRulesSkip、和 DataQualityEvaluationResult）。这表示已通过的记录、失败的记录、跳过的行级评估规则以及总体行级结果。

10. 将输出写入 Amazon S3 存储桶，以分析数据并可视化结果。

```
#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")
```

数据质量定义语言（DQDL）引用

数据质量定义语言 (DQDL) 是一种特定领域的语言，用于定义 Glue 数据 AWS 质量规则。

本指南介绍了关键的 DQDL 概念，以帮助您理解该语言。它还通过语法和示例为 DQDL 规则类型提供了参考。在使用本指南之前，我们建议您熟悉 AWS Glue 数据质量。有关更多信息，请参阅 AWS Glue 数据质量。

> **Note**
>
> DynamicRules 仅在 AWS Glue ETL 中支持。

目录

- 总和
- SchemaMatch
- 独特性
- UniqueValueRatio
- DetectAnomalies

DL 语法

DQDL 文档区分大小写，并包含一个规则集，该规则集将各个数据质量规则组合在一起。要构造规则集，必须创建一个名为 Rules（大写）的列表，由一对方括号分隔。该列表应包含一个或多个以逗号分隔的 DQDL 规则，如下例所示。

```
Rules = [
   IsComplete "order-id",
   IsUnique "order-id"
]
```

规则结构

DQDL 规则的结构取决于规则类型。但是，DQDL 规则通常符合以下格式。

```
<RuleType> <Parameter> <Parameter> <Expression>
```

RuleType 是要配置的规则类型的区分大小写名称。例如，IsComplete、IsUnique 或 CustomSql。每种规则类型的规则参数都不同。有关 DQDL 规则类型及其参数的完整参考，请参阅DL 规则类型引用。

复合规则

DQDL 支持以下逻辑运算符，您可以使用这些运算符来组合规则。这些规则被称为复合规则。

以及

当且仅当逻辑 and 运算符连接的规则为 true，逻辑运算符才会产生结果 true。否则，组合规则将导致 false。与 and 运算符连接的每条规则都必须用圆括号括起来。

以下示例使用 and 运算符将两个 DL 规则组合。

```
(IsComplete "id") and (IsUnique "id")
```

或者

当且仅当逻辑 or 运算符连接的一个或多个规则为 true，逻辑运算符才会产生结果 true。与 or 运算符连接的每条规则都必须用圆括号括起来。

以下示例使用 or 运算符将两个 DL 规则组合。

```
(RowCount "id" > 100) or (IsPrimaryKey "id")
```

您可以使用同一个运算符连接多个规则，因此允许使用以下规则组合。

```
(Mean "Star_Rating" > 3) and (Mean "Order_Total" > 500) and (IsComplete "Order_Id")
```

但是，您不能将逻辑运算符组合成一个表达式。例如，不允许以下组合。

```
(Mean "Star_Rating" > 3) and (Mean "Order_Total" > 500) or (IsComplete "Order_Id")
```

复合规则的工作原理

默认情况下，复合规则将作为整个数据集或表中的单个规则进行评估，然后合并结果。换句话说，它会首先评估整列，然后应用运算符。下面举例说明这种默认行为：

```
# Dataset

+------+------+
|myCol1|myCol2|
+------+------+
|     2|     1|
|     0|     3|
+------+------+

# Overall outcome

+-------------------------------------------------------------+-------+
|Rule                                                         |Outcome|
+-------------------------------------------------------------+-------+
|(ColumnValues "myCol1" > 1) OR (ColumnValues "myCol2" > 2)|Failed |
```

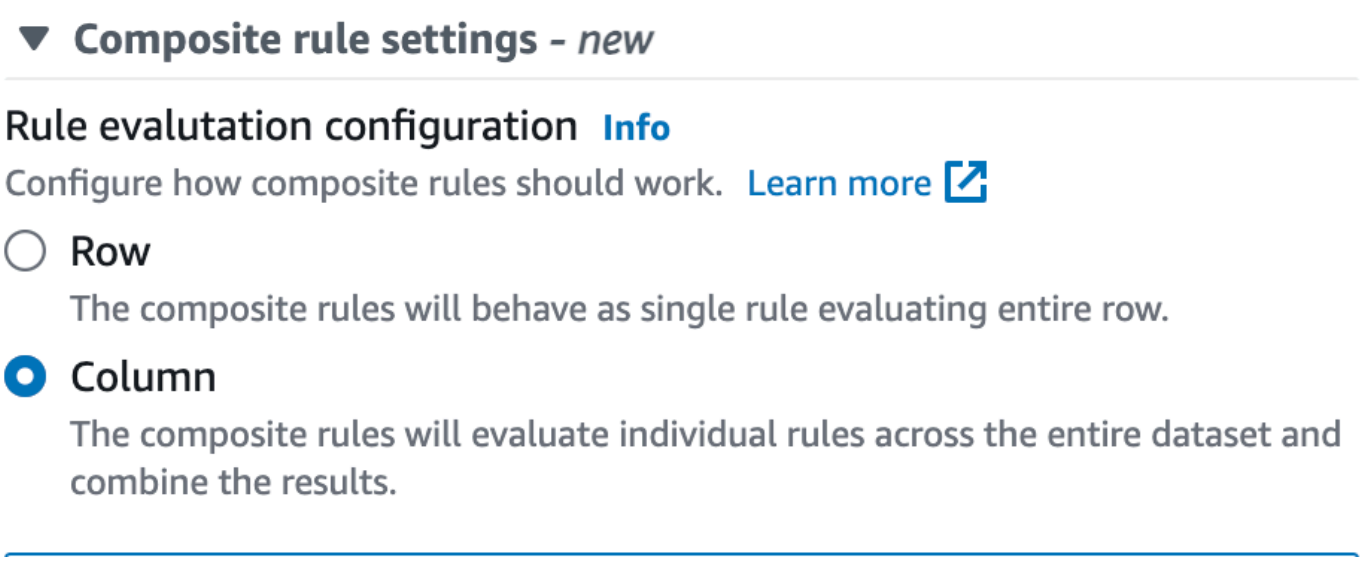
```
+-----------------------------------------------------------+-------+
```

在上面的示例中，AWS Glue Data Quality 会首先评估会导致失效的 (ColumnValues "myCol1" > 1)。然后它将评估同时也会出现失效的 (ColumnValues "myCol2" > 2)。两个结果的组合将被标记为 FAILED。

但是，如果您更喜欢与 SQL 类似的行为，即需要评估整行，则必须显式设置 ruleEvaluation.scope 参数，如下面代码片段中的 additionalOptions 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 数据目录中，您可以在用户界面中轻松设置此选项，如下所示。



设置完成后，复合规则将作为评估整行的单个规则运行。以下示例说明了此行为。

```
# 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
                |
+------+------+-----------------------------------------------------------------
+---------------------------+
```

此功能不支持某些规则，因为其总体结果依赖阈值或比率。不支持的规则列举如下。

依赖比率的规则：

- 完整性
- DatasetMatch
- ReferentialIntegrity

- 独特性

依赖阈值的规则：

当以下规则包含阈值时，则不支持这些规则。但是，不涉及 `with threshold` 的规则仍受支持。

- ColumnDataType
- ColumnValues
- CustomSQL

Expressions

如果规则类型不生成布尔响应，则必须提供表达式作为参数才能创建布尔响应。例如，以下规则根据表达式检查列中所有值的均值（平均值），以返回真或假结果。

```
Mean "colA" between 80 and 100
```

某些规则类型（例如 `IsUnique` 和 `IsComplete`）已经返回布尔响应。

下表列出了您可在 DL 规则中使用的表达式。

支持的 DQDL 表达式

Expression	描述	示例
=*x*	如果规则类型响应等于 *x*，则解析为 `true`。	`Completeness "colA" = "1.0", ColumnValues "colA" = "2022-06-30"`
!=*x*	x 如果规则类型响应不等于 *x*，则解析为 true。	`ColumnValues "colA" != "a", ColumnValues "colA" != "2022-06-30"`
> *x*	如果规则类型响应大于 *x*，则解析为 `true`。	`ColumnValues "colA" > 10`
< *x*	如果规则类型响应小于 *x*，则解析为 `true`。	`ColumnValues "colA" < 1000,`

Expression	描述	示例
		`ColumnValues "colA" < "2022-06-30"`
>= *x*	如果规则类型响应大于 *x*，则解析为 `true`。	`ColumnValues "colA" >= 10`
<= *x*	如果规则类型响应小于或等于 *x*，则解析为 `true`。	`ColumnValues "colA" <= 1000`
介于 *x* 与 *y* 之间	如果规则类型响应在指定范围内（不包括），则解析为 `true`。仅将此表达式类型用于数字和日期类型。	`Mean "colA" between 8 and 100, ColumnValues "colA" between "2022-05-31" and "2022-06-30"`
not between *x* and *y*	如果规则类型响应不在指定范围内（包括首尾数），则解析为 true。您只能将此表达式类型用于数字和日期类型。	`ColumnValues "colA" not between "2022-05-31" and "2022-06-30"`
在 [*a, b, c, ...*] 中	如果规则类型响应在指定的集合中，则解析为 `true`。	`ColumnValues "colA" in [ 1, 2, 3 ], ColumnValues "colA" in [ "a", "b", "c" ]`
not in [*a, b, c, ...*]	如果规则类型响应不在指定的集合中，则解析为 `true`。	`ColumnValues "colA" not in [ 1, 2, 3 ], ColumnValues "colA" not in [ "a", "b", "c" ]`
匹配 */ab+c/i*	如果规则类型响应与正则表达式匹配，则解析为 `true`。	`ColumnValues "colA" matches "[a-zA-Z]*"`
not matches */ab+c/i*	`true`如果规则类型响应与正则表达式不匹配，则解析为。	`ColumnValues "colA" not matches "[a-zA-Z]*"`

Expression	描述	示例
now( )	仅适用于创建日期表达式的 ColumnValues 规则类型。	`ColumnValues "load_date" > (now() - 3 days)`

matches/in […]/not matches/not in [...] with threshold	指定符合规则条件的值的百分比。仅适用于 ColumnValues 、ColumnDataType 和 CustomSQL 规则类型。	`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`

NULL、EMPTY 和 WHITESPACE_ONLY 的关键字

如果要验证字符串列是否为零、空或属于仅包含空格的字符串，则可以使用以下关键字：

- NULL/null – 对于字符串列中的 null 值，此关键字解析为 true。

  如果超过 50% 的数据没有零值，则 `ColumnValues "colA" != NULL with threshold > 0.5` 将返回 true。

  对于所有具有零值或长度大于 5 的行，`(ColumnValues "colA" = NULL) or (ColumnLength "colA" > 5)` 都将返回 true。请注意，这将需要使用 “compositeRuleEvaluation.method” = “ROW” 选项。
- EMPTY/empty – 对于字符串列中的空字符串（“”）值，此关键字将解析为 true。某些数据格式会将字符串列中的零值转换为空字符串。此关键字有助于过滤掉数据中的空字符串。

  如果一行为空、“a”或“b”，`(ColumnValues "colA" = EMPTY) or (ColumnValues "colA" in ["a", "b"])` 将返回 true。请注意，这需要使用 “compositeRuleEvaluation.method” = “ROW” 选项。
- WHITESPACES_ONLY /whitespaces_only – 对于字符串列中只有空格（“ ”）值的字符串，此关键字解析为 true。

  如果一行既不是“a”或“b”，也不只是空格，`ColumnValues "colA" not in ["a", "b", WHITESPACES_ONLY]` 将返回 true。

支持的规则：

- ColumnValues

对于基于数值或日期的表达式，如果要验证列是否包含零值，则可以使用以下关键字。

- NULL/null – 对于字符串列中的零值，此关键字解析为 true。

  如果列中的日期为 2023-01-01 或零，ColumnValues "colA" in [NULL, "2023-01-01"] 将返回 true。

  对于所有具有零值或值介于 1 到 9 之间的行，(ColumnValues "colA" = NULL) or (ColumnValues "colA" between 1 and 9) 都将返回 true。请注意，这将需要使用 “compositeRuleEvaluation.method” = “ROW” 选项。

  支持的规则：

  - ColumnValues

使用 Where 子句筛选

您可以在制定规则时筛选数据。当你想应用条件规则时，这很有用。

```
<DQDL Rule> where "<valid SparkSQL where clause> "
```

必须使用where关键字指定过滤器，然后使用引号括起有效的 sparkSQL 语句。("")

如果要将规则添加 where 子句添加到具有阈值的规则中，则应在阈值条件之前指定 where 子句。

```
<DQDL Rule> where "valid SparkSQL statement>" with threshold <threshold condition>
```

使用此语法，您可以编写如下规则。

```
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)"
```

我们将验证所提供的 sparkSQL 语句是否有效。如果无效，则规则评估将失败，我们将IllegalArgumentException使用以下格式抛出 a：

```
Rule <DQDL Rule> where "<invalid SparkSQL>" has provided an invalid where clause :
<SparkSQL Error>
```

开启行级错误记录识别时的 Where 子句行为

借 AWS 助 Glue 数据质量，您可以识别失败的特定记录。将 where 子句应用于支持行级结果的规则时，我们会将由 where 子句过滤掉的行标记为`Passed`。

如果您希望将筛选出的行分别标记为`SKIPPED`，则可以为 ETL 作业设置以下内容`additionalOptions`。

```
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"
        }
      """
    )
  )
}
```

例如，请参阅以下规则和 dataFrame：

```
IsComplete att2 where "att1 = 'a'"
```

id	att1	att2	行级结果（默认）	行级结果（跳过的选项）	注释
1	a	f	通过了	通过了	
2	b	d	通过了	SKIPPED	行已被过滤掉，因为不att1是"a"
3	a	null	FAILED	FAILED	
4	a	f	通过了	通过了	
5	b	null	通过了	SKIPPED	行已被过滤掉，因为不att1是"a"
6	a	f	通过了	通过了	

动态规则

现在，您可以编写动态规则，将规则生成的当前指标与其历史值进行比较。这些历史比较通过在表达式中使用 `last()` 运算符启用。例如，当前运行中的行数大于同一数据集之前最近一次的行数时，`RowCount > last()` 规则就会成功。`last()` 采用一个可选的自然数参数，描述要考虑的先前指标数量；`k >= 1` 将引用的最后一个 k 指标 `last(k)`。

- 如果没有可用的数据点，`last(k)` 将返回默认值 0.0。
- 如果少于可用的 k 指标，则 `last(k)` 将返回所有之前的指标。

要形成有效的表达式，请使用 `last(k)`，其中 `k > 1` 需要聚合函数以将多个历史结果简化为单个数字。例如，`RowCount > avg(last(5))` 将检查当前数据集的行数是否严格大于同一数据集最后五行计数的平均值。`RowCount > last(5)` 将产生错误，因为无法将当前数据集的行数与列表进行有意义的比较。

支持的聚合函数：

- avg
- median
- max
- min
- sum
- std（标准偏差）
- abs（绝对值）
- index(last(k), i) 将允许从最后一个 k 中选择第 i 最新的值。i 为零索引，因此 index(last(3), 0) 将返回最新的数据点，并且 index(last(3), 3) 将导致错误，因为只有三个数据点，而我们尝试编制第四最新数据点的索引。

示例表达式

ColumnCorrelation

- ColumnCorrelation "colA" "colB" < avg(last(10))

DistinctValuesCount

- DistinctValuesCount "colA" between min(last(10))-1 and max(last(10))+1

大多数具有数字条件或阈值的规则类型都支持动态规则；请参阅提供的分析器和规则表，以确定您的规则类型是否支持动态规则。

分析器

> **Note**
>
> AWS Glue 数据目录不支持分析器。

DQDL 规则使用名为分析器的功能收集有关您的数据的信息。规则的布尔表达式使用此信息确定规则应该成功还是失败。例如，该 RowCount 规则RowCount > 5 将使用行计数分析器来发现数据集中的行数，并将该计数与表达式进行比较，> 5以检查当前数据集中是否存在超过五行。

有时，我们建议创建分析器（而不是编写规则），然后让其生成可用于检测异常的统计信息。对于此类实例，您可以创建分析器。分析器在以下方面与规则不同。

特征	分析器	规则
规则集的一部分	是	是
生成统计信息	是	是
生成观测值	是	是
可以评估和断言条件	否	是
您可以配置操作，例如在失败时停止作业、继续处理作业	否	是

分析器无需规则即可独立存在，因此您可以快速配置分析器并逐步构建数据质量规则。

可以在规则集的 Analyzers 块中输入某些规则类型，以运行分析器所需的规则并收集信息，而无需对任何条件进行检查。有些分析器与规则无关，只能在 Analyzers 块中输入。下表显示每个项目是作为规则还是作为独立分析器支持，以及每种规则类型的其他详细信息。

带分析器的示例规则集

以下规则集使用：

- 一条动态规则，用于检查数据集在过去三次作业运行中，是否增长超过其尾随平均值
- DistinctValuesCount 分析器，用于记录数据集 Name 列中不同值的数量
- ColumnLength 分析器，用于跟踪一段时间内最小和最大 Name 大小

可以在作业运行的“数据质量”选项卡中，查看分析器指标结果。

```
Rules = [
   RowCount > avg(last(3))
]
Analyzers = [
   DistinctValuesCount "Name",
   ColumnLength "Name"
]
```

注释

您可以使用“#”字符在 DQDL 文档中添加注释。DQDL 会忽略在“#”字符之后以及直到行尾的任何内容。

```
Rules = [
    # More items should generally mean a higher price, so correlation should be
 positive
    ColumnCorrelation "price" "num_items" > 0
]
```

DL 规则类型引用

本节提供了 Glue 数据质量支持的每 AWS 种规则类型的参考。

> **Note**
>
> - DQDL 目前不支持嵌套或列表类型的列数据。
> - 下表中带括号的值将替换为规则参数中提供的信息。
> - 规则通常需要一个额外的表达式参数。

RuleType	描述	参数	报告的指标	是否支持作为规则？	是否支持作为分析器？	返回行级结果？	动态规则支持？	生成观测值	支持 Where 子句语法？
Aggregate Match	通过比较总销售额等摘要指标，检查两个数据集是否匹配。如	一个或多个聚合	当第一个和第二个聚合列名匹配时： Column.[C	是	否	否	否	否	否

RuleTyp	描述	参数	报告的指标	是否支持作为规则？	是否支持作为分析器？	返回行级结果？	动态规则支持？	生成观测值	支持 Where 子句语法？
	果所有数据都是从源系统提取的，金融机构可以进行比较。		olumn]. gregate tch 当第一个和第二个聚合列名不同时： Column. [C olumn1, lumn2]. gregate tch						

RuleTyp	描述	参数	报告的指标	是否支持作为规则？	是否支持作为分析器？	返回行级结果？	动态规则支持？	生成观测值	支持 Where 子句语法？

AllStatistics	独立分析器，为提供的列或数据集中的所有列收集多个指标。	单个列名，或“AllColumns”	对于所有类型的列：
			`Dataset.RowCou`
			`Column.[Column].mpleters`
			`Column.[Column].iquenes`
			字符串值列的其他指标：
			`ColumnLgth metrics`
			数字值列的其他指标：
				否	是	否	否	否	否

RuleTyp	描述	参数	报告的指标	是否支持作为规则？	是否支持作为分析器？	返回行级结果？	动态规则支持？	生成观测值	支持 Where 子句语法？
			Column\ues metrics						
ColumnCorrelation	检查两列的关联程度。	正好两个列名	Multicolumn.[Column1],[Column2].ColumnCorrelation	是	是	否	是	否	是
ColumnCount	检查是否删除了任何列。	无	Dataset.ColumnCount	是	否	否	是	是	不支持
ColumnDataType	检查列是否符合数据类型。	只有一个列名	Column.[Column].ColumnDataType.Compliance	是	否	否	是，在行级阈值表达式中	否	是

RuleType	描述	参数	报告的指标	是否支持作为规则？	是否支持作为分析器？	返回行级结果？	动态规则支持？	生成观测值	支持 Where 子句语法？
ColumnExists	检查数据集中是否存在列。这允许客户构建自助服务数据平台以确保某些列可用。	只有一个列名	不适用	是	否	否	否	否	否

RuleTyp	描述	参数	报告的指标	是否支持作为规则？	是否支持作为分析器？	返回行级结果？	动态规则支持？	生成观测值	支持 Where 子句语法？
ColumnL gth	检查数据长度是否一致。	只有一个列名	Column. [C olumn]. ximumLe th Column. [C olumn]. nimumLe th 提供行级阈值时的其他指标： Column. [C olumn]. lumnVal s.Compl nce	是	是	是，如已提供行级阈值	否	是。仅通过分析最小长度和最大长度来生成观测值	是

RuleTyp	描述	参数	报告的指标	是否支持作为规则？	是否支持作为分析器？	返回行级结果？	动态规则支持？	生成观测值	支持 Where 子句语法？
ColumnN esMatchP ttern	检查列名是否与定义的模式匹配。对于治理团队来说，强制列名一致性很有用。	列名的正则表达式	Dataset .Columm mesPatt nMatchR io	是	否	否	否	否	否

RuleTyp	描述	参数	报告的指标	是否支持作为规则？	是否支持作为分析器？	返回行级结果？	动态规则支持？	生成观测值	支持 Where 子句语法？
ColumnV ues	根据定义的值检查数据是否一致。此规则支持正则表达式。	只有一个列名	Column. [C olumn]. ximum Column. [C olumn]. nimum 提供行级阈值时的其他指标： Column. [C olumn]. lumnVa s.Comp nce	是	是	是，如已提供行级阈值	否	是。仅通过分析最小值和最大值来生成观测值	是
完整性	检查数据中是否有任何空白或 NULL。	只有一个列名	Column. [C olumn]. mpleter s	是	是	是	是	是	是

RuleTyp	描述	参数	报告的指标	是否支持作为规则？	是否支持作为分析器？	返回行级结果？	动态规则支持？	生成观测值	支持 Where 子句语法？
CustomS	客户可以在 SQL 中实施几乎任何类型的数据质量检查。	SQL 语句 （可选）行级阈值	Dataset.CustomL 提供行级阈值时的其他指标： Dataset.CustomL.Compliance	是	不支持	是，如已提供行级阈值	是	否	否
DataFres ness	检查数据是否是最新的。	只有一个列名	Column.[Column].DataFreshness.Compliance	是	否	是	否	否	是
DatasetM tch	比较两个数据集并确定它们是否同步。	引用数据集的名称 列映射 （可选）用于检查匹配项的列	Dataset[ReferenceDatasetAlias].DatasetMatch	是	否	是	是	否	否

RuleType	描述	参数	报告的指标	是否支持作为规则？	是否支持作为分析器？	返回行级结果？	动态规则支持？	生成观测值	支持 Where 子句语法？
DistinctValuesCount	检查是否存在重复值。	只有一个列名	Column.[Column].stinctValuesCount	是	是	是	是	是	是
DetectAnomalies	检查其他规则类型报告的指标中是否存在异常。	规则类型	规则类型参数报告的指标	是	否	否	否	否	否
熵	检查数据的熵。	只有一个列名	Column.[Column].tropy	是	是	否	是	否	是
IsComplete	检查数据是否 100% 完整。	只有一个列名	Column.[Column].mpleteness	是	否	是	否	否	是

RuleTyp	描述	参数	报告的指标	是否支持作为规则？	是否支持作为分析器？	返回行级结果？	动态规则支持？	生成观测值	支持 Where 子句语法？
IsPrimary Key	检查列是否为主键（不是 NULL 且是唯一的）。	只有一个列名	对于单列： Column. [C olumn]. iquenes 对于多列： Multicc mn[Comn elimite olumns] niquene	是	否	是	否	否	是
IsUnique	检查数据是否 100% 唯一。	只有一个列名	Column. [C olumn]. iquenes	是	否	是	否	否	是
平均值	检查均值是否与设定的阈值相符。	只有一个列名	Column. [C olumn]. an	是	是	是	是	否	是

RuleTyp	描述	参数	报告的指标	是否支持作为规则？	是否支持作为分析器？	返回行级结果？	动态规则支持？	生成观测值	支持 Where 子句语法？
Referent alIntegri ty	检查两个数据集是否具有引用完整性。	数据集中的一个或多个列名 引用数据集中的一个或多个列名	Column [R eferenc ataset/ as].Rel ential egrity	是	否	是	是	否	否
RowCou	检查记录计数是否与阈值匹配。	无	Dataset .RowCou	是	是	否	是	是	是
RowCou atch	检查两个数据集之间的记录计数是否匹配。	引用数据集别名	Dataset [Referer Dataset ias].Ro ountMat	是	否	否	是	否	否
Standarc eviation	检查标准偏差是否与阈值匹配。	只有一个列名	Column [C olumn]. andardD iation	是	是	是	是	否	是

RuleTyp	描述	参数	报告的指标	是否支持作为规则？	是否支持作为分析器？	返回行级结果？	动态规则支持？	生成观测值	支持 Where 子句语法？
Schemal ch	检查两个数据集之间的架构是否匹配。	引用数据集别名	Dataset [Referer Dataset ias].Sc maMatch	是	否	否	是	否	否
总和	检查总和是否与设定的阈值相符。	只有一个列名	Column [C olumn]. m	是	是	否	是	否	是
独特性	检查数据集的唯一性是否与阈值匹配。	只有一个列名	Column [C olumn]. iquenes	是	是	是	是	否	是
UniqueV ueRatio	检查唯一值比率是否与阈值匹配。	只有一个列名	Column [C olumn]. iqueVal Ratio	是	是	是	是	否	是

主题

- AggregateMatch
- ColumnCorrelation
- ColumnCount
- ColumnDataType

- ColumnExists
- ColumnLength
- ColumnNamesMatchPattern
- ColumnValues
- 完整性
- CustomSQL
- DataFreshness
- DatasetMatch
- DistinctValuesCount
- 熵
- IsComplete
- IsPrimaryKey
- IsUnique
- 平均值
- ReferentialIntegrity
- RowCount
- RowCountMatch
- StandardDeviation
- 总和
- SchemaMatch
- 独特性
- UniqueValueRatio
- DetectAnomalies

AggregateMatch

根据给定表达式检查两列聚合的比率。此规则类型适用于多个数据集。对两列聚合进行评估，并通过将第一列聚合的结果除以第二列聚合的结果得出比率。根据提供的表达式检查比率以生成布尔响应。

语法

列聚合

```
ColumnExists <AGG_OPERATION> (<OPTIONAL_REFERENCE_ALIAS>.<COL_NAME>)
```

- AGG_OPERATION – 用于聚合的操作。目前支持 sum 和 avg。

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数
- OPTIONAL_REFERENCE_ALIAS – 如果列来自引用数据集而不是主数据集，则需要提供此参数。如果您在 AWS Glue 数据目录中使用此规则，则您的参考别名必须遵循 “” 格式<database_name>。<table_name>。 <column_name>

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数
- COL_NAME – 要聚合列的名称。

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数

示例：平均值

```
"avg(rating)"
```

示例：总和

```
"sum(amount)"
```

示例：引用数据集中各列的平均值

```
"avg(reference.rating)"
```

规则

```
AggregateMatch <AGG_EXP_1> <AGG_EXP_2> <EXPRESSION>
```

- AGG_EXP_1 – 第一列聚合。

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数
- AGG_EXP_2 – 第二列聚合。

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数

支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数

- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：使用 sum 进行聚合匹配

以下示例规则检查 amount 列中值的总和是否完全等于 total_amount 列中值的总和。

```
AggregateMatch "sum(amount)" "sum(total_amount)" = 1.0
```

示例：使用 average 进行聚合匹配

以下示例规则检查 ratings 列中值的平均值是否等于 reference 数据集中 ratings 列中值的至少 90% 平均值。在 ETL 或 Data Catalog 体验版中，引用数据集作为附加数据来源提供。

在 AWS Glue ETL 中，你可以使用：

```
AggregateMatch "avg(ratings)" "avg(reference.ratings)" >= 0.9
```

在 AWS Glue 数据目录中，你可以使用：

```
AggregateMatch "avg(ratings)" "avg(database_name.tablename.ratings)" >= 0.9
```

零值行为

在计算聚合方法（总和/均值）时，AggregateMatch 规则将忽略含有零值的行。例如：

```
+---+-----------+
|id |units      |
+---+-----------+
|100|0          |
|101|null       |
|102|20         |
|103|null       |
|104|40         |
+---+-----------+
```

列 units 的均值将为 (0 + 20 + 40)/3 = 20。在此计算中不会考虑第 101 行和第 103 行。

ColumnCorrelation

根据给定表达式检查两列之间的相关性。 AWS Glue Data Quality 使用 Pearson 相关系数来测量两列之间的线性相关性。结果是一个介于 -1 和 1 之间的数字，用于衡量关系的强度和方向。

语法

```
ColumnCorrelation <COL_1_NAME> <COL_2_NAME> <EXPRESSION>
```

- COL_1_NAME — 要根据其评估数据质量规则的第一列的名称。

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数
- COL_2_NAME — 要根据其评估数据质量规则的第二列的名称。

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数
- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：列相关性

以下示例规则检查列 `height` 和 `weight` 之间的相关系数是否具有很强的正相关性（系数值大于 0.8）。

```
ColumnCorrelation "height" "weight" > 0.8
```

```
ColumnCorrelation "weightinkgs" "Salary" > 0.8 where "weightinkgs > 40"
```

示例动态规则

- `ColumnCorrelation "colA" "colB" between min(last(10)) and max(last(10))`
- `ColumnCorrelation "colA" "colB" < avg(last(5)) + std(last(5))`

零值行为

在相关性计算中，`ColumnCorrelation` 规则将忽略包含 NULL 值的行。例如：

```
+---+-----------+
|id |units      |
```

```
+---+-----------+
|100|0          |
|101|null       |
|102|20         |
|103|null       |
|104|40         |
+---+-----------+
```

第 101 行和第 103 行将被忽略，ColumnCorrelation 将为 1.0。

ColumnCount

根据给定表达式检查主数据集的列数。在表达式中，您可以使用 > 和 < 等运算符指定列数或列范围。

语法

```
ColumnCount <EXPRESSION>
```

- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：列数数字校验

以下示例规则检查列数是否在给定范围内。

```
ColumnCount between 10 and 20
```

示例动态规则

- `ColumnCount >= avg(last(10))`
- `ColumnCount between min(last(10))-1 and max(last(10))+1`

ColumnDataType

根据提供的预期类型检查给定列中值的固有数据类型。接受 `with threshold` 表达式以检查列中值的子集。

语法

```
ColumnDataType <COL_NAME> = <EXPECTED_TYPE>
```

```
ColumnDataType <COL_NAME> = <EXPECTED_TYPE> with threshold <EXPRESSION>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：字符串类型

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数
- EXPECTED_TYPE – 列中值的预期类型。

  支持的值：布尔值、日期、时间戳、整数、双精度、浮点数、长整型

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数
- EXPRESSION – 一个可选表达式，用于指定应为预期类型的值的百分比。

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数

示例：将整数作为字符串的列数据类型

以下示例规则检查给定列中字符串类型的值是否实际为整数。

```
ColumnDataType "colA" = "INTEGER"
```

示例：列数据类型整数作为字符串检查值的子集

以下示例规则检查给定列中字符串类型超过 90% 的值是否实际为整数。

```
ColumnDataType "colA" = "INTEGER" with threshold > 0.9
```

ColumnExists

检查列是否存在。

语法

```
ColumnExists <COL_NAME>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：任何列类型

示例：列存在

以下示例规则检查名为 Middle_Name 的列是否存在。

```
ColumnExists "Middle_Name"
```

ColumnLength

检查列中每行的长度是否符合给定表达式。

语法

```
ColumnLength <COL_NAME><EXPRESSION>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：字符串
- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：列行长

以下示例规则检查名为 Postal_Code 的列中每行的值长度是否为 5 个字符。

```
ColumnLength "Postal_Code" = 5
ColumnLength "weightinkgs" = 2 where "weightinkgs > 10"
```

零值行为

ColumnLength 规则将 NULL 视为长度为 0 的字符串。对于 NULL 行：

```
ColumnLength "Postal_Code" > 4 # this will fail
```

```
ColumnLength "Postal_Code" < 6 # this will succeed
```

以下示例复合规则提供了一种让 NULL 值显式失效的方法：

```
(ColumnLength "Postal_Code" > 4) AND (ColumnValues != NULL)
```

ColumnNamesMatchPattern

检查主数据集中所有列的名称是否与给定的正则表达式相匹配。

语法

```
ColumnNamesMatchPattern <PATTERN>
```

- PATTERN - 您要评估数据质量规则依据的模式。

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数

示例：列名匹配模式

以下示例规则检查所有列是否以前缀“aws_”开头

```
ColumnNamesMatchPattern "aws_.*"
ColumnNamesMatchPattern "aws_.*" where "weightinkgs > 10"
```

ColumnValues

针对列中的值运行表达式。

语法

```
ColumnValues <COL_NAME> <EXPRESSION>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：任何列类型
- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：允许的值

以下示例规则检查指定列中的每个值是否在允许的值集中（包括零值、空值以及仅包含空格的字符串）。

```
ColumnValues "Country" in [ "US", "CA", "UK", NULL, EMPTY, WHITESPACES_ONLY ]
ColumnValues "gender" in ["F", "M"] where "weightinkgs < 10"
```

示例：正则表达式

以下示例规则根据正则表达式检查列中的值。

```
ColumnValues "First_Name" matches "[a-zA-Z]*"
```

示例：日期值

以下示例规则根据日期表达式检查日期列中的值。

```
ColumnValues "Load_Date" > (now() - 3 days)
```

示例：数值

以下示例规则检查列值是否与特定的数字约束条件相匹配。

```
ColumnValues "Customer_ID" between 1 and 2000
```

零值行为

对于所有 ColumnValues 规则（!= 和 NOT IN 除外），NULL 行都不能通过该规则。如果由于零值而导致不能通过规则，则失效原因将显示如下：

```
Value: NULL does not meet the constraint requirement!
```

以下示例复合规则提供了一种显式允许 NULL 值的方法：

```
(ColumnValues "Age" > 21) OR (ColumnValues "Age" = NULL)
```

使用!=和not in语法的否定 ColumnValues 规则将适用于NULL行。例如：

```
ColumnValues "Age" != 21
```

```
ColumnValues "Age" not in [21, 22, 23]
```

以下示例提供了一种让 NULL 值显式失效的方法

```
(ColumnValues "Age" != 21) AND (ColumnValues "Age" != NULL)
```

```
ColumnValues "Age" not in [21, 22, 23, NULL]
```

完整性

根据给定表达式检查列中完整（非空）值的百分比。

语法

```
Completeness <COL_NAME> <EXPRESSION>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：任何列类型
- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：空值百分比

以下示例规则检查列中是否有超过 95% 的值是完整的。

```
Completeness "First_Name" > 0.95
Completeness "First_Name" > 0.95 where "weightinkgs > 10"
```

示例动态规则

- `Completeness "colA" between min(last(5)) - 1 and max(last(5)) + 1`
- `Completeness "colA" <= avg(last(10))`

零值行为

关于 CSV 数据格式的注意事项：CSV 列上的空行可能会显示多种行为。

- 如果列为 `String` 类型，则空行将被识别为空字符串，并且不会不通过 `Completeness` 规则。
- 如果列属于类似于 `Int` 的其他数据类型，则空行将被识别为 `NULL`，并且将不会通过 `Completeness` 规则。

CustomSQL

此规则类型已扩展为支持两个用例：

- 针对数据集运行自定义 SQL 语句，并根据给定表达式检查返回值。
- 运行自定义 SQL 语句，在 SELECT 语句中指定列名，与某个条件进行比较以获得行级结果。

语法

```
CustomSql <SQL_STATEMENT> <EXPRESSION>
```

- SQL_STATEMENT — 返回单个数值的 SQL 语句，由双引号包围。
- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：用于检索整体规则结果的自定义 SQL

此示例规则使用 SQL 语句检索数据集的记录数。然后，该规则检查记录数是否介于 10 到 20 之间。

```
CustomSql "select count(*) from primary" between 10 and 20
```

示例：用于检索行级结果的自定义 SQL

此示例规则使用 SQL 语句，在 SELECT 语句中指定列名，与某个条件进行比较以获得行级结果。阈值条件表达式定义了整个规则失败的记录数的阈值。请注意，规则不能同时包含条件和关键字。

```
CustomSql "select Name from primary where Age  > 18"
```

或者

```
CustomSql "select Name from primary where Age > 18" with threshold  > 3
```

> **Important**
>
> `primary` 别名代表要评估的数据集的名称。在控制台上处理可视化 ETL 任务时，`primary` 始终表示传递给 `EvaluateDataQuality.apply()` 转换的 `DynamicFrame`。当您使用 AWS Glue 数据目录对表运行数据质量任务时，`primary`表示该表。

如果您在 AWS Glue Data Catalog 中，也可以使用实际的表名：

```
CustomSql "select count(*) from database.table" between 10 and 20
```

您也可以联接多个表来比较不同的数据元素：

```
CustomSql "select count(*) from database.table inner join database.table2 on id1 = id2"
 between 10 and 20
```

在 AWS Glue ETL 中，CustomSQL 可以识别未通过数据质量检查的记录。要使此功能起作用，您需要返回作为您正在评估数据质量的主表一部分的记录。作为查询一部分返回的记录被视为成功，未返回的记录被视为失败。

以下规则将确保将年龄小于 100 的记录标识为成功记录，并将超过该年龄的记录标记为失败。

```
CustomSql "select id from primary where age < 100"
```

当 50% 的记录年龄大于 10 时，此 CustomSQL 规则将通过，并且还将标识失败的记录。此 CustomSQL 返回的记录将被视为通过，而未返回的记录将被视为失败。

```
CustomSQL "select ID, CustomerID from primary where age > 10" with threshold > 0.5
```

注意：如果您返回数据集中不可用的记录，CustomSQL 规则将失败。

DataFreshness

通过评估当前时间和日期列值之间的差异，来检查列中数据的新鲜度。您可以为此规则类型指定基于时间的表达式，以确保列值是最新的。

语法

```
DataFreshness <COL_NAME> <EXPRESSION>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：日期
- EXPRESSION — 以小时或天为单位的数字表达式。必须在表达式中指定时间单位。

示例：数据新鲜度

以下示例规则检查数据的新鲜度。

```
DataFreshness "Order_Date" <= 24 hours
DataFreshness "Order_Date" between 2 days and 5 days
```

零值行为

对于含有 NULL 值的行，将不能通过 DataFreshness 规则。如果由于零值而导致不能通过规则，则失效原因将显示如下：

```
80.00 % of rows passed the threshold
```

其中 20% 未通过该规则的行包含带 NULL 的行。

以下示例复合规则提供了一种显式允许 NULL 值的方法：

```
(DataFreshness "Order_Date" <= 24 hours) OR (ColumnValues "Order_Date" = NULL)
```

Amazon S3 对象的数据新鲜度

有时，您需要根据 Amazon S3 文件的创建时间来验证数据的新鲜度。为此，您可以使用以下代码获取时间戳并将其添加到您的数据框中，然后应用数据新鲜度检查。

```
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

检查主数据集中的数据是否与引用数据集中的数据匹配。使用提供的键列映射将两个数据集联接起来。如果您只想检查这些列中的数据是否相等，则可以提供其他列映射。请注意，DataSetMatch为了起作用，您的连接键应该是唯一的，并且不应为 NULL（必须是主键）。如果您不满足这些条件，则会收到错误消息：“提供的键映射不适合给定的数据帧”。如果您无法使用唯一的联接密钥，请考虑使用其他规则类型，例如AggregateMatch匹配摘要数据。

语法

```
DatasetMatch <REFERENCE_DATASET_ALIAS> <JOIN CONDITION WITH
 MAPPING> <OPTIONAL_MATCH_COLUMN_MAPPINGS> <EXPRESSION>
```

- REFERENCE_DATASET_ALIAS – 用于比较主数据集数据的引用数据集的别名。
- KEY_COLUMN_MAPPINGS – 以逗号分隔的列名列表，这些列名构成了数据集中的键。如果两个数据集中的列名不相同，则必须使用 -> 分隔它们
- OPTIONAL_MATCH_COLUMN_MAPPINGS – 如果您只想检查某些列中的匹配数据，则可以提供此参数。它使用与键列映射相同的语法。如果未提供此参数，我们将匹配所有剩余列中的数据。其余的非键列在两个数据集中必须具有相同的名称。
- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：使用 ID 列匹配集合数据集

以下示例规则使用“ID”列来检查主数据集中是否有超过 90% 与引用数据集相匹配。在本例中，它会比较所有列。

```
DatasetMatch "reference" "ID" >= 0.9
```

示例：使用多个键列匹配集合数据集

在以下示例中，主数据集和引用数据集的键列名称不同。ID_1 和 ID_2 共同构成主数据集中的复合键。ID_ref1 和 ID_ref2 共同构成引用数据集中的复合键。在这种情况下，您可以使用特殊语法来提供列名。

```
DatasetMatch "reference" "ID_1->ID_ref1,ID_ref2->ID_ref2" >= 0.9
```

示例：使用多个键列匹配集合数据集并检查特定列是否匹配

此示例建立在前一个示例的基础上。我们要检查是否只有包含金额的列匹配。此列在主数据集中名为 Amount1，在引用数据集中名为 Amount2。您希望完全匹配。

```
DatasetMatch "reference" "ID_1->ID_ref1,ID_ref2->ID_ref2" "Amount1->Amount2" >= 0.9
```

DistinctValuesCount

根据给定的表达式检查列中的不同值的数量。

语法

```
DistinctValuesCount <COL_NAME> <EXPRESSION>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：任何列类型
- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：不同的列值计数

以下示例规则检查名为 State 的列是否包含 3 个以上的不同值。

```
DistinctValuesCount "State" > 3
DistinctValuesCount "Customer_ID" < 6  where "Customer_ID < 10"
```

示例动态规则

- `DistinctValuesCount "colA" between avg(last(10))-1 and avg(last(10))+1`
- `DistinctValuesCount "colA" <= index(last(10),2) + std(last(5))`

熵

检查列的熵值是否与给定表达式匹配。熵测量消息中包含的信息级别。给定列中值的概率分布，熵描述了识别一个值需要多少位。

语法

```
Entropy <COL_NAME> <EXPRESSION>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：任何列类型
- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：列熵

以下示例规则检查名 Feedback 为的列的熵值是否大于一。

```
Entropy "Star_Rating" > 1
Entropy "First_Name" > 1  where "Customer_ID < 10"
```

示例动态规则

- `Entropy "colA" < max(last(10))`
- `Entropy "colA" between min(last(10)) and max(last(10))`

IsComplete

检查列中的所有值是否完整（非空）。

语法

```
IsComplete <COL_NAME>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：任何列类型

示例：空值

以下示例检查名为 email 的列中的所有值是否均为非空值。

```
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"
```

零值行为

关于 CSV 数据格式的注意事项：CSV 列上的空行可能会显示多种行为。

- 如果列为 String 类型，则空行将被识别为空字符串，并且不会不通过 Completeness 规则。
- 如果列属于类似于 Int 的其他数据类型，则空行将被识别为 NULL，并且将不会通过 Completeness 规则。

IsPrimaryKey

检查列是否包含主键。如果列中的所有值都是唯一且完整（非空值），则该列包含主键。

语法

```
IsPrimaryKey <COL_NAME>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：任何列类型

示例：主键：

以下示例规则检查名为 `Customer_ID` 的列是否包含主键。

```
IsPrimaryKey "Customer_ID"
IsPrimaryKey "Customer_ID" where "Customer_ID < 10"
```

示例：包含多列的主键。以下任何示例均有效。

```
IsPrimaryKey "colA" "colB"
IsPrimaryKey "colA" "colB" "colC"
IsPrimaryKey colA "colB" "colC"
```

IsUnique

检查列中的所有值是否唯一，并返回布尔值。

语法

```
IsUnique <COL_NAME>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：任何列类型

示例：唯一列值

以下示例规则检查名为 email 的列中的所有值是否均为非空值。

```
IsUnique "email"
IsUnique "Customer_ID" where "Customer_ID < 10"]
```

平均值

检查列中所有值的均值（平均值）是否与给定表达式匹配。

语法

```
Mean <COL_NAME> <EXPRESSION>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数
- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：平均值

以下示例规则检查列中所有值的平均值是否超过阈值。

```
Mean "Star_Rating" > 3
Mean "Salary" < 6200 where "Customer_ID < 10"
```

示例动态规则

- `Mean "colA" > avg(last(10)) + std(last(2))`
- `Mean "colA" between min(last(5)) - 1 and max(last(5)) + 1`

零值行为

在计算均值时，Mean 规则将忽略含有 NULL 值的行。例如：

```
+---+-----------+
|id |units      |
+---+-----------+
|100|0          |
```

```
|101|null       |
|102|20         |
|103|null       |
|104|40         |
+---+-----------+
```

列 units 的均值将为 (0 + 20 + 40)/3 = 20。在此计算中不会考虑第 101 行和第 103 行。

ReferentialIntegrity

检查主数据集中一组列的值在多大程度上是引用数据集中一组列值的子集。

语法

```
ReferentialIntegrity <PRIMARY_COLS> <REFERENCE_DATASET_COLS> <EXPRESSION>
```

- PRIMARY_COLS – 主数据集中列名的逗号分隔列表。

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数
- REFERENCE_DATASET_COLS – 此参数包含用句点分隔的两个部分。第一部分是引用数据集的别名。第二部分是用大括号括起的引用数据集中以逗号分隔的列名列表。

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数
- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：检查邮政编码列 的引用完整性

以下示例规则检查主数据集 zipcode 列中是否有 90% 以上的值存在于 reference 数据集的 zipcode 列中。

```
ReferentialIntegrity "zipcode" "reference.zipcode" >= 0.9
```

示例：检查城市和州列的引用完整性

在以下示例中，包含城市和州信息的列存在于主数据集和引用数据集中。两个数据集中的列名不同。该规则检查主数据集中各列的值集是否与引用数据集中各列的值集完全相等。

```
ReferentialIntegrity "city,state" "reference.{ref_city,ref_state}" = 1.0
```

示例动态规则

- `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

根据给定表达式检查数据集的行数。在表达式中，您可以使用 > 和 < 等运算符指定行数或行范围。

语法

```
RowCount <EXPRESSION>
```

- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：行数数字校验

以下示例规则检查行数是否在给定范围内。

```
RowCount between 10 and 100
RowCount between 1 and 50 where "Customer_ID < 10"
```

示例动态规则

```
RowCount > avg(lats(10)) *0.8
```

RowCountMatch

检查主数据集的行数和引用数据集的行数与给定表达式的比率。

语法

```
RowCountMatch <REFERENCE_DATASET_ALIAS> <EXPRESSION>
```

- REFERENCE_DATASET_ALIAS – 用于比较行数的引用数据集的别名。

支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数

- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：根据引用数据集检查行数

以下示例规则检查主数据集的行数是否至少为引用数据集行数的 90%。

```
RowCountMatch "reference" >= 0.9
```

StandardDeviation

根据给定表达式检查列中所有值的标准差。

语法

```
StandardDeviation <COL_NAME> <EXPRESSION>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数
- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：标准差

以下示例规则检查名为 `colA` 的列中值的标准差是否小于指定值。

```
StandardDeviation "Star_Rating" < 1.5
StandardDeviation "Salary" < 3500 where "Customer_ID < 10"
```

示例动态规则

- `StandardDeviation "colA" > avg(last(10) + 0.1`
- `StandardDeviation "colA" between min(last(10)) - 1 and max(last(10)) + 1`

零值行为

在计算标准差时，StandardDeviation 规则将忽略含有 NULL 值的行。例如：

```
+---+-----------+-----------+
|id |units1     |units2     |
+---+-----------+-----------+
|100|0          |0          |
|101|null       |0          |
|102|20         |20         |
|103|null       |0          |
|104|40         |40         |
+---+-----------+-----------+
```

列 units1 的标准差将不考虑第 101 行和第 103 行，结果为 16.33。列 units2 的标准差将为 16。

总和

根据给定表达式检查列中所有值的总和。

语法

```
Sum <COL_NAME> <EXPRESSION>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数
- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：总和

以下示例规则检查列中所有值的总和是否超过给定阈值。

```
Sum "transaction_total" > 500000
Sum "Salary" < 55600 where "Customer_ID < 10"
```

示例动态规则

- `Sum "ColA" > avg(last(10))`
- `Sum "colA" between min(last(10)) - 1 and max(last(10)) + 1`

零值行为

在计算总和时，Sum 规则将忽略含有 NULL 值的行。例如：

```
+---+-----------+
|id |units      |
+---+-----------+
|100|0          |
|101|null       |
|102|20         |
|103|null       |
|104|40         |
+---+-----------+
```

列 units 的总和将不考虑第 101 行和第 103 行，结果为 (0 + 20 + 40) = 60。

SchemaMatch

检查主数据集中的架构是否与引用数据集中的架构匹配。架构检查逐列完成。如果名称相同且类型相同，则两列的架构相匹配。列的顺序不重要。

语法

```
SchemaMatch <REFERENCE_DATASET_ALIAS> <EXPRESSION>
```

- REFERENCE_DATASET_ALIAS – 用于比较架构的引用数据集的别名。

  支持的列类型：字节、十进制、双精度、浮点数、整数、长整数、短整数
- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例： SchemaMatch

以下示例规则检查主数据集的架构是否与引用数据集中的架构完全匹配。

```
SchemaMatch "reference" = 1.0
```

独特性

根据给定表达式检查列中唯一值的百分比。唯一值只出现一次。

语法

```
Uniqueness <COL_NAME> <EXPRESSION>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：任何列类型
- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：唯一性百分比

以下示例规则检查列中唯一值的百分比是否与某些数字标准相匹配。

```
Uniqueness "email" = 1.0
Uniqueness "Customer_ID" != 1.0 where "Customer_ID < 10"
```

示例动态规则

- `Uniqueness "colA" between min(last(10)) and max(last(10))`
- `Uniqueness "colA" >= avg(last(10))`

UniqueValueRatio

根据给定表达式检查列中唯一值比率。唯一值比率是唯一值除以列中所有不同值的数量得出的分数。唯一值仅出现一次，而不同值至少出现一次。

例如，集合 [a, a, b] 包含一个唯一值 (b) 和两个不同值（a 和 b）。因此，集合的唯一值比率为 ½ = 0.5。

语法

```
UniqueValueRatio <COL_NAME> <EXPRESSION>
```

- COL_NAME — 要根据其评估数据质量规则的列的名称。

  支持的列类型：任何列类型

- EXPRESSION — 针对规则类型响应运行以生成布尔值的表达式。有关更多信息，请参阅 Expressions。

示例：唯一值比率

此示例检查列与一系列值的唯一值比率。

```
UniqueValueRatio "test_score" between 0 and 0.5
UniqueValueRatio "Customer_ID" between 0 and 0.9 where "Customer_ID < 10"
```

示例动态规则

- `UniqueValueRatio "colA" > avg(last(10))`
- `UniqueValueRatio "colA" <= index(last(10),2) + std(last(5))`

DetectAnomalies

检测给定数据质量规则的异常。每次执行 DetectAnomalies 规则都会导致保存给定规则的评估值。当收集到足够的数据时，异常检测算法会获取该给定规则的所有历史数据并运行异常检测。DetectAnomalies 检测到异常时规则失败。可通过观测值获得有关检测到哪些异常的更多信息。

语法

```
    DetectAnomalies <RULE_NAME> <RULE_PARAMETERS>
```

RULE_NAME – 要根据其评估和检测异常的规则的名称。支持的规则：

- "RowCount"
- "Completeness"
- "Uniqueness"
- "Mean"
- "Sum"
- "StandardDeviation"
- "Entropy"
- "DistinctValuesCount"

- "UniqueValueRatio"
- "ColumnLength"
- "ColumnValues"
- "ColumnCorrelation"

RULE_PARAMETERS– 有些规则需要其他参数才能运行。请参阅给定的规则文档以了解所需的参数。

示例：的异常 RowCount

例如，如果我们要检测 RowCount 异常，我们会提供 RowCount 规则名称。

```
DetectAnomalies "RowCount"
```

示例：的异常 ColumnLength

例如，如果我们要检测 ColumnLength 异常，我们会提供 ColumnLength 规则名称和列名。

```
DetectAnomalies "ColumnLength" "id"
```

使用 API 来衡量和管理数据质量

本主题介绍如何使用 API 来衡量和管理数据质量。

目录

先决条件

- 确保您的 boto3 版本是最新版本，以便它包含最新的 AWS Glue Data Quality API。
- 确保您的 AWS CLI 版本是最新版本，以便包含最新的 CLI。

如果您使用 AWS Glue 作业来运行这些 API，则可以使用以下选项将 boto3 库更新到最新版本：

```
–additional-python-modules boto3==<version>
```

使用 AWS Glue Data Quality 建议

要启动 AWS Glue Data Quality 建议，请运行：

```
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']
```

对于建议运行，您可以使用 pushDownPredicates 或 catalogPartitionPredicates 来提高性能，并且只能在目录源的特定分区上运行建议。

```
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>'
  )
```

要获取 AWS Glue Data Quality 建议的结果，请运行：

```
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 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
```

从上面的响应对象中，您可以提取运行推荐的规则集，以便在后续步骤中使用：

```
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"
]
```

要获取可以筛选和列出的所有建议的列表，请执行以下操作：

```
response = client.list_data_quality_rule_recommendation_runs(
    Filter={
        'DataSource': {
            'GlueTable': {
                'DatabaseName': '<database_name>',
                'TableName': '<table_name>'
            }
        }
)
```

要取消现有 AWS Glue Data Quality 建议任务，请执行以下操作：

```
response = client.cancel_data_quality_rule_recommendation_run(
    RunId='dqrun-d4b6b01957fdd79e59866365bf9cb0e40fxxxxxxx'
)
```

使用 AWS Glue Data Quality 规则集

要创建 AWS Glue Data Quality 规则集，请执行以下操作：

```
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>'
    }
)
```

要创建数据质量规则集，请执行以下操作：

```
response = client.get_data_quality_ruleset(
    Name='<ruleset_name>'
)
print(response)
```

然后，您可以使用此 API 提取规则集：

```
print(response['Ruleset'])
```

要列出表的所有数据质量规则集，请执行以下操作：

```
response = client.list_data_quality_rulesets()
```

您可以使用 API 中的筛选条件来筛选附加到特定数据库或表的所有规则集：

```
response = client.list_data_quality_rulesets(
    Filter={
        'TargetTable': {
            'TableName': '<table_name>',
            'DatabaseName': '<database_name>'
        }
    },
)
```

要更新数据质量规则集，请执行以下操作：

```
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
```

要删除数据质量规则集，请执行以下操作：

```
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 运行

要启动 AWS Glue Data Quality 运行，请执行以下操作：

```
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']
```

请记住，您可以传递 pushDownPredicate 或 catalogPartitionPredicate 参数，以确保您的数据质量运行仅针对目录表中的一组特定分区。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>',
    ]
```

```
)
```

要获取有关 AWS Glue Data Quality 运行的信息，请执行以下操作：

```
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
```

要获取 AWS Glue Data Quality 运行的结果，请执行以下操作：

对于给定的 AWS Glue Data Quality 运行，您可以使用以下方法提取运行的评估结果：

```
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'])
```

要列出所有的 AWS Glue Data Quality 运行，请执行以下操作：

```
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
```

您可以修改筛选器子句，使其仅显示特定时间之间的结果或针对特定表运行的结果。

要停止正在进行的 AWS Glue Data Quality 运行，请执行以下操作：

```
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
```

处理 AWS Glue Data Quality 结果

要获取 AWS Glue Data Quality 运行结果，请执行以下操作：

```
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
```

要取消现有 AWS Glue Data Quality 建议任务，请执行以下操作：

给定一个 AWS Glue Data Quality 运行 ID，您可以提取结果 ID，然后得到实际结果，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'])
```

设置警报、部署和计划

本主题介绍如何为 AWS Glue 数据质量设置警报、部署和计划。

目录

  - 将通知格式化为电子邮件
- 在 CloudWatch 集成中设置警报和通知
- 查询数据质量结果以构建控制面板
- 使用部署数据质量规则 AWS CloudFormation
- 计划数据质量规则

在 Amazon EventBridge 集成中设置提醒和通知

AWS Glue Data Quality 支持发布 EventBridge 事件，这些事件是在数据质量规则集评估运行完成后发出的。这样，当数据质量规则失效时，您可以轻松设置警报。

以下是您在 Data Catalog 中评估数据质量规则集时的示例事件。有了这些信息，您就可以查看亚马逊提供的数据 EventBridge。您可以发出其他 API 调用以获取更多详细信息。例如，使用结果 ID 调用 `get_data_quality_result` API 以获取特定执行的详细信息。

```
{
    "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
    }
```

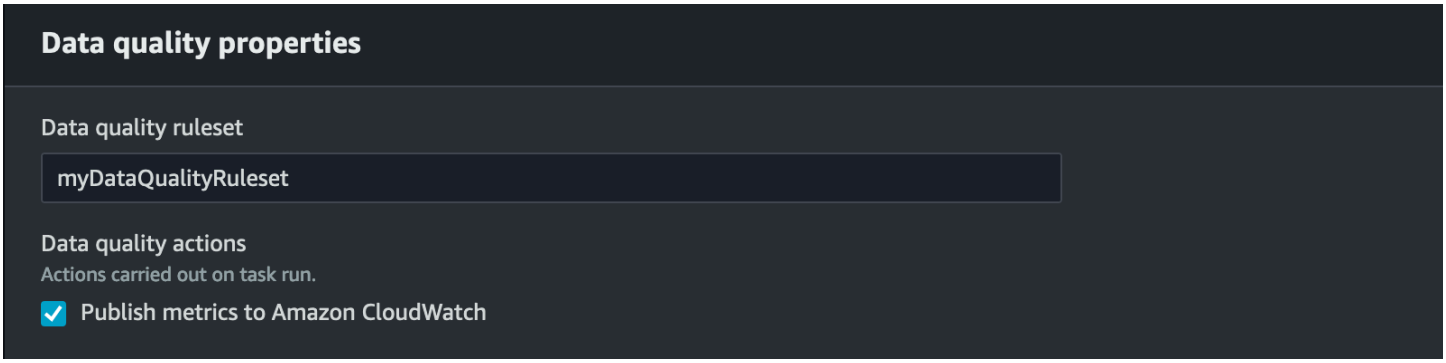
```
}
```

以下是您在评估 G AWS lue ETL 或 Glue Studio 笔记本中的数据质量规则集时发布的示例事件 AWS 。

```
{
    "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
    }
}
```

要使数据质量评估同时在数据目录和 ETL 作业中运行，则必须保持选中状态的 Amazon CloudWatch“将指标发布到” 选项（默认情况下处于选中状态）才能 EventBridge 发布。

设置 EventBridge 通知



要接收发出的事件并定义目标，您必须配置 Amazon EventBridge 规则。要创建规则，请执行以下操作：

1. 打开 Amazon EventBridge 控制台。
2. 在导航栏的总线部分下选择规则。
3. 选择 Create Rule。
4. 在定义规则详细信息中：
   a. 对于名称，请输入 `myDQRule`。
   b. 输入描述（可选）。
   c. 对于事件总线，请选择您的事件总线。如果您没有，请将其保留为默认值。
   d. 对于“规则类型”，选择具有事件模式的规则，然后选择下一步。
5. 在构建事件模式中：
   a. 对于事件源，请选择AWS 事件或 EventBridge 合作伙伴事件。
   b. 跳过示例事件部分。
   c. 对于创建方法，选择使用模式表单。
   d. 对于事件模式：
      i. 为事件源选择 AWS 服务。
      ii. 为 AWS 服务选择 Glue 数据质量。
      iii. 为“事件类型”选择可用的数据质量评估结果。
      iv. 为“特定状态”选择失败。然后，您将看到类似于以下内容的事件模式：

```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
  "detail": {
    "state": ["FAILED"]
```

```
  }
}
```

v. 有关更多连接选项，请参阅 事件模式的其他配置选项。

6. 在选择目标上：

   a. 对于目标类型，选择 AWS 服务。

   b. 使用选择目标下拉列表选择要连接的所需 AWS 服务（SNS、Lambda、SQS 等），然后选择下一步。

7. 在配置标签上，单击添加新标签以添加可选标签，然后选择下一步。

8. 您会看到所有选择的摘要页面。选择底部的创建规则。

事件模式的其他配置选项

除了根据成功或失败筛选事件外，您可能还需要根据不同的参数进一步筛选事件。

为此，请转到“事件模式”部分，然后选择编辑模式以指定其他参数。请注意，事件模式中的字段区分大小写。以下是配置事件模式的示例。

要从评估特定规则集的特定表中捕获事件，请使用以下类型的模式：

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

要在 ETL 体验中捕获来自特定作业的事件，请使用以下类型的模式：

```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
```

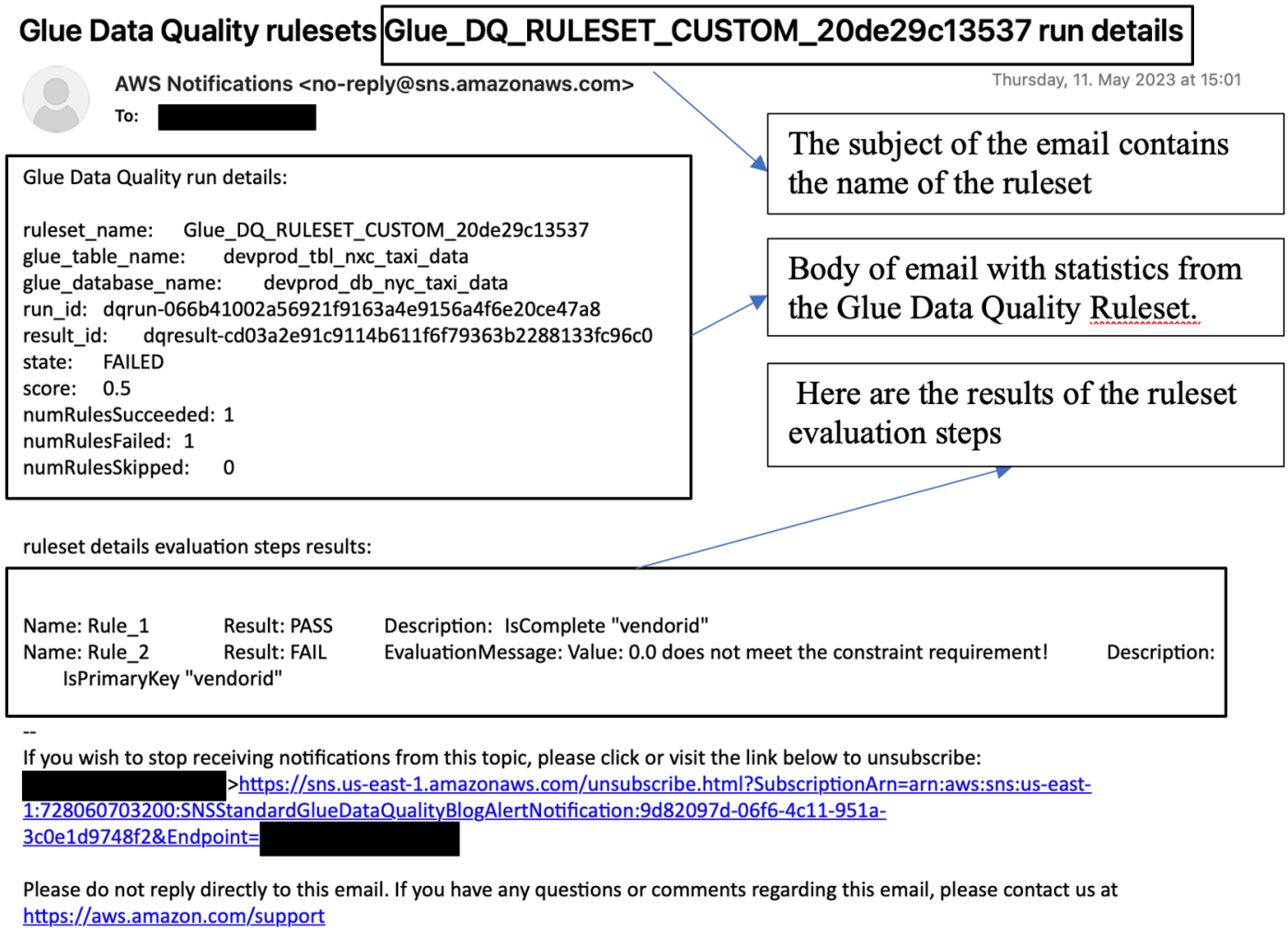
```
  "detail": {
    "context": {
      "contextType": ["GLUE_JOB"],
      "jobName": ["dq_evaluation_job1", "dq_evaluation_job2"]
    },
    "state": ["FAILED"]
  }
}
```

要捕获分数低于特定阈值（例如 70%）的事件，请执行以下操作：

```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
  "detail": {
    "score": [{
      "numeric": ["<=", 0.7]
    }]
  }
}
```

将通知格式化为电子邮件

有时，您需要向业务团队发送格式完善的电子邮件通知。您可以使用 Amazon EventBridge 和 AWS Lambda 来实现这一目标。



以下示例代码可用于格式化数据质量通知以生成电子邮件。

```
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 集成中设置警报和通知

我们推荐的方法是使用 Amazon 设置数据质量提醒 EventBridge，因为亚马逊 EventBridge 需要一次性设置才能提醒买家。但是， CloudWatch 由于熟悉亚马逊，有些买家更喜欢亚马逊。对于此类客户，我们提供与 Amazon 的集成 CloudWatch。

每 AWS 次 Glue 数据质量评估都会在每次数据质量运行时发出一对名为`glue.data.quality.rules.passed`（表示通过的规则数量）和`glue.data.quality.rules.failed`（表示失败的规则数）的指标。您可以使用此发出的指标创建警报，以便在给定的数据质量运行低于阈值时提醒用户。要开始设置可通过 Amazon SNS 通知发送电子邮件的警报，请按照以下步骤操作：

要开始设置可通过 Amazon SNS 通知发送电子邮件的警报，请按照以下步骤操作：

1. 打开 Amazon CloudWatch 控制台。
2. 在指标下选择所有指标。您将在自定义命名空间下看到一个名为 Glue Data Quality 的额外命名空间。

   > **Note**
   >
   > 开始运行 Glu AWS e 数据质量时，请确保已启用 “将指标发布到亚马逊 CloudWatch” 复选框。否则，该特定运行的指标将不会发布到 Amazon CloudWatch。

   在 `Glue Data Quality` 命名空间下，您可以看到每个表、每个规则集发出的指标。对于本主题，如果该值超过 1，我们将使用 `glue.data.quality.rules.failed` 规则和警报（表示，如果我们看到许多失败的规则评估大于 1，我们希望收到通知）。
3. 要创建警报，请在警报下选择所有警报。
4. 选择创建警报。
5. 选择选择指标。
6. 选择与您创建的表格相对应的 `glue.data.quality.rules.failed` 指标，然后选择选择指标。
7. 在指定指标和条件选项卡下的指标部分下：

   a. 对于 Statistic（统计数据），选择 Sum（总计）。

b. 对于周期，选择 1 分钟。

8. 在条件部分下：

a. 对于阈值类型，选择静态。

b. 对于每当 glue.data.quality.rules.failed 为...，选择大于/等于。

c. 对于不止于...，输入 1 作为阈值。

这些选择意味着，如果 glue.data.quality.rules.failed 指标发出的值大于或等于 1，我们将触发警报。但是，如果没有数据，我们会将其视为可接受。

9. 选择下一步。

10.在配置操作中：

a. 对于警报状态触发器，选择报警中。

b. 对于向以下 SNS 主题发送通知部分，选择创建新主题以通过新的 SNS 主题发送通知。

c. 对于将接收通知的电子邮件端点，请输入您的电子邮件地址。然后单击创建主题。

d. 选择下一步。

11对于警报名称，输入 myFirstDQAlarm，然后选择下一步。

12您会看到所有选择的摘要页面。选择底部的创建警报。

现在，您可以从 Amazon 警报控制面板中看到正在创建的 CloudWatch 警报。

查询数据质量结果以构建控制面板

您可能需要构建一个控制面板来显示您的数据质量结果。有两种方式可执行此操作：

EventBridge 使用以下代码设置亚马逊，将数据写入 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')
    }
```

写入 Amazon S3 后，您可以使用 AWS Glue 抓取工具注册到 Athena 并查询表。

在数据质量评估期间配置 Amazon S3 位置：

在 AWS Glue 数据目录或 AWS Glue ETL 中运行数据质量任务时，您可以提供一个 Amazon S3 位置，用于将数据质量结果写入亚马逊 S3。您可以使用以下语法通过引用目标来创建表，以读取数据质量结果。

请注意：必须分别运行 `CREATE EXTERNAL TABLE` 和 `MSCK REPAIR TABLE` 查询。

```
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>;
```

创建上表后，即可使用 Amazon Athena 运行分析查询。

使用部署数据质量规则 AWS CloudFormation

您可以使用 AWS CloudFormation 来创建数据质量规则。有关更多信息，AWS CloudFormation 请参阅 AWS Glue。

计划数据质量规则

您可以使用以下方法计划数据质量规则：

- 从数据目录中安排数据质量规则：任何代码用户都无法使用此选项轻松安排数据质量扫描。 AWS Glue Data Quality 将在亚马逊上创建时间表 EventBridge。要计划数据质量规则，请执行以下操作：
  - 导航到规则集并单击运行。
  - 在运行频率中，选择所需的计划并提供任务名称。此任务名称是您在中的日程安排的名称 EventBridge。
- 使用 Amazon EventBridge 和 AWS Step Functions 来协调数据质量规则的评估和建议。

对 AWS Glue 数据质量错误进行故障排除

如果您在 AWS Glue 数据质量中遇到错误，请使用以下解决方案来帮助您找到问题的根源并进行修复。

目录

错误：缺少 AWS Glue 数据质量模块

错误消息：没有名为“awsgluedq”的模块。

解决方法：当你在不支持的版本中运行 AWS Glue Data Quality 时，就会发生此错误。 AWS 仅有 Glue 版本 3.0 及更高版本支持 Glue 数据质量。

错误： AWS Lake Formation 权限不足

错误消息：用户类异常：impact_sdg_complation 上的 Lake Formation 权限不足（服务`com.amazonaws.services.glue.model.AccessDeniedException`：AWS Glue；状态码：400；错误代码：；请求编号：465ae693-b7ba-4df0-a4e4-6b17xxxxxxxx AccessDeniedException；代理：空）。

解决方案：您必须在 AWS Lake Formation 中提供足够的权限。

错误：规则集的名称不唯一

错误消息：用户类中存在异常：... services.glue.model。 AlreadyExistsException：另一个同名的规则集已经存在。

解决方法：规则集是全局的并且必须是唯一的。

错误：带有特殊字符的表

错误消息：用户类中存在异常：org.apache.spark.sql。 AnalysisException: 无法解析 “C” 给定输入列：
[primary.data_end_time、primary.data_start_time、primary.end_time、primary.data_start_time、primary.end

解决方法：目前存在一个限制，即无法在带有特殊字符（例如 “.”）的表上执行 AWS Glue Data Quality。

错误：规则集过大时发生溢出错误

错误消息：用户类中存在异常：java.lang。 StackOverflowError。

解决方法：如果您的大型规则集大于 2K 规则，则可能会遇到此问题。将您的规则分解为多个规则集。

错误：整体规则状态为失败

错误条件：我的规则集已成功，但我的整体规则状态为失败。

解决方案：之所以出现此错误，很可能是因为您在发布 CloudWatch 时选择了向 Amazon 发布指标的选项。如果您的数据集位于 VPC 中，则您的 VPC 可能不允 AWS 许 Glue 向亚马逊发布指标 CloudWatch。在这种情况下，您>必须为您的 VPC 设置终端节点才能访问亚马逊 CloudWatch。

AnalysisException: 无法验证是否存在默认数据库

错误条件:: 无法验证默认数据库是否存在 AnalysisException：com.amazonaws.services.glue.model。AccessDeniedException: 默认情况下 Lake Formation 权限不足（服务：AWS Glue；状态码：400；错误代码： AccessDeniedException；请求编号：XXXXXXX-XXXX-XXXX-XXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXXXX-XXXXXXX-XXXXXXX-XXX

解决方法：在 AWS Glue 作业的目录集成中，AWS Glue 始终尝试使用 AWS Glue `GetDatabase` API 检查默认数据库是否存在。如果未授予 `DESCRIBE` Lake Formation 权限或授予了 `GetDatabase` IAM 权限，则在验证默认数据库是否存在时，作业将失败。

要解决这个问题，请执行以下操作：

1. 在 Lake Formation 中添加默认数据库的 DESCRIBE 权限。
2. 在 Lake Formation 中将附加到 AWS Glue 作业的 IAM 角色配置为数据库创建者。这将自动创建默认数据库，并为该角色授予所需的 Lake Formation 权限。
3. 禁用 `--enable-data-catalog` 选项。（它在 AWS Glue Studio 中显示为使用 Data Catalog 作为 Hive 元存储）。

   如果作业中不需要 Spark SQL Data Catalog 集成，可以禁用。

错误消息：提供的键映射不适用于给定的数据帧

错误条件：提供的键映射不适用于给定的数据帧。

解决方案：您使用的是DataSetMatch规则类型，并且联接键有重复项。您的交集键必须唯一，并且不能为 NULL。如果您无法使用唯一的联接键，请考虑使用其他规则类型，例如AggregateMatch匹配摘要数据。

用户类中的异常：java.lang。 RuntimeException : 无法获取数据。查看登录 CloudWatch 以获取更多详细信息

错误条件：用户类异常：java.lang。 RuntimeException : 无法获取数据。查看登录 CloudWatch 以获取更多详细信息。

解决方案：当您在基于 Amazon S3 的表上创建 DQ 规则并与 Amazon RDS 或进行比较时，就会发生这种情况。 Amazon Redshift在这些情况下，AWS Glue 无法加载连接。相反，请尝试在 Amazon Redshift 或 Amazon RDS 数据集上设置 DQ 规则。这是一个已知错误。

启动错误：从 S3 下载存储桶时出错

错误条件：启动错误：从 S3 下载存储桶时出错：`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)` 。

解决方案：传递给 AWS Glue Data Quality 的角色权限必须允许从之前的 Amazon S3 位置进行读取。此 IAM policy 应附加到该角色：

```
{
  "Sid": "allowS3",
```

```
  "Effect": "Allow",
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::aws-glue-ml-data-quality-assets-<region>/*"
}
```

有关详细权限，请参阅 Data Quality authorization。这些库是评估数据集的数据质量所必需的。

InvalidInputException（状态：400）：无法解析 DataQuality 规则

错误条件： InvalidInputException（状态：400）：无法解析 DataQuality 规则。

解决方法：出现此错误的可能性很多。一种可能性是您的规则可能使用单引号。确认其已用双引号括起来。例如：

```
Rules = [
ColumnValues "tipo_vinculo" in ["CODO", "DOCO", "COCO", "DODO"] AND "categoria" = 'ES'
    AND "cod_bandera" = 'CEP'
```

将其更改为：

```
Rules = [
(ColumnValues "tipovinculo" in [ "CODO", "DOCO", "COCO", "DODO"]) AND (ColumnValues
 "categoria" = "ES")
    AND (ColumnValues "codbandera" = "CEP")
]
```

错误：Eventbridge 没有根据我设置的计划触发 Glue DQ 作业

错误条件：Eventbridge 没有根据我设置的计划触发 AWS Glue Data Quality 作业。

解决方法：触发作业的角色可能没有正确的权限。确保您用于启动作业的角色具有计划评估运行所需的 IAM 设置中提及的权限。

CustomSQL 错误

错误条件： `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`。

解决方法：SQL 查询有效，但请确认仅从主表中选择列。从主函数中选择聚合函数（例如 sum、count 列）可能会导致此错误。

错误条件： `There was a problem when executing your SQL statement: cannot resolve "Col"`。

解决方法：主表中不存在此列。

错误条件： `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"`。

解决方法：在 SQL 查询中，当您将主表与其他引用表联接时，确认您的 select 语句仅包含主表中的列名，以便为主表生成行级结果。

动态规则

错误条件: `Dynamic rules require job context, and cannot be evaluated in interactive session or data preview.`。

原因：当规则集中存在动态 DQ 规则时，此错误消息可能会出现在您的数据预览结果，或其他交互式会话中。动态规则引用与特定作业名称和评估上下文相关的历史指标，因此无法在交互式会话中对其进行评估。

解决方法：运行 AWS Glue 作业将生成历史指标，可在以后的作业运行中为相同的作业引用这些指标。

错误条件：

- `[RuleType] rule only supports simple atomic operands in thresholds.`.
- `Function last not yet implemented for [RuleType] rule.`

解决方法：数字表达式中的所有 DQDL 规则类型通常都支持动态规则（请参阅 DQDL 参考）。但是，尚不支持某些生成多个指标 ColumnValues 和 ColumnLength的规则。

错误条件： `Binary expression operands must resolve to a single number.`。

原因：动态规则支持二进制表达式，如 RowCount > avg(last(5)) * 0.9。此处二进制表达式为 avg(last(5)) * 0.9。此规则有效，因为两个操作数 avg(last(5)) 和 0.9 均解析为一个数字。RowCount > last(5) * 0.9 是一个错误示例，因为 last(5) 会生成一个无法与当前行数进行有意义比较的列表。

解决方法：使用聚合函数将列表值操作数缩减为单个数字。

错误条件：

- 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：动态规则可用于将数据集的某些特征，与其历史值进行比较。如果提供了正整数参数，则最后一个函数可以检索多个历史值。例如，last(5) 将检索您的规则在作业运行时观测到的最近五个值。

解决方法：必须使用聚合函数将这些值缩减为单个数字，才能与当前作业运行时观测到的值进行有意义的比较。

有效示例：

- RowCount >= avg(last(5))
- RowCount > last(1)
- RowCount < last()

无效示例：RowCount > last(5)。

错误条件：

- 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.

解决方法：在编写动态规则时，您可以使用 index 聚合函数从列表中选择一个历史值。例如 RowCount > index(last(5)，1) 将检查在当前作业中观测到的行数，是否严格大于在作业中观测到的次新行数。index 从零开始编制索引。

错误条件： `IllegalArgumentException: Parsing Error: Rule Type: DetectAnomalies is not valid`。

解决方法：异常检测仅在 AWS Glue 4.0 中提供。

错误条件： `IllegalArgumentException: Parsing Error: Unexpected condition for rule of type ... no viable alternative at input ...`。

注意：`...` 是动态的。示例：`IllegalArgumentException: Parsing Error: Unexpected condition for rule of type RowCount with number return type, line 4:19 no viable alternative at input '>last'`。

解决方法：异常检测仅在 AWS Glue 4.0 中提供。

用户类中的异常：org.apache.spark.sql。 AnalysisException: org.apache.hadoop.hive.ql.metadata。 HiveException

错误条件：`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：你在 Glue 数据目录中使用 Apache Iceberg，而 AWS GI AWS ue 数据目录中的 “输入格式” 属性为空。

解决方法：当您在 DQ 规则中使用 CustomSQL 规则类型时，会出现此问题。解决这个问题的一种方法是使用“主要”或将目录名称 `glue_catalog.` 添加到 `<database>.<table> in Custom ruletype`。

UNCLASSIVED_ERROR; IllegalArgumentException: 解析错误：未提供规则或分析器。，输入时没有可行的替代方案

错误条件：`UNCLASSIFIED_ERROR; IllegalArgumentException: Parsing Error: No rules or analyzers provided., no viable alternative at input`

解决方案：DQDL 不可解析。在某些情况下，可能会发生这种情况。如果您使用的是复合规则，请确保它们有右括号。

```
(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 数据集成在 AWS Glue

Amazon Q 数据集成 AWS Glue 是一种新的生成式 AI 功能 AWS Glue ，它使数据工程师和 ETL 开发人员能够使用自然语言构建数据集成作业。工程师和开发人员可以让 Amazon Q 撰写作业、解决问题并回答有关 AWS Glue 数据集成的问题。

什么是 Amazon Q？

> **Note**
>
> 由 Amazon Bedrock 提供支持： AWS 实现自动滥用检测。由于 Amazon Q 数据集成功能基于 Amazon Bedrock 构建，因此用户可以充分利用 Amazon Bedrock 中实施的控制措施，以便安全、负责任地使用人工智能（AI）。

Amazon Q 是一款生成式人工智能 (AI) 驱动的对话助手，可以帮助您理解、构建、扩展和操作 AWS 应用程序。为Amazon Q提供支持的模型已添加了高质量的 AWS 内容，可为您提供更完整、更具可操作性和参考性的答案，从而加快您的构建。 AWS有关更多信息，请参阅什么是 Amazon Q？

什么是 AWS Glue中的 Amazon Q 数据集成？

中的 Amazon Q 数据集成 AWS Glue 包括以下功能：

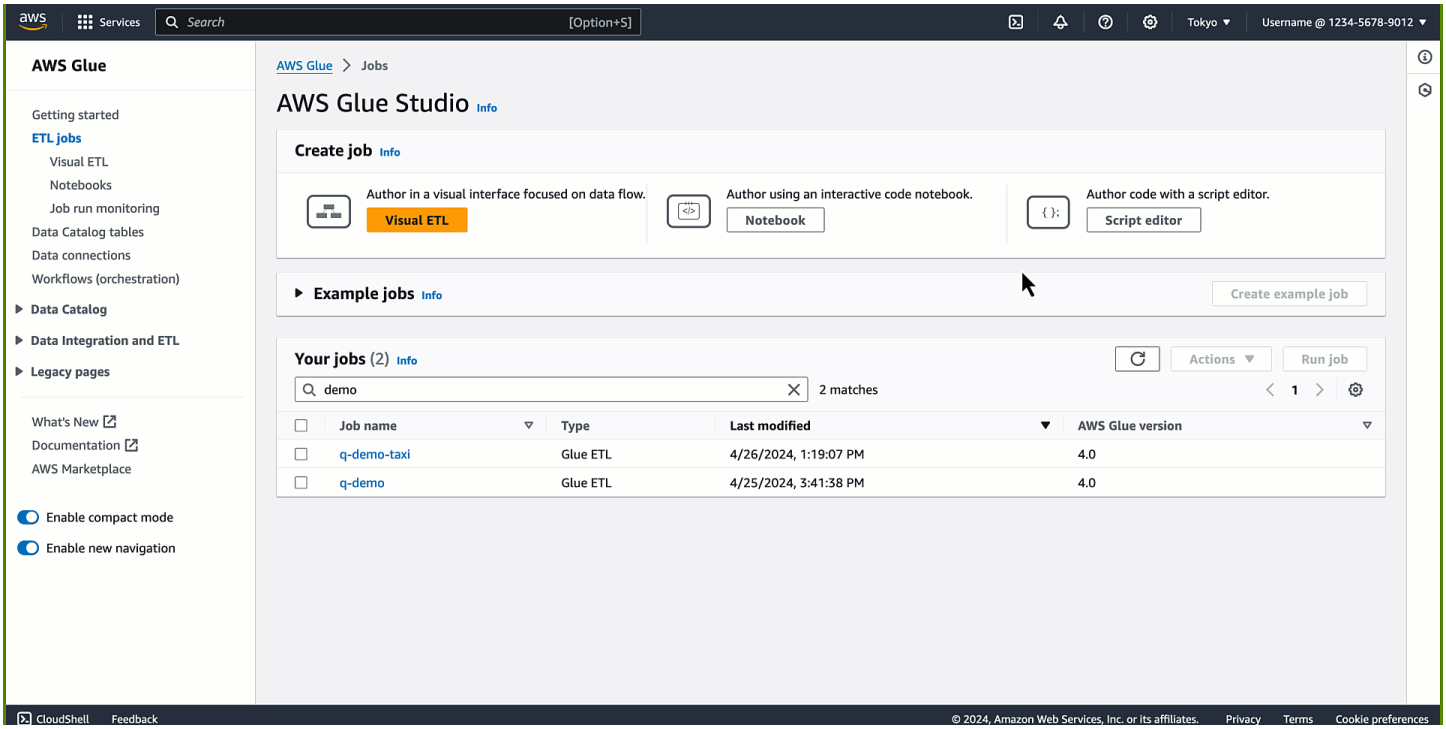
- 聊天 — 中的 Amazon Q 数据集成 AWS Glue 可以用英语回答有关 AWS Glue 数据集成领域的自然语言问题，例如 AWS Glue 源和目标连接器、 AWS Glue ETL 作业、数据目录 AWS Lake Formation、爬虫和其他功能文档和最佳实践。Amazon Q 数据集成以 step-by-step 说明作为 AWS Glue 响应，并包括对其信息源的引用。
- 数据集成代码生成 — Amazon Q 中的数据集成 AWS Glue 可以回答有关 AWS Glue ETL 脚本的问题，并根据英语自然语言问题生成新代码。
- 疑难解答 — 中的 AWS Glue Amazon Q 数据集成旨在帮助您了解 AWS Glue 任务中的错误，并提供问题根源和解决问题的 step-by-step 说明。

> **Note**
>
> 中的 Amazon Q 数据集成 AWS Glue 不会使用您的对话背景来告知您对话期间的未来回复。与 Amazon Q 数据集成的每次对话 AWS Glue 都独立于您之前或将来的对话。

使用 AWS Glue中的 Amazon Q 数据集成？

在 Amazon Q 面板中，您可以请求 Amazon Q 为 AWS Glue ETL 脚本生成代码，或者回答有关 AWS Glue 功能的问题或对错误进行故障排除。响应是一个 ETL 脚本， PySpark 其中包含自定义脚本、查看和执行脚本的 step-by-step 说明。对于问题，将根据数据集成知识库生成回复，并含有摘要和来源 URL 供参考。

例如，您可以让 Amazon Q “请提供一个从 Snowflake 读取、重命名字段并写入 Redshift 的 Glue 脚本”，作为响应，Amazon Q 数据集成 AWS Glue 将返回一个可以执行所请求操作的 AWS Glue 任务脚本。您可以查看生成的代码，确保其满足请求的意图。如果满意，则可以将其部署为生产中的 AWS Glue 作业。您可以要求集成说明错误和故障，并提出解决方案，从而对作业进行故障排除。Amazon Q 可以回答有关 AWS Glue 我们的数据集成最佳实践的问题。



以下是示例问题，演示了 Amazon Q 数据集成 AWS Glue 如何帮助您在此基础上再接再厉 AWS Glue：

AWS Glue ETL 代码生成：

- 编写一个从 S3 读取 JSON、使用应用映射转换字段并写入 Amazon Redshift 的 AWS Glue 脚本
- 如何编写 AWS Glue 脚本，用于从 DynamoDB 读取、应用转换并以 Parquet DropNullFields 的形式写入 S3？
- 给我一个 AWS Glue 脚本，它可以从 MySQL 读取，根据我的业务逻辑删除一些字段，然后写入 Snowflake
- 写一个 AWS Glue 任务来从 DynamoDB 读取然后以 JSON 的形式写入 S3
- 帮我开发一个 S3 AWS Glue 数据目录的 AWS Glue 脚本
- 写一个 AWS Glue 任务来从 S3 读取 JSON，删除空值然后写入 Redshift

AWS Glue 功能说明：

- 如何使用 AWS Glue 数据质量？
- 如何使用 AWS Glue 招聘书签？
- 如何启用 AWS Glue 自动缩放？
- AWS Glue 动态帧和 Spark 数据帧有什么区别？
- 支持哪些不同类型的连接 AWS Glue？

AWS Glue 疑难解答：

- 如何解决 AWS Glue 作业中的内存不足 (OOM) 错误？
- 在设置 AWS Glue 数据质量时，您可能会看到哪些错误消息？如何修复它们？
- 如何修复错误为 Amazon S3 访问被拒绝的 AWS Glue 任务？
- 如何解决 AWS Glue 作业数据随机排列的问题？

与 Amazon Q 数据集成交互的最佳实践

以下是与 Amazon Q 数据集成交互的最佳实践：

- 在与 Amazon Q 数据集成交互时，请提出具体问题，在有复杂请求时进行迭代，并验证答案是否准确。
- 在以自然语言提供数据集成提示时，请尽可能具体，以帮助助手准确了解您的需求。与其询问 “从 S3 中提取数据”，不如提供更多详细信息，例如 “编写从 S3 中提取 JSON 文件的 AWS Glue 脚本”。

- 在运行生成的脚本之前，请对其进行检查，以确保准确性。如果生成的脚本有错误或与您的意图不符，请向助手提供有关如何更正该脚本的说明。
- 生成式人工智能技术是一项全新的技术，其回复中可能会出现错误，有时将这种错误称为幻觉。在您的环境或工作负载中使用代码之前，请对所有代码进行测试并检查是否存在错误和漏洞。

AWS Glue 服务改进中的 Amazon Q 数据集成

为了帮助 Amazon Q 数据集成 AWS Glue 提供最相关的 AWS 服务信息，我们可能会使用 Amazon Q 中的某些内容，例如您向 Amazon Q 提出的问题及其回复，以改进服务。

有关我们使用哪些内容以及如何选择退出的信息，请参阅 Amazon Q 开发者用户指南中的 Amazon Q 开发者服务改进。

注意事项

在 AWS Glue中使用 Amazon Q 数据集成之前，请考虑以下各项：

- 目前，代码生成仅适用于内 PySpark 核。生成的代码适用于基于 Python Spark 的 AWS Glue 作业。
- 有关中支持的 Amazon Q 数据集成代码生成功能组合的信息 AWS Glue，请参阅支持的代码生成功能。

在中设置 Amazon Q 数据集成 AWS Glue

以下各节提供在 AWS Glue中设置 Amazon Q 数据集成的信息。

主题

- 配置 IAM 权限

配置 IAM 权限

本主题介绍您为 Amazon Q 聊天体验和 AWS Glue Studio 笔记本体验配置的 IAM 权限。

主题

- 为 Amazon Q 聊天配置 IAM 权限
- 为 AWS Glue Studio 笔记本配置 IAM 权限

为 Amazon Q 聊天配置 IAM 权限

向 Amazon Q 数据集成中使用的 API 授予权限 AWS Glue 需要相应的 AWS 身份和访问管理 (IAM) 权限。您可以通过将以下自定义 AWS 策略附加到您的 IAM 身份（例如用户、角色或群组）来获取权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:StartCompletion",
                "glue:GetCompletion"
            ],
            "Resource": [
                "arn:aws:glue:*:*:completion/*"
            ]
        }
    ]
}
```

为 AWS Glue Studio 笔记本配置 IAM 权限

要在 AWS Glue Studio 笔记本中启用 Amazon Q 数据集成，请确保为笔记本 IAM 角色附加以下权限：

```
{
    "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 数据集成中 AWS Glue 没有通过 AWS 软件开发工具包提供的 API 可供您以编程方式使用。IAM 策略中使用了以下两个 API，用于通过 Amazon Q 聊天面板或 AWS Glue Studio 笔记本实现这种体验：`StartCompletion`和`GetCompletion`。

分配权限

要提供访问权限，请为您的用户、组或角色添加权限：

- AWS IAM 身份中心中的用户和群组：创建权限集。按照《AWS IAM Identity Center 用户指南》中 Create a permission set 部分的说明进行操作。
- 通过身份提供商在 IAM 中管理的用户：创建身份联合验证角色。按照《IAM 用户指南》中为第三方身份提供商创建角色（联合身份验证）的说明进行操作。
- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

支持的代码生成功能

以下是 AWS Glue中 Amazon Q 数据集成的代码生成功能的组合。

来源	Transformation	目标
具有以下格式类型的 S3：json、csv、parquet、hudi、delta	ApplyMapping	具有以下格式类型的 S3：json、csv、avro、orc、parquet、hudi、delta

来源	Transformation	目标
Glue 数据目录	RenameField	Glue 数据目录
Amazon Redshift	DropFields	Amazon Redshift
MySQL	SelectFields	MySQL
Postgres	DropNullFields	Postgres
Oracle	筛选条件	Oracle
SQL Server	Spigot	SQL Server
DynamoDB	自定义 SQL 代码	DynamoDB
Snowflake	聚合	Snowflake
MongoDB	DropDuplicates	MongoDB
自定义 JDBC 连接器	联接	自定义 JDBC 连接器
自定义 Spark 连接器	Union	自定义 Spark 连接器
谷歌 BigQuery		谷歌 BigQuery
Teradata		Teradata
亚马逊 OpenSearch 服务		亚马逊 OpenSearch 服务
Vertica		Vertica
天蓝 SQL		Azure DQL
SAP HANA		SAP HANA
蔚蓝宇宙		蔚蓝宇宙

示例交互

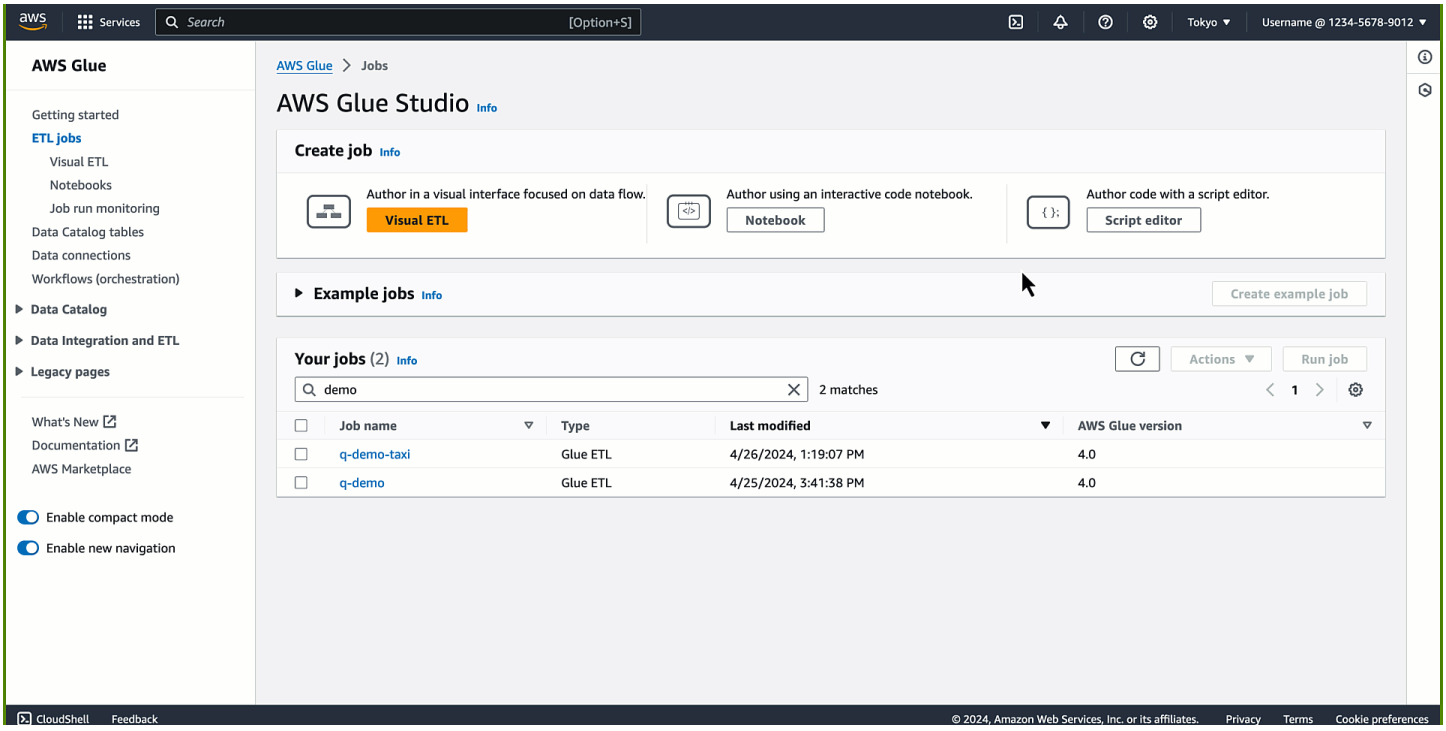
Amazon Q 数据集成 AWS Glue 允许您在 Amazon Q 面板中输入您的问题。您可以输入有关 AWS Glue所提供数据集成功能的问题。系统将返回详细答案以及参考文档。

另一个用例是生成 AWS Glue ETL 作业脚本。您可以询问有关如何执行数据提取、转换、加载作业的问题。将返回生成的 PySpark 脚本。

主题

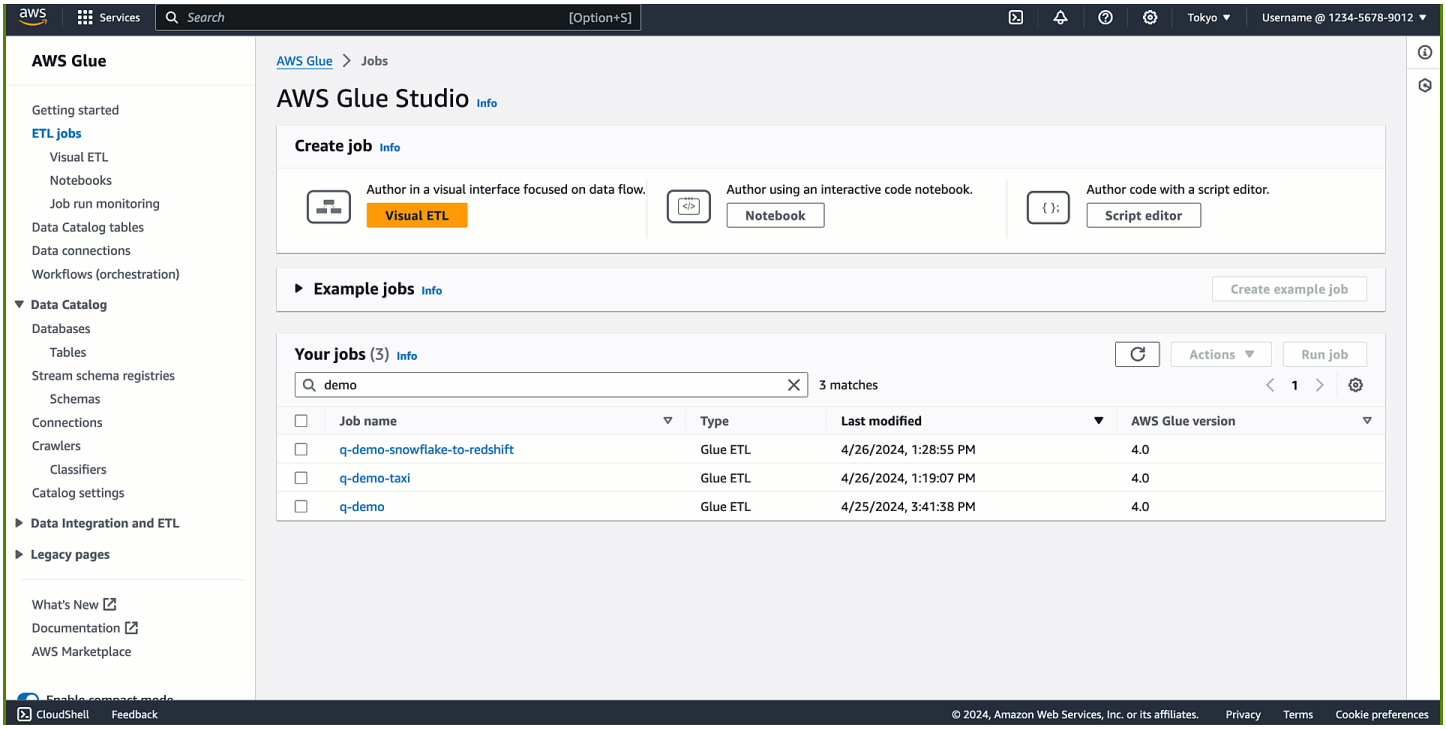
亚马逊 Q 聊天互动

在 AWS Glue 控制台上，开始创作新作业，然后询问 Amazon Q：“请提供一个从 Snowflake 读取、重命名字段并写入 Redshift 的 Glue 脚本。”



您会注意到代码已生成。通过这个回复，你可以学习和理解如何为自己的目的编写 AWS Glue 代码。您可以将生成的代码复制/粘贴到脚本编辑器并配置占位符。在作业上配置 AWS 身份和访问管理 (IAM) Access Management 角色 AWS Glue 和连接后，保存并运行该作业。任务完成后，您可以开始在 Amazon Redshift 中查询从 Snowflake 导出的表。

以下提示读取来自两个不同来源的数据，分别对它们进行筛选和投影，在公用键上联接，然后将输出写入第三个目标。提问 Amazon Q：“我想从 S3 中读取 Parquet 格式的数据，然后选择一些字段。我还想从 DynamoDB 读取数据，选择一些字段，然后筛选一些行。我想合并这两个数据集并将结果写入 OpenSearch。

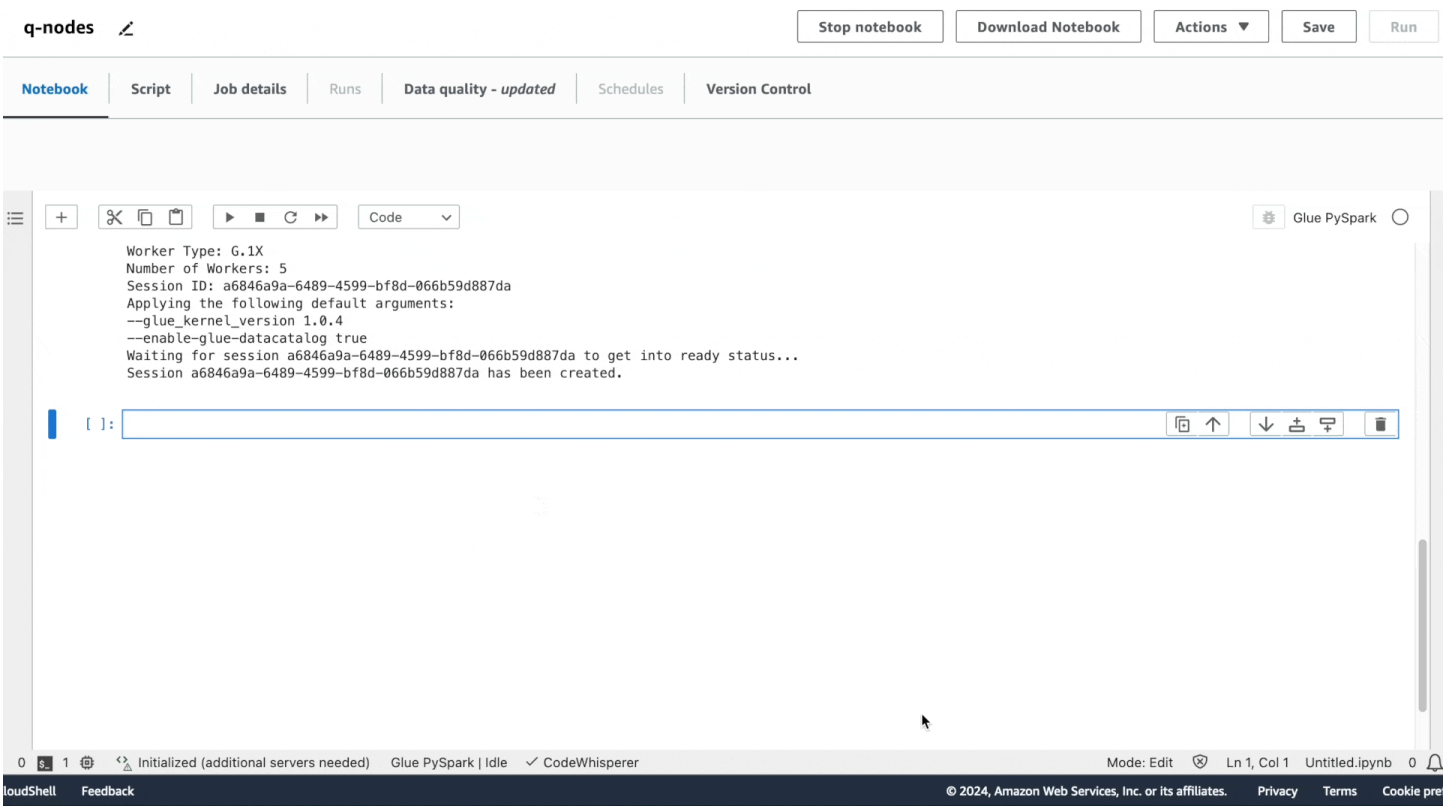


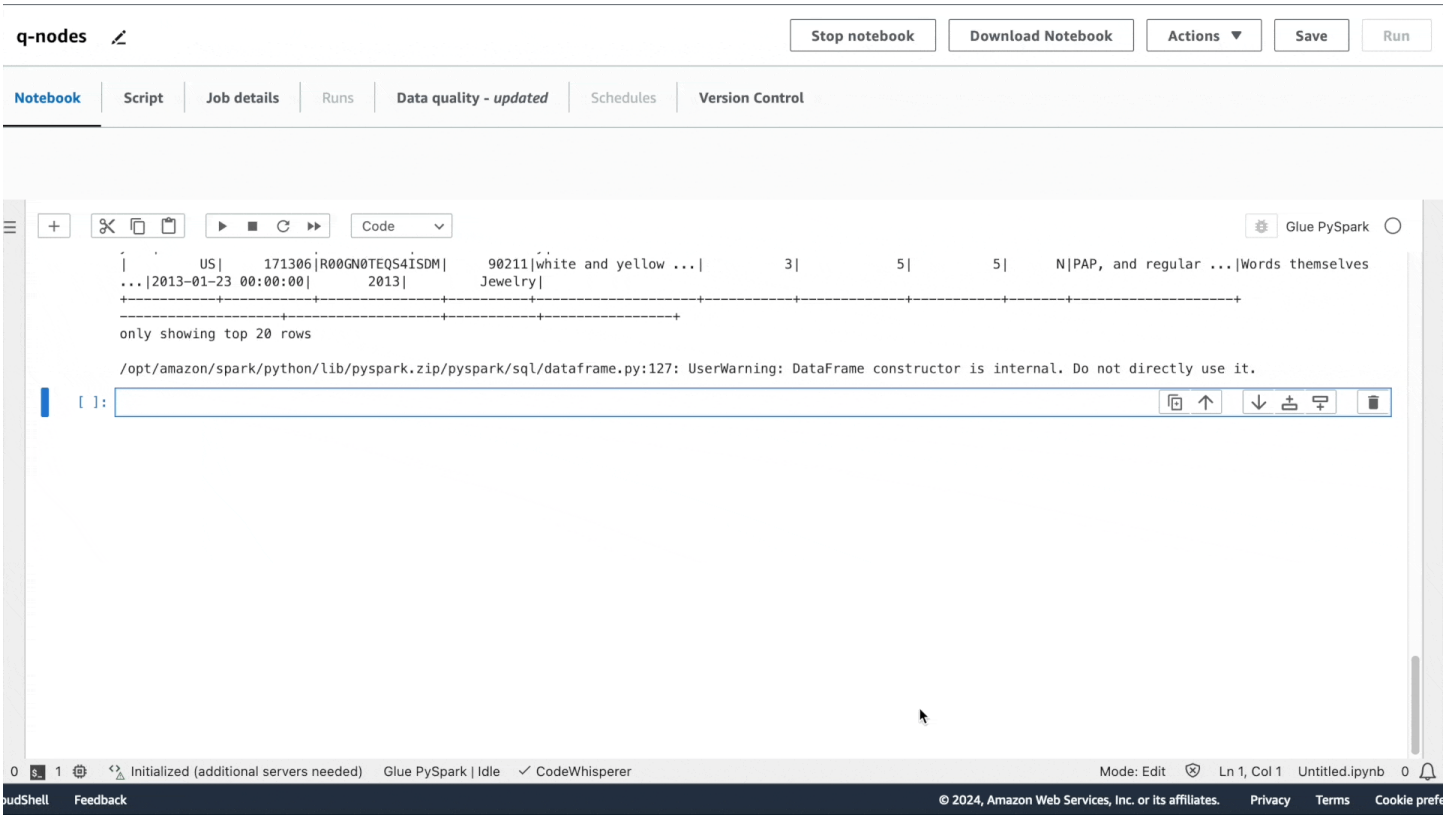
代码已生成。任务完成后，您的索引将在中可用 OpenSearch 并可供下游工作负载使用。

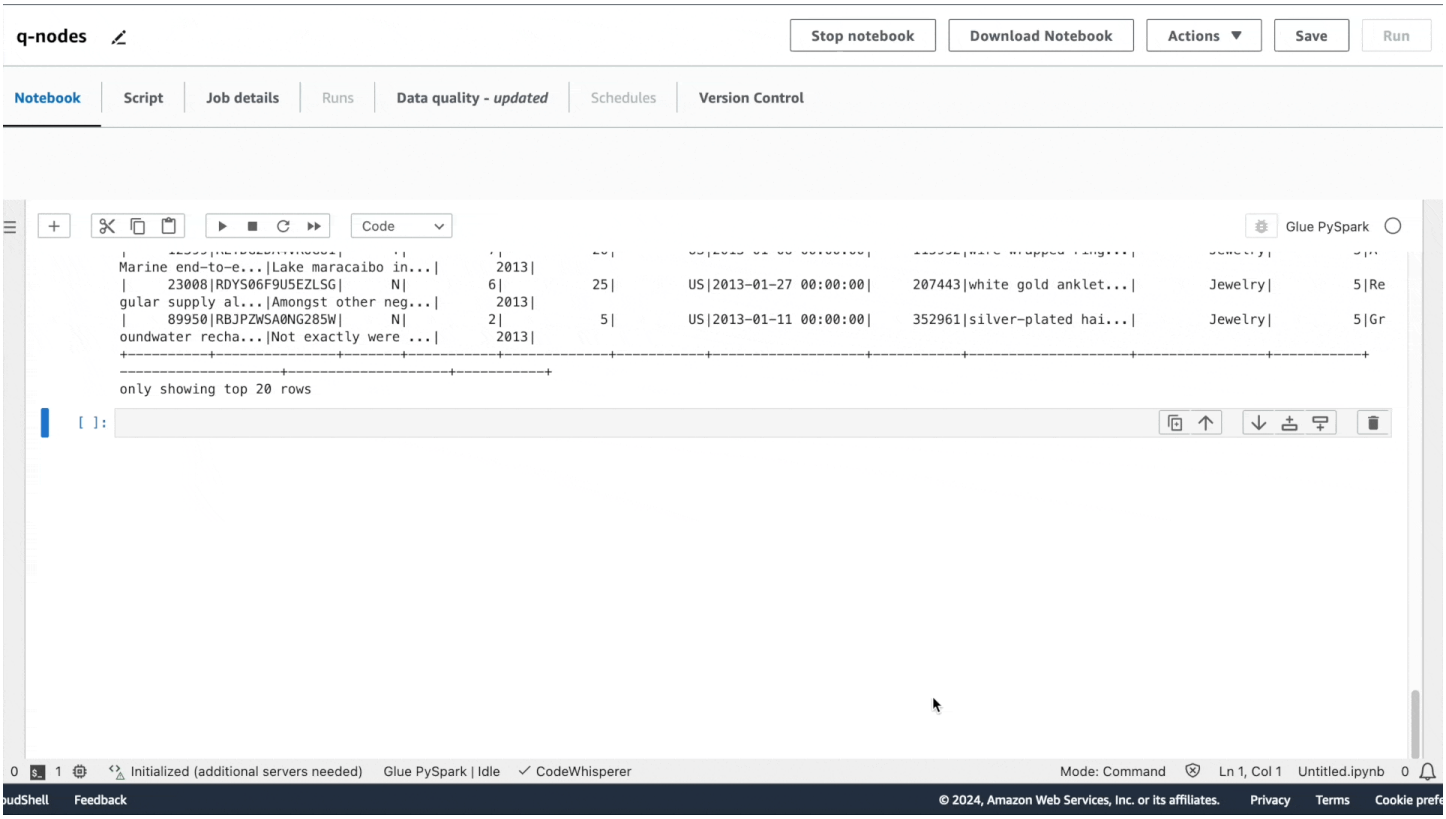
AWS Glue 工作室笔记本互动

添加一个新单元格并输入您的评论以描述您想要实现的目标。按 Tab 键和 Enter 键后，将显示推荐的代码。

第一个意图是提取数据：“给我读取 Glue Data Catalog 表的代码”，然后是 “给我代码以应用 star_rating>3 的过滤器转换” 和 “给我把帧作为 Parquet 写入 S3 的代码”。


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

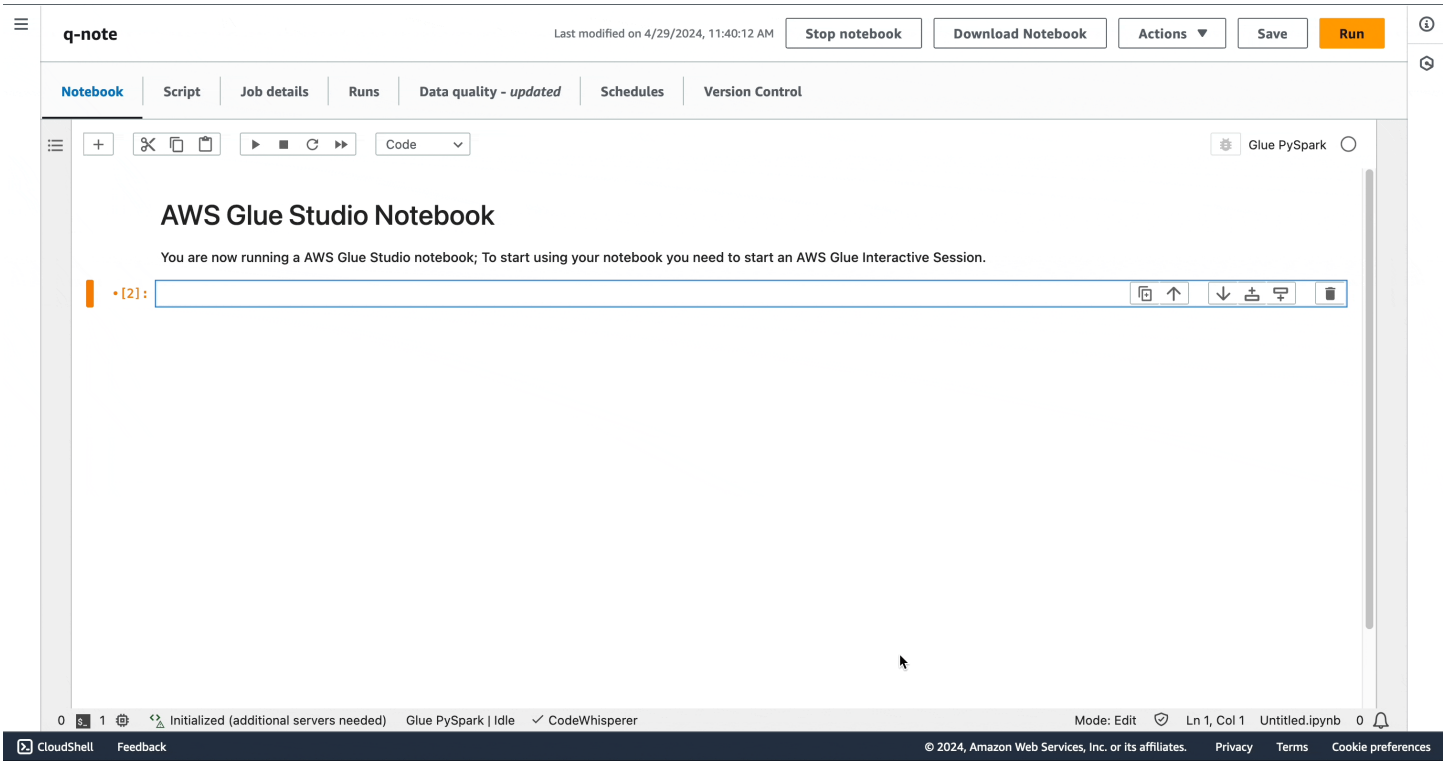


与 Amazon Q 聊天体验类似，推荐使用该代码。如果按 Tab 键，则会选择推荐的代码。

您可以通过在生成的代码中填写源代码的相应选项来运行每个单元。在运行过程中的任何时候，您也可以使用该show( )方法预览数据集的样本。

复杂的提示

您可以使用单个复杂提示生成完整的脚本。“我有 S3 中的 JSON 数据和 Oracle 中的数据需要合并。请提供一个 Glue 脚本，该脚本可以从两个来源读取，进行联接，然后将结果写入 Redshift。”



您可能会注意到，在笔记本上，Amazon Q 数据集成 AWS Glue 生成的代码片段与 Amazon Q 聊天中生成的代码片段相同。

您可以通过选择 “运行” 或以编程方式将笔记本作为作业运行。

AWS Glue 中的编排

以下几个部分提供有关在 AWS Glue 中编排作业的信息。

主题

使用触发器启动作业和爬网程序

在 AWS Glue 中，您可以创建称为触发器的数据目录对象，用于手动或自动启动一个或多个爬网程序或提取、转换、加载（ETL）任务。使用触发器，您可以设计一个相互依赖的作业和爬网程序链条。

> **Note**
>
> 您也可以通过定义 工作流程来实现相同的目的。工作流程是创建复杂的多作业 ETL 操作的首选方式。有关更多信息，请参阅 the section called “使用蓝图和工作流执行复杂的 ETL 活动”。

主题

AWS Glue 触发器

触发器在触发时可以启动指定的作业和爬网程序。触发器可以根据需要、基于计划或基于事件组合触发。

> **Note**
>
> 一个触发器只能激活两个爬网程序。如果要爬取多个数据存储，请为每个爬网程序使用多个源，而不是同时运行多个爬网程序。

触发器可以处于几种状态之一。触发器可以处于 CREATED、ACTIVATED 或 DEACTIVATED 状态。此外，还有一些过渡状态，例如 ACTIVATING。要暂停触发器的触发，您可以将其停用。之后您可以重新激活它。

有三种类型的触发器：

已安排

基于 cron 的定时触发器。

您可以为一组作业或爬网程序创建基于计划的触发器。您可以指定约束条件，例如作业或爬网程序的运行频率、它们在一周中的哪几天运行，以及具体在什么时间运行。这些约束基于 cron。当您为触发器设置计划时，需要考虑 cron 的功能和限制。例如，如果您选择在每月第 31 天运行您的爬网程序，请记住，有些月份没有 31 天。有关 cron 的更多信息，请参阅用于作业和爬网程序的基于时间的计划。

条件

在上一个作业或爬网程序或多个作业或爬网程序满足条件列表中的条件时触发的触发器。

创建条件触发器时，您可以指定要监控的作业列表和爬网程序列表。对于每个受监控的作业或爬网程序，您可以指定要监控的状态，例如成功、失败、超时等。当受监控的作业或爬网程序进入指定的状态时，触发器就会触发。您可以将触发器配置为在任何或所有受监控的事件发生时触发。

例如，您可以将触发器 T1 配置为在作业 J1 和作业 J2 都成功完成时启动作业 J3，并将另一个触发器 T2 配置为在作业 J1 或作业 J2 失败时启动作业 J4。

下表列出了触发器监控的作业和爬网程序完成状态（事件）。

作业完成状态	爬网程序完成状态
• SUCCEEDED • STOPPED • FAILED • TIMEOUT	• SUCCEEDED • FAILED • CANCELLED

按需

在激活时触发的触发器。按需触发器永远不会进入 ACTIVATED 或 DEACTIVATED 状态。它们始终处于 CREATED 状态。

因此，它们一旦存在就可以立即触发。而对于计划触发器和条件触发器，则可以设置一个标记，用于在创建时激活它们。

> **Important**
>
> 作为其他作业或爬网程序完成的结果而运行的作业或爬网程序称为依赖项。仅当完成的作业或爬网程序由触发器启动时，才会启动依赖作业或爬网程序。依赖链中的所有作业或爬网程序都必须是单个计划或按需触发器的子代。

通过触发器传递作业参数

触发器可以将参数传递给它所启动的作业。参数包括作业参数、超时值、安全配置等。如果触发器启动多个作业，则参数会传递给每个作业。

以下是触发器传递作业参数的规则：

- 如果键值对中的键与默认作业参数匹配，则传递的参数将覆盖默认参数。如果键与默认参数不匹配，则该参数将作为附加参数传递给作业。
- 如果键值对中的键与不可覆盖的参数匹配，则会忽略传递的参数。

有关更多信息，请参阅 AWS Glue API 中的the section called “触发”。

添加触发器

您可以使用 AWS Glue 控制台、 AWS Command Line Interface (AWS CLI) 或 AWS Glue API 添加触发器。

> **Note**
>
> 目前， AWS Glue 控制台在使用触发器时仅支持作业，而不支持爬网程序。您可以使用 AWS CLI 或 AWS Glue API 来配置同时支持作业和爬网程序的触发器。

添加触发器（控制台）

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格中，于 ETL 下，选择 Triggers (触发器)。选择 Add trigger (添加触发器)。

3. 提供以下属性：

   名称

   赋予您的触发器一个唯一的名称。

   触发器类型

   指定下列项之一：

   - Schedule (计划)：触发器按特定的频率和时间触发。
   - Job events (作业事件)：条件触发器。当列表中的任何或所有作业进入其指定状态时，触发器就会触发。要让触发器触发，受监控的作业必须由触发器启动。对于您选择的任何作业，您只能监控一个作业事件（完成状态）。
   - On-demand (按需)：触发器会在激活时触发。

4. 完成触发器向导。在 Review (审查) 页面上，您可以选择 Enable trigger on creation (在创建时启用触发器)，立即激活 Schedule (计划) 和 Job events (任务事件)（条件）触发。

添加触发器 (AWS CLI)

- 输入类似以下的命令。

```
aws glue create-trigger --name MyTrigger --type SCHEDULED --schedule  "cron(0 12 *
 * ? *)" --actions CrawlerName=MyCrawler --start-on-creation
```

此命令会创建一个名为 `MyTrigger` 的计划触发器 ，该触发器每天在 UTC 时间中午 12:00 运行，并启动一个名为 `MyCrawler` 的爬网程序。该触发器在创建时处于激活状态。

有关更多信息，请参阅 the section called “AWS Glue 触发器”。

用于作业和爬网程序的基于时间的计划

您可以在 AWS Glue 中定义用于作业和爬网程序的基于时间的计划。这些计划的定义使用类似于 Unix 的 cron 语法。您可以按 协调世界时 (UTC) 指定时间，计划的最小精度是 5 分钟。

要了解有关配置任务和爬网程序以使用计划运行的详细信息，请参阅使用触发器启动作业和爬网程序。

Cron 表达式

Cron 表达式有六个必填字段，之间以空格分隔。

语法

```
cron(Minutes Hours Day-of-month Month Day-of-week Year)
```

字段	值	通配符
分钟	0-59	, - * /
小时	0-23	, - * /
日期	1-31	, - * ? / L W
月	1-12 或 JAN-DEC	, - * /
星期几	1-7 或 SUN-SAT	, - * ? / L
年	1970-2199	, - * /

通配符

- , (逗号) 通配符包含其他值。在 Month 字段中，JAN,FEB,MAR 将包含 January、February 和 March。
- - (破折号) 通配符用于指定范围。在 Day 字段中，1-15 将包含指定月份的 1-15 日。
- * (星号) 通配符包含该字段中的所有值。在 Hours 字段中，* 将包含每个小时。
- / (正斜杠) 通配符用于指定增量。在 Minutes 字段中，您可以输入 **1/10** 以指定从一个小时的第一分钟开始的每个第十分钟 (例如，第 11 分钟、第 21 分钟和第 31 分钟) 。
- ? (问号) 通配符用于指定一个或另一个。在 Day-of-month 字段中，您可以输入 7，如果您不介意 7 日是星期几，则可以在“星期几”字段中输入 ?。
- 或 字段中的 Day-of-monthLDay-of-week 通配符用于指定月或周的最后一天。
- Day-of-month 字段中的 W 通配符用于指定工作日。在 Day-of-month 字段中，3W 用于指定最靠近当月的第三周的日。

限制

- 您无法在同一 cron 表达式中为 Day-of-month 和 Day-of-week 字段同时指定值。如果您在其中一个字段中指定了值，则必须在另一个字段中使用 ? (问号) 。

- 不支持产生的速率快于 5 分钟的 Cron 表达式。

示例

在创建计划时，您可以使用以下示例 cron 字符串。

分钟	小时	日期	月份	星期几	年	含义
0	10	*	*	?	*	每天上午的 10:00 (UTC) 运行
15	12	*	*	?	*	每天在下午 12:15 (UTC) 运行
0	18	?	*	MON-FRI	*	每星期一到星期五的下午 6:00 (UTC) 运行
0	8	1	*	?	*	每月第 1 天上午 8:00 (UTC) 运行
0/15	*	*	*	?	*	每 15 分钟运行一次
0/10	*	?	*	MON-FRI	*	从星期一到星期五，每 10 分钟运行一次
0/5	8-17	?	*	MON-FRI	*	每星期一到星期五的上午 8:00 和下午 5:55 (UTC) 之

分钟	小时	日期	月份	星期几	年	含义
						间，每 5 分钟运行一次

例如，要按计划在每天 12:15 UTC 运行，请指定：

```
cron(15 12 * * ? *)
```

激活和停用触发器

您可以使用AWS Glue控制台、 AWS Command Line Interface (AWS CLI) 或 AWS Glue API 激活或停用触发器。

激活或停用触发器（控制台）

1. 登录 AWS Management Console 并打开AWS Glue控制台，网址为 https://console.aws.amazon.com/glue/。
2. 在导航窗格中，于 ETL 下，选择 Triggers (触发器)。
3. 选中所需触发器旁边的复选框，然后在 Action (操作) 菜单上选择 Enable trigger (启用触发器) 以激活触发器，或选择 Disable trigger (禁用触发器) 以停用触发器。

激活或停用触发器 (AWS CLI)

- 输入以下命令之一。

  ```
  aws glue start-trigger --name MyTrigger

  aws glue stop-trigger --name MyTrigger
  ```

  启动触发器会激活触发器，停止触发器会停用触发器。当您激活按需触发器时，它会立即触发。

有关更多信息，请参阅 the section called “AWS Glue 触发器”。

使用 AWS Glue 中的蓝图和工作流执行复杂的 ETL 活动

您组织的一些复杂的提取、转换和加载（ETL）流程最好通过使用多个相互依赖的 AWS Glue 任务和爬网程序来实施。使用 AWS Glue 工作流，您可以设计一个复杂的多任务、多爬网程序 ETL 流程，AWS Glue 可以将其作为单个实体运行和跟踪。创建工作流并指定工作流中的任务、爬网程序和触发器后，您可以按需或按计划运行工作流。

主题

AWS Glue 中的工作流概述

在 AWS Glue 中，可以使用工作流程创建和可视化涉及多个爬网程序、作业和触发器的复杂的提取、转换和加载 (ETL) 活动。每个工作流都管理其所有任务和爬网程序的执行和监控。当工作流运行每个组件时，它会记录执行进度和状态。这将为您提供大型任务的概览和每个步骤的详细信息。AWS Glue 控制台以图表形式呈现工作流。

您可以使用 AWS Glue 蓝图创建工作流，也可以使用AWS Management Console或 AWS Glue API 手动构建组件工作流程。有关蓝图的更多信息，请参阅 the section called “蓝图概览”。

工作流中的触发器可以启动任务和爬网程序，也可以由任务或爬网程序触发。使用触发器，您可以创建相互依赖的任务和爬网程序的大型链。除了定义任务和爬网程序依赖关系的工作流中的触发器之外，每个工作流都有启动触发器。有三种类型的启动触发器：

- 计划 – 工作流程根据您定义的计划启动。计划可以是每天、每周、每月执行等，也可以是基于 cron 表达式的自定义计划。
- 按需：工作流将从 AWS Glue 控制台、API、或 AWS CLI 手动启动。
- EventBridge 事件 – 在发生单个 Amazon EventBridge 事件或一批 Amazon EventBridge 事件时启动该工作流。使用此触发器类型，AWS Glue 可作为事件驱动架构中的事件使用者。任何 EventBridge 事件都可以启动工作流。常见的使用案例是收到了 Amazon S3 存储桶（S3 PutObject 操作）中的新对象。

  等到收到指定数量的事件或经过指定的时间后，才表示可以使用一批事件来启动工作流程。创建 EventBridge 事件触发器时，您可以有选择地指定批处理条件。如果指定批处理条件，则必须指定批处理大小（事件数），并且有选择地指定批处理时间（秒数）。默认和最大批处理时间为 900 秒（15 分钟）。首先满足的批处理条件启动工作流程。批处理时间在第一个事件到达时开始计算。如果在创建触发器时未指定批处理条件，则批处理大小默认为 1。

  工作流程启动后，系统将重置批处理条件，事件触发器开始监控下一个要满足的批处理条件，以重新启动工作流。

  下表显示了批处理大小和批处理时间如何协同运行来触发工作流。

Batch 大小	Batch 时间	生成的触发条件
10		工作流将在 10 个 EventBridge 事件到达时触发，或者在第一个事件到达 15 分钟后触发，以先发生者为准。(如果未指定时间长度，则默认为 15 分钟。)
10	2 分钟	工作流将在 10 个 EventBridge 事件到达时触发，或者在第一个事件到达 2 分钟后触发，以先发生者为准。
1		在第一个事件到达时触发工作流程。时间长度无关紧要。如果在创建 EventBridge 事件触发器时未指定批处理条件，则批处理大小默认为 1。

  GetWorkflowRun API 操作返回触发工作流的批处理条件。

无论工作流程如何启动，您都可以在创建工作流时指定并发工作流运行的最大数量。

如果某个事件或某批事件启动的工作流最终运行失败，则不再考虑使用该事件或该批事件来启动工作流程运行。只有在下一个事件或下一批事件到达时，才会启动新的工作流运行。

> **Important**
>
> 将工作流中任务、爬网程序和触发器的总数限制为 100 个或更少。如果包含超过 100 个，则在尝试恢复或停止工作流运行时可能会出错。

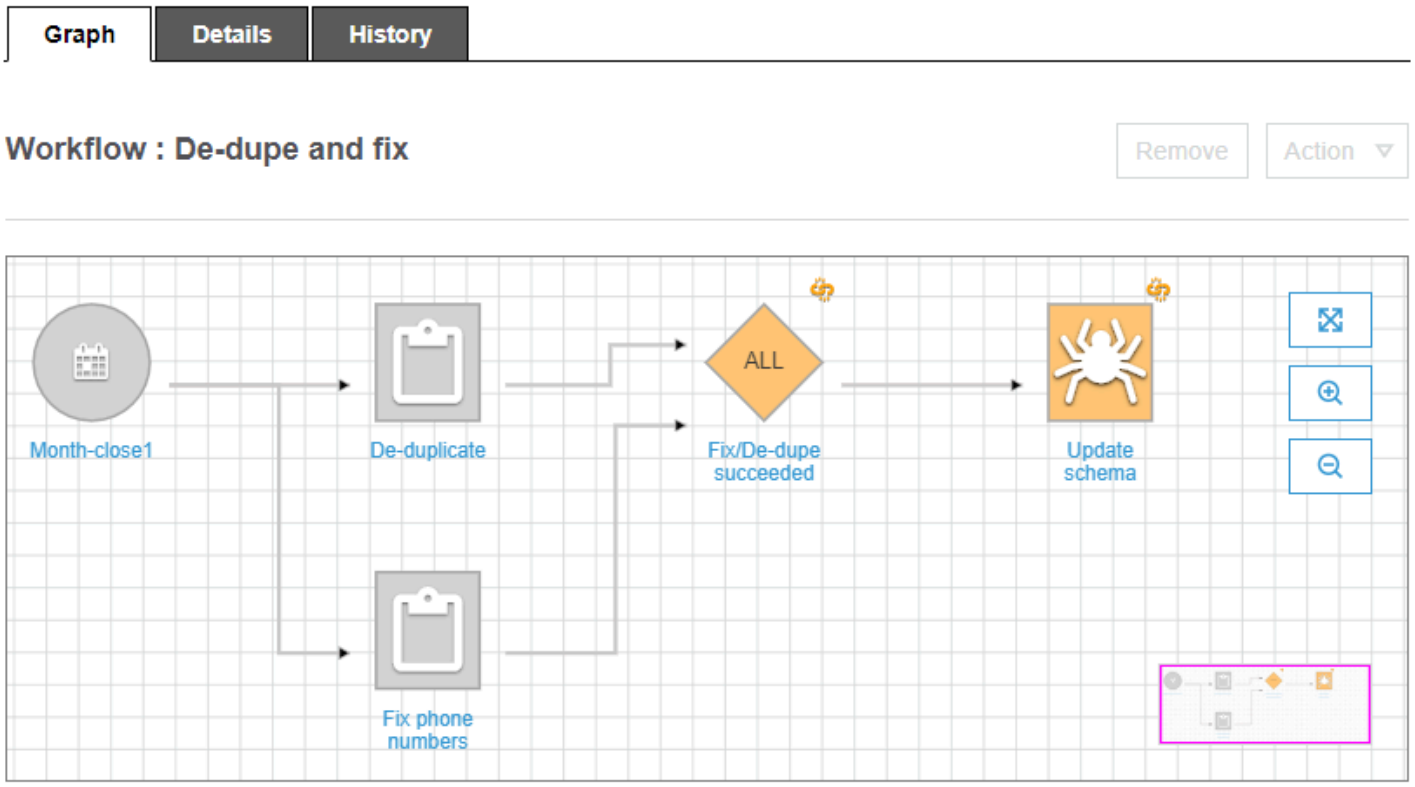
如果工作流运行超过为工作流设置的并发限制，即使满足事件条件，也不会启动工作流运行。建议根据预期的事件量调整工作流并发限制。AWS Glue 不会重试因超出并发限制而失败的工作流运行。同样，建议根据预期的事件量调整工作流中的任务和爬网程序的并发限制。

工作流运行属性

要在工作流程运行中共享和管理状态，您可以定义默认工作流程运行属性。这些属性是名称/值对，可用于工作流程中的所有作业。利用 AWS Glue API，任务可以检索工作流运行属性，并针对工作流中后面的任务修改它们。

工作流图

下图显示 AWS Glue 控制台上的基本工作流图。您的工作流程可能拥有几十个组件。



此工作流由计划触发器 Month-close1 启动，该计划触发器启动两个任务 De-duplicate 和 Fix phone numbers。在成功完成这两个任务后，事件触发器 Fix/De-dupe succeeded 将启动爬网程序 Update schema。

静态和动态工作流视图

对于每个工作流程，存在静态视图 和动态视图 的概念。静态视图表示工作流程的设计。动态视图是一个运行时视图，包含每个任务和爬网程序的最新运行信息。运行信息包含成功状态和错误详细信息。

当工作流程正在运行时，控制台将显示一个动态视图，它以图形方式指示作业已完成且尚未运行。您还可以使用 AWS Glue API 检索正在运行的工作流程的动态视图。有关更多信息，请参阅 使用 AWS Glue API 查询工作流。

另请参阅

- the section called “从蓝图创建工作流”
- the section called “手动创建和构建工作流”
- 工作流程（用于工作流 API）

在 AWS Glue 中手动创建和构建工作流

您可使用 AWS Glue 控制台，一次对一个节点手动创建和构建工作流。

工作流程包含作业、爬网程序和触发器。在手动创建工作流之前，请创建工作流要包含的任务和爬网程序。最好是指定工作流的按需运行爬网程序。可以在构建工作流程时创建新的触发器，也可以将现有触发器克隆 到工作流程中。在克隆触发器时，与触发器关联的所有目录对象触发它的任务或爬网程序，以及它启动的任务或爬网程序将添加到工作流中。

> **Important**
>
> 将工作流中任务、爬网程序和触发器的总数限制为 100 个或更少。如果包含超过 100 个，则在尝试恢复或停止工作流运行时可能会出错。

您可以通过向工作流程图表添加触发器并为每个触发器定义监视的事件和操作来构建工作流程。首先，您可以启动触发器（它可以是按需触发器或计划触发器），并通过添加事件（条件）触发器来完成图表。

步骤 1：创建工作流程

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格中，在 ETL 下，选择 Workflows (工作流程)。
3. 选择 Add workflow (添加工作流程)，并完成 Add a new ETL workflow (添加新的 ETL 工作流程) 表。

   您添加的任何可选默认运行属性都可用作工作流程中所有作业的参数。有关更多信息，请参阅 在 AWS Glue 中获取并设置工作流运行属性。
4. 选择 Add workflow (添加工作流程)。

   新的工作流程将显示在 Workflows (工作流程) 页面上的列表中。

步骤 2：添加启动触发器

1. 在 Workflows (工作流程) 页面上，选择新工作流程。然后，在页面底部，确保 Graph (图表) 选项卡处于选中状态。
2. 选择 Add trigger (添加触发器)，然后在 Add trigger (添加触发器) 对话框中，执行下列操作之一：

- 选择 Clone existing (克隆现有项)，然后选择要克隆的触发器。然后，选择 Add (添加)。

  触发器与它监视的任务和爬网程序以及它启动的任务和爬网程序一起显示在图表中。

  如果您错误地选择了错误的触发器，请在图表上选择该触发器，然后选择 Remove (删除)。

- 选择 Add new (添加新项)，然后完成 Add trigger (添加触发器) 表。

  1. 对于 Trigger type (触发器类型)，选择 Schedule (计划)、On demand (按需)，或者 EventBridge event (EventBridge 事件)。

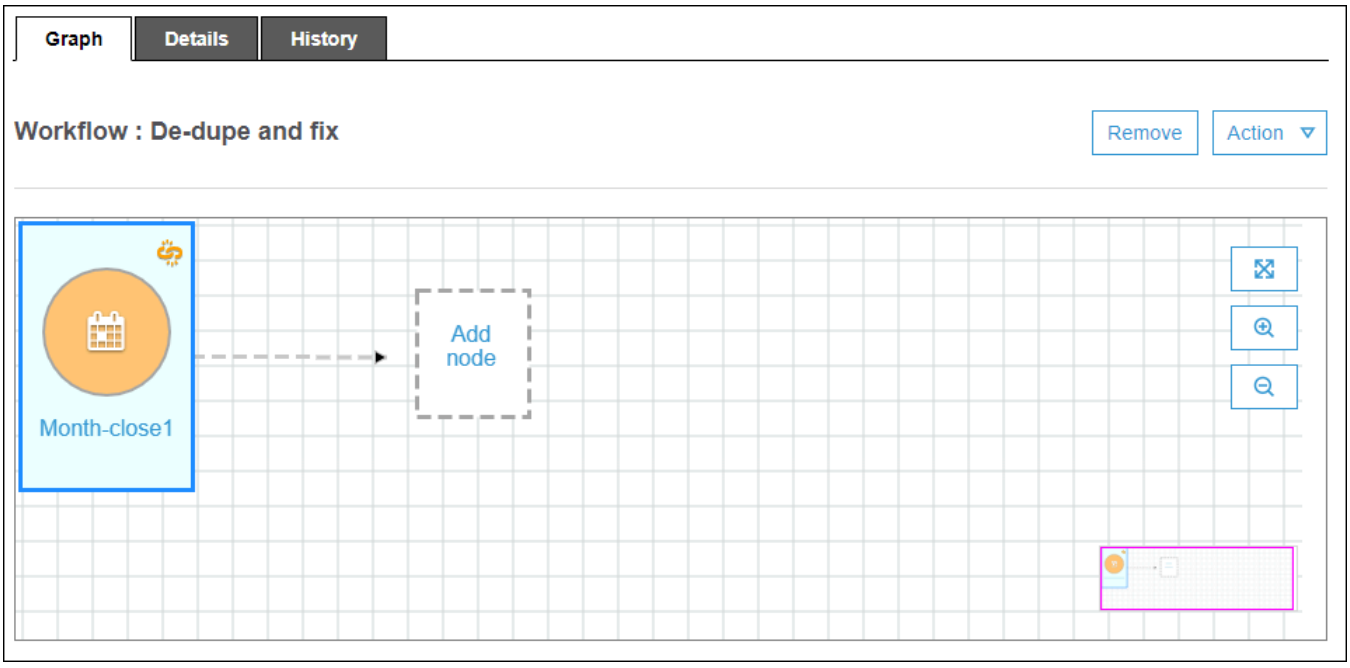
     对于触发器类型 Schedule (计划)，选择一个 Frequency (频率) 选项。选择 Custom (自定义) 以输入 `cron` 表达式。

     对于触发器类型 EventBridge event (EventBridge 事件)，输入 Number of events (事件数) (批次大小) ，然后根据需要输入 Time delay (时间延迟) (批处理时间) 。如果省略 Time delay (时间延迟)，则批处理时间默认为 15 分钟。有关更多信息，请参阅 [AWS Glue 中的工作流概述]。

  2. 选择 添加。

  触发器将与占位符节点 (标记为 Add node (添加节点)) 一起显示在图表中。在以下示例中，启动触发器是名为 `Month-close1` 的计划触发器。

  此时，尚未保存触发器。



3. 如果您已添加新触发器，请完成以下步骤：

a. 请执行以下操作之一：

- 选择占位符节点（Add node (添加节点)）。
- 确保选择启动触发器，然后在图表上方的 Action (操作) 菜单上，选择 Add jobs/crawlers to trigger (将作业/爬网程序添加到触发器)。

b. 在 Add jobs(s) and crawler(s) to trigger (将作业和爬网程序添加到触发器) 对话框中，选择一个或多个作业或爬网程序，然后选择 Add (添加)。

将保存触发器，并且选定作业或爬网程序会与触发器中的连接器一起显示在图表中。

如果您错误地添加了错误的作业或爬网程序，则可以选择触发器或连接器，并选择 Remove (删除)。

步骤 3：添加更多触发器

通过添加更多 Event (事件) 类型的触发器来继续构建工作流。要放大/缩小图表画布，请使用图表右侧的图标。对于要添加的每个触发器，请完成以下步骤：

> **Note**
>
> 没有用于保存工作流的操作。添加最后一个触发器并将操作分配给触发器后，工作流将完成并保存。您可以稍后返回并添加更多节点。

1. 请执行以下操作之一：

- 要克隆现有触发器，请确保未选择图表上的任何节点，然后在 Action (操作) 菜单上，选择 Add trigger (添加触发器)。
- 要添加监视图表上的特定作业或爬网程序的新触发器，请选择作业或爬网程序节点，然后选择 Add trigger (添加触发器) 占位符节点。

  您可以在稍后的步骤中添加更多作业或爬网程序以监视此触发器。

2. 在 Add trigger (添加触发器) 对话框中，执行下列操作之一：

- 选择 Add new (添加新项)，然后完成 Add trigger (添加触发器) 表。然后，选择 Add (添加)。

  触发器将显示在图表中。您将在后面的步骤中完成触发器。
- 选择 Clone existing (克隆现有项)，然后选择要克隆的触发器。然后，选择 Add (添加)。

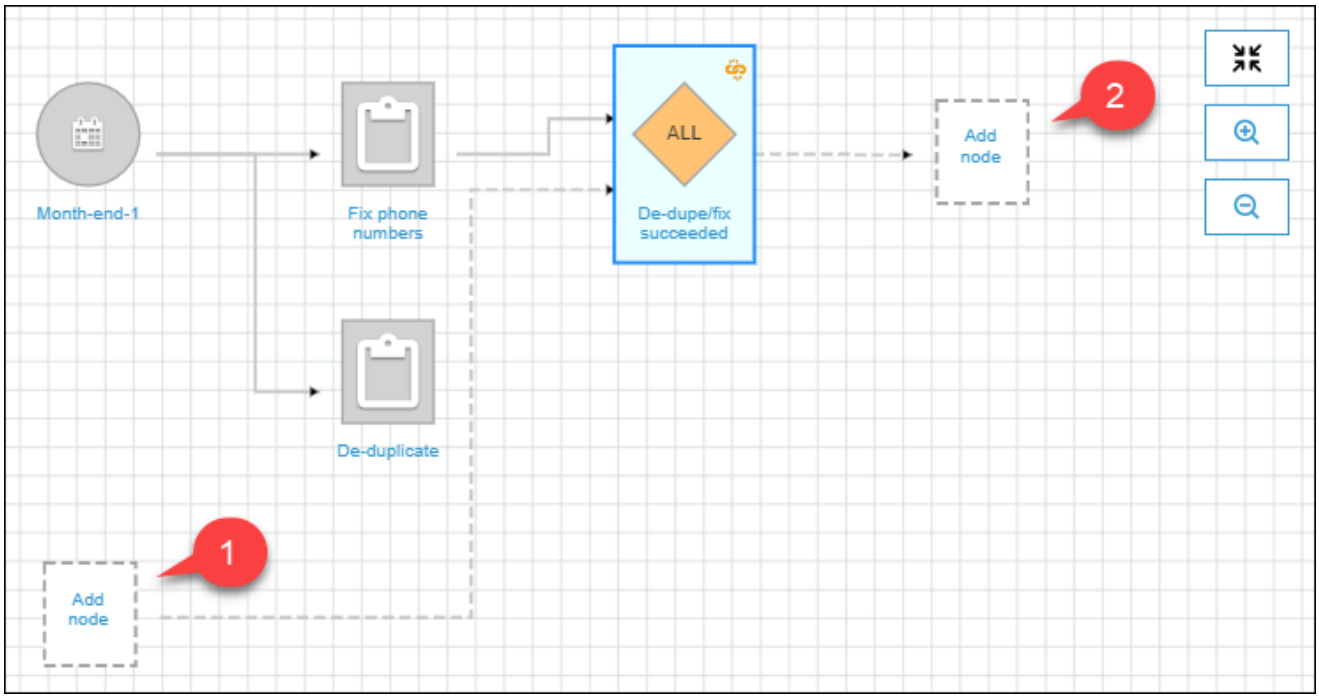
触发器与它监视的任务和爬网程序以及它启动的任务和爬网程序一起显示在图表中。

如果您错误地选择了错误的触发器，请在图表上选择该触发器，然后选择 Remove (删除)。

3. 如果您已添加新触发器，请完成以下步骤：

   a. 选择新触发器。

      如下图所示，选中触发器 `De-dupe/fix succeeded`，为要监视的（1）事件和（2）操作显示了占位符节点。



   b. （如果触发器已监视某个事件，并且您想添加更多作业或爬网程序以进行监视，则为可选。）选择要监视的事件占位符节点，然后在 Add job(s) and crawler(s) to watch (添加要监视的作业和爬网程序) 对话框中，选择一个或多个作业或爬网程序。选择要监视的事件（SUCCEEDED、FAILED 等）并选择 Add (添加)。

   c. 确保已选择触发器，然后选择操作占位符节点。

   d. 在 Add jobs(s) and crawler(s) to watch (添加要监视的作业和爬网程序) 对话框中，选择一个或多个作业或爬网程序，然后选择 Add (添加)。

      选定作业和爬网程序会与触发器中的连接器一起显示在图表中。

有关工作流和蓝图的更多信息，请参阅以下主题。

- AWS Glue 中的工作流概述
- 在 AWS Glue 中运行和监控工作流
- 在 AWS Glue 中从蓝图创建工作流

使用 Amazon EventBridge 事件启动 AWS Glue 工作流

Amazon EventBridge 又名 CloudWatch Events，可助力您自动执行您的 AWS 服务并自动响应系统事件，例如应用程序可用性问题或资源更改。AWS 服务中的事件将近乎实时传输到 EventBridge。您可以编写简单规则来指示您关注的事件，并指示要在事件匹配规则时执行的自动化操作。

通过 EventBridge 支持，AWS Glue 可以在事件驱动型架构中充当事件创建者和使用者。对于工作流，AWS Glue 作为使用者支持任何类型的 EventBridge 事件。可能最常见的使用案例是 Amazon S3 存储桶中新对象的到达。如果数据以不规则或未定义的间隔到达，您可以尽可能在此数据到达时处理数据。

> **Note**
>
> AWS Glue 不保证传送 EventBridge 消息。如果 EventBridge 传送重复消息，AWS Glue 不会执行重复数据删除。您必须基于自己的使用案例来管理幂等性。
> 请确保正确配置 EventBridge 规则，避免发送不必要事件。

开始前的准备工作

如果您要使用 Amazon S3 数据事件启动工作流，则必须确保关注的 S3 存储桶的事件记录到 AWS CloudTrail 和 EventBridge。为此，您必须创建 CloudTrail 跟踪。有关更多信息，请参阅为您的 AWS 账户创建跟踪。

使用 EventBridge 事件启动工作流

> **Note**
>
> 在以下命令中：
>
> - 将 *<workflow-name>* 替换为要分配给工作流的名称。
> - 将 *<trigger-name>* 替换为要分配给触发器的名称。
> - 将 *<bucket-name>* 替换为 Amazon S3 存储桶的名称。

- 将 *<account-id>* 替换为有效 AWS 账户 ID。
- 将 *<region>* 替换为区域的名称（例如，us-east-1）。
- 将 *<rule-name>* 替换为要分配给 EventBridge 规则的名称。

1. 确保您拥有 AWS Identity and Access Management（IAM）权限，可以创建和查看 EventBridge 规则和目标。以下是您可以附加的一个示例策略。您可能需要将其范围缩小，以限制操作和资源。

```
{
    "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. 创建 EventBridge 服务将事件传递到 AWS Glue 时可承担的 IAM 角色。

   a. 在 IAM 控制台的 Create role (创建角色) 页面上，选择 AWS Service (亚马逊云科技服务)。然后，选择服务 CloudWatch Events。

   b. 完成 Create role (创建角色) 向导。向导会自动附加 `CloudWatchEventsBuiltInTargetExecutionAccess` 和 `CloudWatchEventsInvocationAccess` 策略。

   c. 将下面的内联策略附加到角色。此策略允许 EventBridge 服务将事件定向到 AWS Glue。

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

```
    {
      "Effect": "Allow",
      "Action": [
        "glue:notifyEvent"
      ],
      "Resource": [
        "arn:aws:glue:<region>:<account-id>:workflow/<workflow-name>"
      ]
    }
  ]
}
```

3. 输入以下命令以创建工作流。

   请参阅《AWS CLI 命令参考》中的创建工作流，了解有关其他可选命令行参数的信息。

   ```
   aws glue create-workflow --name <workflow-name>
   ```

4. 输入以下命令，为工作流创建 EventBridge 事件触发器。这将是工作流的启动触发器。将 *<actions>* 替换为要执行的操作（要启动的任务和爬网程序）。

   请参阅《AWS CLI 命令参考》中的创建触发器，了解有关如何编码 `actions` 参数的信息。

   ```
   aws glue create-trigger --workflow-name <workflow-name> --type EVENT --
   name <trigger-name> --actions <actions>
   ```

   如果您希望工作流由批量事件（而不是单个 EventBridge 事件）触发，请改为输入以下命令。

   ```
   aws glue create-trigger --workflow-name <workflow-name> --type EVENT
    --name <trigger-name> --event-batching-condition BatchSize=<number-of-
   events>,BatchWindow=<seconds> --actions <actions>
   ```

   对于 `event-batching-condition` 参数，`BatchSize` 为必需项，`BatchWindow` 为可选项。如果忽略 `BatchWindow`，则窗口默认为 900 秒，这是最大窗口大小。

   Example

   下面的示例会创建一个触发器，该触发器在三个 EventBridge 事件到达后，或者第一个事件到达后五分钟（以先到者为准），启动 `eventtest` 工作流。

```
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 中创建规则。

   a. 在您的首选文本编辑器中创建规则详细信息的 JSON 对象。

   以下示例将 Amazon S3 事件指定为事件源，PutObject 为事件名称，存储桶名称为请求参数。当新对象到达存储桶时，此规则将启动工作流。

```
{
  "source": [
    "aws.s3"
  ],
  "detail-type": [
    "AWS API Call via CloudTrail"
  ],
  "detail": {
    "eventSource": [
      "s3.amazonaws.com"
    ],
    "eventName": [
      "PutObject"
    ],
    "requestParameters": {
      "bucketName": [
        "<bucket-name>"
      ]
    }
  }
}
```

   要在新对象到达存储桶内的文件夹时启动工作流，您可以将以下代码替换为 requestParameters。

```
    "requestParameters": {
      "bucketName": [
        "<bucket-name>"
      ]
      "key" : [{ "prefix" : "<folder1>/<folder2>/*"}}]
  }
```

b. 使用您的首选工具将规则 JSON 对象转义字符串转换为转义字符串。

```
{\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. 运行以下命令，创建您可编辑的 JSON 参数模板，以便将输入参数指定给后续 `put-rule` 命令。在文件中保存输出。在此示例中，文件名为 `ruleCommand`。

```
aws events put-rule --name <rule-name> --generate-cli-skeleton >ruleCommand
```

有关 `--generate-cli-skeleton` 参数的更多信息，请参阅《AWS 命令行界面用户指南》中的从 JSON 或 YAML 输入文件生成 AWS CLI 骨架和输入参数。

输出文件应与以下内容类似。

```
{
    "Name": "",
    "ScheduleExpression": "",
    "EventPattern": "",
    "State": "ENABLED",
    "Description": "",
    "RoleArn": "",
    "Tags": [
        {
            "Key": "",
            "Value": ""
        }
    ],
    "EventBusName": ""
}
```

d. 编辑文件，选择性地删除参数，并且至少指定 `Name`、`EventPattern` 和 `State` 参数。对于 `EventPattern` 参数，请为您在上一步中创建的规则详细信息提供转义字符串。

```
{
    "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**
>
> 在完成工作流构建之前，最好将规则保持禁用状态。

e. 输入以下 put-rule 命令，该命令从文件 ruleCommand 中读取输入参数。

```
aws events put-rule --name <rule-name> --cli-input-json file://ruleCommand
```

以下输出代表成功。

```
{
    "RuleArn": "<rule-arn>"
}
```

6. 输入以下命令，将策略附加到目标。目标是指 AWS Glue 中的工作流。将 *<role-name>* 替换为您在此程序开始时创建的角色。

```
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>
```

以下输出代表成功。

```
{
    "FailedEntryCount": 0,
    "FailedEntries": []
}
```

7. 输入以下命令，确认规则和目标连接成功。

```
aws events list-rule-names-by-target --target-arn arn:aws:glue:<region>:<account-
id>:workflow/<workflow-name>
```

以下输出代表成功，其中 *<rule-name>* 是所创建规则的名称。

```
{
    "RuleNames": [
        "<rule-name>"
    ]
}
```

8. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
9. 选择工作流，确认启动触发器及其操作（触发器启动的任务或爬网程序）在工作流图表上显示。然后，继续执行 步骤 3：添加更多触发器 中的程序。或者，使用 AWS Glue API 或 AWS Command Line Interface，将更多组件添加到工作流。
10. 完全指定工作流后，启用规则。

```
aws events enable-rule --name <rule-name>
```

工作流现在可以通过单个 EventBridge 事件或批量事件启动。

> **另请参阅**
>
> - Amazon EventBridge 用户指南
> - AWS Glue 中的工作流概述
> - 在 AWS Glue 中手动创建和构建工作流

查看启动工作流的 EventBridge 事件

您可以查看启动工作流的 Amazon EventBridge 事件的事件 ID。如果工作流由一批事件启动，您可以查看批处理中所有事件的事件 ID。

对于批处理大小大于 1 的工作流，您还可以查看工作流启动的批处理条件：批处理大小中事件数的到达或批处理窗口的到期。

查看工作流启动的 EventBridge 事件（控制台）

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格中，选择 Workflows (工作流)。
3. 选择工作流。然后在底部，选择 History (历史记录) 选项卡。
4. 选择工作流运行，然后选择 View run details (查看运行详细信息)。
5. 在运行详细信息页面上，找到 Run properties (运行属性) 字段，然后查找 aws:eventIds 密钥。

   该密钥的值是一系列 EventBridge 事件 ID。

查看工作流启动的 EventBridge 事件（AWS API）

- 在 Python 脚本中包含以下代码。

```
workflow_params =
 glue_client.get_workflow_run_properties(Name=workflow_name,RunId=workflow_run_id)
batched_events = workflow_params['aws:eventIds']
```

  `batched_events` 将是一系列字符串，其中每个字符串均为一个事件 ID。

> 另请参阅
>
> - Amazon EventBridge 用户指南
> - the section called “工作流概述”

在 AWS Glue 中运行和监控工作流

如果工作流程的启动触发器是按需触发器，请从 AWS Glue 控制台启动工作流程。完成以下步骤，运行并监控工作流。如果工作流失败，您可以查看运行图，确定失败的节点。为帮助进行故障排除，如果工作流根据蓝图创建，您可以查看蓝图运行，了解工作流创建所用的蓝图参数值。有关更多信息，请参阅 the section called “查看蓝图运行”。

您可以使用 AWS Glue 控制台、API、或 AWS Command Line Interface（AWS CLI）运行和监控工作流。

运行和监控工作流（控制台）

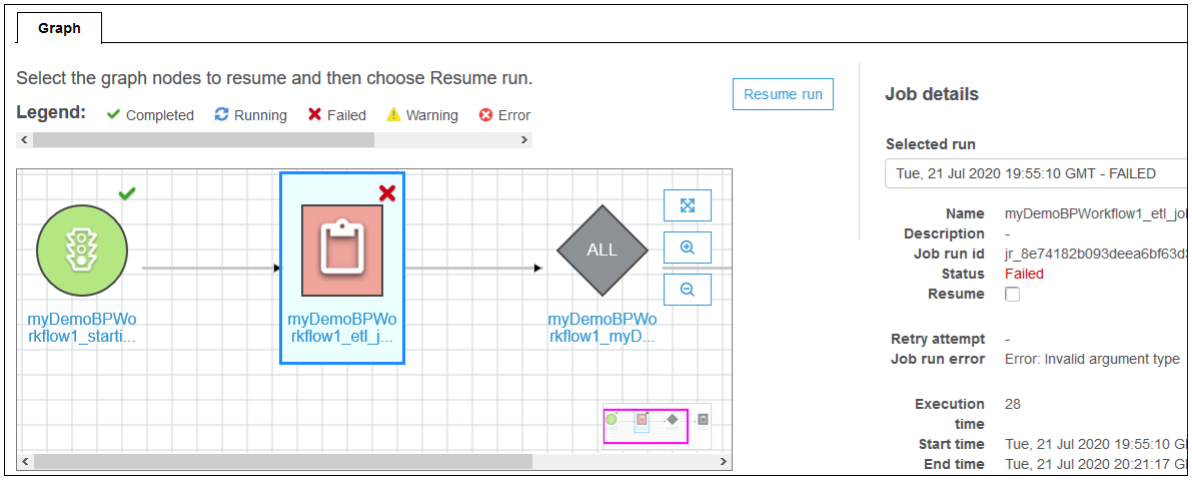
1. 通过 https://console.aws.amazon.com/glue/ 打开 AWS Glue 控制台。
2. 在导航窗格中，在 ETL 下，选择 Workflows (工作流程)。
3. 选择工作流程。在 Actions (操作) 菜单上，选择 Run (运行)。
4. 选中工作流列表中的 Last run status (上次运行状态) 列。选择刷新按钮，查看正在进行的工作流状态。
5. 当工作流正在运行或者已完成（或失败）后，请按照以下步骤查看运行详细信息。

   a. 确保已选择工作流，然后选择 History (历史记录) 选项卡。

   b. 选择当前或最近的工作流运行，然后选择 View run details (查看运行详细信息)。

      工作流运行时图会显示当前运行状态。

   c. 选择图表中的任意节点，查看该节点的详细信息和状态。



运行和监控工作流（AWS CLI）

1. 输入以下 命令。将 *<workflow-name>* 替换为要运行的工作流程。

```
aws glue start-workflow-run --name <workflow-name>
```

   如果工作流程成功启动，则此命令将返回运行 ID。

2. 使用 `get-workflow-run` 命令查看工作流运行状态。提供工作流名称和运行 ID。

```
aws glue get-workflow-run --name myWorkflow --run-id
 wr_d2af14217e8eae775ba7b1fc6fc7a42c795aed3cbcd8763f9415452e2dbc8705
```

下面是示例命令输出。

```
{
    "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
        }
    }
}
```

另请参阅：

- the section called “工作流概述”
- the section called “蓝图概览”

停止工作流运行

您可以使用 AWS Glue 控制台、AWS Command Line Interface (AWS CLI) 或 AWS Glue API 停止工作流程运行。停止工作流程运行时，所有正在运行的作业和爬网程序将立即终止，尚未启动的作业和爬网程序从不会启动。停止所有正在运行的作业和爬网程序可能需要一分钟。工作流程运行状态从

Running (正在运行) 变为 Stopping (正在停止)；当工作流程运行完全停止时，状态将变为 Stopped (已停止)。

工作流程运行停止后，您可以查看运行图，以查看哪些作业和爬网程序已完成，哪些从未启动。然后，您可以确定是否必须执行任何步骤来确保数据的完整性。停止工作流程运行会导致不执行自动回滚操作。

停止工作流程运行（控制台）

1. 通过 https://console.aws.amazon.com/glue/ 打开 AWS Glue 控制台。
2. 在导航窗格中，在 ETL 下，选择 Workflows (工作流程)。
3. 选择正在运行的工作流程，然后选择 History (历史记录) 选项卡。
4. 选择工作流程运行，然后选择 Stop run (停止运行)。

   运行状态更改为 Stopping (正在停止)。
5. （可选）选择工作流程运行，选择 View run details (查看运行详细信息)，然后查看运行图。

停止工作流程运行 (AWS CLI)

- 输入以下 命令。将 *<workflow-name>* 替换为工作流程的名称，并将 *<run-id>* 替换为要停止的工作流程运行的运行 ID。

```
aws glue stop-workflow-run --name <workflow-name> --run-id <run-id>
```

以下是 stop-workflow-run 命令的示例。

```
aws glue stop-workflow-run --name my-workflow --run-id
 wr_137b88917411d128081069901e4a80595d97f719282094b7f271d09576770354
```

修复和恢复工作流运行

如果工作流中的一个或多个节点（任务或爬网程序）未成功完成，这意味着工作流仅部分运行。找到根本原因并进行更正后，您可以选择要从中恢复工作流运行的一个或多个节点，然后恢复工作流运行。然后，所选节点以及这些节点下游的所有节点就会运行。

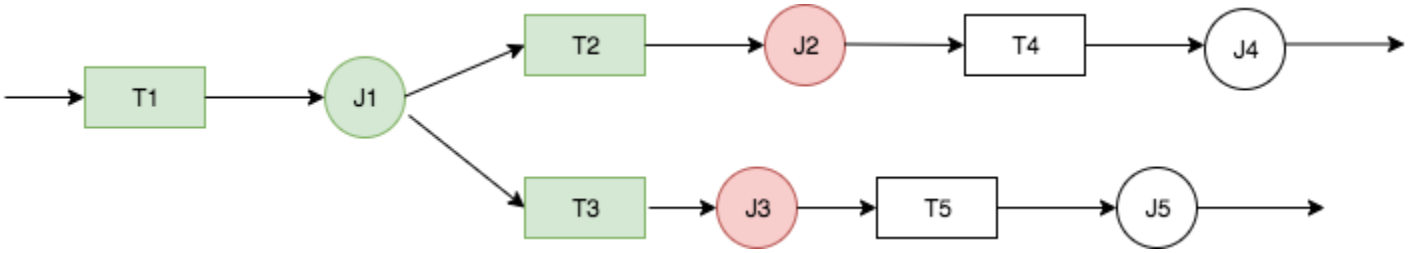
主题

- 恢复工作流运行：工作原理

- 恢复工作流运行
- 工作流运行恢复的注释和限制

恢复工作流运行：工作原理

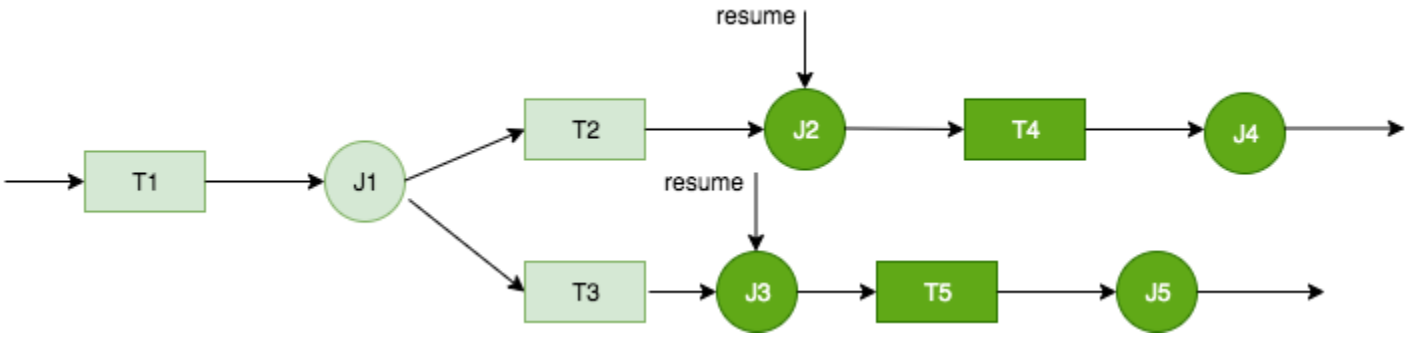
请考虑下图中的工作流 W1。



工作流运行进程如下所示：

1. 触发器 T1 启动任务 J1。
2. J1 成功完成，从而触发触发器 T2 和 T3，分别运行任务 J2 和 J3。
3. 任务 J2 和 J3 失败。
4. 触发器 T4 和 T5 取决于 J2 和 J3 的成功完成结果，所以它们不会触发，任务 J4 和 J5 也不会运行。工作流 W1 仅部分运行。

现在，假设导致 J2 和 J3 失败的问题得到更正。J2 和 J3 被选为工作流运行的恢复起点。



工作流运行恢复如下所示：

1. 任务 J2 和 J3 成功运行。
2. 触发触发器 T4 和 T5。
3. 任务 J4 和 J5 成功运行。

恢复的工作流运行作为具有新运行 ID 的单独工作流运行被跟踪。查看工作流历史记录时，您可以查看任意工作流运行的上次运行 ID。在以下屏幕截图中的示例中，具有运行 ID `wr_c7a22...`（第二行）的工作流运行有未完成的节点。用户修复问题并恢复工作流运行，从而导致运行 ID `wr_a07e55...`（第一行）。

> **Note**
>
> 在本次讨论的其余部分，术语“恢复的工作流运行”是指恢复上次工作流运行时创建的工作流运行。“原始工作流运行”是指仅部分运行且需要恢复的工作流运行。

“恢复的工作流运行”图

在恢复的工作流运行中，尽管仅运行部分节点，但运行图是完整图。也就是说，恢复的工作流中未运行的节点将从原始工作流运行的运行图中复制。原始工作流运行中运行的复制任务和爬网程序节点包括运行详细信息。

再次考虑上图中的工作流 W1。从 J2 和 J3 开始恢复工作流运行时，恢复的工作流运行的运行图会显示所有任务（J1 到 J5）以及所有触发器（T1 到 T5）。从原始工作流运行中复制 J1 的运行详细信息。

工作流运行快照

启动工作流运行时，AWS Glue 会在该时间点拍摄工作流设计图的快照。该快照在工作流运行期间使用。如果您在运行启动后对任何触发器进行更改，这些更改不会影响当前工作流运行。快照确保工作流运行以一致方式继续执行。

快照确保仅触发器不可改变。您在工作流运行期间对下游任务和爬网程序所做的更改对当前运行有效。

恢复工作流运行

请按照以下步骤操作，恢复工作流运行。您可以使用 AWS Glue 控制台、API、或 AWS Command Line Interface（AWS CLI）恢复工作流。

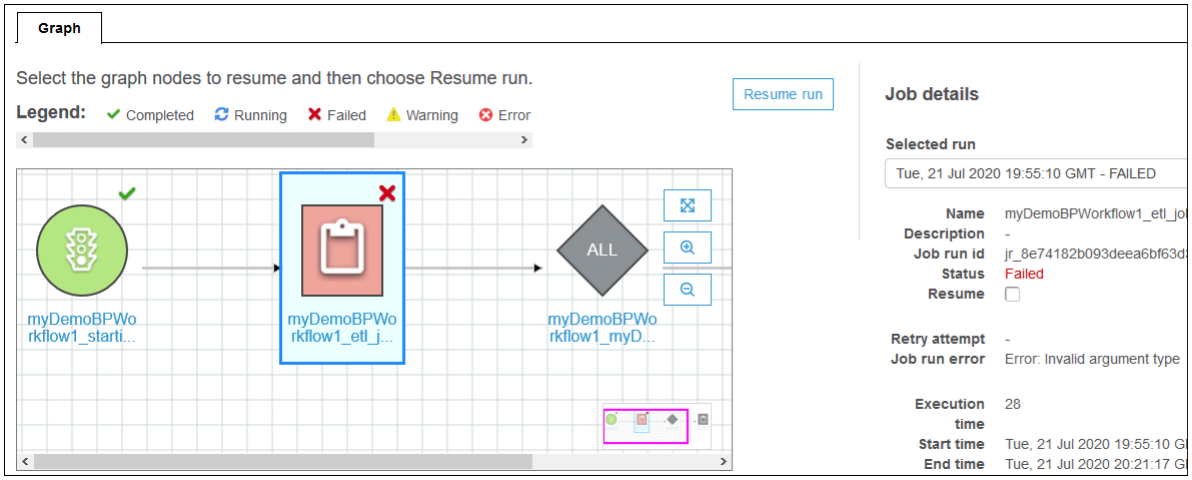
恢复工作流运行（控制台）

1. 通过 https://console.aws.amazon.com/glue/ 打开 AWS Glue 控制台。

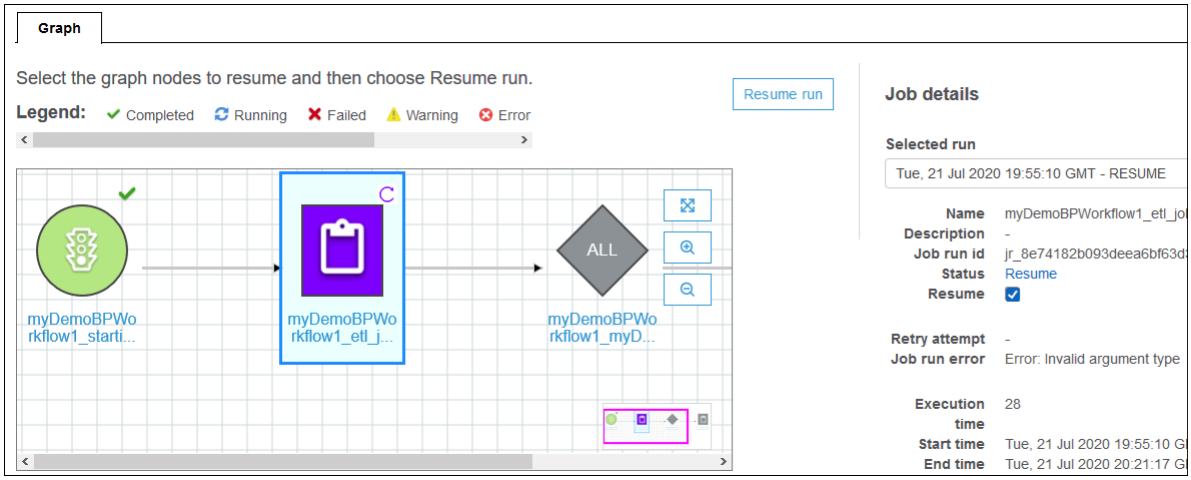
   以有权查看工作流和恢复工作流运行的用户身份登录。

   > **Note**
   >
   > 要恢复工作流运行，您需要 `glue:ResumeWorkflowRun` AWS Identity and Access Management（IAM）权限。

2. 在导航窗格中，选择 Workflows (工作流)。
3. 选择工作流，然后选择 History (历史记录) 选项卡。
4. 选择仅部分运行的工作流运行，然后选择 View run details (查看运行详细信息)。
5. 在运行图中，选择您要重新启动并且从中恢复工作流运行的第一个（或唯一）节点。
6. 在图形右侧的详细信息窗格中，选中 Resume (恢复) 复选框。



   节点会更改颜色，并在右上角显示一个小恢复图标。



7. 完成前面 2 个步骤，以便其他节点重新启动。
8. 选择 Resume run (恢复运行)。

恢复工作流运行（AWS CLI）

1. 确保您拥有 `glue:ResumeWorkflowRun` IAM 权限。
2. 检索要重新启动的节点的节点 ID。

   a. 运行原始工作流运行的 `get-workflow-run` 命令。提供工作流名称和运行 ID，然后添加 `--include-graph` 选项，如以下示例所示。从控制台上的 History (历史记录) 选项卡中获取运行 ID，或者运行 `get-workflow` 命令以获取运行 ID。

   ```
   aws glue get-workflow-run --name cloudtrailtest1 --run-id
    wr_a07e55f2087afdd415a404403f644a4265278f68b13ba3da08c71924ebe3c3a8 --include-
   graph
   ```

   命令以大型 JSON 对象的形式返回图形的节点和边缘。

   b. 按照节点对象的 `Type` 和 `Name` 属性，找到感兴趣的节点。

   以下是输出中的示例节点对象。

   ```
   {
       "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. 从节点对象的 UniqueId 属性中获取节点 ID。

3. 运行 resume-workflow-run 命令。提供工作流名称、运行 ID 和节点 ID 列表（使用空格分隔），如以下示例所示。

```
aws glue resume-workflow-run --name cloudtrailtest1 --run-id
 wr_a07e55f2087afdd415a404403f644a4265278f68b13ba3da08c71924ebe3c3a8 --node-
ids wnode_ca1f63e918fb855e063aed2f42ec5762ccf71b80082ae2eb5daeb8052442f2f3
 wnode_d1b2563c503078b153142ee76ce545fe5ceef66e053628a786ddd74a05da86fd
```

命令会输出已恢复（新）工作流运行的运行 ID 以及将启动的节点列表。

```
{
    "RunId": "wr_2ada0d3209a262fc1156e4291134b3bd643491bcfb0ceead30bd3e4efac24de9",
    "NodeIds": [
        "wnode_ca1f63e918fb855e063aed2f42ec5762ccf71b80082ae2eb5daeb8052442f2f3"
    ]
}
```

请注意，尽管示例 `resume-workflow-run` 命令列出了两个要重新启动的节点，但示例输出指明仅重新启动了一个节点。这是因为，一个节点位于另一个节点的下游，下游节点无论如何会按工作流的正常流程重新启动。

工作流运行恢复的注释和限制

恢复工作流运行时，请牢记以下注释和限制。

- 仅当工作流处于 `COMPLETED` 状态时，您可恢复工作流。

> **Note**
>
> 即使工作流运行中有一个或多个节点未完成，工作流运行状态也会显示为 `COMPLETED`。请务必检查运行图，发现任何未成功完成的节点。

- 您可以从原始工作流运行尝试运行的任务或爬网程序节点恢复工作流运行。您不能从触发器节点恢复工作流运行。
- 重新启动节点不会重置其状态。部分处理的数据都不会回滚。
- 您可以多次恢复同一工作流运行。如果恢复的工作流运行仅部分运行，您可以解决该问题并恢复已恢复的运行。
- 如果您选择两个要重新启动的节点，并且它们彼此依赖，则上游节点将在下游节点之前运行。事实上，无需选择下游节点，因为它将按照工作流的正常流程运行。

在 AWS Glue 中获取并设置工作流运行属性

使用工作流程运行属性在 AWS Glue 工作流程中的作业之间共享和管理状态。您可以在创建工作流程时设置默认运行属性。然后，当您的作业运行时，它们可以检索运行属性值，并可选择修改它们以输入到工作流程中稍后的作业。当任务修改运行属性时，仅工作流运行有新值。默认运行属性不受影响。

如果您的 AWS Glue 作业不属于工作流程的一部分，则不会设置这些属性。

以下 Python 示例代码来自提取、转换和加载（ETL）任务，该代码演示如何获取工作流运行属性。

```
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']
```

以下代码通过将 `target_format` 运行属性设置为 `'csv'` 来继续。

```
workflow_params['target_format'] = 'csv'
glue_client.put_workflow_run_properties(Name=workflow_name, RunId=workflow_run_id,
 RunProperties=workflow_params)
```

有关更多信息，请参阅下列内容：

- GetWorkflowRunProperties 操作（Python：get_workflow_run_properties）
- PutWorkflowRunProperties 操作（Python：put_workflow_run_properties）

使用 AWS Glue API 查询工作流

AWS Glue 提供了用于管理工作流程的丰富 API。您可以使用 AWS Glue API 检索工作流程的静态视图或正在运行的工作流程的动态视图。有关更多信息，请参阅 工作流程。

主题

- 查询静态视图
- 查询动态视图

查询静态视图

可使用 `GetWorkflow` API 操作获取指示工作流程设计的静态视图。此操作返回由节点和边缘组成的有向图，其中节点表示触发器、作业或爬网程序。边缘定义节点之间的关系。它们由 AWS Glue 控制台中图表上的连接器（箭头）表示。

您还可以将此操作用于常用的图形处理库，如 NetworkX、igraph、JGraphT 和 Java 通用网络/图表 (JUNG) 框架。由于所有这些库都以类似方式表示图表，因此需要最少的转换。

根据与工作流程关联的触发器的最新定义，此 API 返回的静态视图是最新视图。

图表定义

工作流程图表 G 是一个有序对 (N, E)，其中 N 是一组节点，E 是一组边缘。节点 是图表中由一个唯一数字标识的顶点。节点的类型可以是触发器、任务或爬网程序。例如：`{name:T1, type:Trigger, uniqueId:1}, {name:J1, type:Job, uniqueId:2}`。

边缘 是一个 2 元组的形式 (`src, dest`)，其中 `src` 和 `dest` 是节点，并且存在从 `src` 到 `dest` 的定向边缘。

查询静态视图的示例

考虑条件触发器 T，这将在作业 J1 完成时触发作业 J2。

```
J1 ---> T ---> J2
```

节点：J1、T、J2

边缘：(J1, T)、(T, J2)

查询动态视图

使用 `GetWorkflowRun` API 操作获取正在运行的工作流程的动态视图。此操作返回图表的相同的静态视图以及与工作流程运行相关的元数据。

对于运行，表示 `GetWorkflowRun` 调用中的任务的节点具有一组任务运行，它们作为最新工作流运行的一部分启动。您可以使用此列表在图表本身中显示每个作业的运行状态。对于尚未运行的下游依赖项，此字段设置为 `null`。图表信息使您能够随时了解任何工作流程的当前状态。

此 API 返回的动态视图基于工作流运行开始时出现的静态视图。

运行时节点示例：`{name:T1, type: Trigger, uniqueId:1}`、`{name:J1, type:Job, uniqueId:2, jobDetails:{jobRuns}}`、`{name:C1, type:Crawler, uniqueId:3, crawlerDetails:{crawls}}`

示例 1：动态视图

以下示例说明了一个包含两个触发器的简单工作流程。

- 节点：t1、j1、t2、j2
- 边缘：(t1, j1)、(j1, t2)、(t2, j2)

GetWorkflow 响应包含以下内容。

```
{
    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 响应包含以下内容。

```
{
    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
        }
}
```

示例 2：带一个条件触发器的多个作业

以下示例显示具有多个作业和一个条件触发器 (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 中的蓝图和工作流限制

以下是蓝图和工作流限制。

蓝图限制

请记住以下蓝图限制：

- 蓝图必须在 Amazon S3 存储桶所在的相同 AWS 区域中注册。

- 要跨 AWS 账户共享蓝图，您必须授予关于 Amazon S3 中蓝图 ZIP 格式存档的读取权限。客户如果对蓝图 ZIP 格式存档具有读取权限，可以在他们的 AWS 账户中注册蓝图并使用。
- 蓝图参数集存储为单个 JSON 对象。此对象的最大长度为 128KB。
- 蓝图 ZIP 格式归档文件的最大未压缩大小为 5MB。最大未压缩大小为 1MB。
- 将工作流中任务、爬网程序和触发器的总数限制为 100 个或更少。如果包含超过 100 个，则在尝试恢复或停止工作流运行时可能会出错。

工作流限制

请记住以下工作流限制：其中一些注释更多地针对手动创建工作流的用户。

- Amazon EventBridge 事件触发器的最大批处理大小为 100。最大窗口大小为 900 秒（15 分钟）。
- 一个触发器只能与一个工作流程关联。
- 仅允许一个启动触发器（按需或计划）。
- 如果工作流程中的某个作业或爬网程序由工作流外部的触发器启动，则工作流程中依赖于该作业或爬网程序完成（成功或其他状态）的任何触发器都不会触发。
- 同样，如果工作流中的某个任务或爬网程序具有触发器，而该触发器依赖于工作流内外部的任务或爬网程序完成（成功或其他状态），则如果任务或爬网程序是从工作流内部启动的，只有工作流内部的触发器在任务或爬网程序完成时触发。

排查 AWS Glue 中的蓝图错误

如果您在使用 AWS Glue 蓝图时遇到错误，则可以使用以下解决方案来帮助您查找问题的根源并解决问题。

主题

- 错误：在指定的布局生成器路径中找不到模块
- 错误：连接字段中的验证错误

错误：缺少 PySpark 模块

AWS Glue 返回错误“Unknown error executing layout generator function ModuleNotFoundError: No module named 'pyspark'”。

解压蓝图存档文件时，它可能类似以下任一情况：

```
$ 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
```

对于第一种情况，与蓝图相关的所有文件均位于名为 compaction 的文件夹下面，然后转换为名为 compaction.zip 的 zip 格式文件。

对于第二种情况，蓝图所需的所有文件均不在文件夹内，而是作为根文件添加到 zip 格式文件 compaction.zip 下面。

允许使用上述两种格式创建文件。但是，请确保 `blueprint.cfg` 有正确路径指向布局生成脚本中函数的名称。

示例

对于案例 1：`blueprint.cfg` 应该具有 `layoutGenerator`，如下所示：

```
layoutGenerator": "compaction.layout.generate_layout"
```

对于案例 2：blueprint.cfg 应该具有 layoutGenerator，如下所示：

```
layoutGenerator": "layout.generate_layout"
```

如果没有正确包含此路径，您可能会看到指示的错误。例如，如果您拥有案例 2 中提到的文件夹结构，并且具有案例 1 中指明的 layoutGenerator，您可能会看到上面的错误。

错误：缺少蓝图配置文件

AWS Glue 返回错误“Unknown error executing layout generator function FileNotFoundError: [Errno 2] No such file or directory: '/tmp/compaction/blueprint.cfg'”。

blueprint.cfg 应置于 ZIP 格式存档的根级别，或者置于与 ZIP 存档文件具有相同名称的文件夹内。

当我们提取蓝图 ZIP 格式存档文件时，blueprint.cfg 预计位于以下某个路径。如果您在以下路径找不到该文件，您可能会看到上述错误。

```
$ unzip compaction.zip
Archive:  compaction.zip
   creating: compaction/
  inflating: compaction/blueprint.cfg

$ unzip compaction.zip
Archive:  compaction.zip
  inflating: blueprint.cfg
```

错误：缺少导入的文件

AWS Glue 返回错误“Unknown error executing layout generator function FileNotFoundError: [Errno 2] No such file or directory:* *'demo-project/foo.py'”。

如果布局生成脚本具有读取其他文件的功能，请确保提供要导入的文件的完整路径。例如，Conversion.py 脚本可能会在 Layout.py 中引用。有关更多信息，请参阅示例蓝图项目。

错误：未授权在资源上执行 iamPassRole

AWS Glue 返回错误“User: arn:aws:sts::123456789012:assumed-role/AWSGlueServiceRole/GlueSession is not authorized to perform: iam:PassRole on resource: arn:aws:iam::123456789012:role/AWSGlueServiceRole”

如果工作流中的任务和爬网程序承担的角色是通过蓝图创建工作流时传递的角色，则蓝图角色本身需要包含 `iam:PassRole` 权限。

如果工作流中的任务和爬网程序承担的角色不是通过蓝图创建工作流实体时传递的角色，则蓝图角色对于此类其他角色（不是蓝图角色）需要包含 `iam:PassRole` 权限。

有关更多信息，请参阅蓝图角色的权限。

错误：无效 cron 计划

AWS Glue 返回错误“The schedule cron(0 0 * * * *) is invalid.”

提供有效 cron 表达式。有关更多信息，请参阅用于作业和爬网程序的基于时间的计划。

错误：已存在具有相同名称的触发器

AWS Glue 返回错误“Trigger with name 'foo_starting_trigger' already submitted with different configuration”。

蓝图不要求您在用于工作流创建的布局脚本中定义触发器。触发器创建由蓝图库根据两项操作之间定义的依赖关系进行管理。

触发器的命名如下所示：

- 对于工作流中的启动触发器，命名为 <workflow_name>_starting_trigger。
- 如果工作流中的节点（任务/爬网程序）取决于一个或多个上游节点的完成情况；AWS Glue 会定义以下名称的触发器：<workflow_name>_<node_name>_trigger

此错误意味着已存在具有相同名称的触发器。您可以删除现有触发器并重新运行工作流创建。

> **Note**
>
> 删除工作流不会删除工作流内的节点。虽然工作流已删除，但触发器可能会保留。因此，您可能不会收到“workflow already exists”错误，但如果创建工作流、删除工作流并尝试从同一蓝图使用相同的名称重新创建，您可能会收到“trigger already exists”错误。

错误：名称为 foo 的工作流已存在。

工作流名称应该唯一。请尝试使用其他名称。

错误：在指定的布局生成器路径中找不到模块

AWS Glue 返回错误“Unknown error executing layout generator function ModuleNotFoundError: No module named 'crawl_s3_locations'”。

```
layoutGenerator": "crawl_s3_locations.layout.generate_layout"
```

例如，如果您具有上述 LayoutGenerator 路径，当您解压缩蓝图归档文件时，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
```

解压归档文件时，如果蓝图归档文件如下所示，则可能会出现上述错误。

```
$ unzip crawl_s3_locations.zip
Archive:  crawl_s3_locations.zip
  inflating: blueprint.cfg
  inflating: layout.py
  inflating: README.md
```

您可能会发现，没有名为 `crawl_s3_locations` 的文件夹，并且当 `layoutGenerator` 路径通过模块 `crawl_s3_locations` 指向布局文件时，您可能会收到上述错误。

错误：连接字段中的验证错误

AWS Glue 返回错误“Unknown error executing layout generator function TypeError: Value ['foo'] for key Connections should be of type <class 'dict'>!”。

这是验证错误。`Job` 类中的 `Connections` 字段需要一个字典，但却提供了导致错误的值列表。

```
User input was list of values
Connections= ['string']

Should be a dict like the following
Connections*=*{'Connections': ['string']}
```

为了在从蓝图创建工作流时避免这些运行时错误，您可以验证工作流、任务和爬网程序定义，如测试蓝图中概述。

请参阅 AWS Glue 蓝图类参考中的语法，定义布局脚本中的 AWS Glue 作业、爬网程序和工作流。

AWS Glue 蓝图的角色权限

以下是常见的角色，以及针对 AWS Glue 蓝图角色的 AWS Identity and Access Management（IAM）权限策略建议。

主题

蓝图角色

以下是 AWS Glue 蓝图生命周期中通常涉及的角色。

角色	描述
AWS Glue 开发人员	开发、测试和发布蓝图。
AWS Glue 管理员	蓝图上的注册、维护和授予权限。
数据分析人员	运行蓝图以创建工作流。

有关更多信息，请参阅 the section called “蓝图概览”。

蓝图角色的权限

以下是针对每个蓝图角色的建议权限。

蓝图的 AWS Glue 开发人员权限

AWS Glue 开发人员必须具有写入用于发布蓝图的 Amazon S3 存储桶的权限。通常，开发人员在上传蓝图后注册蓝图。在这种情况下，开发人员需要具有 the section called “蓝图的 AWS Glue 管理员

权限” 中列出的权限。此外，如果开发人员希望在蓝图注册后测试蓝图，还需要具有 the section called “蓝图的数据分析人员权限” 中列出的权限。

蓝图的 AWS Glue 管理员权限

以下策略授予注册、查看和维护 AWS Glue 蓝图的权限。

> **Important**
>
> 在以下策略中，将 *<s3-bucket-name>* 和 *<prefix>* 替换为 Amazon S3 路径，以便上传蓝图 ZIP 格式归档来进行注册。

```
{
    "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>/*"
        }
    ]
}
```

蓝图的数据分析人员权限

以下策略授予运行蓝图以及查看生成的工作流和工作流组件的权限。它还授予通过 AWS Glue 创建工作流和工作流组件时承担的角色 PassRole。

该策略授予对任何资源的权限。如果要配置对单个蓝图的精细访问，请使用以下蓝图 ARN 格式：

```
arn:aws:glue:<region>:<account-id>:blueprint/<blueprint-name>
```

> **Important**
>
> 在以下策略中，将 *<account-id>* 替换为有效的AWS账户并将 *<role-name>* 替换为用于运行蓝图的角色的名称。请参阅the section called “蓝图角色的权限”了解此角色所需的权限。

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

蓝图角色的权限

以下是针对用于从蓝图创建工作流的 IAM 角色的建议权限。此角色必须与 `glue.amazonaws.com` 具有信任关系。

> **Important**
>
> 在以下策略中，将 *<account-id>* 替换为有效的AWS账户并将 *<role-name>* 替换为角色的名称。

```
{
    "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**
>
> 如果工作流中的任务和爬网程序担任此角色以外的角色，则此策略必须包含 `iam:PassRole` 权限，而不是蓝图角色上的权限。

AWS Glue 中的开发蓝图

您的组织可能有一组类似的 ETL 用例，这些用例可以通过参数化单个工作流以处理所有这些用例而受益。为了满足这一需求，AWS Glue 使您能够定义蓝图，您可以使用它来生成工作流。蓝图接受参数，因此数据分析人员可以从单个蓝图中创建不同的工作流来处理类似的 ETL 用例。创建蓝图后，您可以将其重复用于不同的部门、团队和项目。

主题

AWS Glue 中的蓝图概览

> **Note**
>
> 蓝图功能目前在 AWS Glue 控制台的以下区域不可用：亚太地区（雅加达）和中东（阿联酋）。

AWS Glue 蓝图提供了一种创建和共享 AWS Glue 工作流的方法。当存在可用于类似用例的复杂 ETL 流程时，您可以创建单个蓝图，而不是为每个使用案例创建一个 AWS Glue 工作流。

蓝图指定要包含在工作流中的任务和爬网程序，并指定工作流用户在运行蓝图创建工作流时提供的参数。参数的使用使单个蓝图能够为各种类似的用例生成工作流。有关工作流的更多信息，请参阅AWS Glue 中的工作流概述。

以下是蓝图的示例用例：

- 您想要对现有数据集进行分区。蓝图的输入参数包括 Amazon Simple Storage Service（Amazon S3）源和目标路径以及分区列列表。
- 您希望将 Amazon DynamoDB 表快照到 SQL 数据存储（如 Amazon Redshift）中。蓝图的输入参数包括 DynamoDB 表名称和 AWS Glue 连接，它指定 Amazon Redshift 集群和目标数据库。
- 您想将多个 Amazon S3 路径中的 CSV 数据转换为 Parquet。您希望 AWS Glue 工作流为每个路径包含一个单独的爬网程序和任务。输入参数是 AWS Glue 数据目录中的目标数据库和以逗号分隔的 Amazon S3 路径列表。请注意，在这种情况下，工作流创建的爬网程序和任务的数量是可变的。

蓝图组件

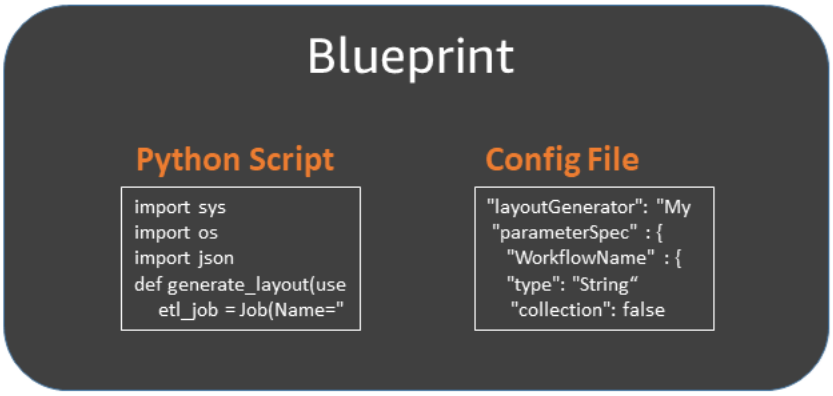
蓝图是包含以下组件的 ZIP 格式归档：

- Python 布局生成器脚本

  包含指定工作流布局的函数 – 为工作流创建的爬网程序和任务、任务和爬网程序属性以及任务和爬网程序之间的依赖关系。该函数接受蓝图参数并返回 AWS Glue 用于生成工作流的工作流结构（JSON 对象）。因为您使用 Python 脚本来生成工作流，因此您可以添加适合您的用例的逻辑。
- 一个配置文件

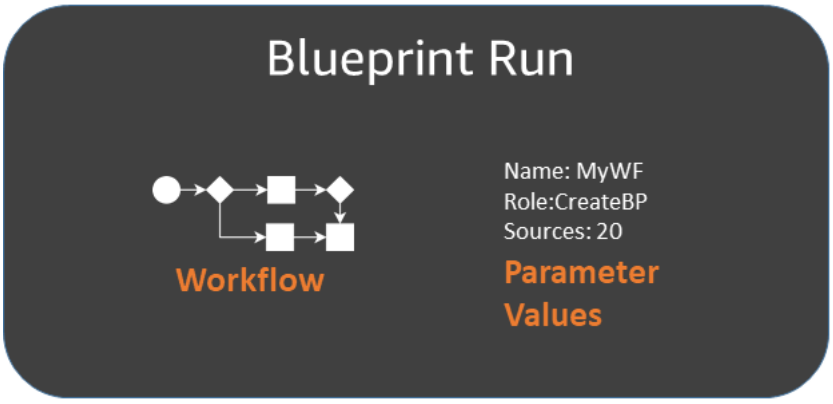
  指定生成工作流布局的 Python 函数的完全限定名称。还指定脚本使用的所有蓝图参数的名称、数据类型和其他属性。
- （可选）ETL 脚本和支持文件

  作为高级用例，您可以参数化任务使用的 ETL 脚本的位置。您可以在 ZIP 格式归档中包含任务脚本文件，并为要复制脚本的 Amazon S3 位置指定蓝图参数。布局生成器脚本可以将 ETL 脚本复制到指定位置，并将该位置指定为任务脚本位置属性。您还可以包含任何库或其他支持文件，前提是您的脚本处理它们。



蓝图运行

从蓝图创建工作流时，AWS Glue 会运行蓝图，这会开启一个异步过程来创建工作流以及工作流封装的任务、爬网程序和触发器。AWS Glue 使用蓝图运行来编排工作流及其组件的创建。您可以通过查看蓝图运行状态来查看创建过程的状态。蓝图运行还存储您为蓝图参数提供的值。



您可以使用 AWS Glue 控制台或 AWS Command Line Interface（AWS CLI）查看蓝图运行。查看工作流或对工作流进行问题排查时，您始终可以返回蓝图运行以查看用于创建工作流的蓝图参数值。

蓝图的生命周期

蓝图经过开发、测试、向 AWS Glue 注册并运行以创建工作流。蓝图生命周期中通常涉及三个角色。

角色	任务
AWS Glue 开发人员	• 编写工作流布局脚本并创建配置文件。 • 使用 AWS Glue 服务提供的库在本地测试蓝图。 • 创建脚本、配置文件和支持文件的 ZIP 格式归档，并将归档发布到 Amazon S3 中的某个位置。

角色	任务
	• 将存储桶策略添加到 Amazon S3 存储桶，以向 AWS Glue 管理员的 AWS 账户授予对存储桶对象的读取权限。 • 向 AWS Glue 管理员授予对 Amazon S3 中 ZIP 格式归档的 IAM 读取权限。
AWS Glue 管理员	• 使用 AWS Glue 注册蓝图。AWS Glue 会将 ZIP 格式归档的副本复制到预留的 Amazon S3 位置。 • 向数据分析人员授予对蓝图的 IAM 权限。
数据分析人员	• 运行蓝图以创建工作流，并提供蓝图参数值。检查蓝图运行状态以确保已成功生成工作流和工作流组件。 • 运行工作流并对其进行故障排除。在运行工作流之前，可以通过在 AWS Glue 控制台上查看工作流设计图来验证工作流。

另请参阅

- AWS Glue 中的开发蓝图
- 在 AWS Glue 中从蓝图创建工作流
- AWS Glue 蓝图的角色权限

AWS Glue 中的开发蓝图

作为 AWS Glue 开发人员，您可以创建和发布蓝图，以便数据分析人员可以使用这些蓝图来生成工作流。

主题

- 开发蓝图概述
- 开发蓝图的先决条件
- 编写蓝图代码
- 示例蓝图项目

- 测试蓝图
- 发布蓝图
- AWS Glue 蓝图类参考
- 蓝图示例

> 另请参阅
>
> - AWS Glue 中的蓝图概览

开发蓝图概述

开发过程的第一步是确定将受益于蓝图的常见使用案例。典型使用案例涉及一个反复出现的 ETL 问题，您认为此问题应该以常规方式解决。接下来，设计一个实现通用使用案例的蓝图，并定义蓝图输入参数，这些参数都可从通用使用案例中定义特定使用案例。

蓝图由包含蓝图参数配置文件的项目以及定义所要生成工作流布局的脚本组成。布局定义了所要创建的任务和爬网程序（或蓝图脚本术语中的实体）。

您不会直接在布局脚本中指定任何触发器。相反，您可以编写代码来指定脚本创建的任务和爬网程序之间的依赖关系。AWS Glue 根据您的依赖项规范生成触发器。布局脚本的输出是一个工作流对象，其中包含所有工作流实体的规范。

您可以使用以下 AWS Glue 蓝图库构建工作流对象：

- `awsglue.blueprint.base_resource` – 库使用的基本资源库。
- `awsglue.blueprint.workflow` – 用于定义 `Workflow` 类的库。
- `awsglue.blueprint.job` – 用于定义 `Job` 类的库。
- `awsglue.blueprint.crawler` – 用于定义 `Crawler` 类的库。

唯一支持布局生成的其他库是可用于 Python shell 的库。

在发布蓝图之前，您可以使用蓝图库中定义的方法在本地测试蓝图。

当您准备好将蓝图提供给数据分析人员时，您可以将脚本、参数配置文件和任何支持文件（如附加脚本和库）打包到单个可部署资产中。然后，您将资产上传到 Amazon S3，并要求管理员将其注册到 AWS Glue。

有关更多示例蓝图项目的信息，请参阅示例蓝图项目和蓝图示例。

开发蓝图的先决条件

要开发蓝图，您应熟悉使用 AWS Glue 和为 Apache Spark ETL 任务或 Python shell 任务编写脚本。此外，您还必须完成以下设置任务。

- 下载 4 个 AWS Python 库，以在蓝图布局脚本中使用。
- 设置 AWS SDK。
- 设置 AWS CLI。

下载 Python 库

从 GitHub 下载以下库，并将它们安装到您的项目中：

- 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

设置 AWS Java SDK

对于 AWS Java SDK，您必须添加 `jar` 文件，其中包含适用于蓝图的 API。

1. 如果您还没有这样做，请设置 AWS SDK for Java。
   - 对于 Java 1.x，请遵循《AWS SDK for Java 开发人员指南》中的设置 AWS SDK for Java。
   - 对于 Java 2.x，请遵循《AWS SDK for Java 2.x 开发人员指南》中的设置 AWS SDK for Java 2.x。
2. 下载客户端 `jar` 文件，该文件拥有访问适用于蓝图的 API 权限。
   - 对于 Java 1.x：s3://awsglue-custom-blueprints-preview-artifacts/awsglue-java-sdk-preview/AWSGlueJavaClient-1.11.x.jar
   - 对于 Java 2.x：s3://awsglue-custom-blueprints-preview-artifacts/awsglue-java-sdk-v2-preview/AwsJavaSdk-Glue-2.0.jar
3. 将客户端 `jar` 添加到 Java 类路径前，覆盖由 AWS Java SDK 提供的 AWS Glue 客户端。

```
export CLASSPATH=<path-to-preview-client-jar>:$CLASSPATH
```

4. （可选）使用以下 Java 应用程序测试 SDK。应用程序应输出空列表。

   将 accessKey 和 secretKey 替换为您的凭证，然后将 us-east-1 替换为您的区域。

```
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));
    }
}
```

设置 AWS Python SDK

以下步骤假定您的计算机上已安装 Python 版本 2.7 或更高版本，或版本 3.6 或更高版本。

1. 下载下面的 boto3 wheel 文件。如果提示打开或保存文件，请保存文件 s3://awsglue-custom-blueprints-preview-artifacts/aws-python-sdk-preview/boto3-1.17.31-py2.py3-none-any.whl
2. 下载以下 botocore wheel 文件：s3://awsglue-custom-blueprints-preview-artifacts/aws-python-sdk-preview/botocore-1.20.31-py2.py3-none-any.whl
3. 检查您的 Python 版本。

```
python --version
```

4. 根据您的 Python 版本，输入以下命令（适用于 Linux）：

   - 对于 Python 2.7 或更高版本。

     ```
     python3 -m pip install --user virtualenv
     source env/bin/activate
     ```

   - 对于 Python 3.6 或更高版本。

     ```
     python3 -m venv python-sdk-test
     source python-sdk-test/bin/activate
     ```

5. 安装 botocore wheel 文件。

   ```
   python3 -m pip install <download-directory>/botocore-1.20.31-py2.py3-none-any.whl
   ```

6. 安装 boto3 wheel 文件。

   ```
   python3 -m pip install <download-directory>/boto3-1.17.31-py2.py3-none-any.whl
   ```

7. 在 `~/.aws/credentials` 和 `~/.aws/config` 文件中配置您的凭证和默认区域。有关更多信息，请参阅《AWS Command Line Interface 用户指南》中的配置 AWS CLI。

8. （可选）测试您的设置。以下命令应该返回空列表。

   将 `us-east-1` 替换为您的区域。

   ```
   $ python
   >>> import boto3
   >>> glue = boto3.client('glue', 'us-east-1')
   >>> glue.list_blueprints()
   ```

设置预览 AWS CLI

1. 如果您还没有这样做，请安装和/或更新适用于电脑的 AWS Command Line Interface（AWS CLI）。执行该操作的最简单方法是使用 `pip`，Python 实用安装程序：

   ```
   pip install awscli --upgrade --user
   ```

   您可在此处查找 AWS CLI 的完整安装说明：安装 AWS Command Line Interface。

2. 下载 AWS CLI wheel 文件：s3://awsglue-custom-blueprints-preview-artifacts/awscli-preview-build/awscli-1.19.31-py2.py3-none-any.whl
3. 安装 AWS CLI wheel 文件。

```
python3 -m pip install awscli-1.19.31-py2.py3-none-any.whl
```

4. 运行 `aws configure` 命令。配置 AWS 凭证（包括访问密钥和私有密钥）和 AWS 区域。您可在此处找到有关配置 AWS CLI 的信息：配置 AWS CLI。
5. 测试 AWS CLI。以下命令应该返回空列表。

   将 `us-east-1` 替换为您的区域。

```
aws glue list-blueprints --region us-east-1
```

编写蓝图代码

您创建的每个蓝图项目必须至少包含以下文件：

- 定义工作流的 Python 布局脚本。该脚本包含一个函数，用于定义工作流中的实体（任务和爬网程序）以及它们之间的依赖关系。
- 配置文件 `blueprint.cfg`，它定义了：
  - 工作流布局定义函数的完整路径。
  - 蓝图接受的参数。

主题

- 创建蓝图布局脚本
- 创建配置文件
- 指定蓝图参数

创建蓝图布局脚本

蓝图布局脚本必须包含在工作流中生成实体的函数。您可以随意命名此函数。AWS Glue 使用配置文件确定函数的完全限定名称。

您的布局函数执行以下操作：

- （可选）实例化 Job 类来创建 Job 对象，并传递诸如 Command 和 Role 的参数。如果您使用 AWS Glue 控制台或 API 来创建任务，这些是您要指定的任务属性。
- （可选）实例化 Crawler 类来创建 Crawler 对象，并传递名称、角色和目标参数。
- 要指示对象（工作流实体）之间的依赖项，请将 DependsOn 和 WaitForDependencies 等其他参数传递到 Job() 和 Crawler()。此部分的后面将说明这些参数。
- 实例化 Workflow 类以创建返回到 AWS Glue 的工作流对象，传递 Name 参数、Entities 参数和可选的 OnSchedule 参数。Entities 参数指定要包含在工作流中的所有任务和爬网程序。如需了解如何构建 Entities 对象，请参阅此部分后面的示例项目。
- 返回 Workflow 对象。

有关 Job、Crawler 和 Workflow 类的定义，请参阅AWS Glue 蓝图类参考。

该布局函数接受以下输入参数。

参数	描述
user_params	蓝图参数名称和值的 Python 词典。有关更多信息，请参阅 指定蓝图参数。
system_params	Python 词典包含两个属性：region 和 accountId 。

这里是一个文件名为 Layout.py 的简单布局生成器脚本：

```
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
```

示例脚本会导入所需的蓝图库，并包含 generate_layout 函数，用于生成包含两个任务的工作流。这是一个非常简单的脚本。更复杂的脚本可以使用额外的逻辑和参数来生成具有许多任务和爬网程序（甚至可变数量的任务和爬网程序）的工作流。

使用 DependsOn 参数

DependsOn 参数是此实体对工作流中其他实体所具有的依赖关系的字典表示形式。格式如下。

```
DependsOn = {dependency1 : state, dependency2 : state, ...}
```

此字典中的密钥表示实体的对象引用，而不是名称，而值是对应于要监视的状态的字符串。AWS Glue 推断合适的触发器。有关有效状态，请参阅条件结构。

例如，任务可能依赖于爬网程序的成功完成。如果您定义了名为 crawler2 的爬网程序对象，如下所示：

```
crawler2 = Crawler(Name="my_crawler", ...)
```

然后一个依赖于 crawler2 的对象将包含一个构造函数参数，例如：

```
DependsOn = {crawler2 : "SUCCEEDED"}
```

例如：

```
job1 = Job(Name="Job1", ..., DependsOn = {crawler2 : "SUCCEEDED", ...})
```

如果实体省略了 DependsOn，则该实体依赖于工作流启动触发器。

使用 WaitForDependencies 参数

WaitForDependencies 参数定义任务或爬网程序实体是应该等待它依赖的全部实体完成，还是等待任意实体完成。

允许的值为“AND”或”ANY“。

使用 OnSchedule 参数

Workflow 类构造函数的 OnSchedule 参数是 cron 表达式，用于定义工作流的启动触发器。

如果指定此参数，AWS Glue 创建一个带有相应计划的计划触发器。如果未指定启动触发器，则工作流的启动触发器是按需触发器。

创建配置文件

蓝图配置文件是必需的文件，用于定义生成工作流的脚本入口点以及蓝图接受的参数。文件必须命名为 blueprint.cfg。

下面是示例配置文件。

```
{
    "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 属性指定生成布局的脚本中函数的完全限定名称。

parameterSpec 属性指定此蓝图接受的参数。有关更多信息，请参阅 指定蓝图参数。

> **Important**
>
> 配置文件必须包含工作流名称作为蓝图参数，或者必须在布局脚本中生成唯一的工作流名称。

指定蓝图参数

配置文件包含 parameterSpec JSON 对象中的蓝图参数规范。parameterSpec 包含一个或多个参数对象。

```
"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>": {
       ...
    }
  }
```

以下是编码每个参数对象的规则：

- 参数名称和 type 是必需的。所有其他属性均为可选属性。
- 如果您指定 defaultValue 属性，则该参数是可选的。否则，参数是必需参数，并且从蓝图创建工作流的数据分析员必须为其提供值。

- 如果您将 `collection` 属性设置为 `true`，参数可以采用值的集合。集合可以是任意数据类型。
- 如果您指定 `allowedValues`，则 AWS Glue 控制台显示一个值下拉列表，供数据分析人员在从蓝图创建工作流时进行选择。

以下是 `type` 允许的值：

参数数据类型	注意事项
`String`	-
`Integer`	-
`Double`	-
`Boolean`	可能的值为 `true` 和 `false`。AWS Glue 控制台的 Create a workflow from <blueprint>（从 <蓝图> 创建工作流）页面生成的一个复选框。
`S3Uri`	从 `s3://` 开始完成 Amazon S3 路径。Create a workflow from <blueprint> (从 <blueprint> 创建工作流) 页面生成文本字段和 Browse (浏览) 按钮。
`S3Bucket`	仅限 Amazon S3 存储桶名称。在 Create a workflow from <blueprint> (从 <blueprint> 创建工作流) 页生成存储桶选取器。
`IAMRoleArn`	AWS Identity and Access Management（IAM）角色的 Amazon Resource Name（ARN）。在 Create a workflow from <blueprint> (从 <blueprint> 创建工作流) 页生成角色选取器。
`IAMRoleName`	IAM 角色的名称。在 Create a workflow from <blueprint> (从 <blueprint> 创建工作流) 页生成角色选取器。

示例蓝图项目

数据格式转换是一种常见的提取、转换和加载（ETL）使用案例。在典型的分析工作负载中，基于列的文件格式（如 Parquet 或 ORC）优先于 CSV 或 JSON 等文本格式。此示例蓝图使您能够将 CSV/JSON/ 等格式的数据转换为 Amazon S3 文件的 Parquet。

此蓝图采用由蓝图参数定义的 S3 路径列表，将数据转换为 Parquet 格式，并将其写入另一个蓝图参数指定的 S3 位置。布局脚本为每个路径创建一个爬网程序和任务。布局脚本还将 `Conversion.py` 中

的 ETL 脚本上载到另一个蓝图参数指定的 S3 存储桶。然后，布局脚本将上载的脚本指定为每个任务的 ETL 脚本。项目的 ZIP 格式归档文件包含布局脚本、ETL 脚本和蓝图配置文件。

有关更多示例蓝图项目的信息，请参阅蓝图示例。

以下是布局脚本，位于文件 Layout.py。

```
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
```

以下是相应的蓝图配置文件 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."
        }
    }
```

```
}
```

文件 Conversion.py 中的以下脚本是上传的 ETL 脚本。请注意，它会在转换过程中保留分区方案。

```
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**
>
> 只能提供两个 Amazon S3 路径作为对示例蓝图的输入。这是因为 AWS Glue 触发器仅限于调用两个爬网程序操作。

测试蓝图

在开发代码时，应执行本地测试以验证工作流布局是否正确。

本地测试不会生成 AWS Glue 任务、爬网程序或触发器。相反，您可以在本地运行布局脚本并使用 `to_json()` 和 `validate()` 方法打印对象并查找错误。这些方法在库中定义的所有三个类中都可用。

可通过两种方式处理 AWS Glue 传递到您的布局函数的 `user_params` 和 `system_params` 参数。您的测试平台代码可以创建示例蓝图参数值的字典，并将其作为 `user_params` 参数传递到布局函数。或者，您可以移除对 `user_params` 的引用并使用硬编码字符串替代他们。

如果您的代码使用 `system_params` 参数中的 `region` 和 `accountId` 属性，您可以传入所拥有的 `system_params` 字典。

要测试蓝图

1. 在包含库的目录中启动 Python 解释器，或将蓝图文件和提供的库加载到首选的集成开发环境（IDE）中。
2. 确保您的代码导入提供的库。
3. 将代码添加到布局函数以调用在任何实体上或在 `Workflow` 对象上的 `validate()` 或 `to_json()`。例如，如果您的代码创建了一个名为 `mycrawler` 的 `Crawler` 对象，您可以调用 `validate()`，如下所示。

   ```
   mycrawler.validate()
   ```

   您可以打印 `mycrawler`，如下所示：

   ```
   print(mycrawler.to_json())
   ```

   如果您对一个对象调用 `to_json`，则不需要同时调用 `validate()`，因为 `to_json()` 调用了 `validate()`。

对工作流对象调用这些方法非常有用。假设您的脚本命名工作流对象 my_workflow，验证并打印工作流对象，如下所示。

```
print(my_workflow.to_json())
```

有关 to_json() 和 validate() 的更多信息，请参阅 类方法。

您还可以导入 pprint 并美观地打印列工作流对象，如本部分后面的示例所示。

4. 运行代码，修复错误，最后删除对 validate() 或者 to_json() 的调用。

Example

以下示例说明如何构建示例蓝图参数字典并将其作为 user_params 参数传入布局函数 generate_compaction_workflow。它还说明了如何美观地打印生成的工作流对象。

```
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)
```

发布蓝图

开发蓝图后，您必须将其上传到 Amazon S3 中。您必须对用于发布蓝图的 Amazon S3 存储桶具有写入权限。您还必须确保注册蓝图的 AWS Glue 管理员具有 Amazon S3 存储桶的读取权限。如需用于角色的 AWS Identity and Access Management（IAM）权限策略和用于 AWS Glue 蓝图的角色方面的相关建议，请参阅AWS Glue 蓝图的角色权限。

要发布蓝图

1. 创建必要的脚本、资源和蓝图配置文件。
2. 将所有文件添加到 zip 格式归档中，并将 zip 格式文件上传到 Amazon S3。对与用户将注册和运行蓝图的区域，使用与其相同区域的 S3 存储桶。

   您可以使用以下命令从命令行中创建 zip 格式文件。

```
zip -r folder.zip folder
```

3. 添加一个存储桶策略，该策略将授予 AWS 所需账户的读取权限。以下是策略示例。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-blueprints/*"
        }
    ]
}
```

4. 对 AWS Glue 管理员或对任何注册蓝图的人员授予 Amazon S3 存储桶的 IAM `s3:GetObject` 访问权限。有关授予管理员的示例策略，请参阅蓝图的 AWS Glue 管理员权限。

完成蓝图的本地测试后，您可能还需要在 AWS Glue 上测试蓝图。要在 AWS Glue 上测试蓝图，则必须注册。您可以使用 IAM 授权或使用单独的测试账户来限制可以查看已注册蓝图的人员。

> 另请参阅：
>
> - 在 AWS Glue 中注册蓝图

AWS Glue 蓝图类参考

适用于 AWS Glue 蓝图的库定义了在工作流布局脚本中使用的三个类：`Job`、`Crawler` 和 `Workflow`。

主题

- 作业类
- 爬网程序类
- 工作流类

- 类方法

作业类

Job 类表示一个 AWS Glue ETL 任务。

强制构造函数参数

以下是适用于 Job 类的强制构造函数参数。

参数名	类型	描述
Name	str	要分配给任务的名称。AWS Glue 将随机生成的后缀添加到名称中，以区分其他蓝图运行创建的任务。
Role	str	任务在执行时应担任角色的 Amazon Resource Name（ARN）。
Command	dict	任务命令，如指定在 API 文档中的 JobCommand 结构。

可选构造函数参数

以下是适用于 Job 类的可选构造函数参数。

参数名	类型	描述
DependsOn	dict	任务所依赖的工作流实体的列表。有关更多信息，请参阅 使用 DependsOn 参数。
WaitForDependencies	str	指示任务是应该等待它依赖的全部实体完成，还是等待任意实体完成。有关更多信息，请参阅 使用 WaitForDependencies 参数。如果任务仅依赖于一个实体，则省略。
（任务属性）	-	任何任务属性都在 AWS Glue API 文档中的Job 结构内列出（CreatedOn 和 LastModifiedOn 除外）。

爬网程序类

Crawler 类表示一个 AWS Glue 爬网程序。

强制构造函数参数

以下是适用于 Crawler 类的强制构造函数参数。

参数名	类型	描述
Name	str	要分配给爬网程序的名称。AWS Glue 将随机生成的后缀添加到名称中，以区分其他蓝图运行创建的爬网程序。
Role	str	运行时爬网程序应担任角色的 ARN。
Targets	dict	要网络爬取的目标集合。Targets 类构造函数在 API 文档中的CrawlerTargets 结构内定义。全部 Targets 构造函数参数是可选的，但您必须至少传递一个。

可选构造函数参数

以下是适用于 Crawler 类的可选构造函数参数。

参数名	类型	描述
DependsOn	dict	爬网程序所依赖的工作流实体列表。有关更多信息，请参阅 使用 DependsOn 参数。
WaitForDependencies	str	指示爬网程序是应该等待它依赖的全部实体完成，还是等待任意实体完成。有关更多信息，请参阅 使用 WaitForDependencies 参数。如果爬网程序仅依赖于一个实体，则省略。
（爬网程序属性）	-	所有爬网程序属性都在 AWS Glue API 文档中的Crawler 结构内列出，以下属性例外： • State

参数名	类型	描述
		• CrawlElapsedTime • CreationTime • LastUpdated • LastCrawl • Version

工作流类

Workflow 类表示一个 AWS Glue 工作流。工作流布局脚本返回 Workflow 对象。AWS Glue 基于此对象创建工作流。

强制构造函数参数

以下是适用于 Workflow 类的强制构造函数参数。

参数名	类型	描述
Name	str	要向工作流分配的名称。
Entities	Entities	要包含在工作流中的实体（任务和爬网程序）的集合。Entities 类构造函数接受 Jobs 参数（Job 对象列表）和 Crawlers 参数（Crawler 对象列表）。

可选构造函数参数

以下是适用于 Workflow 类的可选构造函数参数。

参数名	类型	描述
Description	str	请参阅 Workflow 结构。
DefaultRunProperties	dict	请参阅 Workflow 结构。

参数名	类型	描述
OnSchedule	str	cron 表达式

类方法

所有三个类都包括以下方法。

validate()

验证对象的属性，如果发现错误，则输出消息并退出。如果没有错误，则不生成输出。对于 Workflow 类，会在工作流中的每个实体上调用自己。

to_json()

将对象序列化为 JSON。还调用 validate()。对于 Workflow 类，JSON 对象包括任务和爬网程序列表，以及由任务和爬网程序依赖关系规范生成的触发器列表。

蓝图示例

有许多示例蓝图项目可用于 AWS Glue 蓝图 Github 存储库。这些示例仅供参考，不能用于生产。

示例项目的标题是：

- 压缩：此蓝图创建一个任务，根据所需的文件大小将输入文件压缩到更大的数据块中。
- 转换：此蓝图将各种标准文件格式的输入文件转换为 Apache Parquet 格式，该格式针对分析工作负载进行了优化。
- 网络爬取 Amazon S3 位置：此蓝图对多个 Amazon S3 位置进行网络爬取，将元数据表添加到数据目录。
- 自定义到数据目录的连接：此蓝图使用 AWS Glue 自定义连接器访问数据存储，读取记录，并根据记录架构填充 AWS Glue 数据目录中的表定义。
- 编码：此蓝图将您的非 UTF 文件转换为 UTF 编码的文件。
- 分区：此蓝图创建一个分区任务，根据特定的分区键将输出文件放入分区。
- 将 Amazon S3 数据导入到 DynamoDB 表中：此蓝图将数据从 Amazon S3 导入到 DynamoDB 表中。
- 所管理的标准表：此蓝图将 AWS Glue 数据目录表导入到 Lake Formation 表中。

在 AWS Glue 中注册蓝图

在 AWS Glue 开发人员已编码蓝图并将 ZIP 格式归档上传到 Amazon Simple Storage Service（Amazon S3）后，AWS Glue 管理员必须注册蓝图。注册蓝图使其可供使用。

注册蓝图时，AWS Glue 将蓝图归档复制到预留的 Amazon S3 位置。然后，您可以从上传位置删除归档。

如要注册蓝图，您需要对包含上传归档的 Amazon S3 位置具有读取权限。您还需要 AWS Identity and Access Management（IAM）权限 `glue:CreateBlueprint`。有关必须注册、查看和维护蓝图的 AWS Glue 管理员的建议权限，请参阅 蓝图的 AWS Glue 管理员权限。

您可以使用 AWS Glue 控制台、AWS Glue API 或 AWS Command Line Interface（AWS CLI）来注册蓝图。

注册蓝图（控制台）

1. 确保您对 Amazon S3 中的蓝图 ZIP 格式归档具有读取权限（`s3:GetObject`）。
2. 通过 https://console.aws.amazon.com/glue/ 打开 AWS Glue 控制台。

   以具有蓝图注册权限的用户身份登录。切换到包含蓝图 ZIP 格式归档的 Amazon S3 存储桶所在的相同 AWS 区域。
3. 在导航窗格中，选择 blueprints（蓝图）。然后在 blueprints（蓝图）页面上，选择 Add blueprint（添加蓝图）。
4. 输入蓝图名称和可选说明。
5. 在 ZIP archive location (S3) (ZIP 归档位置 (S3)) 中，输入已上传蓝图 ZIP 格式归档的 Amazon S3 路径。在路径中包含归档文件名，并以 `s3://` 作为路径的开头。
6. （可选）添加一个或多个标签。
7. 选择 Add blueprint (添加蓝图)。

   返回 blueprints（蓝图）页面，蓝图状态显示为 `CREATING`。选择刷新按钮，直到状态更改为 `ACTIVE` 或者 `FAILED`。
8. 如果状态为 `FAILED`，则选择蓝图，然后在 Actions (操作) 菜单上，选择 View (查看)。

   详细信息页面会显示故障原因。如果错误消息为“Unable to access object at location...”或者“Access denied on object at location...”，请检查以下要求：

   - 您在登录时使用的用户身份必须对 Amazon S3 中的蓝图 ZIP 格式归档具有读取权限。

- 包含 ZIP 格式归档的 Amazon S3 存储桶必须具有存储桶策略，该策略会为您的 AWS 账户 ID 授予对象读取权限。有关更多信息，请参阅 AWS Glue 中的开发蓝图。
- 您使用的 Amazon S3 存储桶必须位于您在控制台上登录的相同区域。

9. 确保数据分析人员对蓝图具有相应的权限。

   针对数据分析人员的建议 IAM policy 如蓝图的数据分析人员权限所述。该策略授予对任何资源的 `glue:GetBlueprint` 权限。如果您的策略在资源级别更精细，则授予数据分析人员对此新建资源的权限。

注册蓝图（AWS CLI）

1. 输入以下 命令。

```
aws glue create-blueprint --name <blueprint-name> [--description <description>] --
blueprint-location s3://<s3-path>/<archive-filename>
```

2. 要检查蓝图状态，请输入以下命令。重复该命令，直到状态变为 `ACTIVE` 或者 `FAILED`。

```
aws glue get-blueprint --name <blueprint-name>
```

   如果状态为 `FAILED` 并且错误消息为“Unable to access object at location...”或“Access denied on object at location...”，请检查以下要求：

   - 您在登录时使用的用户身份必须对 Amazon S3 中的蓝图 ZIP 格式归档具有读取权限。
   - 包含 ZIP 格式归档的 Amazon S3 存储桶必须具有存储桶策略，该策略会为您的 AWS 账户 ID 授予对象读取权限。有关更多信息，请参阅 发布蓝图。
   - 您使用的 Amazon S3 存储桶必须位于您在控制台上登录的相同区域。

> 另请参阅：
>
> - AWS Glue 中的蓝图概览

在 AWS Glue 中查看蓝图

查看蓝图，了解蓝图描述、状态和参数规范，并下载蓝图 ZIP 格式归档。

您可以使用 AWS Glue 控制台、AWS Glue API 或 AWS Command Line Interface（AWS CLI）来查看蓝图。

查看蓝图（控制台）

1. 打开 AWS Glue 控制台，地址：https://console.aws.amazon.com/glue/。
2. 在导航窗格中，选择 blueprints（蓝图）。
3. 在 blueprints（蓝图）页面上，选择一个蓝图。然后，在 Actions (操作) 菜单上，选择 View (查看)。

查看蓝图（AWS CLI）

- 输入以下命令，仅查看蓝图名称、描述和状态。将 *<blueprint-name>* 替换为要查看的蓝图的名称。

```
aws glue get-blueprint --name <blueprint-name>
```

命令输出类似如下。

```
{
    "Blueprint": {
        "Name": "myDemoBP",
        "CreatedOn": 1587414516.92,
        "LastModifiedOn": 1587428838.671,
        "BlueprintLocation": "s3://awsexamplebucket1/demo/
DemoBlueprintProject.zip",
        "Status": "ACTIVE"
    }
}
```

输入以下命令，同时查看参数规范。

```
aws glue get-blueprint --name <blueprint-name>  --include-parameter-spec
```

命令输出类似如下。

```
{
    "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"
    }
}
```

添加 `--include-blueprint` 参数以便在输出中包含 URL，您可以将其粘贴到浏览器，以下载 AWS Glue 存储的蓝图 ZIP 格式归档。

> 另请参阅：
>
> - AWS Glue 中的蓝图概览

在 AWS Glue 中更新蓝图

如果您具有修订的布局脚本、修订的蓝图参数集或修订的支持文件，则可以更新蓝图。更新蓝图将创建一个新版本。

更新蓝图不会影响根据蓝图创建的现有工作流。

您可以通过使用 AWS Glue 控制台、AWS Glue API 或 AWS Command Line Interface（AWS CLI）更新蓝图。

以下程序假设 AWS Glue 开发人员已经创建更新后的蓝图 ZIP 格式归档并将其上传到 Amazon S3。

更新蓝图（控制台）

1. 确保您对 Amazon S3 中的蓝图 ZIP 格式归档具有读取权限（`s3:GetObject`）。
2. 通过 https://console.aws.amazon.com/glue/ 打开 AWS Glue 控制台。

以具有蓝图更新权限的用户身份登录。切换到包含蓝图 ZIP 格式归档的 Amazon S3 存储桶所在的相同 AWS 区域。

3. 在导航窗格中，选择 blueprints（蓝图）。
4. 在 blueprints（蓝图）页面上，选择一个蓝图，然后在 Actions（操作）菜单上选择 Edit（编辑）。
5. 在 Edit a blueprint (编辑蓝图) 页面上，更新蓝图 Description (描述) 或者 ZIP archive location (S3) (ZIP 归档位置（S3）)。请务必在路径中包含归档文件的名称。
6. 选择 Save (保存)。

   返回 blueprints（蓝图）页面，蓝图状态显示为 UPDATING。选择刷新按钮，直到状态更改为 ACTIVE 或者 FAILED。
7. 如果状态为 FAILED，则选择蓝图，然后在 Actions (操作) 菜单上，选择 View (查看)。

   详细信息页面会显示故障原因。如果错误消息为“Unable to access object at location...”或者“Access denied on object at location...”，请检查以下要求：

   - 您在登录时使用的用户身份必须对 Amazon S3 中的蓝图 ZIP 格式归档具有读取权限。
   - 包含 ZIP 格式归档的 Amazon S3 存储桶必须具有存储桶策略，该策略会为您的 AWS 账户 ID 授予对象读取权限。有关更多信息，请参阅 发布蓝图。
   - 您使用的 Amazon S3 存储桶必须位于您在控制台上登录的相同区域。

> **Note**
>
> 如果更新失败，则下次蓝图运行将使用已成功注册或更新的最新蓝图版本。

更新蓝图（AWS CLI）

1. 输入以下 命令。

```
aws glue update-blueprint --name <blueprint-name> [--description <description>] --
blueprint-location s3://<s3-path>/<archive-filename>
```

2. 要检查蓝图状态，请输入以下命令。重复该命令，直到状态变为 ACTIVE 或者 FAILED。

```
aws glue get-blueprint --name <blueprint-name>
```

如果状态为 FAILED 并且错误消息为“Unable to access object at location...”或“Access denied on object at location...”，请检查以下要求：

- 您在登录时使用的用户身份必须对 Amazon S3 中的蓝图 ZIP 格式归档具有读取权限。
- 包含 ZIP 格式归档的 Amazon S3 存储桶必须具有存储桶策略，该策略会为您的 AWS 账户 ID 授予对象读取权限。有关更多信息，请参阅 发布蓝图。
- 您使用的 Amazon S3 存储桶必须位于您在控制台上登录的相同区域。

另请参阅

- AWS Glue 中的蓝图概览

在 AWS Glue 中从蓝图创建工作流

您可以手动创建 AWS Glue 工作流，一次添加一个组件，也可以从 AWS Glue 蓝图创建工作流。AWS Glue 包括常见用例的蓝图。您的 AWS Glue 开发人员可以创建其他蓝图。

Important

将工作流中任务、爬网程序和触发器的总数限制为 100 个或更少。如果包含超过 100 个，则在尝试恢复或停止工作流运行时可能会出错。

使用蓝图时，您可以根据蓝图定义的常见使用案例快速生成特定使用案例的工作流。您可以通过为蓝图参数提供值来定义特定使用案例。例如，对数据集进行分区的蓝图可以将 Amazon S3 源和目标路径作为参数。

AWS Glue 通过运行蓝图从蓝图创建工作流。蓝图运行保存您提供的参数值，并用于跟踪工作流及其组件的创建进度和结果。对工作流进行问题排查时，您可以查看蓝图运行以确定用于创建工作流的蓝图参数值。

要创建和查看工作流，您需要特定的 IAM 权限。有关建议的 IAM policy，请参阅蓝图的数据分析人员权限。

您可以通过使用 AWS Glue 控制台、AWS Glue API 或 AWS Command Line Interface（AWS CLI）从蓝图创建工作流。

从蓝图（控制台）创建工作流

1. 通过 https://console.aws.amazon.com/glue/ 打开 AWS Glue 控制台。

   以拥有创建工作流权限的用户身份登录。
2. 在导航窗格中，选择 blueprints（蓝图）。
3. 选择一个蓝图，然后在 Actions (操作) 菜单中，选择 Create workflow (创建工作流)。
4. 在 Create a workflow from <blueprint-name> (从 <blueprint-name> 创建工作流) 页面上，输入以下信息：

   蓝图参数

   这些因蓝图设计而异。有关参数的问题，请咨询开发人员。蓝图通常包含工作流名称的参数。

   IAM 角色

   AWS Glue 在创建工作流及其组件时承担的角色。该角色必须具有创建和删除工作流、任务、爬网程序和触发器的权限。有关该角色的建议策略，请参阅 蓝图角色的权限。
5. 选择 Submit (提交)。

   此时会显示 Blueprint Details (蓝图详细信息) 页面，并在底部显示蓝图运行列表。
6. 在蓝图运行列表中，检查最上面的蓝图运行以了解工作流创建状态。

   初始状态为 RUNNING。选择刷新按钮，直到状态变为 SUCCEEDED 或者 FAILED。
7. 请执行以下操作之一：

   - 如果完成状态为 SUCCEEDED，您可以转到 Workflows (工作流) 页面，选择新创建的工作流，然后运行它。在运行工作流之前，您可以查看设计图。
   - 如果完成状态为 FAILED，则选择蓝图运行，然后在 Actions (操作) 菜单上，选择 View (查看) 以查看错误消息。

有关工作流和蓝图的更多信息，请参阅以下主题。

- AWS Glue 中的工作流概述
- 在 AWS Glue 中更新蓝图
- 在 AWS Glue 中手动创建和构建工作流

在 AWS Glue 中查看蓝图运行

查看蓝图运行，了解以下信息：

- 已创建的工作流的名称。
- 用于创建工作流的蓝图参数值。
- 工作流创建操作的状态。

您可以使用 AWS Glue 控制台、AWS Glue API 或 AWS Command Line Interface（AWS CLI）来查看蓝图运行。

查看蓝图运行（控制台）

1. 打开 AWS Glue 控制台，地址：https://console.aws.amazon.com/glue/。
2. 在导航窗格中，选择 blueprints（蓝图）。
3. 在 blueprints（蓝图）页面上，选择一个蓝图。然后，在 Actions (操作) 菜单上，选择 View (查看)。
4. 在 Blueprint Details (蓝图详细信息) 页面底部，选择一个蓝图运行，然后在 Actions (操作) 菜单上，选择 View (查看)。

查看蓝图运行（AWS CLI）

- 输入以下 命令。将 *<blueprint-name>* 替换为蓝图的名称。将 *<blueprint-run-id>* 替换为蓝图运行 ID。

```
aws glue get-blueprint-run --blueprint-name <blueprint-name> --run-id <blueprint-run-id>
```

另请参阅：

- AWS Glue 中的蓝图概览

适用于 AWS Glue 的 AWS CloudFormation

AWS CloudFormation 是可创建许多 AWS 资源的服务。AWS Glue 提供了 API 操作以在 AWS Glue Data Catalog 中创建对象。但是，在 AWS CloudFormation 模板文件中定义并创建 AWS Glue 对象和其他相关 AWS 资源对象可能更方便。然后，您可以自动化创建对象的过程。

AWS CloudFormation 提供了简化的语法 JSON（JavaScript 对象表示法）或 YAML（YAML Ain't 标记语言）来表示 AWS 资源的创建。您可以使用 AWS CloudFormation 模板来定义数据目录对象，如数据库、表、分区、爬网程序、分类器和连接。您还可以定义 ETL 对象，如作业、触发器和开发终端节点。您可创建一个模板来描述所需的所有 AWS 资源，而 AWS CloudFormation 则可为您预配和配置这些资源。

相关详情，请参阅《AWS CloudFormation 用户指南》中的什么是 AWS CloudFormation？以及使用 AWS CloudFormation 模板。

如果您计划使用与 AWS Glue 兼容的 AWS CloudFormation 模板，则您作为管理员，必须授予对 AWS CloudFormation 及其依赖的 AWS 服务和操作的访问权。要授予创建 AWS CloudFormation 资源的权限，请将以下策略附加到使用 AWS CloudFormation 的用户：

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

下表包含 AWS CloudFormation 模板可代表您执行的操作。它包括指向您可添加到 AWS CloudFormation 模板的 AWS 资源类型及其属性类型的相关信息的链接。

AWS Glue 资源	AWS CloudFormation 模板	AWS Glue 示例
分类器	AWS::Glue::Classifier	Grok 分类器、JSON 分类器、XML 分类器

AWS Glue 资源	AWS CloudFormation 模板	AWS Glue 示例
Connection	AWS::Glue::Connection	MySQL 连接
爬网程序	AWS::Glue::Crawler	Amazon S3 爬网程序、MySQL 爬网程序
数据库	AWS::Glue::Database	空数据库、具有表的数据库
开发终端节点	AWS::Glue::DevEndpoint	开发终端节点
任务	AWS::Glue::Job	Amazon S3 任务、JDBC 任务
机器学习转换	AWS::Glue::MLTransform	机器学习转换
数据质量规则集	AWS::Glue::DataQualityRuleset	数据质量规则集、使用 EventBridge 调度器的数据质量规则集
分区	AWS::Glue::Partition	表的分区
表	AWS::Glue::Table	数据库中的表
触发器	AWS::Glue::Trigger	按需触发器、计划触发器、条件触发器

要开始使用，请使用以下示例模板并使用您自己的元数据对其进行自定义。然后，使用 AWS CloudFormation 控制台创建 AWS CloudFormation 堆栈以将对象添加到 AWS Glue 和任何关联的服务。AWS Glue 对象中的许多字段都是可选字段。这些模板说明了必填字段或正常运行的 AWS Glue 对象所需的字段。

AWS CloudFormation 模板可以采用 JSON 或 YAML 格式。在这些示例中，使用了 YAML 以便于阅读。这些示例包含注释 (#) 以介绍模板中定义的值。

AWS CloudFormation 模板可以包含 `Parameters` 部分。可以在示例文本中或在将 YAML 文件提交到 AWS CloudFormation 控制台以创建堆栈时更改此部分。模板的 `Resources` 部分包含 AWS Glue 和相关对象的定义。AWS CloudFormation 模板语法定义可能包含包括更详细的属性语法的属性。可能并非所有属性都是创建 AWS Glue 对象所必需的。这些示例显示创建 AWS Glue 对象时常用的属性的示例值。

AWS Glue 数据库的示例 AWS CloudFormation 模板

数据目录中的 AWS Glue 数据库包含元数据表。数据库包含非常少的属性，可在数据目录中使用 AWS CloudFormation 模板进行创建。提供了以下示例模板以帮助您入门并说明如何将 AWS CloudFormation 堆栈与 AWS Glue 一起使用。示例模板创建的唯一资源是名为 `cfn-mysampledatabase` 的数据库。您可以更改它，方法是编辑示例的文本，或在提交 YAML 时在 AWS CloudFormation 控制台上更改值。

下面显示创建 AWS Glue 数据库时常用的属性的示例值。有关 AWS Glue 的 AWS CloudFormation 数据库模板的更多信息，请参阅 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 数据库、表和分区的示例 AWS CloudFormation 模板

AWS Glue 表包含定义要使用 ETL 脚本处理的数据的结构和位置的元数据。在表中，可以定义分区以并行处理您的数据。分区是您使用键定义的数据块。例如，使用月份作为键，1 月的所有数据包含在同一分区中。在 AWS Glue 中，数据库可以包含表，表可以包含分区。

以下示例显示如何使用 AWS CloudFormation 模板填充数据库、表和分区。基本数据格式为 `csv` 并使用逗号 (,) 分隔。因为数据库必须先存在才能包含表，表必须先存在才能创建分区，所以模板在创建这些对象时使用 `DependsOn` 语句来定义它们的依赖关系。

此示例中的值定义一个包含公开可用的 Amazon S3 存储桶中的航班数据的表。为方便说明，仅定义了一些数据列和一个分区键。还在数据目录中定义了 4 个分区。还在 `StorageDescriptor` 字段中显示了用于描述基本数据的存储的一些字段。

```
---
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
```

AWS Glue grok 分类器的示例 AWS CloudFormation 模板

AWS Glue 分类器确定数据的架构。一种类型的自定义分类器使用 grok 模式来匹配您的数据。如果模式匹配，则使用自定义分类器来创建您的表的架构并将 `classification` 设置为分类器定义中设置的值。

此示例创建了一个分类器，该分类器创建了具有一个名为 `message` 的列的架构并将 classification 设置为 `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
```

AWS Glue JSON 分类器的示例 AWS CloudFormation 模板

AWS Glue 分类器确定数据的架构。一种类型的自定义分类器使用 JsonPath 字符串，该字符串定义供分类器分类的 JSON 数据。AWS Glue 支持小部分适用于 JsonPath 的运算符，如编写 JsonPath 自定义分类器中所述。

如果模式匹配，则使用自定义分类器来创建您的表的架构。

此示例创建了一个分类器，该分类器创建一个架构，其每条记录都在对象的 Records3 数组中。

```
---
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[*]
```

AWS Glue XML 分类器的示例 AWS CloudFormation 模板

AWS Glue 分类器确定数据的架构。一种类型的自定义分类器指定 XML 标签来指定包含要分析的 XML 文档中的每条记录的元素。如果模式匹配，则使用自定义分类器来创建您的表的架构并将 `classification` 设置为分类器定义中设置的值。

此示例创建了一个分类器，该分类器创建了一个其每条记录位于 `Record` 标签中的架构并将分类设置为 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>
```

Amazon S3 的 AWS Glue 爬网程序的示例 AWS CloudFormation 模板

AWS Glue 爬网程序在数据目录中创建与您的数据对应的元数据表。然后，可以在您的 ETL 任务中使用这些表定义作为源和目标。

此示例在数据目录中创建爬网程序、所需的 IAM 角色和 AWS Glue 数据库。当此爬网程序运行时，它会代入 IAM 角色并在数据库中为公用航班数据创建一个表。使用前缀“cfn_sample_1_”创建此表。此模板创建的 IAM 角色允许全局权限；您可能希望创建自定义角色。此分类器没有定义任何自定义分类器。默认使用 AWS Glue 内置分类器。

当您将此示例提交到 AWS CloudFormation 控制台时，您必须确认要创建 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\"}}}"
```

AWS Glue 连接的示例 AWS CloudFormation 模板

数据目录中的 AWS Glue 连接包含连接到 JDBC 数据库所需的 JDBC 和网络信息。在您连接到 JDBC 数据库以运行 ETL 作业或对其进行爬网时，会使用此信息。

此示例创建到名为 devdb 的 Amazon RDS MySQL 数据库的连接。使用此连接时，还必须提供 IAM 角色、数据库凭证和网络连接值。请参阅模板中的必填字段的详细信息。

```
---
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
```

JDBC 的 AWS Glue 爬网程序的示例 AWS CloudFormation 模板

AWS Glue 爬网程序在数据目录中创建与您的数据对应的元数据表。然后，可以在您的 ETL 任务中使用这些表定义作为源和目标。

此示例在数据目录中创建爬网程序、所需的 IAM 角色和 AWS Glue 数据库。当此爬网程序运行时，它会代入 IAM 角色并在数据库中为存储在 MySQL 数据库中的公用航班数据创建一个表。使用前缀“cfn_jdbc_1_”创建此表。此模板创建的 IAM 角色允许全局权限；您可能希望创建自定义角色。不能为 JDBC 数据定义自定义分类器。默认使用 AWS Glue 内置分类器。

当您将此示例提交到 AWS CloudFormation 控制台时，您必须确认要创建 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\"}}}"
```

Amazon S3 到 Amazon S3 AWS Glue 作业的示例 AWS CloudFormation 模板

数据目录中的 AWS Glue 任务包含在 AWS Glue 中运行脚本所需的参数值。

此示例创建从 csv 格式的 Amazon S3 存储桶读取航班数据并将其写入 Amazon S3 Parquet 文件的任务。此任务运行的脚本必须已存在。您可以使用 AWS Glue 控制台为您的环境生成 ETL 脚本。在运行此任务时，还必须提供具有正确权限的 IAM 角色。

模板中显示了常用参数值。例如，AllocatedCapacity（DPU）默认为 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
```

JDBC 到 Amazon S3 的 AWS Glue 作业的示例 AWS CloudFormation 模板

数据目录中的 AWS Glue 任务包含在 AWS Glue 中运行脚本所需的参数值。

此示例创建从名为 `cfn-connection-mysql-flights-1` 的连接所定义的 MySQL JDBC 数据库读取航班数据并将其写入 Amazon S3 Parquet 文件的任务。此任务运行的脚本必须已存在。您可以使用 AWS Glue 控制台为您的环境生成 ETL 脚本。在运行此任务时，还必须提供具有正确权限的 IAM 角色。

模板中显示了常用参数值。例如，`AllocatedCapacity`（DPU）默认为 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 按需触发器的示例 AWS CloudFormation 模板

数据目录中的 AWS Glue 触发器包含在它触发时启动任务运行所需的参数值。在您启用按需触发器时，该按需触发器触发。

此示例创建启动一个名为 `cfn-job-S3-to-S3-1` 的作业的按需触发器。

```
---
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 计划触发器的示例 AWS CloudFormation 模板

数据目录中的 AWS Glue 触发器包含在它触发时启动任务运行所需的参数值。在启用计划触发器并弹出 cron 计时器时，该计划触发器触发。

此示例创建启动一个名为 `cfn-job-S3-to-S3-1` 的作业的计划触发器。计时器是在工作日每隔 10 分钟运行一次作业的 cron 表达式。

```
---
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 条件触发器的示例 AWS CloudFormation 模板

数据目录中的 AWS Glue 触发器包含在它触发时启动任务运行所需的参数值。在启用条件触发器并满足其条件 (如作业成功完成) 时，该条件触发器触发。

此示例创建启动一个名为 cfn-job-S3-to-S3-1 的作业的条件触发器。在名为 cfn-job-S3-to-S3-2 的作业成功完成时，此作业启动。

```
---
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 开发终端节点的示例 AWS CloudFormation 模板

AWS Glue 机器学习转换是用于清理数据的自定义转换。当前有一个名为 FindMatches 的可用转换。通过 FindMatches 转换，您可以识别数据集中的重复或匹配记录，即使记录没有公共唯一标识符且没有完全匹配的字段也是如此。

此示例创建机器学习转换。有关创建机器学习转换所需参数的更多信息，请参阅 与 AWS Lake Formation FindMatches 匹配的记录。

```
---
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 规则集的示例 AWS CloudFormation 模板

AWS Glue 数据质量规则集包含可以在 Data Catalog 中的表上进行评估的规则。将规则集放在目标表上后，您可以进入 Data Catalog 并运行评估，根据规则集中的这些规则运行数据。这些规则可能各不相同，从评估行数到评估数据的引用完整性。

以下示例是 CloudFormation 模板，该模板在指定的目标表上创建包含各种规则的规则集。

```
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 调度器的 AWS Glue Data Quality 规则集的示例 AWS CloudFormation 模板

AWS Glue 数据质量规则集包含可以在 Data Catalog 中的表上进行评估的规则。将规则集放在目标表上后，您可以进入 Data Catalog 并运行评估，根据规则集中的这些规则运行数据。您不必手动进入 Data Catalog 来评估规则集，而是可以在我们的 CloudFormation 模板中添加 EventBridge 调度器，按定时间隔为您计划这些规则集评估。

以下示例是一个 CloudFormation 模板，它创建了一个 Data Quality 规则集和一个 EventBridge 调度器，用于每五分钟评估一次上述规则集。

```
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"
  # 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 开发终端节点的示例 AWS CloudFormation 模板

AWS Glue 开发终端节点是可用于开发和测试您的 AWS Glue 脚本的环境。

此示例使用成功创建开发终端节点所需的最少网络参数值创建开发终端节点。有关设置开发终端节点所需的参数的更多信息，请参阅为 AWS Glue 设置开发网络。

您需要提供现有的 IAM 角色 ARN（Amazon Resource Name）来创建开发终端节点。如果您计划在开发终端节点上创建笔记本电脑服务器，请提供有效的 RSA 公有密钥并保持对应的私有密钥可用。

> **Note**
>
> 对于您创建的任何与开发终端节点关联的 notebook 服务器，都可以对其进行管理。因此，如果您删除开发终端节点以删除笔记本服务器，您必须在 AWS CloudFormation 控制台上删除 AWS CloudFormation 堆栈。

```
---
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 编程指南

脚本包含从源中提取数据、转换数据并将数据加载到目标中的代码。AWS Glue 在启动作业时运行脚本。

AWS Glue ETL 脚本使用 Python 或 Scala 编码。虽然所有作业类型都可以用 Python 编写，但 AWS Glue for Spark 作业也可以用 Scala 编写。当您为 AWS Glue Studio 中的作业自动生成源代码逻辑时，会创建脚本。您可以编辑此脚本，也可以提供自己的脚本来处理您的 ETL 作业。

提供您自己的自定义脚本

脚本在 AWS Glue 中执行提取、转换和加载 (ETL) 工作。当您为作业自动生成源代码逻辑时，将会创建一个脚本。您可以编辑这个生成的脚本，也可以提供您自己的自定义脚本。

要在 AWS Glue 中提供您自己的自定义脚本，请遵循以下常规步骤：

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 选择 ETL 作业选项卡，然后查看创建作业部分。选择脚本编辑器选项。
3. 在 This job runs 下，选择以下内容之一：
   - 使用样板代码创建新脚本
   - 上传和编辑现有脚本
4. 在作业详细信息页面中，选择运行您的自定义脚本所需的 IAM 角色。有关更多信息，请参阅 AWS Glue 的身份和访问管理。
5. 选择您的脚本引用的任何连接。需要这些对象才能连接到必要的 JDBC 数据存储。

   弹性网络接口 (ENI) 是一种虚拟网络接口，您可以将其连接至 Virtual Private Cloud（VPC）中的实例。选择连接到脚本中使用的数据存储所需的弹性网络接口。
6. 提供特定于您的作业类型的额外配置，包括参数。有关您的作业类型配置的更多信息，请参阅 使用 AWS Glue Studio 构建可视化 ETL 作业 一节。
7. 在脚本选项卡上，粘贴或编写您的自定义脚本。

使用本节中的内容来指导编写自定义脚本的过程。

有关在 AWS Glue 中添加作业的更多信息，请参阅使用 AWS Glue Studio 构建可视化 ETL 作业。

有关分步指导，请参阅 控制台中的 Add jobAWS Glue 教程。

Spark 脚本编程

使用 AWS Glue 可轻松编写或自动生成提取、转换和加载 (ETL) 脚本，以及测试并运行这些脚本。本部分介绍 AWS Glue 引入的 Apache Spark 扩展，并提供如何在 Python 和 Scala 中编写和运行 ETL 脚本的示例。

> **Important**
>
> 不同版本的 AWS Glue 支持不同版本的 Apache Spark。您的自定义脚本必须与受支持的 Apache Spark 版本兼容。有关 AWS Glue 版本的信息，请参阅 Glue version job property。

主题

- 教程：编写 AWS Glue for Spark 脚本
- 在 AWS Glue ETL 脚本中编程 PySpark
- 在 Scala 中编写 AWS Glue ETL 脚本
- AWS Glue for Spark ETL 编程的功能和优化

教程：编写 AWS Glue for Spark 脚本

本教程向您介绍编写 AWS Glue 脚本的过程。脚本可以与作业一起按计划运行，也可以与交互式会话交互运行。有关作业的更多信息，请参阅使用 AWS Glue Studio 构建可视化 ETL 作业。有关交互式会话的更多信息，请参阅 the section called “AWS Glue 交互式会话概览”。

AWS Glue Studio 可视化编辑器提供了一个图形化的无代码界面，用于构建 AWS Glue 作业。 AWS Glue 脚本支持可视化作业。它们可让您访问用于使用 Apache Spark 程序的扩展工具集。您可以从 AWS Glue 脚本中访问原生 Spark API 以及简化提取、转换和加载 (ETL) 工作流程的 Gl AWS ue 库。

在本教程中，您将提取、转换和加载停车罚单数据集。完成这项工作的脚本在形式和功能上与 AWS 大数据博客上的 “使用 AWS Glue Studio简化ETL” 中生成的脚本相同，后者介绍了Glue Studio可视化编辑器。 AWS 通过在作业中运行此脚本，您可以将其与可视化作业进行比较，并了解 AWS Glue ETL 脚本的工作原理。这可以让您为使用可视化作业中尚不可用的其他功能做好准备。

在本教程中，您使用 Python 语言和库。Scala 中提供了类似的功能。完成本教程后，您应该能够生成和检查 Scala 脚本示例，以了解如何执行 Scala G AWS lue ETL 脚本编写过程。

先决条件

本教程包含以下先决条件：

- 与 Glue Stu AWS dio 博客文章相同的先决条件，后者指导你运行 AWS CloudFormation 模板。

  此模板使用 AWS Glue 数据目录来管理中可用的停车罚单数据集`s3://aws-bigdata-blog/artifacts/gluestudio/`。它创建了以下将引用的资源：
- AWS Glue Studio角色 - 要运行 AWS Glue 任务的 IAM 角色
- AWS Glue StudioAmazon S3 存储桶 - 用于存储与博客相关的文件的 Amazon S3 存储桶的名称
- AWS Glue StudioTicketsYYZDB – AWS Glue 数据目录数据库
- AWS Glue StudioTableTickets— 用作源的数据目录表
- AWS Glue StudioTableTrials— 用作源的数据目录表
- AWS Glue StudioParkingTicketCount — 用作目标的数据目录表
- 该脚本在 Glue Stud AWS io 博客文章中生成。如果博客文章发生更改，脚本也可以在以下文本中使用。

生成示例脚本

您可以将 AWS Glue Studio 可视化编辑器用作强大的代码生成工具，为要编写的脚本创建脚手架。您将使用此工具创建示例脚本。

如果您想跳过这些步骤，教程提供了此脚本。

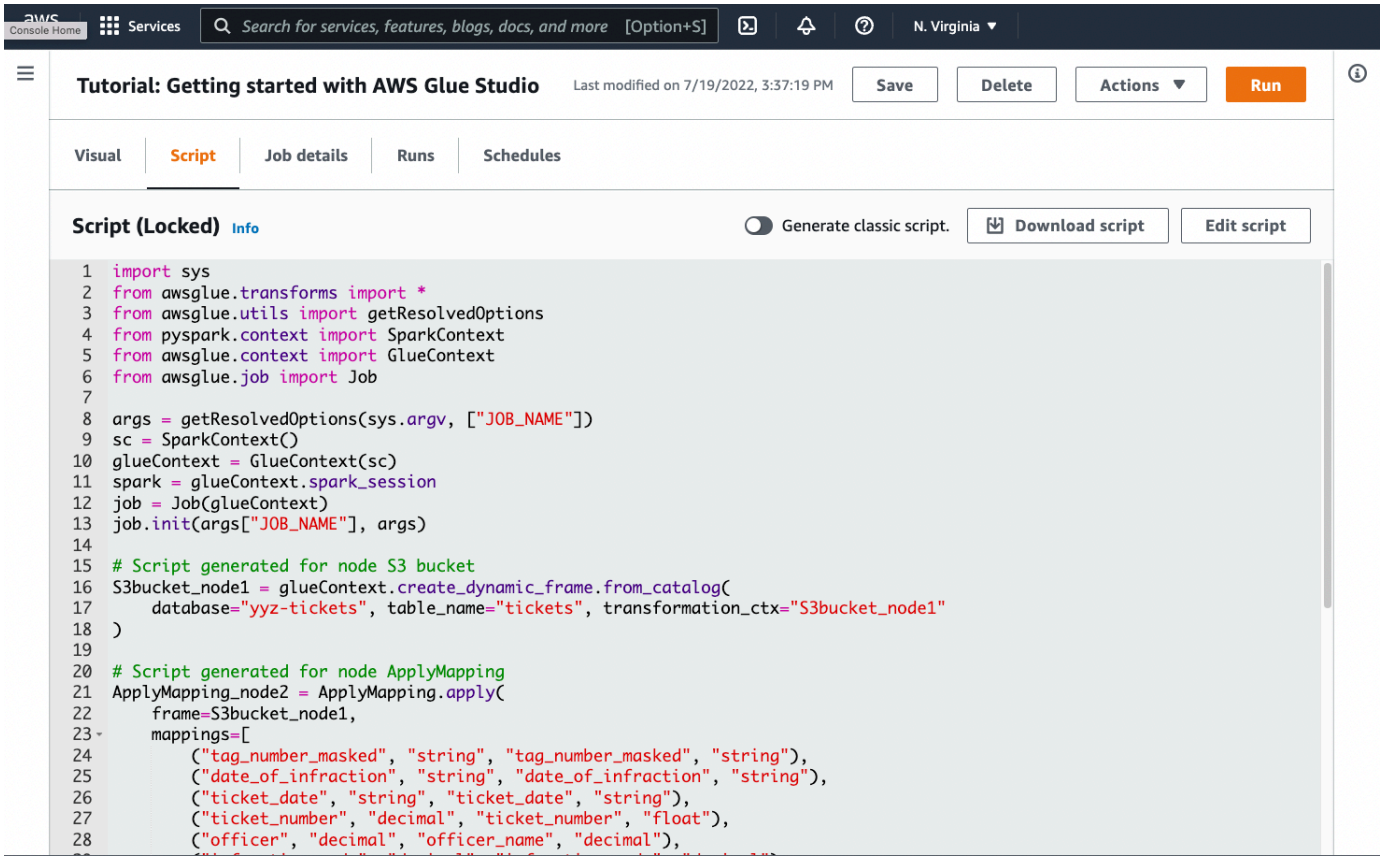
教程示例脚本

```
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()
```

生成示例脚本

1. 完成 Glue AWS Studio 教程。要完成本教程，请参阅通过示例作业在 AWS Glue Studio 中创建作业。
2. 导航到作业页面上的 Script（脚本）选项卡，如以下屏幕截图所示：



3. 复制 Script（脚本）选项卡的完整内容。通过在 Job details（作业详细信息）中设置脚本语言，您可以在生成 Python 或 Scala 代码之间来回切换。

第 1 步。创建作业并粘贴您的脚本

在此步骤中，您将在中创建一个 AWS Glue 作业 AWS Management Console。这会设置一个配置，允许 AWS Glue 运行你的脚本。它还为您创建了一个存储和编辑脚本的位置。

创建作业

1. 在中 AWS Management Console，导航到 AWS Glue 登录页面。
2. 在侧面的导航窗格中，选择 Jobs（作业）。
3. 选择 Create job（创建作业）中的 Spark script editor（Spark 脚本编辑器），然后选择 Create（创建）。
4. 可选 - 将脚本的全文粘贴到 Script（脚本）窗格中。或者，您可以按照教程进行操作。

第 2 步。导入 AWS Glue 库

您需要将脚本设置为与脚本外部定义的代码和配置进行交互。这项工作是在 AWS Glue Studio 的幕后完成的。

在此步骤中，您执行以下操作。

- 导入并初始化 `GlueContext` 对象。从脚本编写的角度来看，这是最重要的导入。这公开了定义源数据集和目标数据集的标准方法，是所有 ETL 脚本的起点。要了解有关 `GlueContext` 类的更多信息，请参阅 GlueContext 班级。
- 初始化 `SparkContext` 和 `SparkSession`。它们允许您配置 AWS Glue 作业中可用的 Spark 引擎。您无需在入门 AWS Glue 脚本中直接使用它们。
- 调用 `getResolvedOptions` 以准备在脚本中使用的作业参数。有关解析作业参数的更多信息，请参阅 the section called “getResolvedOptions”。
- 初始化 `Job`。该`Job`对象设置配置并跟踪各种可选 AWS Glue 功能的状态。脚本可在没有 `Job` 对象的情况下运行，但最佳实践是对其进行初始化，以便以后集成这些功能时不会混淆。

  其中一项功能是您可以选择在本教程中进行配置的作业书签。您可以在下一节 the section called “可选 - 启用作业书签” 中了解作业书签。

在此过程中，您将编写以下代码。此代码是生成的示例脚本的一部分。

```
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)
```

导入 AWS Glue 库

- 复制此部分的代码并将其粘贴到 Script（脚本）编辑器中。

> **Note**
>
> 您可能会认为复制代码是一种糟糕的工程实践。在本教程中，我们建议这样做是为了鼓励您在所有 AWS Glue ETL 脚本中一致地命名核心变量。

第 3 步。从源提取数据

在任何 ETL 过程中，您首先需要定义要更改的源数据集。在 AWS Glue Studio 可视化编辑器中，您可以通过创建源节点来提供此信息。

在此步骤中，您为 `create_dynamic_frame.from_catalog` 方法提供 `database` 和 `table_name`，以从 AWS Glue 数据目录中配置的源提取数据。

在上一步中，您初始化了 `GlueContext` 对象。可使用此对象查找用于配置源的方法，例如 `create_dynamic_frame.from_catalog`。

在此过程中，您将使用 `create_dynamic_frame.from_catalog` 编写以下代码。此代码是生成的示例脚本的一部分。

```
S3bucket_node1 = glueContext.create_dynamic_frame.from_catalog(
    database="yyz-tickets", table_name="tickets", transformation_ctx="S3bucket_node1"
    )
```

从源提取数据

1. 查看文档，找到从 Glue 数据目录中定义的源中 AWS 提取数据的方法。`GlueContext`这些方法记录在 the section called “GlueContext” 中。选择 create_dynamic_frame.from_catalog 方法。在 `glueContext` 上调用此方法。
2. 查看 `create_dynamic_frame.from_catalog` 的文档。此方法需要 `database` 和 `table_name` 参数。为 `create_dynamic_frame.from_catalog` 提供必要的参数。

   AWS Glue 数据目录存储有关源数据的位置和格式的信息，是在先决条件部分中设置的。您不必直接向脚本提供该信息。
3. 可选 - 为方法提供 `transformation_ctx` 参数，以支持作业书签。您可以在下一节 the section called “可选 - 启用作业书签” 中了解作业书签。

> **Note**
>
> 提取数据的常用方法
> the section called “create_dynamic_frame_from_catalog”用于连接 Glue 数据 AWS 目录中的表。
> 如果需要直接为作业提供描述源结构和位置的配置，请参阅 the section called “create_dynamic_frame_from_options” 方法。与使用 `create_dynamic_frame.from_catalog` 时相比，您需要提供更详细的参数来描述数据。请参阅有关 `format_options` 和 `connection_parameters` 的补充文档以确定所需参数。有关如何提供源数据格式相关脚本信息的说明，请参阅 the section called “数据格式选项”。有关如何提供源数据位置相关脚本信息的说明，请参阅 the section called “连接参数”。
> 如果从流式传输源读取信息，您可以通过 the section called “create_data_frame_from_catalog” 或 the section called “create_data_frame_from_options” 方法为作业提供源信息。请注意，这些方法会返回 Apache Spark `DataFrames`。
> 我们生成的代码将调用 `create_dynamic_frame.from_catalog`，同时参考文档指向 `create_dynamic_frame_from_catalog`。这些方法最终调用相同的代码，并且包含在内以编写更纯净的代码。您可以通过查看 Python 包装器的源代码来验证这一点，该源代码位于 aws-glue-libs。

第 4 步。使用 AWS Glue 转换数据

在 ETL 过程中提取源数据后，您需要描述想要如何更改数据。您可以通过在 AWS Glue Studio 可视化编辑器中创建变换节点来提供这些信息。

在此步骤中，为 `ApplyMapping` 方法提供当前和所需字段名称和类型的映射以转换 `DynamicFrame`。

您可执行以下转换。

- 删除四个 `location` 和 `province` 键。
- 将 `officer` 的名称更改为 `officer_name`。
- 将 `ticket_number` 和 `set_fine_amount` 的类型更改为 `float`。

`create_dynamic_frame.from_catalog` 为您提供一个 `DynamicFrame` 对象。A `DynamicFrame` 表示 Glue 中的数据 AWS 集。 AWS Glue 变换是会发生变`DynamicFrames`化的操作。

> **Note**
>
> 什么是 DynamicFrame？
> DynamicFrame 是一个抽象概念，可让您将数据集与数据中条目的名称和类型的描述联系起来。在 Apache Spark 中，存在一种类似的抽象概念，叫做。 DataFrame有关说明 DataFrames，请参阅 Spark SQL 指南。
> 借助 DynamicFrames，您可以动态描述数据集架构。考虑一个带有价格列的数据集，其中一些条目将价格存储为字符串，而另一些条目则将价格存储为双精度。 AWS Glue 计算架构 on-the-fly ——它为每行创建一条自描述记录。
> 不一致的字段（如价格）在框架的架构中用类型（ChoiceType）显式表示。您可以解决不一致的字段，方法是使用 DropFields 删除或使用 ResolveChoice 解析这些字段。这些是 DynamicFrame 上可用的转换。然后，您可以使用 writeDynamicFrame 将数据写回数据湖。
> 您可以从 DynamicFrame 类的方法中调用许多相同的转换，从而得到更易读的脚本。有关 DynamicFrame 的更多信息，请参阅the section called “DynamicFrame”。

在此过程中，您将使用 ApplyMapping 编写以下代码。此代码是生成的示例脚本的一部分。

```
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",
)
```

使用 AWS Glue 转换数据

1. 查看文档以确定用于更改和删除字段的转换。有关更多信息，请参阅 the section called “GlueTransform”。选择 ApplyMapping 转换。有关 ApplyMapping 的更多信息，请参阅the section called “ApplyMapping”。对 ApplyMapping 转换对象调用 apply。

   > **Note**
   >
   > 什么是 ApplyMapping？
   > ApplyMapping 采用 DynamicFrame 并对其进行转换。它需要一个表示字段转换的元组列表，即“映射”。前两个元组元素（字段名称和类型）用于标识框架中的字段。后两个参数也是字段名称和类型。
   > ApplyMapping 将源字段转换为目标名称并键入一个新的名称DynamicFrame，返回该名称。未提供的字段在返回值中删除。
   > 您可以使用 DynamicFrame 上的 apply_mapping 方法调用相同的转换，而不是调用 apply，以创建更流畅、可读的代码。有关更多信息，请参阅 the section called “apply_mapping”。

2. 查看 ApplyMapping 的文档以确定所需的参数。请参阅 the section called “ApplyMapping”。您会发现此方法需要 frame 和 mappings 参数。为 ApplyMapping 提供必要的参数。
3. 可选 - 为方法提供 transformation_ctx 以支持作业书签。您可以在下一节 the section called “可选 - 启用作业书签” 中了解作业书签。

> **Note**
>
> Apache Spark 功能
> 我们提供转换，以简化作业中的 ETL 工作流。您还可以访问作业中 Spark 程序提供的、为更通用的目的而构建的库。为使用这些库，您将在 DynamicFrame 和 DataFrame 之间转换。您可以使用 the section called “toDF” 创建 DataFrame。然后，您可以使用上提供的方法 DataFrame 来转换您的数据集。有关这些方法的更多信息，请参阅DataFrame。然后，你可以使用向后转换the section called “fromDF”，使用 AWS Glue 操作将帧加载到目标。

第 5 步。将数据加载到目标中

转换数据后，通常将转换后的数据存储在与源不同的位置。您可以通过在 Glue Studio 可视化编辑器中 AWS 创建目标节点来执行此操作。

在此步骤中，您将为 write_dynamic_frame.from_options 方法提供 connection_type、connection_options、format 和 format_options 以将数据加载到 Amazon S3 中的目标桶中。

在步骤 1 中，您初始化了 GlueContext 对象。在 AWS Glue 中，你可以在这里找到用于配置目标的方法，就像源一样。

在此过程中，您将使用 write_dynamic_frame.from_options 编写以下代码。此代码是生成的示例脚本的一部分。

```
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",
    )
```

将数据加载到目标中

1. 查看文档，找到将数据加载到目标 Amazon S3 桶中的方法。这些方法记录在 the section called “GlueContext” 中。选择 the section called “write_dynamic_frame_from_options” 方法。在 glueContext 上调用此方法。

   > **Note**
   >
   > 加载数据的常用方法
   > write_dynamic_frame.from_options 是用于加载数据的最常用方法。它支持 Glue 中可用的所有 AWS 目标。
   > 如果您要写入在 Glue 连接中定义的 JDBC 目标，请使用该方法。 AWS the section called “write_dynamic_frame_from_jdbc_conf” AWS Glue 连接存储有关如何连接到数据源的信息。这样就无需在 connection_options 中提供该信息。但是，您仍需要使用 connection_options 来提供 dbtable。
   > write_dynamic_frame.from_catalog 并非加载数据的常用方法。此方法在不更新基础数据集的情况下更新 AWS Glue 数据目录，并且与其他更改基础数据集的过程结合使用。有关更多信息，请参阅 the section called “更新架构并添加新分区”。

2. 查看 the section called “write_dynamic_frame_from_options” 的文档。此方法需要 `frame`、`connection_type`、`format`、`connection_options` 和 `format_options`。在 `glueContext` 上调用此方法。

   a. 请参阅有关 `format_options` 和 `format` 的补充文档以确定您需要的参数。有关数据格式说明，请参阅 the section called “数据格式选项”。

   b. 请参阅有关 `connection_type` 和 `connection_options` 的补充文档以确定您需要的参数。有关连接说明，请参阅 the section called “连接参数”。

   c. 为 `write_dynamic_frame.from_options` 提供必要的参数。此方法与 `create_dynamic_frame.from_options` 的配置类似。

3. 可选 - 为 `transformation_ctx` 提供 `write_dynamic_frame.from_options` 以支持作业书签。您可以在下一节 the section called “可选 - 启用作业书签” 中了解作业书签。

第 6 步。提交 **Job** 对象

您在步骤 1 中初始化了 `Job` 对象。您需要在脚本结束时手动结束其生命周期。某些可选功能需要此操作才能正常运行。这项工作是在 AWS Glue Studio 的幕后完成的。

在此步骤中，调用 `commit` 对象上的 `Job` 方法。

在此过程中，您将编写以下代码。此代码是生成的示例脚本的一部分。

```
job.commit()
```

提交 **Job** 对象

1. 如果尚未完成，请执行前面部分中概述的可选步骤以包含 `transformation_ctx`。
2. 调用 `commit`。

可选 - 启用作业书签

在之前的每个步骤中，都会指示您设置 `transformation_ctx` 参数。这与称为作业书签的功能有关。

借助作业书签，您可以使用重复运行的作业节省时间和金钱，而不是使用可以轻松跟踪以前工作的数据集。Job 书签可追踪之前运行的数据集中 AWS Glue 变换的进度。通过跟踪之前运行的结束位置，AWS Glue 可以将其工作限制在以前未处理过的行上。有关作业书签的更多信息，请参阅 the section called “使用作业书签跟踪已处理的数据”。

要启用作业书签，请先将 `transformation_ctx` 添加到提供的函数中，如前面的示例所述。作业书签状态在运行期间保持不变。`transformation_ctx` 参数是用于访问该状态的键。这些语句本身没有任何作用。您还需要激活作业配置中的功能。

在此过程中，您可以使用 AWS Management Console启用作业书签。

启用作业书签

1. 导航到相应作业的 Job details（作业详细信息）部分。
2. 将 Job bookmark（作业书签）设置为 Enable（启用）。

第 7 步。将代码作为作业运行

在此步骤中，运行作业以验证是否已成功完成本教程。这可以通过点击按钮来完成，就像在 AWS Glue Studio可视化编辑器中一样。

将代码作为作业运行

1. 选择标题栏上的 Untitled job（未命名作业）以编辑和设置作业名称。
2. 导航到 Job details（作业详细信息）选项卡。为作业分配 IAM Role（IAM 角色）。你可以在 AWS Glue Studio 教程的先决条件中使用 AWS CloudFormation 模板创建的模板。如果已完成该教程，则角色应作为 `AWS Glue StudioRole` 提供。
3. 选择 Save（保存）以保存您的脚本。
4. 选择 Run（运行）以运行您的作业。
5. 导航到 Runs（运行）选项卡以验证作业完成。
6. 导航到 *`DOC-EXAMPLE-BUCKET`*（`write_dynamic_frame.from_options` 的目标）。确认输出符合您的预期。

有关配置和管理作业的更多信息，请查阅 the section called “提供您自己的自定义脚本”。

更多信息

Apache Spark 库和方法在 Glue 脚本中 AWS 可用。您可以查看 Spark 文档以了解如何使用这些随附的库。有关更多信息，请参阅 Spark 源存储库的示例部分。

AWS 默认情况下，Glue 2.0+ 包含几个常见的 Python 库。还有一些机制可以将你自己的依赖项加载到 Scala 或 Python 环境中的 AWS Glue 作业中。有关 Python 依赖项的信息，请参阅 the section called “Python 库”。

有关如何在 Python 中使用 AWS Glue 功能的更多示例，请参阅the section called “Python 示例”。Scala 和 Python 作业具有同等的功能，因此我们的 Python 示例应该会让您了解如何在 Scala 中进行类似的工作。

在 AWS Glue ETL 脚本中编程 PySpark

您可以在 GitHub 网站的示例存储库AWS Glue中找到 Python 代码示例和实用工具。AWS Glue

将 Python 和 AWS Glue 一起使用

AWS Glue 支持 PySpark Python 方言的扩展，用于编写提取、转换和加载 (ETL) 作业脚本。本节介绍如何在 ETL 脚本中以及如何通过 AWS Glue API 使用 Python。

- 进行设置以便将 Python 与 AWS Glue 一起使用
- 在 Python 中调用 AWS Glue API
- 将 Python 库与 AWS Glue 一起使用
- AWS Glue Python 代码示例

AWS Glue PySpark 扩展

AWS Glue已为 PySpark Python 方言创建了以下扩展。

- 使用 `getResolvedOptions` 访问参数
- PySpark 扩展类型
- DynamicFrame 类
- DynamicFrameCollection 类
- DynamicFrameWriter 类
- DynamicFrameReader 班级
- GlueContext 班级

AWS Glue PySpark 变换

AWS Glue已创建以下转换类以在 PySpark ETL 操作中使用。

- GlueTransform 基类
- ApplyMapping 类

- DropFields 类
- DropNullFields 类
- ErrorsAsDynamicFrame 类
- FillMissingValues 类
- Filter 类
- FindIncrementalMatches 类
- FindMatches 类
- FlatMap 类
- Join 类
- Map 类
- MapToCollection 类
- mergeDynamicFrame
- Relationalize 类
- RenameField 类
- ResolveChoice 类
- SelectFields 类
- SelectFromCollection 类
- Spigot 类
- SplitFields 类
- SplitRows 类
- Unbox 类
- UnnestFrame 类

进行设置以便将 Python 与 AWS Glue 一起使用

使用 Python 为 Spark 作业开发 ETL 脚本。ETL 任务支持的 Python 版本取决于任务的 AWS Glue 版本。有关 AWS Glue 版本的更多信息，请参阅 Glue version job property。

设置您的系统以便将 Python 与 AWS Glue 一起使用

按照以下步骤安装 Python 并能够调用 AWS Glue API。

1. 如果您还没有安装 Python，请从 Python.org 下载页面进行下载和安装。

2. 按照 AWS CLI 文档中所述安装 AWS Command Line Interface（AWS CLI）。

   AWS CLI 不是使用 Python 所直接必需的。但是，安装和配置它是使用账户凭证设置 AWS 并验证它们是否工作的方便方法。

3. 按照 Boto3 快速入门所述安装 AWS SDK for Python（Boto 3）。

   Boto 3 资源 API 尚不可用于 AWS Glue。目前，只有 Boto 3 客户端 API 可用。

   有关 Boto 3 的更多信息，请参阅 AWS SDK for Python（Boto3）入门。

您可以在 GitHub 网站上的 AWS Glue 示例存储库中找到 AWS Glue 的 Python 代码示例和实用程序。

在 Python 中调用 AWS Glue API

请注意，Boto 3 资源 API 尚不可用于 AWS Glue。目前，只有 Boto 3 客户端 API 可用。

Python 中的 AWS Glue API 名称

Java 和其他编程语言中的 AWS Glue API 名称通常是 CamelCased。但是，当从 Python 调用时，这些通用名称将更改为小写，部分的某些名称用下划线字符隔开，使它们更“Pythonic”。在 AWS Glue API 参考文档中，这些 Pythonic 名称在通用 CamelCased 名称之后列在括号中。

但是，尽管 AWS Glue API 名称自身转换为小写，其参数名称仍保持大写。请务必记住这一点，因为在调用 AWS Glue API 时会按名称传递参数，如下一节中所述。

在 AWS Glue 中传递和访问 Python 参数

在 Python 对 AWS Glue API 的调用中，最好按名称显式传递参数。例如：

```
job = glue.create_job(Name='sample', Role='Glue_DefaultRole',
                      Command={'Name': 'glueetl',
                               'ScriptLocation': 's3://my_script_bucket/scripts/
my_etl_script.py'})
```

了解 Python 创建您可以在 Job 结构 或 JobRun 结构 中指定为 ETL 脚本参数的名称/值元组的字典是很有帮助的。然后，Boto 3 通过 REST API 调用，以 JSON 格式将其传递给 AWS Glue。这意味着，当您在脚本中访问这些参数时，不能依赖它们的顺序。

例如，假设您在 Python Lambda 处理程序函数中启动 JobRun，并且您希望指定多个参数。您的代码看起来可能类似于：

```
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 } )
```

要在 ETL 脚本中可靠地访问这些参数，请使用 AWS Glue 的 `getResolvedOptions` 函数，然后从生成的字典访问它们：

```
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']
```

如果您想传递属于嵌套 JSON 字符串的参数，以便在它传递给您的 AWS Glue ETL 任务时保存参数值，您必须在开始任务运行之前对参数字符串进行编码，然后在引用任务脚本之前对参数字符串进行解码。例如，考虑以下参数字符串：

```
glue_client.start_job_run(JobName = "gluejobname", Arguments={
"--my_curly_braces_string": '{"a": {"b": {"c": [{"d": {"e": 42}}]}}}'
})
```

要正确传递此参数，您应将参数编码为 Base64 编码的字符串。

```
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}")
...
```

示例：创建并运行作业

下面的示例显示如何使用 Python 调用 AWS Glue API 来创建和运行 ETL 作业。

创建并运行作业

1. 创建 AWS Glue 客户端的实例。

```
import boto3
glue = boto3.client(service_name='glue', region_name='us-east-1',
              endpoint_url='https://glue.us-east-1.amazonaws.com')
```

2. 创建作业。您必须使用 `glueetl` 作为名称的 ETL 命令，如以下代码所示：

```
myJob = glue.create_job(Name='sample', Role='Glue_DefaultRole',
                          Command={'Name': 'glueetl',
                                   'ScriptLocation': 's3://my_script_bucket/
scripts/my_etl_script.py'})
```

3. 启动您在上一步中创建的作业的新运行：

```
myNewJobRun = glue.start_job_run(JobName=myJob['Name'])
```

4. 获取作业状态：

```
status = glue.get_job_run(JobName=myJob['Name'], RunId=myNewJobRun['JobRunId'])
```

5. 打印作业运行的当前状态：

```
print(status['JobRun']['JobRunState'])
```

将 Python 库与 AWS Glue 一起使用

AWS Glue 允许您安装额外的 Python 模块和库，以便与 AWS Glue ETL 一起使用。

主题

- 使用 pip 在 AWS Glue 2.0+ 中安装其他 Python 模块
- 包括具有 PySpark 原生功能的 Python 文件
- 使用视觉对象转换的编程脚本
- AWS Glue 中已提供的 Python 模块
- 压缩库以用于包含
- 在 AWS Glue Studio 笔记本中加载 Python
- 在开发终端节点中加载 Python 库
- 在作业中使用 Python 库或 JobRun

使用 pip 在 AWS Glue 2.0+ 中安装其他 Python 模块

AWS Glue 使用 Python Package Installer（pip3）安装 AWS Glue ETL 使用的其他模块。您可以将 `--additional-python-modules` 参数与逗号分隔的 Python 模块列表结合使用，以添加新模块或更改现有模块的版本。通过将分发上传到 Amazon S3，然后在模块列表中包含到 Amazon S3 对象的路径，您可以安装库的自定义分发。

您可以使用 `--python-modules-installer-option` 参数将其他选项传递给 pip3。例如，您可以传递 `"--upgrade"` 以升级 `"--additional-python-modules"` 指定的包。有关更多示例，请参阅使用 GI AWS ue 2.0 为 Spark ETL 工作负载从轮子上构建 Python 模块。

如果您的 Python 依赖关系暂时依赖于原生的编译代码，则可能会遇到以下限制： AWS Glue 不支持在作业环境中编译原生代码。但是， AWS Glue 作业在亚马逊 Linux 2 环境中运行。您可通过 Wheel 分发以编译后的形式提供原生依赖项。

例如，要更新或添加新的 `scikit-learn` 模块，请使用以下键/值：`"--additional-python-modules", "scikit-learn==0.21.3"`。

此外，在 `--additional-python-modules` 选项中，您可以指定指向 Python Wheel 模块的 Amazon S3 路径。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
```

您可以在AWS Glue控制台的 Job 参数字段`--additional-python-modules`中指定，或者通过更改 AWS SDK 中的作业参数来指定。有关设置作业参数的更多信息，请参阅 the section called “任务参数”。

包括具有 PySpark 原生功能的 Python 文件

AWS Glue用于在 AWS G PySpark lue ETL 作业中包含 Python 文件。如果可用，您希望使用 `--additional-python-modules` 管理依赖项。您可以使用 `--extra-py-files` 作业参数以包含 Python 文件。依赖关系必须托管在 Amazon S3 中，参数值应为逗号分隔的 Amazon S3 路径列表，且不含空格。此功能的行为类似于您将在 Spark 中使用的 Python 依赖项管理。有关 Spark 中的 Python 依赖关系管理的更多信息，请参阅 Apache Spark 文档中的使用 PySpark 原生功能页面。 `--extra-py-files`在其他代码未打包的情况下，或者在使用现有工具链迁移 Spark 程序以管理依赖关系时非常有用。为使依赖项工具可维护，您必须在提交之前捆绑依赖项。

使用视觉对象转换的编程脚本

使用 AWS Glue Studio 可视化界面创建 AWS Glue 作业时，您可以使用托管数据转换节点和自定义视觉变换来转换数据。有关托管式数据转换节点的更多信息，请参阅 the section called “编辑 AWS Glue 托管数据转换节点”。有关自定义视觉对象转换的更多信息，请参阅 the section called “ 自定义视觉转换 ”。只有当作业的语言设置为使用 Python 时，才能生成使用视觉对象转换的脚本。

使用视觉变换生成 AWS Glue 作业时， AWS Glue Studio 将使用作业配置中的`--extra-py-files`参数将这些变换包含在运行时环境中。有关任务参数的更多信息，请参阅 the section called “任务参数”。对生成的脚本或运行时环境进行更改时，需要保留此作业配置以确保脚本的成功运行。

AWS Glue 中已提供的 Python 模块

要更改这些已提供模块的版本，请使用 `--additional-python-modules` 作业参数提供新版本。

AWS Glue version 2.0

AWS Glue 2.0 版包括以下开箱即用的 Python 模块：

- 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
- PyMySQL==0.9.3

- 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 版包括以下开箱即用的 Python 模块：

- 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
- PyMySQL==1.0.2
- 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 版包括以下开箱即用的 Python 模块：

- 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
- kaleido==0.2.1
- 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
- plotly==5.16.0
- pmdarima==2.0.1
- ptvsd==4.3.2
- pyarrow==10.0.0
- pydevd==2.5.0
- pyhocon==0.3.58
- PyMySQL==1.0.2
- 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

压缩库以用于包含

除非库包含在单个 `.py` 文件中，否则它应打包到 `.zip` 存档中。包目录应该位于存档文件的根部，并且必须包含一个针对该包的 `__init__.py` 文件。然后，Python 将能够以正常方式导入包。

如果您的库仅在一个 `.py` 文件中包含单个 Python 模块，您无需将其放入 `.zip` 文件。

在 AWS Glue Studio 笔记本中加载 Python

要在 Glue Studio 笔记本中 AWS 指定 Python 库，请参阅安装其他 Python 模块。

在开发终端节点中加载 Python 库

如果对不同的 ETL 脚本使用不同的库集，则可以为每个集设置单独的开发终端节点，也可以覆盖每次您切换脚本时开发终端节点加载的库 `.zip` 文件。

在创建开发终端节点时，您可以使用控制台为其指定一个或多个库 .zip 文件。在分配名称和 IAM 角色后，请选择 Script Libraries and job parameters (optional) (脚本库和任务参数（可选）)，然后在 Python library path (脚本库路径) 框中输入库 `.zip` 文件的完整 Amazon S3 路径。例如：

```
s3://bucket/prefix/site-packages.zip
```

如果需要，您可以指定文件的多个完整路径并使用逗号分隔，但不能有空格，如下所示：

```
s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/lib_X.zip
```

如果您更新这些 .zip 文件，则可以使用控制台将其重新导入到您的开发终端节点。导航到所涉开发人员终端节点，选中它旁边的框，然后从 Action 菜单中选择 Update ETL libraries。

类似地，您可以使用 AWS Glue API 指定库文件。当您通过调用 CreateDevEndpoint 操作（Python：create_dev_endpoint）创建开发终端节点时，可以通过如下所示的调用在 ExtraPythonLibsS3Path 参数中指定库的一个或多个完整路径：

```
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")
```

当您更新一个开发终端节点时，还可以通过在调用 UpdateDevEndpoint (update_dev_endpoint) 时使用 DevEndpointCustomLibraries 对象并将 UpdateEtlLibraries 参数设置为 True 来更新它所加载的库。

在作业中使用 Python 库或 JobRun

当您在控制台上创建新任务时，可以通过选择 Script Libraries and job parameters (optional) (脚本库和任务参数 (可选)) 并输入完整的 Amazon S3 库路径来指定一个或多个库 .zip 格式文件（方法与创建开发终端节点时相同）：

```
s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/lib_X.zip
```

如果您正在调用 CreateJob（create_job），则可以使用 --extra-py-files 默认参数指定默认库的一个或多个完整路径，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'})
```

然后，当你启动时 JobRun，你可以用不同的库设置来覆盖默认的库设置：

```
runId = glue.start_job_run(JobName='sampleJob',
                           Arguments={'--extra-py-files': 's3://bucket/prefix/
lib_B.zip'})
```

AWS Glue Python 代码示例

- 代码示例：对数据进行联接和关系化
- 代码示例：使用 ResolveChoice、Lambda 和 ApplyMapping 进行数据准备

代码示例：对数据进行联接和关系化

本示例使用从 http://everypolitician.org/ 下载到 Amazon Simple Storage Service（Amazon S3）中的 `sample-dataset` 存储桶的数据集：`s3://awsglue-datasets/examples/us-legislators/all`。该数据集包含有关美国议员及其在美国众议院和参议院中占有的席位的数据（JSON 格式），并且已针对本教程进行了轻微修改且在公共 Amazon S3 存储桶中提供。

您可以在 GitHub 网站上 AWS Glue 示例存储库的 `join_and_relationalize.py` 文件中找到本示例的源代码。

利用此数据，本教程将介绍如何执行以下操作：

- 使用 AWS Glue 爬网程序对存储在公有 Amazon S3 存储桶中的对象进行分类并将其架构保存到 AWS Glue Data Catalog。
- 检查生成自爬网的表元数据和架构。
- 编写 Python 提取、转移和加载（ETL）脚本，该脚本使用数据目录中的元数据执行以下操作：
  - 将不同源文件中的数据加入到单个数据表中 (即，使数据非规范化)。
  - 按议员类型筛选已加入到单独的表中的表。
  - 将生成的数据写入单独的 Apache Parquet 文件以供日后分析。

在 AWS 上运行时，调试 Python 或 PySpark 脚本的首选方法是在 AWS Glue Studio 上使用笔记本。

步骤 1：爬取 Amazon S3 存储桶中的数据

1. 登录 AWS Management Console 并打开位于 https://console.aws.amazon.com/glue/ 的 AWS Glue 控制台。

2. 按照 配置爬网程序 中的步骤操作，创建可将 s3://awsglue-datasets/examples/us-legislators/all 数据集网络爬取到 AWS Glue Data Catalog 中名为 legislators 的数据库的新爬网程序。示例数据已位于此公共 Amazon S3 存储桶中。
3. 运行新爬网程序，然后检查 legislators 数据库。

   该爬网程序将创建以下元数据表：

   - persons_json
   - memberships_json
   - organizations_json
   - events_json
   - areas_json
   - countries_r_json

   这是一个包含议员及其历史记录的半规范化表集合。

步骤 2：向开发终端节点笔记本中添加样板文件脚本

将以下样板文件脚本粘贴到开发终端节点笔记本中以导入所需的 AWS Glue 库，然后设置单个 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())
```

步骤 3：检查数据目录中数据的架构

接下来，您可以轻松地从 AWS Glue Data Catalog 检查 DynamicFrame，并检查数据的架构。例如，要查看 persons_json 表的架构，请在您的笔记本中添加以下内容：

```
persons = glueContext.create_dynamic_frame.from_catalog(
```

```
             database="legislators",
             table_name="persons_json")
print "Count: ", persons.count()
persons.printSchema()
```

下面是来自打印调用的输出：

```
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
```

表中的每个人都是某个美国国会机构的成员。

要查看 memberships_json 表的架构，请键入以下内容：

```
memberships = glueContext.create_dynamic_frame.from_catalog(
                 database="legislators",
                 table_name="memberships_json")
print "Count: ", memberships.count()
memberships.printSchema()
```

您可以在一个 (扩展) 代码行中执行所有这些操作：

```
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 是美国国会的党派和两大议院，即参议院和众议院。要查看 organizations_json 表的架构，请键入以下内容：

```
orgs = glueContext.create_dynamic_frame.from_catalog(
           database="legislators",
           table_name="organizations_json")
print "Count: ", orgs.count()
orgs.printSchema()
```

您可以在一个 (扩展) 代码行中执行所有这些操作：

```
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
```

步骤 4：筛选数据

接下来，仅保留您需要的字段，然后将 `id` 重命名为 `org_id`。该数据集足够小，方便您完整查看。

`toDF()` 将 DynamicFrame 转换为 Apache Spark `DataFrame`，因此您可以应用已存在于 Apache Spark SQL 中的转换：

```
orgs = orgs.drop_fields(['other_names',
                         'identifiers']).rename_field(
                             'id', 'org_id').rename_field(
                                'name', 'org_name')
orgs.toDF().show()
```

下面显示了输出：

```
+--------------+--------------------+--------------------+--------------------+-----
+-----------+--------------------+
|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|
+--------------+--------------------+--------------------+--------------------+-----
+-----------+--------------------+
```

键入以下内容以查看显示在 memberships 中的 organizations：

```
memberships.select_fields(['organization_id']).toDF().distinct().show()
```

下面显示了输出：

```
+--------------------+
|     organization_id|
+--------------------+
|d56acebe-8fdc-47b...|
|8fa6c3d2-71dc-478...|
+--------------------+
```

步骤 5：整合内容

现在，使用 AWS Glue 联接这些关系表并创建一个包含议员 memberships 及其对应的 organizations 的完整历史记录表。

1. 首先，联接 id 和 person_id 上的 persons 和 memberships。
2. 接下来，将结果与 org_id 和 organization_id 上的 orgs 联接。
3. 然后，删除多余的字段 person_id 和 org_id。

您可以在一个 (扩展) 代码行中执行所有这些操作：

```
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()
```

您可以在一个 (扩展) 代码行中执行所有这些操作：

```
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
```

您现在有了可用于分析的最终表。您可以采用紧凑且高效的格式写入该表以供分析（即 Parquet），该格式可让您在 AWS Glue、Amazon Athena 或 Amazon Redshift Spectrum 中运行 SQL。

以下调用将跨多个文件写入该表以在日后执行分析时支持快速并行读取：

```
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")
```

要将所有历史记录数据放入单个文件，您必须将其转换为数据帧，为其重新分区然后写入它：

```
s_history = l_history.toDF().repartition(1)
s_history.write.parquet('s3://glue-sample-target/output-dir/legislator_single')
```

或者，如果您希望按参议院和众议院分隔该数据：

```
l_history.toDF().write.parquet('s3://glue-sample-target/output-dir/legislator_part',
```

```
                                    partitionBy=['org_name'])
```

步骤 6：转换关系数据库的数据

利用 AWS Glue，您可以轻松将数据写入到关系数据库（如 Amazon Redshift），甚至对半结构化数据也是如此。它提供了一种转换 `relationalize`，这将展平 DynamicFrames，无论帧中的对象的复杂度可能如何。

使用本示例中的 `l_history` DynamicFrame，传入根表的名称 (`hist_root`) 和 `relationalize` 的临时工作路径。这将返回 DynamicFrameCollection。您随后可以列出该集合中 DynamicFrames 的名称：

```
dfc = l_history.relationalize("hist_root", "s3://glue-sample-target/temp-dir/")
dfc.keys()
```

以下是 `keys` 调用的输出：

```
[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 将该历史记录表拆分为 6 个新表：1 个包含 DynamicFrame 中每个对象的记录的根表以及 5 个用于数组的辅助表。关系数据库中的数组处理通常不够理想，尤其是在这些数组变大时。将这些数组分成不同的表会使查询进展得快得多。

接下来，通过检查 `contact_details` 来查看分隔：

```
l_history.select_fields('contact_details').printSchema()
dfc.select('hist_root_contact_details').toDF().where("id = 10 or id =
 75").orderBy(['id','index']).show()
```

以下是 `show` 调用的输出：

```
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 字段是原始 DynamicFrame 中的一个结构数组。这些数组中的每个元素都是辅助表中的单独的行，通过 index 编制索引。此处的 id 是具有键 contact_details 的 hist_root 表的外键：

```
dfc.select('hist_root').toDF().where(
    "contact_details = 10 or contact_details = 75").select(
       ['id', 'given_name', 'family_name', 'contact_details']).show()
```

下面是输出：

```
+--------------------+----------+-----------+---------------+
|                  id|given_name|family_name|contact_details|
+--------------------+----------+-----------+---------------+
|f4fc30ee-7b42-432...|      Mike|       Ross|             10|
|e3c60f34-7d1b-4c0...|   Shelley|     Capito|             75|
+--------------------+----------+-----------+---------------+
```

请注意，这些命令中先后使用了 toDF() 和 where 表示式来筛选您要查看的行。

因此，将 hist_root 表与辅助表联接可让您执行以下操作：

- 将数据加载到不支持数组的数据库中。
- 使用 SQL 查询数组中的每个单独的项目。

使用 AWS Glue 连接安全地存储和访问您的 Amazon Redshift 凭证。有关如何创建您自己的连接的信息，请参阅 连接到数据。

现在，您已准备就绪，可以通过一次遍历一个 DynamicFrames 来将数据写入到连接：

```
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)
```

根据您的关系数据库类型，连接设置将有所不同：

- 有关写入到 Amazon Redshift 的说明，请参阅 the section called “Redshift 连接”。
- 有关其他数据库，请参阅 AWS Glue for Spark 中适用于 ETL 的连接类型和选项。

结论

总的来说，AWS Glue 非常灵活。它让您只需几行代码即可完成通常需要编写几天才能实现的功能。您可以在 GitHub 上 join_and_relationalize.py 示例的 Python 文件 AWS Glue 中找到完整的源到目标 ETL 脚本。

代码示例：使用 ResolveChoice、Lambda 和 ApplyMapping 进行数据准备

此示例使用的数据集包括从两个 Data.CMS.gov 数据集下载的医疗保健提供商付款数据：“Inpatient Prospective Payment System Provider Summary for the Top 100 Diagnosis-Related Groups - FY2011”和“Inpatient Charge Data FY 2011”。下载该数据后，我们修改了数据集，以在文件末尾引入了几个错误的记录。这个修改过的文件位于 s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv 上的公有 Amazon S3 存储桶。

您可以在 AWS Glue 示例 GitHub 存储库中的 data_cleaning_and_lambda.py文件中找到本示例的源代码。

在 AWS 上运行时，调试 Python 或 PySpark 脚本的首选方法是在 AWS Glue Studio 上使用笔记本。

步骤 1：爬取 Amazon S3 存储桶中的数据

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 按照 配置爬网程序 中描述的过程进行操作，创建新的爬网程序，它可以网络爬取 s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv 文件，而且可以将生成的元数据放入 AWS Glue 数据目录中一个名为 payments 的数据库。
3. 运行新爬网程序，然后检查 payments 数据库。在读取该文件的开头以确定其格式和分隔符之后，您应该发现爬网程序已经在数据库中创建了一个名为 medicare 的元数据表。

   新 medicare 表的架构如下所示：

```
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
```

步骤 2：向开发终端节点笔记本中添加样板文件脚本

将以下样板文件脚本粘贴到开发终端节点笔记本中以导入所需的 AWS Glue 库，然后设置单个 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())
```

步骤 3：比较不同的架构解析

接下来，您可以查看由 Apache Spark DataFrame 识别的架构是否与您的 AWS Glue 爬网程序记录的架构相同。运行此代码：

```
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()
```

下面是来自 printSchema 调用的输出：

```
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)
```

接下来，查看 AWS Glue DynamicFrame 生成的架构：

```
medicare_dynamicframe = glueContext.create_dynamic_frame.from_catalog(
       database = "payments",
       table_name = "medicare")
medicare_dynamicframe.printSchema()
```

printSchema 中的输出如下所示：

```
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
```

DynamicFrame 生成一个架构，在其中 provider id 可以是 long 或 string 类型。DataFrame 架构将 Provider Id 列为 string 类型，数据目录将 provider id 列为 bigint 类型。

哪一个是正确的？ 文件末尾有两条记录 (共计 16 万条记录)，该列中有 string 值。这些是为说明问题而引入的错误记录。

为解决此类问题，AWS Glue DynamicFrame 引入了 choice 类型的概念。在这种情况下，DynamicFrame 显示 long 和 string 值都出现在该列中。AWS Glue 爬网程序错过了 string 值，因为它仅被视为数据的一个 2 MB 前缀。Apache Spark DataFrame 考虑了整个数据集，但它被迫将最一般的类型分配给该列，即 string。事实上，当存在复杂类型或不熟悉的变体时，Spark 通常会采用最一般的情况。

要查询 provider id 列，请先解析选择类型。您可以在 DynamicFrame 中使用 resolveChoice 转换方法，通过 cast:long 选项将这些 string 值转换为 long 值：

```
medicare_res = medicare_dynamicframe.resolveChoice(specs = [('provider
 id','cast:long')])
medicare_res.printSchema()
```

printSchema 输出现在是：

```
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
```

其中，该值是无法强制转换的 `string`，AWS Glue 插入了一个 `null`。

另一个选项是将选择类型转换为一个 `struct`，以保持两种类型的值。

接下来，查看异常的行：

```
medicare_res.toDF().where("'provider id' is NULL").show()
```

您看到以下内容：

```
+--------------------+----------+--------------+----------------------+-------------
+-------------+----------------+-------------------------------------+---------------
+-----------------------+---------------------+-------------------------+
|      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|
+--------------------+----------+--------------+----------------------+-------------
+-------------+----------------+-------------------------------------+---------------
+-----------------------+---------------------+-------------------------+
```

现在，删除两个格式错误的记录，如下所示：

```
medicare_dataframe = medicare_res.toDF()
medicare_dataframe = medicare_dataframe.where("'provider id' is NOT NULL")
```

步骤 4：映射数据和使用 Apache Spark Lambda 函数

AWS Glue 尚未直接支持 Lambda 函数，也称为用户定义函数。但是，您始终可以将 DynamicFrame 和 Apache Spark DataFrame 相互转换，以便除 DynamicFrames 的特殊功能外，还能利用 Spark 功能。

接下来，将付款信息转化为数字，以便 Amazon Redshift 或 Amazon Athena 这样的分析引擎可以更快地进行数字处理：

```
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 调用中的输出如下所示：

```
+--------+-------+-------+
|     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
```

这些仍然是数据中的字符串。我们可以使用强大的 apply_mapping 转换方法来删除、重命名、转换和嵌套数据，以便其他数据编程语言和系统可以轻松地访问它：

```
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 输出如下所示：

```
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
```

将数据重新变成 Spark DataFrame 后，您可以显示它现在的外观：

```
medicare_nest_dyf.toDF().show()
```

您可以在一个 (扩展) 代码行中执行所有这些操作：

```
+--------------------+--------------------+--------------+--------------------+
|                 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
```

步骤 5：将数据写入到 Apache Parquet

有了 AWS Glue，可以很容易地以诸如 Apache Parquet 这样的格式编写数据，以便关系数据库可以有效地使用它：

```
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 扩展参考

AWS Glue 为 PySpark Python 方言创建了以下扩展。

- 使用 getResolvedOptions 访问参数
- PySpark 扩展类型
- DynamicFrame 类
- DynamicFrameCollection 类
- DynamicFrameWriter 类
- DynamicFrameReader 班级
- GlueContext 班级

使用 **getResolvedOptions** 访问参数

AWS Glue `getResolvedOptions(args, options)` 实用程序函数为您提供了访问您在运行作业时传递到脚本的参数的权限。要使用此函数，请先将其从 AWS Glue `utils` 模块以及 `sys` 模块导入：

```
import sys
from awsglue.utils import getResolvedOptions
```

getResolvedOptions(args, options)

- `args` – `sys.argv` 中包含的参数的列表。
- `options` – 要检索的参数名称的 Python 数组。

Example 检索传递到 JobRun 的参数

假设您在某个脚本中（或许是在某个 Lambda 函数内）创建了 JobRun：

```
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 } )
```

要检索传递的参数，您可以使用 getResolvedOptions 函数，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']
```

请注意，每个参数将定义为以两个连字符开头，则将在不带连字符的脚本中引用。参数只使用下划线，而不是连字符。您的参数需要遵循此规则才能得到解决。

PySpark 扩展类型

AWS Glue PySpark 扩展所使用的类型。

DataType

其他 AWS Glue 类型的基类。

__init__(properties={})

- properties – 数据类型的属性 (可选)。

typeName(cls)

返回 AWS Glue 类型类的类型（即，类名，其“Type”会从末尾删除）。

- cls – 一个派生自 AWS Glue 的 DataType 类实例。

jsonValue( )

返回一个包含类的数据类型和属性的 JSON 对象：

```
{
  "dataType": typeName,
  "properties": properties
}
```

AtomicType 和简单衍生

继承自并扩展 DataType 类，并且充当所有 AWS Glue 原子数据类型的基类。

`fromJsonValue(cls, json_value)`

使用 JSON 对象中的值初始化类实例。

- `cls` – 一个要初始化的 AWS Glue 类型类。
- `json_value` – 要从其中加载键-值对的 JSON 对象。

以下类型是 AtomicType 类的简单衍生：

- `BinaryType` – 二进制数据。
- `BooleanType` – 布尔值。
- `ByteType` – 一个字节值。
- `DateType` – 一个日期时间值。
- `DoubleType` – 一个双精度浮点值。
- `IntegerType` – 一个整数值。
- `LongType` – 一个长整数值。
- `NullType` – 一个空值。
- `ShortType` – 一个短整数值。
- `StringType` – 一个文本字符串。
- `TimestampType` – 一个时间戳值 (通常以秒为单位，从 1/1/1970 开始)。
- `UnknownType` – 一个未知类型的值。

DecimalType(AtomicType)

继承自并扩展 AtomicType 类以表示十进制数字 (以十进制数字表示的数字，与二进制以 2 为底数的数字相对)。

__init__(precision=10, scale=2, properties={})

- precision – 十进制数中的位数 (可选；默认值为 10)。
- scale – 小数点右侧的位数 (可选；默认值为 2)。
- properties – 十进制数字的属性 (可选)。

EnumType(AtomicType)

继承自并扩展 AtomicType 类以表示有效选项的枚举。

__init__(options)

- options – 正被枚举的选项的列表。

集合类型

- ArrayType(DataType)
- ChoiceType(DataType)
- MapType(DataType)
- Field(Object)
- StructType(DataType)
- EntityType(DataType)

ArrayType(DataType)

__init__(elementType=UnknownType(), properties={})

- elementType – 数组中的元素的类型 (可选；默认值为 UnknownType)。
- properties – 数组的属性 (可选)。

ChoiceType(DataType)

__init__(choices=[], properties={})

- choices – 可能选项的列表 (可选)。
- properties – 这些选项的属性 (可选)。

add(new_choice)

将新选项添加到可能的选项列表中。

- new_choice – 要添加到可能选项列表中的选项。

merge(new_choices)

将新选项列表与现有选项列表合并。

- new_choices – 要与现有选项列表合并的新选项列表。

MapType(DataType)

__init__(valueType=UnknownType, properties={})

- valueType – 映射中的值的类型 (可选；默认值为 UnknownType)。
- properties – 映射的属性 (可选)。

Field(Object)

根据从 DataType 派生的对象创建一个字段对象。

__init__(name, dataType, properties={})

- name – 要为字段分配的名称。
- dataType – 要从中创建字段的对象。
- properties – 字段的属性 (可选)。

StructType(DataType)

定义数据结构 (struct)。

__init__(fields=[], properties={})

- fields – 要在结构中包括的字段的列表 (类型为 Field) (可选)。

- properties – 结构的属性 (可选)。

add(field)

- field – 要添加到结构中的类型 Field 的对象。

hasField(field)

如果此结构具有同名字段，则返回 True，否则返回 False。

- field – 一个字段名称，或其名称被使用的类型 Field 的对象。

getField(field)

- field – 一个字段名称，或其名称被使用的类型 Field 的对象。如果此结构具有同名字段，则返回它。

EntityType(DataType)

```
__init__(entity, base_type, properties)
```

此类尚未实现。

其他类型

- DataSource(object)
- DataSink(object)

DataSource(object)

__init__(j_source, sql_ctx, name)

- j_source – 数据源。
- sql_ctx – SQL 上下文。
- name – 数据源名称。

setFormat(format, **options)

- format – 要为数据源设置的格式。
- options- 要为数据源设置的选项的集合。有关格式选项的更多信息，请参阅 the section called “数据格式选项”。

getFrame()

为数据源返回 DynamicFrame。

DataSink(object)

__init__(j_sink, sql_ctx)

- j_sink – 要创建的堆栈。
- sql_ctx – 数据接收器的 SQL 上下文。

setFormat(format, **options)

- format – 要为数据接收器设置的格式。
- options- 要为数据接收器设置的选项的集合。有关格式选项的更多信息，请参阅 the section called “数据格式选项”。

setAccumulableSize(size)

- size – 要设置的可累积大小 (以字节为单位)。

writeFrame(dynamic_frame, info="")

- dynamic_frame – 要编写的 DynamicFrame。
- info – 有关 DynamicFrame 的信息 (可选)。

write(dynamic_frame_or_dfc, info="")

写入 DynamicFrame 或 DynamicFrameCollection。

- dynamic_frame_or_dfc – 将被写入的 DynamicFrame 对象或 DynamicFrameCollection 对象。
- info – 有关将被写入的 DynamicFrame 或 DynamicFrames 的信息 (可选)。

DynamicFrame 类

Apache Spark 中的主要抽象之一是 SparkSQL DataFrame，它类似于在 R 和 Pandas 中找到的 DataFrame 构造。DataFrame 类似于表格，支持函数风格（map/reduce/filter/等）操作和 SQL 操作（select、project、aggregate）。

尽管 DataFrames 功能强大且运用广泛，但在提取、转换和加载 (ETL) 操作方面存在局限性。最明显的是，它们需要指定了架构后才能加载任何数据。SparkSQL 通过对数据进行两次扫描解决了这一问题 – 第一次为了推断架构，第二次为了加载数据。然而，此推断仍有局限性，且不能解决数据混乱的实际问题。例如，同样的字段在不同记录中可能属于不同类型。对此，Apache Spark 通常没有好的办法，只能使用原始字段文本将类型报告为 string。这或许不正确，同时您可能希望更精细地控制解决架构差异的方式。对于大型数据集而言，每扫描一次源数据都会付出高昂代价。

为了解决这些局限性，AWS Glue 推出了 DynamicFrame。DynamicFrame 类似于 DataFrame，不同之处在于每个记录都是自描述的，因此初始并不需要架构。相反，AWS Glue 会在需要时实时计算一个架构，并使用选择（或联合）类型显式编码架构不一致之处。您可以解决这些不一致之处，以使您的数据集兼容需要固定架构的数据存储。

同样，一个 DynamicRecord 表示 DynamicFrame 中的一条逻辑记录。它类似于 Spark DataFrame 中的一行，只不过它是自描述的，可用于不符合固定架构的数据。将 AWS Glue 与 PySpark 一起使用时，通常不需要处理独立 DynamicRecords。相反，您需要通过 DynamicFrame 一起转换数据集。

在解决任何架构不一致问题后，就可以在 DynamicFrames 和 DataFrames 之间来回转换。

– 构造 –

- __init__
- fromDF
- toDF

__init__

__init__(jdf, glue_ctx, name)

- jdf – 对 Java 虚拟机 (JVM) 中数据帧的引用。

- glue_ctx – 一个 GlueContext 班级 对象。
- name – 可选的名称字符串，默认为空。

fromDF

fromDF(dataframe, glue_ctx, name)

通过将 DataFrame 字段转换为 DynamicRecord 字段将 DataFrame 转换为 DynamicFrame。返回新的 DynamicFrame。

一个 DynamicRecord 表示 DynamicFrame 中的一条逻辑记录。它类似于 Spark DataFrame 中的一行，只不过它是自描述的，可用于不符合固定架构的数据。

此函数期望您 DataFrame 中名称重复的列已经被解析。

- dataframe – 要转换的 Apache Spark SQL DataFrame (必需)。
- glue_ctx – 为此转换指定上下文的 GlueContext 班级 对象 (必需)。
- name – 生成的 DynamicFrame 的名称（从 AWS Glue 3.0 起是可选的）。

toDF

toDF(options)

通过将 DynamicRecords 转换为 DataFrame 字段将 DynamicFrame 转换为 Apache Spark DataFrame。返回新的 DataFrame。

一个 DynamicRecord 表示 DynamicFrame 中的一条逻辑记录。它类似于 Spark DataFrame 中的一行，只不过它是自描述的，可用于不符合固定架构的数据。

- options – 一个选项列表。如果您选择 Project 和 Cast 操作类型，则指定目标类型。示例包括以下内容。

```
>>>toDF([ResolveOption("a.b.c", "KeepAsStruct")])
>>>toDF([ResolveOption("a.b.c", "Project", DoubleType())])
```

– 信息 –

- count

- 架构
- printSchema
- show
- repartition
- coalesce

count

count( ) – 返回底层 DataFrame 中的行数。

架构

schema( ) – 返回此 DynamicFrame 的架构，如果此架构不可用，则返回底层 DataFrame 的架构。

有关构成此架构的 DynamicFrame 类型的更多信息，请参阅 the section called “类型”。

printSchema

printSchema( ) – 打印底层 DataFrame 的架构。

show

show(num_rows) – 打印底层 DataFrame 中的行数。

repartition

repartition(numPartitions) – 返回具有 numPartitions 个分区的新 DynamicFrame。

coalesce

coalesce(numPartitions) – 返回具有 numPartitions 个分区的新 DynamicFrame。

– 转换 –

- apply_mapping
- drop_fields
- filter

- join
- 映射
- mergeDynamicFrame
- relationalize
- rename_field
- resolveChoice
- select_fields
- spigot
- split_fields
- split_rows
- unbox
- the section called “联合”
- unnest
- unnest_ddb_json
- write

apply_mapping

apply_mapping(mappings, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

将声明映射应用于此 DynamicFrame 并返回已将那些映射应用于所指定字段的新 DynamicFrame。新 DynamicFrame 中省略了未指定的字段。

- mappings – 映射元组列表（必需）。每个元组包括：（源列，源类型，目标列，目标类型）。

  如果源列的名称有一个点“.”，则必须在名称外加上反引号“``”。例如，要将 this.old.name（字符串）映射到 thisNewName，可以使用以下元组：

  ```
  ("`this.old.name`", "string", "thisNewName", "string")
  ```

- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与此转换的错误报告关联的字符串 (可选)。

- stageThreshold – 此转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。
- totalThreshold – 此转换之前及转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。

示例：使用 apply_mapping 重命名字段并更改字段类型

以下代码示例展示了如何使用 apply_mapping 方法重命名选定字段并更改字段类型。

> **Note**
>
> 要访问本示例中使用的数据集，请参阅 代码示例：对数据进行联接和关系化 并按照 步骤 1：爬取 Amazon S3 存储桶中的数据 中的说明进行操作。

```
# 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()
```

输出

```
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)

调用 FlatMap 类 转换以从 DynamicFrame 中删除字段。返回指定字段已删除的新的 DynamicFrame。

- paths – 字符串列表。每个字符串都包含要删除的字段节点的完整路径。您可以使用点表示法来指定嵌套字段。例如，如果字段 first 在树结构中是字段 name 的子字段，则为路径指定 "name.first"。

  如果字段节点的名称有文字 .，则必须用反引号（ ` ）将名称括起来。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与此转换的错误报告关联的字符串 (可选)。
- stageThreshold – 此转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。
- totalThreshold – 此转换之前及转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。

示例：使用 drop_fields 从 **DynamicFrame** 中删除字段

此代码示例使用 drop_fields 方法从 DynamicFrame 中移除选定的顶级字段和嵌套字段。

示例数据集

该示例使用以下数据集，其在代码中由 EXAMPLE-FRIENDS-DATA 表表示：

```
{"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}]}
```

示例代码

```
# 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()
```

输出

```
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
```

filter

filter(f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

返回新 DynamicFrame，其中包含输入 DynamicFrame 中满足指定谓词函数 f 的所有 DynamicRecords。

- f – 应用到 DynamicFrame 的谓词函数。该函数必须采用 DynamicRecord 作为参数，如果 DynamicRecord 满足筛选要求，则返回 True，否则返回 False (必需)。

  一个 DynamicRecord 表示 DynamicFrame 中的一条逻辑记录。其类似于 Spark DataFrame 中的一行，但其具有自描述性，可用于不符合固定架构的数据。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与此转换的错误报告关联的字符串 (可选)。
- stageThreshold – 此转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。
- totalThreshold – 此转换之前及转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。

示例：使用筛选条件获取筛选后的字段选择

此示例用 filter 方法创建新 DynamicFrame，其中包括筛选后的其他 DynamicFrame 字段选择。

类似于 map 方法，filter 采用一个函数作为实际参数，以将其应用于原 DynamicFrame 中的每个记录。该函数将记录作为输入并返回布尔值。如果返回值为 true，则该记录将包含在生成的 DynamicFrame 中。如果返回值为 false，则该记录将被排除在外。

> **Note**
>
> 要访问本示例中使用的数据集，请参阅 代码示例：使用 ResolveChoice、Lambda 和 ApplyMapping 进行数据准备 并按照 步骤 1：爬取 Amazon S3 存储桶中的数据 中的说明进行操作。

```
# 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())
```

输出

```
Unfiltered record count:  163065
Filtered record count:   564
```

join

join(paths1, paths2, frame2, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

执行与另一个 DynamicFrame 的等式联接并返回生成的 DynamicFrame。

- paths1 – 此帧中要联接的键列表。
- paths2 – 另一帧中要联接的键列表。
- frame2 – 要联接的另一个 DynamicFrame。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与此转换的错误报告关联的字符串 (可选)。
- stageThreshold – 此转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。
- totalThreshold – 此转换之前及转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。

示例：使用联接来组合 **DynamicFrames**

此示例使用 join 方法对三个 DynamicFrames 执行联接。AWSGlue 根据您提供的字段键执行联接。由此生成的 DynamicFrame 包含来自指定密钥相匹配的两个原始帧中的行。

请注意，join 转换会让所有字段保持不变。这意味着您指定要匹配的字段会出现在生成的 DynamicFrame 中，即使这些字段是多余的并且包含相同的密钥。在此示例中，我们在联接后使用 drop_fields 移除这些冗余密钥。

> **Note**
>
> 要访问本示例中使用的数据集，请参阅 代码示例：对数据进行联接和关系化 并按照 步骤 1：爬取 Amazon S3 存储桶中的数据 中的说明进行操作。

```
# 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()
```

输出

```
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
```

映射

map(f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

返回由于将指定映射函数应用到原始 DynamicFrame 中的所有记录而产生的新的 DynamicFrame。

- f – 应用到 DynamicFrame 中所有记录的映射函数。该函数必须采用 DynamicRecord 作为参数并返回一个新的 DynamicRecord (必需)。

  一个 DynamicRecord 表示 DynamicFrame 中的一条逻辑记录。它类似于 Apache Spark DataFrame 中的一行，但其具有自描述性，可用于不符合固定架构的数据。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与转换中的错误关联的字符串（可选）。
- stageThreshold – 在转换出错之前可能在其中发生的最大错误数（可选）。默认值为 0。
- totalThreshold – 在处理出错之前可能全面发生的最大错误数（可选）。默认值为 0。

示例：使用映射将函数应用于 DynamicFrame 中的每个记录

此示例展示了如何使用 map 方法将函数应用于 DynamicFrame 的每个记录。具体而言，此示例将名为 MergeAddress 的函数应用于每个记录，以便将多个地址字段合并为一个 struct 类型。

> **Note**
>
> 要访问本示例中使用的数据集，请参阅 代码示例：使用 ResolveChoice、Lambda 和 ApplyMapping 进行数据准备 并按照 步骤 1：爬取 Amazon S3 存储桶中的数据 中的说明进行操作。

```
# 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()
```

输出

```
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)

基于指定主键的将此 DynamicFrame 与暂存 DynamicFrame 合并以标识记录。不会对重复记录（具有相同主键的记录）去除重复。如果暂存帧中没有匹配的记录，则从源中保留所有记录（包括重复记录）。如果暂存帧具有匹配的记录，则暂存帧中的记录将覆盖 AWS Glue 中的源中的记录。

- stage_dynamic_frame – 要合并的暂存 DynamicFrame。
- primary_keys – 要匹配源和暂存动态帧中的记录的主键字段列表。
- transformation_ctx – 用于检索有关当前转换的元数据的唯一字符串（可选）。
- options – 为此转换提供其他信息的 JSON 名称-值对的字符串。当前未使用此参数。
- info – 一个 String。要与此转换中的错误关联的任何字符串。
- stageThreshold – 一个 Long。给定转换中处理需要排除的错误的数目。
- totalThreshold – 一个 Long。此转换中处理需要排除的错误的总数。

该方法将返回通过将此 DynamicFrame 与暂存 DynamicFrame 合并获取的新 DynamicFrame。

在这些情况下，返回的 DynamicFrame 将包含记录 A：

- 如果 A 在源帧和暂存帧中都存在，则返回暂存帧中的 A。
- 如果 A 在源表中，且 A.primaryKeys 不在 stagingDynamicFrame 中，这意味着未在暂存表中更新 A。

源帧和暂存帧不需要具有相同的架构。

示例：根据主键使用 mergeDynamicFrame 合并两个 **DynamicFrames**

以下代码示例显示了如何使用 mergeDynamicFrame 方法根据主键 id 将 DynamicFrame 与“staging”DynamicFrame 合并。

示例数据集

该示例使用了名为 split_rows_collection 的 DynamicFrameCollection 中的两个 DynamicFrames。以下是 split_rows_collection 中的键列表。

```
dict_keys(['high', 'low'])
```

示例代码

```
# 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()
```

输出

```
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|
+---+-----+-----------------------+-----------------------+
```

relationalize

relationalize(root_table_name, staging_path, options, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

将 DynamicFrame 转换为适用于关系数据库的形式。将数据从 DynamoDB 等 NoSQL 环境移动到 MySQL 等关系数据库时，对 DynamicFrame 进行关系化尤其有用。

该转换将通过取消嵌套列的嵌套并透视数组列来生成帧列表。可使用在取消嵌套阶段生成的联接键将透视数组列联接到根表。

- root_table_name – 根表的名称。
- staging_path – 要将 CSV 格式的透视表分区存储到的路径（可选）。从该路径读取透视表。
- options – 可选参数的词典。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与此转换的错误报告关联的字符串 (可选)。
- stageThreshold – 此转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。
- totalThreshold – 此转换之前及转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。

示例：使用 relatialize 在 **DynamicFrame** 中展平嵌套架构

此代码示例使用 relationalize 方法将嵌套架构展平为适用于关系数据库的形式。

示例数据集

该示例通过以下架构使用了名为 legislators_combined 的 DynamicFrame。legislators_combined 有多个嵌套字段，例如 links、images 和 contact_details，这些字段将通过 relationalize 转换进行展平。

```
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
```

示例代码

```
# 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()
```

输出

以下输出可让您将名为 contact_details 的嵌套字段的架构与 relationalize 转换创建的表进行比较。请注意，表记录可使用名为 id 的外键和表示数组位置的 index 列链接回主表。

```
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)

重命名此 DynamicFrame 中的一个字段并返回包含该重命名字段的新的 DynamicFrame。

- oldName – 要重命名的节点的完整路径。

  如果旧名称中包含点，则 RenameField 将不起作用，除非使用反引号（ ` ）将其引起来。例如，要将 this.old.name 替换为 thisNewName，应按如下方式调用 rename_field。

  ```
  newDyF = oldDyF.rename_field("`this.old.name`", "thisNewName")
  ```

- newName – 新名称，作为完整路径。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与此转换的错误报告关联的字符串 (可选)。
- stageThreshold – 此转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。
- totalThreshold – 此转换之前及转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。

示例：使用 rename_field 重命名 **DynamicFrame** 中的字段

此代码示例可使用 rename_field 方法重命名 DynamicFrame 中的字段。请注意，该示例可使用方法链同时重命名多个字段。

> **Note**
>
> 要访问本示例中使用的数据集，请参阅 代码示例：对数据进行联接和关系化 并按照 步骤 1：爬取 Amazon S3 存储桶中的数据 中的说明进行操作。

示例代码

```
# 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()
```

输出

```
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)

解析此 DynamicFrame 内的一个选择类型并返回新的 DynamicFrame。

- specs – 要解析的特定歧义列表，每个歧义均采用元组形式：(field_path, action)。

  可通过两种方式使用 resolveChoice。第一种是使用 specs 参数指定一系列特定字段以及如何解析它们。resolveChoice 的另一种模式是使用 choice 参数为所有 ChoiceTypes 指定单个解析方法。

  specs 的值被指定为由 (field_path, action) 对组成的元组。field_path 值标识特定歧义元素，action 值标识相应解析。可能的操作如下：

  - cast:*type* – 尝试将所有值转换为指定的类型。例如：cast:int。
  - make_cols – 将每个不同的类型转换为名为 *columnName_type* 的列。通过展平数据来解析潜在的歧义。例如，如果 columnA 是 int 或 string，则解析就是在结果 DynamicFrame 中生成名为 columnA_int 和 columnA_string 的两个列。
  - make_struct – 使用 struct 表示数据，解析潜在的歧义。例如，如果某个列中的数据是 int 或 string，则使用 make_struct 操作会在结果 DynamicFrame 中生成结构列。每个结构都包含 int 和 string。
  - project:*type* – 将所有数据投影到可能的数据类型之一，解析潜在的歧义。例如，如果某个列中的数据是 int 或 string，则使用 project:string 操作会在结果 DynamicFrame 中生成一个列，其中所有 int 值都已转换为字符串。

  如果 field_path 识别到数组，则在数组名称后放置一个空的方括号可避免歧义。例如，假设您正在使用结构如下的数据：

  ```
  "myList": [
    { "price": 100.00 },
    { "price": "$100.00" }
  ]
  ```

  可以通过将 field_path 设置为 "myList[].price"、将 action 设置为 "cast:double" 来选择价格的数字而非字符串版本。

  > **Note**
  >
  > 只能使用 specs 和 choice 参数之一。如果 specs 参数不为 None，则 choice 参数必须为空字符串。反过来，如果 choice 不为空字符串，则 specs 参数必须为 None。

- choice – 为所有 ChoiceTypes 指定单个解析方法。这可以在运行前不知道 ChoiceTypes 的完整列表的情况下使用。除了之前为 specs 列出的操作外，此参数还支持以下操作：

  - match_catalog – 尝试将每个 ChoiceType 转换为指定数据目录表中的对应类型。
- database – 与 match_catalog 操作一起使用的数据目录数据库。
- table_name – 与 match_catalog 操作一起使用的数据目录表。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与此转换的错误报告关联的字符串 (可选)。
- stageThreshold – 此转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。
- totalThreshold – 此转换之前及转换过程中遇到的将导致过程出错的错误数（可选）。默认为零，表示此过程应该不会出错。
- catalog_id – 正在访问的数据目录的目录 ID（数据目录的账户 ID）。如果设置为 None（默认值），它使用调用账户的目录 ID。

示例：使用 resolveChoice 处理包含多种类型的列

此代码示例使用 resolveChoice 方法来指定如何处理包含多种类型值的 DynamicFrame 列。该示例演示了处理不同类型的列的两种常用方法：

- 将该列转换为单一数据类型。
- 在单独的列中保留所有类型。

示例数据集

> **Note**
>
> 要访问本示例中使用的数据集，请参阅 代码示例：使用 ResolveChoice、Lambda 和 ApplyMapping 进行数据准备 并按照 步骤 1：爬取 Amazon S3 存储桶中的数据 中的说明进行操作。

该示例通过以下架构使用了名为 medicare 的 DynamicFrame：

```
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
```

示例代码

```
# 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()
```

输出

```
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)

返回包含选定字段的新 DynamicFrame。

- paths – 字符串列表。每个字符串就是一个指向您要选择的顶层节点的路径。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与此转换的错误报告关联的字符串 (可选)。
- stageThreshold – 此转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。
- totalThreshold – 此转换之前及转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。

示例：使用 select_fields 以利用选定字段创建新 **DynamicFrame**

以下代码示例展示了如何使用 `select_fields` 方法以利用从现有 DynamicFrame 中选定的字段列表来创建新 DynamicFrame。

> **Note**
>
> 要访问本示例中使用的数据集，请参阅 代码示例：对数据进行联接和关系化 并按照 步骤 1：爬取 Amazon S3 存储桶中的数据 中的说明进行操作。

```
# 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()
```

输出

```
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
```

simplify_ddb_json

`simplify_ddb_json(): DynamicFrame`

简化 DynamicFrame 中的嵌套列，其具体位于 DynamoDB JSON 结构中，并返回一个新的简化 DynamicFrame。如果 List 类型中有多种类型或 Map 类型，则不会简化 List 中的元素。请注意，这是一种特定类型的转换，其行为与常规 unnest 转换不同，并且要求数据已存在于 DynamoDB JSON 结构中。有关更多信息，请参阅 DynamoDB JSON。

例如，使用 DynamoDB JSON 结构读取导出的架构可能如下所示：

```
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()` 转换会将此转换为：

```
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
```

示例：使用 simplify_ddb_json 调用 DynamoDB JSON simplify 命令

以下代码示例使用 `simplify_ddb_json` 方法来使用 AWS Glue DynamoDB 导出连接器、调用 DynamoDB JSON simplify 命令，以及打印分区数量。

示例代码

```
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={})

将示例记录写入指定的目标，以帮助您验证作业执行的转换。

- path – 要写入的目标的路径（必需）。
- options – 指定选项的键值对（可选）。"topk" 选项指定应写入第一条 k 记录。"prob" 选项指定选择任何给定记录的可能性（以十进制数字形式表示）。您可以在选择要写入的记录时使用它。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。

示例：使用 spigot 将 **DynamicFrame** 中的示例字段写入 Amazon S3

此代码示例使用 spigot 方法在应用了 select_fields 转换后将示例记录写入 Amazon S3 存储桶。

示例数据集

> **Note**
>
> 要访问本示例中使用的数据集，请参阅 代码示例：对数据进行联接和关系化 并按照 步骤 1：爬取 Amazon S3 存储桶中的数据 中的说明进行操作。

该示例通过以下架构使用了名为 persons 的 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
```

示例代码

```
# 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}
)
```

输出

以下是 spigot 写入 Amazon S3 的数据示例。由于指定了示例代码 options={"topk": 10}，因此示例数据包含前 10 条记录。

```
{"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)

返回一个新的 DynamicFrameCollection，其中包含两个 DynamicFrames。第一个 DynamicFrame 包含所有已拆分的节点，第二个包含其余节点。

- paths – 字符串列表，每个字符串是到一个您要拆分为新的 DynamicFrame 的节点的完整路径。
- name1 – 拆分的 DynamicFrame 的名称字符串。
- name2 – 指定节点拆分后留存的 DynamicFrame 的名称字符串。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与此转换的错误报告关联的字符串 (可选)。

- stageThreshold – 此转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。
- totalThreshold – 此转换之前及转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。

示例：使用 split_fields 将选定字段拆分为单独的 **DynamicFrame**

此代码示例使用 split_fields 方法将指定字段列表拆分为单独的 DynamicFrame。

示例数据集

该示例使用了集合 legislators_relationalized 中名为 l_root_contact_details 的 DynamicFrame。

l_root_contact_details 有以下架构和条目。

```
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|
...
```

示例代码

```
# 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()
```

输出

```
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)

将 DynamicFrame 中的一个或多个行拆分到新的 DynamicFrame 中。

该方法可返回一个新的 DynamicFrameCollection，其中包含两个 DynamicFrames。第一个 DynamicFrame 包含所有已拆分的行，第二个包含其余行。

- comparison_dict – 一个词典，其中的键是到一个列的路径，值是另一个词典，用于将比较器映射到与该列值所比较的值。例如，{"age": {">": 10, "<": 20}} 拆分其年龄列中的值大于 10 但小于 20 的行。
- name1 – 拆分的 DynamicFrame 的名称字符串。
- name2 – 指定节点拆分后留存的 DynamicFrame 的名称字符串。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与此转换的错误报告关联的字符串 (可选)。
- stageThreshold – 此转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。
- totalThreshold – 此转换之前及转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。

示例：使用 split_rows 拆分 **DynamicFrame** 中的行

此代码示例使用 split_rows 方法根据 id 字段值拆分 DynamicFrame 中的行。

示例数据集

该示例使用了从集合 legislators_relationalized 中选择的名为 l_root_contact_details 的 DynamicFrame。

l_root_contact_details 有以下架构和条目。

```
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|
+---+-----+----------------------+------------------------+
```

示例代码

```
# 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()
```

输出

```
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)

取消装箱（重新格式化）DynamicFrame 中的一个字符串字段并返回包含已取消装箱 DynamicRecords 的新的 DynamicFrame。

一个 DynamicRecord 表示 DynamicFrame 中的一条逻辑记录。它类似于 Apache Spark DataFrame 中的一行，但其具有自描述性，可用于不符合固定架构的数据。

- path – 要取消装箱的字符串节点的完整路径。
- format – 格式规范（可选）。您可将其用于 Amazon S3 或支持多种格式的 AWS Glue 连接。相关受支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与此转换的错误报告关联的字符串 (可选)。
- stageThreshold – 此转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。
- totalThreshold – 此转换之前及转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。
- options – 下列一个或多个：
  - separator – 包含分隔符字符的字符串。
  - escaper – 包含转义字符的字符串。
  - skipFirst – 指示是否跳过第一个实例的布尔值。

- withSchema - 包含节点架构的 JSON 表示形式的字符串。架构的 JSON 表示格式由 StructType.json() 的输出定义。
- withHeader – 指示是否包括标头的布尔值。

示例：使用 unbox 将字符串字段拆开为结构

此代码示例使用 unbox 方法将 DynamicFrame 中的字符串字段拆开或重新格式化为结构类型的字段。

示例数据集

该示例通过以下架构和条目使用了名为 DynamicFrame 的 mapped_with_string。

请注意名为 AddressString 的字段。这是拆开为结构示例中的字段。

```
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|
...
```

示例代码

```
# 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()
```

输出

```
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
```

联合

union(frame1, frame2, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

联合两个 DynamicFrame。返回 DynamicFrame，其中包含来自两个输入 DynamicFrame 的所有记录。这种转换可能会从两个 DataFrame 与等效数据的合并中返回不同的结果。如果您需要 Spark DataFrame 联合行为，可以考虑使用 toDF。

- frame1 – 第一个要联合的 DynamicFrame。
- frame2 – 第二个要联合的 DynamicFrame。
- transformation_ctx – （可选）用于标识统计信息/状态信息的唯一字符串
- info – （可选）与转换中的错误关联的任何字符串
- stageThreshold – （可选）在处理出错之前转换中出现的最大错误数
- totalThreshold – （可选）在处理出错之前出现的最大总错误数

unnest

unnest(transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

取消嵌套 DynamicFrame 中的嵌套对象，使其成为顶级对象，并返回新的取消嵌套的 DynamicFrame。

- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与此转换的错误报告关联的字符串 (可选)。
- stageThreshold – 此转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。
- totalThreshold – 此转换之前及转换过程中遇到的将导致过程出错的错误数（可选）。默认值为零，则表示进程不应出错。

示例：使用 unnest 将嵌套字段转换为顶级字段

此代码示例使用 unnest 方法将 DynamicFrame 中的所有嵌套字段展平为顶级字段。

示例数据集

该示例通过以下架构使用了名为 mapped_medicare 的 DynamicFrame。请注意，Address 字段是唯一包含嵌套数据的字段。

```
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
```

示例代码

```
# 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()
```

输出

```
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 DynamicFrame 中的嵌套列，其具体位于 DynamoDB JSON 结构中，并返回一个新的非嵌套 DynamicFrame。属于结构类型数组的列将不会被解除嵌套。请注意，这是一种特定类型的非嵌套转换，其行为与常规 unnest 转换不同，并且要求数据已存在于 DynamoDB JSON 结构中。有关更多信息，请参阅 DynamoDB JSON。

unnest_ddb_json(transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与此转换的错误报告关联的字符串 (可选)。
- stageThreshold – 此转换过程中遇到的将导致过程出错的错误数 (可选，默认为零，表示此过程应该不会出错)。
- totalThreshold – 此转换之前及转换过程中遇到的将导致过程出错的错误数 (可选，默认为零，表示此过程应该不会出错)。

例如，使用 DynamoDB JSON 结构读取导出的架构可能如下所示：

```
root
|-- Item: struct
|    |-- ColA: struct
|    |    |-- S: string
|    |-- ColB: struct
|    |    |-- S: string
|    |-- ColC: struct
|    |    |-- N: string
```

```
|    |-- ColD: struct
|    |    |-- L: array
|    |    |    |-- element: null
```

unnest_ddb_json() 转换会将此转换为：

```
root
|-- ColA: string
|-- ColB: string
|-- ColC: string
|-- ColD: array
|    |-- element: null
```

以下代码示例演示了如何使用 AWS Glue DynamoDB 导出连接器、调用 DynamoDB JSON 解除嵌套命令，以及打印分区数量：

```
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)

从此 DynamicFrame 的 GlueContext 班级 获得指定连接类型的 DataSink(object) 并将其用于格式化和写入此 DynamicFrame 的内容。返回按指定进行格式化和写入的新的 DynamicFrame。

- connection_type – 要使用的连接类型。有效值包括 s3、mysql、postgresql、redshift、sqlserver 和 oracle。
- connection_options – 要使用的连接选项（可选）。对于 connection_type 的 s3，将会定义 Amazon S3 路径。

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

对于 JDBC 连接，必须定义多个属性。请注意，数据库名称必须是 URL 的一部分。它可以选择性地包含在连接选项中。

> **Warning**
>
> 不建议在脚本中存储密码。请考虑使用 boto3 从 AWS Secrets Manager 或 AWS Glue Data Catalog 检索它们。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

- format – 格式规范（可选）。这用于 Amazon Simple Storage Service（Amazon S3）或支持多种格式的 AWS Glue 连接。有关支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- format_options – 指定格式的格式选项。有关支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- accumulator_size - 要使用的可累积大小（以字节为单位）（可选)。

– 错误 –

- assertErrorThreshold

- errorsAsDynamicFrame
- errorsCount
- stageErrorsCount

assertErrorThreshold

`assertErrorThreshold( )` – 创建此 DynamicFrame 的转换中的错误的资产。从底层 DataFrame 返回 `Exception`。

errorsAsDynamicFrame

`errorsAsDynamicFrame( )` – 返回其中包含嵌套的错误记录的 DynamicFrame。

示例：使用 errorsAsDynamicFrame 查看错误记录

以下代码示例展示了如何使用 `errorsAsDynamicFrame` 方法查看 DynamicFrame 的错误记录。

示例数据集

该示例使用以下数据集，您可以将以下数据集作为 JSON 上传到 Amazon S3。请注意，第二条记录的格式有误。当您使用 SparkSQL 时，格式错误的数据通常会中断文件解析。但是，DynamicFrame 会识别格式错误的问题，并将格式错误的行转换为您能单独处理的错误记录。

```
{"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}
```

示例代码

```
# 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]))
```

输出

```
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( )` – 返回 DynamicFrame 中的错误总数。

stageErrorsCount

`stageErrorsCount` – 返回生成此 DynamicFrame 的过程中发生的错误数。

DynamicFrameCollection 类

`DynamicFrameCollection` 是 DynamicFrame 类 对象的字段，在其中键是 DynamicFrames 的名称，值是 DynamicFrame 对象。

__init__

`__init__(dynamic_frames, glue_ctx)`

- `dynamic_frames` – DynamicFrame 类 对象的字典。
- `glue_ctx` – 一个 GlueContext 班级 对象。

键

`keys( )` – 返回此集合中的键的列表，通常由相应的 DynamicFrame 值的名称组成。

值

values(key) – 返回此集合中的 DynamicFrame 值的列表。

Select

select(key)

返回与指定键对应的 DynamicFrame (通常是 DynamicFrame 的名称)。

- key – DynamicFrameCollection 中的键，通常表示 DynamicFrame 的名称。

Map

map(callable, transformation_ctx="")

使用传入函数根据此集合中的 DynamicFrames 创建并返回新的 DynamicFrameCollection。

- callable – 一个函数，它采用 DynamicFrame 以及指定转换上下文作为参数并返回一个 DynamicFrame。
- transformation_ctx – 要由可调用脚本使用的转换上下文 (可选)。

Flatmap

flatmap(f, transformation_ctx="")

使用传入函数根据此集合中的 DynamicFrames 创建并返回新的 DynamicFrameCollection。

- f – 一个函数，它采用 DynamicFrame 作为参数并返回一个 DynamicFrame 或 DynamicFrameCollection。
- transformation_ctx – 要由函数使用的转换上下文 (可选)。

DynamicFrameWriter 类

Methods

- __init__
- from_options
- from_catalog

- from_jdbc_conf

__init__

__init__(glue_context)

- glue_context – 要使用的 GlueContext 班级。

from_options

from_options(frame, connection_type, connection_options={}, format=None, format_options={}, transformation_ctx="")

使用指定的连接和格式编写一个 DynamicFrame。

- frame – 要编写的 DynamicFrame。
- connection_type – 连接类型。有效值包括 s3、mysql、postgresql、redshift、sqlserver 和 oracle。
- connection_options – 连接选项，例如路径和数据库表 (可选)。对于 connection_type 的 s3，将会定义 Amazon S3 路径。

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

对于 JDBC 连接，必须定义多个属性。请注意，数据库名称必须是 URL 的一部分。它可以选择性地包含在连接选项中。

> **Warning**
>
> 不建议在脚本中存储密码。请考虑使用 boto3 从 AWS Secrets Manager 或 AWS Glue Data Catalog 检索它们。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

dbtable 属性是 JDBC 表的名称。对于在数据库中支持架构的 JDBC 数据存储，指定 schema.table-name。如果未提供架构，则使用默认的“public”架构。

有关更多信息，请参阅 AWS Glue for Spark 中适用于 ETL 的连接类型和选项。

- format – 格式规范（可选）。这用于 Amazon Simple Storage Service（Amazon S3）或支持多种格式的 AWS Glue 连接。有关支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- format_options – 指定格式的格式选项。有关支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- transformation_ctx – 要使用的转换上下文 (可选)。

from_catalog

from_catalog(frame, name_space, table_name, redshift_tmp_dir="", transformation_ctx="")

使用指定的目录数据库和表名称编写一个 DynamicFrame。

- frame – 要编写的 DynamicFrame。
- name_space – 要使用的数据库。
- table_name – 要使用的 table_name。
- redshift_tmp_dir – 要使用的 Amazon Redshift 临时目录（可选）。
- transformation_ctx – 要使用的转换上下文 (可选)。
- additional_options 提供给 AWS Glue 的额外选项。

要写入 Lake Formation 受管表，可以使用以下附加选项：

- transactionId – （字符串）用于写入受管表的事务 ID。此事务无法提交或中止，否则写入将失败。
- callDeleteObjectsOnCancel – （布尔值，可选）如果设置为 true（原定设置）、则在将对象写入 Amazon S3 后 AWS Glue 将自动调用 DeleteObjectsOnCancel API。有关更多信息，请参阅 AWS Lake Formation 开发人员指南中的 DeleteObjectsOnCancel。

Example 示例：写入 Lake Formation 中的受管表

```
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 连接信息编写一个 DynamicFrame。

- frame – 要编写的 DynamicFrame。
- catalog_connection – 要使用的目录连接。
- connection_options – 连接选项，例如路径和数据库表 (可选)。
- redshift_tmp_dir – 要使用的 Amazon Redshift 临时目录（可选）。
- transformation_ctx – 要使用的转换上下文 (可选)。

write_dynamic_frame 示例

此示例结合使用 S3 的 connection_type 与 connection_options 中的 POSIX 路径参数，在本地写入输出，从而允许将数据写入本地存储。

```
glueContext.write_dynamic_frame.from_options(\
frame = dyf_splitFields,\
connection_options = {'path': '/home/glue/GlueLocalOutput/'},\
connection_type = 's3',\
format = 'json')
```

DynamicFrameReader 班级

– 方法 –

- __init__
- from_rdd
- from_options
- from_catalog

__init__

__init__(glue_context)

- glue_context – 要使用的 GlueContext 班级。

from_rdd

from_rdd(data, name, schema=None, sampleRatio=None)

从弹性分布式数据集 (RDD) 读取 DynamicFrame。

- data – 要从中读取的数据集。
- name – 要从中读取的名称。
- schema – 要读取的架构 (可选)。
- sampleRatio – 采样率 (可选)。

from_options

from_options(connection_type, connection_options={}, format=None, format_options={}, transformation_ctx="")

使用指定的连接和格式读取 DynamicFrame。

- connection_type – 连接类型。有效值包括 s3、mysql、postgresql、redshift、sqlserver、oracle、dynamodb 和 snowflake。
- connection_options – 连接选项，例如路径和数据库表 (可选)。有关更多信息，请参阅 AWS Glue for Spark 中的 ETL 连接类型和选项。对于 connection_type 的 s3，Amazon S3 路径在数组中定义。

  ```
  connection_options = {"paths": [ "s3://mybucket/object_a", "s3://mybucket/object_b"]}
  ```

  对于 JDBC 连接，必须定义多个属性。请注意，数据库名称必须是 URL 的一部分。它可以选择性地包含在连接选项中。

> **Warning**
>
> 不建议在脚本中存储密码。考虑使用boto3从 AWS Secrets Manager 或 AWS Glue 数据目录中检索它们。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

对于执行并行读取的 JDBC 连接，您可以设置 hashfield 选项。例如：

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path" , "hashfield": "month"}
```

有关更多信息，请参阅 从 JDBC 表并行读取。

- `format` – 格式规范（可选）。这用于 Amazon Simple Storage Service（Amazon S3）或支持多种格式的 AWS Glue 连接。有关支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- `format_options` – 指定格式的格式选项。有关支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- `transformation_ctx` – 要使用的转换上下文 (可选)。
- `push_down_predicate` – 筛选分区，而不必列出并读取数据集中的所有文件。有关更多信息，请参阅使用下推谓词进行预筛选。

from_catalog

`from_catalog(database, table_name, redshift_tmp_dir="", transformation_ctx="", push_down_predicate="", additional_options={})`

使用指定的目录命名空间和表名称读取 `DynamicFrame`。

- `database` – 要从中读取的数据库。
- `table_name` – 要从中读取的表的名称。

- `redshift_tmp_dir` – 要使用的 Amazon Redshift 临时目录（如果不从 Redshift 中读取数据，则此项是可选的）。
- `transformation_ctx` – 要使用的转换上下文 (可选)。
- `push_down_predicate` – 筛选分区，而不必列出并读取数据集中的所有文件。有关更多信息，请参阅 使用下推谓词进行预筛选。
- `additional_options` 提供给 AWS Glue 的额外选项。
  - 要使用执行并行读取的 JDBC 连接，您可以设置 `hashfield`、`hashexpression` 或 `hashpartitions` 选项。例如：

    ```
    additional_options = {"hashfield": "month"}
    ```

    有关更多信息，请参阅 从 JDBC 表并行读取。
  - 要传递目录表达式以根据索引列进行筛选，您可以查看 `catalogPartitionPredicate` 选项。

    `catalogPartitionPredicate` – 要传递目录表达式以根据索引列进行筛选。这样会将筛选下推到服务器端。有关更多信息，请参阅 AWS Glue 分区数据。请注意，`push_down_predicate` 和 `catalogPartitionPredicate` 使用不同的语法。前者使用 Spark SQL 标准语法，后者使用 JSQL 解析器。

    有关更多信息，请参阅 管理 AWS Glue 中用于 ETL 输出的分区。

GlueContext 班级

封装 Apache Spark SparkContext对象，从而提供与 Apache Spark 平台进行交互的机制。

__init__

`__init__(sparkContext)`

- `sparkContext` – 要使用的 Apache Spark 上下文。

Creating

- __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)

创建一个 DataSource 对象，该对象可用于从外部来源读取 DynamicFrames。

- connection_type – 要使用的连接类型，例如 Amazon Simple Storage Service (Amazon S3)、Amazon Redshift 和 JDBC。有效值包括 s3、mysql、postgresql、redshift、sqlserver、oracle 和 dynamodb。
- transformation_ctx – 要使用的转换上下文 (可选)。
- options – 可选名称/值对的集合。有关更多信息，请参阅 AWS Glue for Spark 中适用于 ETL 的连接类型和选项。

以下是使用 getSource 的示例。

```
>>> 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="")

返回一个从 Apache Spark 弹性分布式数据集 (RDD) 创建的 DynamicFrame。

- data – 要使用的数据源。
- name – 要使用的数据的名称。
- schema – 要使用的架构 (可选)。

- sample_ratio – 要使用的采样率 (可选)。
- transformation_ctx – 要使用的转换上下文 (可选)。

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)

返回一个使用数据目录数据库和表名称创建的 DynamicFrame。使用此方法时，您可以format_options通过指定的 Glue Data Cat AWS alog 表提供表格属性，并通过additional_options参数提供其他选项。

- Database – 要从中读取的数据库。
- table_name – 要从中读取的表的名称。
- redshift_tmp_dir – 要使用的 Amazon Redshift 临时目录（可选）。
- transformation_ctx – 要使用的转换上下文 (可选)。
- push_down_predicate – 筛选分区，而不必列出并读取数据集中的所有文件。有关支持的来源和限制，请参阅 AWS Glue ETL 中使用向下推优化读取。有关更多信息，请参阅 使用下推谓词进行预筛选。
- additional_options – 可选名称/值对的集合。可能选项包括 AWS Glue for Spark 中适用于 ETL 的连接类型和选项 中列出的选项，但 endpointUrl、streamName、bootstrap.servers、security.protocol、topicName、classi 和 delimiter 除外。另一个支持的选项是 catalogPartitionPredicate：

  catalogPartitionPredicate – 要传递目录表达式以根据索引列进行筛选。这样会将筛选下推到服务器端。有关更多信息，请参阅 AWS Glue 分区数据。请注意，push_down_predicate 和 catalogPartitionPredicate 使用不同的语法。前者使用 Spark SQL 标准语法，后者使用 JSQL 解析器。
- catalog_id – 正在访问的数据目录 ID（账户 ID）。当为 None 时，将使用调用方的默认账户 ID。

create_dynamic_frame_from_options

create_dynamic_frame_from_options(connection_type, connection_options={}, format=None, format_options={}, transformation_ctx = "")

返回一个使用指定连接和格式创建的 DynamicFrame。

- connection_type – 连接类型，例如 Amazon S3、Amazon Redshift 和 JDBC。有效值包括 s3、mysql、postgresql、redshift、sqlserver、oracle 和 dynamodb。
- connection_options – 连接选项，例如路径和数据库表（可选）。对于 s3 的 connection_type，定义 Amazon S3 路径的列表。

```
connection_options = {"paths": ["s3://aws-glue-target/temp"]}
```

对于 JDBC 连接，必须定义多个属性。请注意，数据库名称必须是 URL 的一部分。它可以选择性地包含在连接选项中。

> **Warning**
>
> 不建议在脚本中存储密码。考虑使用boto3从 AWS Secrets Manager 或 AWS Glue 数据目录中检索它们。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

dbtable 属性是 JDBC 表的名称。对于在数据库中支持架构的 JDBC 数据存储，指定 schema.table-name。如果未提供架构，则使用默认的“public”架构。

有关更多信息，请参阅 AWS Glue for Spark 中适用于 ETL 的连接类型和选项。

- format— 格式规范。这用于 Amazon S3 或支持多种格式的 AWS Glue 连接。有关支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- format_options – 指定格式的格式选项。有关支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- transformation_ctx – 要使用的转换上下文 (可选)。
- push_down_predicate – 筛选分区，而不必列出并读取数据集中的所有文件。有关支持的来源和限制，请参阅 AWS Glue ETL 中使用向下推优化读取。有关更多信息，请参阅使用下推谓词进行预筛选。

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)

返回一个使用数据目录数据库和表名称创建的 DynamicFrame 示例。DynamicFrame 只包含来自数据源的第一个 num 记录。

- database – 要从中读取的数据库。
- table_name – 要从中读取的表的名称。
- num – 返回的动态帧示例中的最大记录数。
- redshift_tmp_dir – 要使用的 Amazon Redshift 临时目录（可选）。
- transformation_ctx – 要使用的转换上下文 (可选)。
- push_down_predicate – 筛选分区，而不必列出并读取数据集中的所有文件。有关更多信息，请参阅 使用下推谓词进行预筛选。
- additional_options – 可选名称/值对的集合。可能选项包括 AWS Glue for Spark 中适用于 ETL 的连接类型和选项 中列出的选项，但 endpointUrl、streamName、bootstrap.servers、security.protocol、topicName、classi 和 delimiter 除外。
- sample_options – 控制采样行为的参数（可选）。Amazon S3 源的当前可用参数：
  - maxSamplePartitions – 采样将读取的最大分区数。默认值为 10
  - maxSampleFilesPerPartition – 采样将在一个分区中读取的最大文件数。默认值为 10。

    这些参数有助于减少文件列表所耗费的时间。例如，假设数据集有 1000 个分区，每个分区有 10 个文件。如果您设置 maxSamplePartitions = 10，以及 maxSampleFilesPerPartition = 10，则取样不会列出所有 10,000 个文件，而是只列出并读取前 10 个分区，每个分区中的前 10 个文件：10*10 = 共 100 个文件。
- catalog_id – 正在访问的数据目录的目录 ID（数据目录的账户 ID）。默认设置为 None。None 默认为服务中调用账户的目录 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 = "")

返回一个使用指定连接和格式创建的 DynamicFrame 示例。DynamicFrame 只包含来自数据源的第一个 num 记录。

- connection_type – 连接类型，例如 Amazon S3、Amazon Redshift 和 JDBC。有效值包括 s3、mysql、postgresql、redshift、sqlserver、oracle 和 dynamodb。
- connection_options – 连接选项，例如路径和数据库表（可选）。有关更多信息，请参阅 AWS Glue for Spark 中适用于 ETL 的连接类型和选项。
- num – 返回的动态帧示例中的最大记录数。
- sample_options – 控制采样行为的参数（可选）。Amazon S3 源的当前可用参数：
  - maxSamplePartitions – 采样将读取的最大分区数。默认值为 10
  - maxSampleFilesPerPartition – 采样将在一个分区中读取的最大文件数。默认值为 10。

    这些参数有助于减少文件列表所耗费的时间。例如，假设数据集有 1000 个分区，每个分区有 10 个文件。如果您设置 maxSamplePartitions = 10，以及 maxSampleFilesPerPartition = 10，则取样不会列出所有 10,000 个文件，而是只列出并读取前 10 个分区，每个分区中的前 10 个文件：10*10 = 共 100 个文件。
- format— 格式规范。这用于 Amazon S3 或支持多种格式的 AWS Glue 连接。有关支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- format_options – 指定格式的格式选项。有关支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- transformation_ctx – 要使用的转换上下文 (可选)。
- push_down_predicate – 筛选分区，而不必列出并读取数据集中的所有文件。有关更多信息，请参阅 使用下推谓词进行预筛选。

add_ingestion_time_columns

add_ingestion_time_columns(dataFrame, timeGranularity = "")

将提取时间列（例如 ingest_year、ingest_month、ingest_day、ingest_hour、ingest_minute）附加到输入 DataFrame。当您指定以 Amazon S3 为目标的数据目录表时，AWS Glue 生成的脚本中会自动生成

此函数。此函数使用输出表上的提取时间列自动更新分区。这允许根据提取时间自动对输出数据进行分区，而不需要在输入数据中显示提取时间列。

- dataFrame – 提取时间列要附加到的 dataFrame。
- timeGranularity – 时间列的粒度。有效值为“day”、“hour”和“minute”。例如，如果“hour”传递到函数，原始 dataFrame 将附加“ingest_year”、“ingest_month”、“ingest_day”和“ingest_hour”时间列。

在附加时间粒度列后返回数据框。

例如：

```
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 = {})

返回一个使用数据目录数据表中的信息创建的 DataFrame。

- database – 要从中读取数据的数据目录数据库。
- table_name – 要从中读取数据的数据目录表的名称。
- transformation_ctx – 要使用的转换上下文 (可选)。
- additional_options – 可选名称/值对的集合。可能选项包括流式传输源的 AWS Glue for Spark 中适用于 ETL 的连接类型和选项 中列出的选项，例如 startingPosition、maxFetchTimeInMs 和 startingOffsets。
  - useSparkDataSource— 设置为 true 时，会强 AWS 制 Glue 使用原生 Spark 数据源 API 来读取表格。Spark 数据源 API 支持以下格式：AVRO、二进制、CSV、JSON、ORC、Parquet 和文本。在数据目录表中，您可以使用 classification 属性指定格式。要了解有关 Spark 数据源 API 的更多信息，请参阅 Apache Spark 官方文档。

    create_data_frame_from_catalog 与 useSparkDataSource 结合使用具有以下优势：
    - 直接返回一个 DataFrame 并提供 create_dynamic_frame.from_catalog().toDF() 的替代项。
    - 支持原生格式的 AWS Lake Formation 表级权限控制。

- 支持读取数据湖格式，无需 AWS Lake Formation 表级权限控制。有关更多信息，请参阅 在 AWS Glue ETL 任务中使用数据湖框架。

  启用后useSparkDataSource，您还可以根据需要在中additional_options添加任何 Spark 数据源选项。 AWS Glue 将这些选项直接传递给 Spark 阅读器。
- useCatalogSchema— 设置为 true 时， AWS Glue 会将数据目录架构应用于生成的数据DataFrame。否则，读取器会从数据中推断出架构。如果启用 useCatalogSchema，则必须同时将 useSparkDataSource 设置为 true。

限制

使用 useSparkDataSource 选项时请考虑以下限制：

- 当你使用时useSparkDataSource， AWS Glue 会在单独的 Spark 会话DataFrame中创建一个与原始 Spark 会话不同的新会话。
- Spark DataFrame 分区过滤不适用于以下 AWS Glue 功能。
  - 作业书签
  - 排除 Amazon S3 存储类
  - 目录分区谓词

  要将分区过滤与这些功能一起使用，可以使用 Glue AWS 下推谓词。有关更多信息，请参阅 使用下推谓词进行预筛选。对未分区列的筛选不受影响。

  以下示例脚本演示了使用 excludeStorageClasses 选项执行分区筛选的错误方法。

```
// 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'")
```

以下示例脚本演示了使用下推谓词以使用 excludeStorageClasses 选项执行分区筛选的正确方法。

```
// 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'")
```

示例：使用 Spark 数据源读取器创建 CSV 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 现已弃用。请改用 getSource() API。返回一个使用指定连接和格式创建的 DataFrame。仅将此函数与 AWS Glue 串流源结合使用。

- connection_type – 流式传输连接类型。有效值包括 kinesis 和 kafka。
- connection_options – 连接选项，不同于 Kinesis 和 Kafka。您可以在 AWS Glue for Spark 中适用于 ETL 的连接类型和选项 找到每个流式传输数据源的所有连接选项列表。请注意流式传输连接选项的以下差异：

  - Kinesis 流式传输源需要 `streamARN`、`startingPosition`、`inferSchema` 和 `classification`。
  - Kinesis 流式传输源需要 `connectionName`、`topicName`、`startingOffsets`、`inferSchema` 和 `classification`。
- `format`— 格式规范。这用于 Amazon S3 或支持多种格式的 AWS Glue 连接。有关所支持格式的信息，请参阅AWS Glue for Spark 中的输入和输出的数据格式选项。
- `format_options` – 指定格式的格式选项。有关所支持的格式选项的信息，请参阅AWS Glue for Spark 中的输入和输出的数据格式选项。
- `transformation_ctx` – 要使用的转换上下文 (可选)。

Amazon Kinesis 流式传输源示例：

```
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 流式传输源示例：

```
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)

将传入的 batch_function 应用于从流式传输源读取的每个微批处理。

- frame— DataFrame 包含当前微批次的。
- batch_function – 应用于每个微处理的函数。
- options – 键值对集合，其中包含有关如何处理微批处理的信息。以下选项为必填：
  - windowSize – 处理每个批处理所花费的时间量。
  - checkpointLocation – 为流式传输 ETL 任务存储检查点的位置。
  - batchMaxRetries – 在该批处理失败时重试的最大次数。默认值为 3。此选项仅适用于 Glue 版本 2.0 及更高版本。

示例：

```
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 中使用数据集

- purge_table
- purge_s3_path
- transition_table
- transition_s3_path

purge_table

purge_table(catalog_id=None, database="", table_name="", options={}, transformation_ctx="")

从 Amazon S3 中删除指定目录的数据库和表的文件。如果删除一个分区中的所有文件，则也会从目录中删除该分区。

如果您希望能恢复已删除的对象，可以在 Amazon S3 存储桶上启用对象版本控制。如果从未启用对象版本控制的存储桶中删除一个对象，则无法恢复该对象。有关如何恢复已启用版本控制的存储桶中的已删除对象的更多信息，请参阅 AWS Support 知识中心中的如何检索已删除的 Amazon S3 对象？。

- `catalog_id` – 正在访问的数据目录的目录 ID（数据目录的账户 ID）。默认设置为 None。None 默认为服务中调用账户的目录 ID。
- `database` – 要使用的数据库。
- `table_name` – 要使用的表的名称。
- `options` – 用于筛选要删除的文件和清单文件生成的选项。
  - `retentionPeriod` – 指定保留文件的时段（以小时为单位）。比保留期更新的文件将被保留。默认情况下设置为 168 小时（7 天）。
  - `partitionPredicate` – 满足此谓词的分区将被删除。这些分区中保留期内的文件不会被删除。设置为 `""` – 默认情况下为空。
  - `excludeStorageClasses` – `excludeStorageClasses` 集中包含存储类的文件不会被删除。默认值是 `Set()` – 一个空集。
  - `manifestFilePath` – 用于生成清单文件的可选路径。所有成功清除的文件将记录在 `Success.csv` 中，所有未能清除的文件将记录在 `Failed.csv` 中
- `transformation_ctx` – 要使用的转换上下文 (可选)。在清单文件路径中使用。

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="")

以递归方式从指定的 Amazon S3 路径中删除文件。

如果您希望能恢复已删除的对象，可以在 Amazon S3 存储桶上启用对象版本控制。如果从未启用对象版本控制的存储桶中删除一个对象，则无法恢复该对象。有关如何使用版本控制恢复存储桶中已删除对象的更多信息，请参阅如何检索已删除的 Amazon S3 对象？ 在 AWS Support 知识中心中。

- s3_path – 要删除的文件 Amazon S3 中的路径，格式为 s3://<*bucket*>/<*prefix*>/
- options – 用于筛选要删除的文件和清单文件生成的选项。
  - retentionPeriod – 指定保留文件的时段（以小时为单位）。比保留期更新的文件将被保留。默认情况下设置为 168 小时（7 天）。
  - excludeStorageClasses – excludeStorageClasses 集中包含存储类的文件不会被删除。默认值是 Set() – 一个空集。
  - manifestFilePath – 用于生成清单文件的可选路径。所有成功清除的文件将记录在 Success.csv 中，所有未能清除的文件将记录在 Failed.csv 中
- transformation_ctx – 要使用的转换上下文 (可选)。在清单文件路径中使用。

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)

为指定目录的数据库和表转换存储在 Amazon S3 上的文件的存储类。

可以在任意两个存储类之间转换。对于 GLACIER 和 DEEP_ARCHIVE 存储类，您可以转换到这些类。但您可以使用 S3 RESTORE 从 GLACIER 和 DEEP_ARCHIVE 存储类进行转换。

如果您要运行从 Amazon S3 读取文件或分区的 AWS Glue ETL 任务，则可以排除某些 Amazon S3 存储类类型。有关更多信息，请参阅排除 Amazon S3 存储类。

- database – 要使用的数据库。
- table_name – 要使用的表的名称。
- transition_to – 要切换到的 Amazon S3 存储类。
- options – 用于筛选要删除的文件和清单文件生成的选项。
  - retentionPeriod – 指定保留文件的时段（以小时为单位）。比保留期更新的文件将被保留。默认情况下设置为 168 小时（7 天）。
  - partitionPredicate – 将转换满足此谓词的分区。这些分区中保留期内的文件不会被转换。设置为 "" – 默认情况下为空。
  - excludeStorageClasses – excludeStorageClasses 集中包含存储类的文件不会被转换。默认值是 Set() – 一个空集。
  - manifestFilePath – 用于生成清单文件的可选路径。所有成功转换的文件将记录在 Success.csv 中，所有未能转换的文件将记录在 Failed.csv 中
  - accountId – 要运行转换的 Amazon Web Services 账户 ID。对于此转换是必需的。
  - roleArn— 运行过渡转换的 AWS 角色。对于此转换是必需的。
- transformation_ctx – 要使用的转换上下文 (可选)。在清单文件路径中使用。
- catalog_id – 正在访问的数据目录的目录 ID（数据目录的账户 ID）。默认设置为 None。None 默认为服务中调用账户的目录 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="")

以递归方式转换指定 Amazon S3 路径中的文件的存储类。

可以在任意两个存储类之间转换。对于 GLACIER 和 DEEP_ARCHIVE 存储类，您可以转换到这些类。但您可以使用 S3 RESTORE 从 GLACIER 和 DEEP_ARCHIVE 存储类进行转换。

如果您要运行从 Amazon S3 读取文件或分区的 AWS Glue ETL 任务，则可以排除某些 Amazon S3 存储类类型。有关更多信息，请参阅排除 Amazon S3 存储类。

- s3_path – 要转换的文件的 Amazon S3 中的路径，格式为 s3://<*bucket*>/<*prefix*>/
- transition_to – 要切换到的 Amazon S3 存储类。
- options – 用于筛选要删除的文件和清单文件生成的选项。
  - retentionPeriod – 指定保留文件的时段（以小时为单位）。比保留期更新的文件将被保留。默认情况下设置为 168 小时（7 天）。
  - partitionPredicate – 将转换满足此谓词的分区。这些分区中保留期内的文件不会被转换。设置为 "" – 默认情况下为空。
  - excludeStorageClasses – excludeStorageClasses 集中包含存储类的文件不会被转换。默认值是 Set() – 一个空集。
  - manifestFilePath – 用于生成清单文件的可选路径。所有成功转换的文件将记录在 Success.csv 中，所有未能转换的文件将记录在 Failed.csv 中
  - accountId – 要运行转换的 Amazon Web Services 账户 ID。对于此转换是必需的。
  - roleArn— 运行过渡转换的 AWS 角色。对于此转换是必需的。
- transformation_ctx – 要使用的转换上下文 (可选)。在清单文件路径中使用。

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"})
```

提取

- extract_jdbc_conf

extract_jdbc_conf

`extract_jdbc_conf(connection_name, catalog_id = None)`

从数据目录中的 `dict` 连接对象返回具有键（包含配置属性）的 AWS Glue。

- `user` - 数据库用户名。
- `password` - 数据库密码。
- `vendor` - 指定供应商（`mysql`、`postgresql`、`oracle`、`sqlserver`、等）。
- `enforceSSL` - 一个布尔字符串，指示是否需要安全连接。
- `customJDBCCert` - 使用指示的 Amazon S3 路径中的特定客户端证书。
- `skipCustomJDBCCertValidation` - 一个布尔字符串，指示 `customJDBCCert` 是否必须由 CA 验证。
- `customJDBCCertString` - 关于自定义证书的其他信息，特定于驱动程序类型。
- `url` -（已弃用）仅包含协议、服务器和端口的 JDBC URL。
- `fullUrl` - 创建连接时输入的 JDBC URL（AWS Glue 版本 3.0 或更高版本中可用）。

检索 JDBC 配置的示例：

```
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'}
```

事务

- start_transaction
- commit_transaction
- cancel_transaction

start_transaction

`start_transaction(read_only)`

开启新事务。内部调用 Lake Formation startTransaction API。

- read_only – (布尔值) 指示此事务应为只读还是读写。使用只读事务 ID 进行的写入将被拒绝。只读事务不需要提交。

返回事务 ID。

commit_transaction

commit_transaction(transaction_id, wait_for_commit = True)

尝试提交指定的事务。可能在事务完成提交之前返回 commit_transaction。内部调用 Lake Formation commitTransaction API。

- transaction_id – (字符串) 要提交的事务。
- wait_for_commit – (布尔值) 确定是否立即返回 commit_transaction。默认值为 true。如果为假，则轮询 commit_transaction 并等待事务提交。等待时间限制为 1 分钟使用指数回退，最多重试 6 次。

返回布尔值，以指示是否完成提交。

cancel_transaction

cancel_transaction(transaction_id)

尝试取消指定的事务。如果事务以前已提交，则返回 TransactionCommittedException 异常。在内部调用 Lake Format CancelTransactionion API。

- transaction_id – (字符串) 要取消的事务。

编写

- 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)

获取一个 DataSink 对象，该对象可用于将 DynamicFrames 写入外部来源。先检查 SparkSQL format 以确保获得预期的接收器。

- connection_type – 要使用的连接类型，例如 Amazon S3、Amazon Redshift 和 JDBC。有效值包括 s3、mysql、postgresql、redshift、sqlserver、oracle、kinesis 和 kafka。
- format – 要使用的 SparkSQL 格式 (可选)。
- transformation_ctx – 要使用的转换上下文 (可选)。
- options – 用于指定连接选项的名称-值对的集合。一些可能的值包括：
  - user 和 password：适用于授权
  - url：数据存储的端点
  - dbtable：目标表的名称
  - bulkSize：插入操作的并行度

可以指定的选项取决于连接类型。有关其他值和示例，请参阅 AWS Glue for Spark 中适用于 ETL 的连接类型和选项。

例如：

```
>>> 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 = "")

写入并返回一个使用指定连接和格式的 DynamicFrame。

- frame – 要编写的 DynamicFrame。
- connection_type – 连接类型，例如 Amazon S3、Amazon Redshift 和 JDBC。有效值包括 s3、mysql、postgresql、redshift、sqlserver、oracle、kinesis 和 kafka。

- connection_options – 连接选项，例如路径和数据库表 (可选)。对于 connection_type 的 s3，将会定义 Amazon S3 路径。

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

对于 JDBC 连接，必须定义多个属性。请注意，数据库名称必须是 URL 的一部分。它可以选择性地包含在连接选项中。

> **Warning**
>
> 不建议在脚本中存储密码。考虑使用boto3从 AWS Secrets Manager 或 AWS Glue 数据目录中检索它们。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

dbtable 属性是 JDBC 表的名称。对于在数据库中支持架构的 JDBC 数据存储，指定 schema.table-name。如果未提供架构，则使用默认的“public”架构。

有关更多信息，请参阅 AWS Glue for Spark 中适用于 ETL 的连接类型和选项。

- format— 格式规范。这用于 Amazon S3 或支持多种格式的 AWS Glue 连接。有关支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- format_options – 指定格式的格式选项。有关支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- transformation_ctx – 要使用的转换上下文 (可选)。

write_from_options

write_from_options(frame_or_dfc, connection_type, connection_options={}, format={}, format_options={}, transformation_ctx = "")

写入并返回一个使用指定连接和格式信息创建的 DynamicFrame or DynamicFrameCollection。

- frame_or_dfc – 要写入的 DynamicFrame 或 DynamicFrameCollection。

- connection_type – 连接类型，例如 Amazon S3、Amazon Redshift 和 JDBC。有效值包括 s3、mysql、postgresql、redshift、sqlserver 和 oracle。
- connection_options – 连接选项，例如路径和数据库表 (可选)。对于 connection_type 的 s3，将会定义 Amazon S3 路径。

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

对于 JDBC 连接，必须定义多个属性。请注意，数据库名称必须是 URL 的一部分。它可以选择性地包含在连接选项中。

> **Warning**
>
> 不建议在脚本中存储密码。考虑使用boto3从 AWS Secrets Manager 或 AWS Glue 数据目录中检索它们。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

dbtable 属性是 JDBC 表的名称。对于在数据库中支持架构的 JDBC 数据存储，指定 schema.table-name。如果未提供架构，则使用默认的“public”架构。

有关更多信息，请参阅 AWS Glue for Spark 中适用于 ETL 的连接类型和选项。

- format— 格式规范。这用于 Amazon S3 或支持多种格式的 AWS Glue 连接。有关支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- format_options – 指定格式的格式选项。有关支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。
- transformation_ctx – 要使用的转换上下文 (可选)。

write_dynamic_frame_from_catalog

write_dynamic_frame_from_catalog(frame, database, table_name, redshift_tmp_dir, transformation_ctx = "", additional_options = {}, catalog_id = None)

写入并返回一个使用数据目录数据库和表中信息的 DynamicFrame。

- frame – 要编写的 DynamicFrame。
- Database – 包含表的数据目录数据库。
- table_name – 与目标关联的数据目录表的名称。
- redshift_tmp_dir – 要使用的 Amazon Redshift 临时目录（可选）。
- transformation_ctx – 要使用的转换上下文 (可选)。
- additional_options – 可选名称/值对的集合。
- catalog_id – 正在访问的数据目录 ID（账户 ID）。当为 None 时，将使用调用方的默认账户 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)

写入并返回一个使用数据目录数据库和表中信息的 DataFrame。此方法支持写入数据湖格式（Hudi、Iceberg 和 Delta Lake）。有关更多信息，请参阅 在 AWS Glue ETL 任务中使用数据湖框架。

- frame – 要编写的 DataFrame。
- Database – 包含表的数据目录数据库。
- table_name – 与目标关联的数据目录表的名称。
- redshift_tmp_dir – 要使用的 Amazon Redshift 临时目录（可选）。
- transformation_ctx – 要使用的转换上下文 (可选)。
- additional_options – 可选名称/值对的集合。
  - useSparkDataSink— 设置为 true 时，会强 AWS 制 Glue 使用原生 Spark Data Sink API 写入表。启用此选项后，您可以根据需要向中添加任何 Spark 数据源选项。additional_options AWS Glue 将这些选项直接传递给 Spark 编写器。
- catalog_id – 正在访问的数据目录 ID（账户 ID）。如果您未指定值，则使用调用方的默认账户 ID。

限制

使用 useSparkDataSink 选项时请考虑以下限制：

- 使用 useSparkDataSink 选项时不支持 enableUpdateCatalog 选项。

示例：使用 Spark 数据源写入器写入 Hudi 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)`

写入并返回一个使用指定 JDBC 和连接信息的 DynamicFrame。

- `frame` – 要编写的 DynamicFrame。
- `catalog_connection` – 要使用的目录连接。
- `connection_options` – 连接选项，例如路径和数据库表 (可选)。有关更多信息，请参阅 AWS Glue for Spark 中适用于 ETL 的连接类型和选项。
- `redshift_tmp_dir` – 要使用的 Amazon Redshift 临时目录（可选）。
- `transformation_ctx` – 要使用的转换上下文 (可选)。
- `catalog_id` – 正在访问的数据目录 ID（账户 ID）。当为 None 时，将使用调用方的默认账户 ID。

write_from_jdbc_conf

`write_from_jdbc_conf(frame_or_dfc, catalog_connection, connection_options={}, redshift_tmp_dir = "", transformation_ctx = "", catalog_id = None)`

写入并返回一个使用指定 JDBC 和连接信息的 DynamicFrame or DynamicFrameCollection。

- `frame_or_dfc` – 要写入的 DynamicFrame 或 DynamicFrameCollection。
- `catalog_connection` – 要使用的目录连接。
- `connection_options` – 连接选项，例如路径和数据库表 (可选)。有关更多信息，请参阅 AWS Glue for Spark 中适用于 ETL 的连接类型和选项。
- `redshift_tmp_dir` – 要使用的 Amazon Redshift 临时目录（可选）。
- `transformation_ctx` – 要使用的转换上下文 (可选)。
- `catalog_id` – 正在访问的数据目录 ID（账户 ID）。当为 None 时，将使用调用方的默认账户 ID。

AWS Glue PySpark 变换参考

AWS Glue 提供了以下内置变换，您可以在 PySpark ETL 操作中使用这些变换。您的数据在名为 a 的数据结构中从一个转换传递到另一个转换 DynamicFrame，该数据结构是 Apache Spark SQL DataFrame 的扩展。DynamicFrame 包含您的数据，并引用其架构来处理您的数据。

此外，其中的大多数转换也将作为 DynamicFrame 类的方法存在。有关更多信息，请参见DynamicFrame 变换。

- GlueTransform 基类
- ApplyMapping 类
- DropFields 类
- DropNullFields 类
- ErrorsAsDynamicFrame 类
- EvaluateDataQuality 类
- FillMissingValues 类
- Filter 类
- FindIncrementalMatches 类
- FindMatches 类

- FlatMap 类
- Join 类
- Map 类
- MapToCollection 类
- mergeDynamicFrame
- Relationalize 类
- RenameField 类
- ResolveChoice 类
- SelectFields 类
- SelectFromCollection 类
- Simplify_ddb_json 类
- Spigot 类
- SplitFields 类
- SplitRows 类
- Unbox 类
- UnnestFrame 类

GlueTransform 基类

所有 `awsglue.transforms` 类继承的基类。

这些类全部定义 `__call__` 方法。它们要么重写以下各节中列出的 `GlueTransform` 类方法，要么在默认情况下使用类名调用它们。

方法

- apply(cls, *args, **kwargs)
- name(cls)
- describeArgs(cls)
- describeReturn(cls)
- describeTransform(cls)
- describeErrors(cls)
- describe(cls)

apply(cls, *args, **kwargs)

通过调用转换类来应用转换，并返回结果。

- `cls` – `self` 类对象。

name(cls)

返回派生的转换类的名称。

- `cls` – `self` 类对象。

describeArgs(cls)

- `cls` – `self` 类对象。

返回各自与命名参数对应的字典的列表，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)"
  },
...
]
```

在未实现它的派生转换中调用时引发 `NotImplementedError` 异常。

describeReturn(cls)

- `cls` – `self` 类对象。

返回具有有关返回类型的信息的字典，格式如下：

```
{
  "type": "(return type)",
```

```
  "description": "(description of output)"
}
```

在未实现它的派生转换中调用时引发 NotImplementedError 异常。

describeTransform(cls)

返回一个描述转换的字符串。

- cls – self 类对象。

在未实现它的派生转换中调用时引发 NotImplementedError 异常。

describeErrors(cls)

- cls – self 类对象。

返回各自描述此转换引发的可能异常的字典的列表，格式如下：

```
[
  {
    "type": "(type of error)",
    "description": "(description of error)"
  },
...
]
```

describe(cls)

- cls – self 类对象。

返回具有以下格式的对象：

```
{
  "transform" : {
    "name" : cls.name( ),
    "args" : cls.describeArgs( ),
    "returns" : cls.describeReturn( ),
    "raises" : cls.describeErrors( ),
    "location" : "internal"
```

```
  }
}
```

ApplyMapping 类

在 DynamicFrame 中应用映射。

示例

我们建议您使用 DynamicFrame.apply_mapping() 方法在 DynamicFrame 中应用映射。要查看代码示例，请参阅 示例：使用 apply_mapping 重命名字段并更改字段类型。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

__call__(frame, mappings, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

将声明映射到指定的 DynamicFrame。

- frame – 要对其应用映射的 DynamicFrame（必需）。
- mappings – 映射元组列表（必需）。每个元组包括：（源列，源类型，目标列，目标类型）。

  如果源列中有一个点“.”在名称里，则必须在名称外加上反引号“``”。例如，要将 this.old.name（字符串）映射到 thisNewName，可以使用以下元组：

  ```
  ("`this.old.name`", "string", "thisNewName", "string")
  ```
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与转换中的错误关联的字符串（可选）。
- stageThreshold – 在转换出错之前可能在其中发生的最大错误数（可选）。默认值为 0。

- totalThreshold – 在处理出错之前可能全面发生的最大错误数（可选）。默认值为 0。

仅返回“映射”元组中指定的 DynamicFrame 字段。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

DropFields 类

删除 DynamicFrame 中的字段。

示例

我们建议您使用 DynamicFrame.drop_fields() 方法从 DynamicFrame 中删除字段。要查看代码示例，请参阅 示例：使用 drop_fields 从 DynamicFrame 中删除字段。

方法

- __call__

- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

__call__(frame, paths, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

删除 DynamicFrame 中的字节。

- `frame` – 要在其中删除节点的 DynamicFrame（必需）。
- `paths` – 要删除的节点的完整路径的列表 (必需)。
- `transformation_ctx` – 用于标识状态信息的唯一字符串 (可选)。
- `info` – 与转换中的错误关联的字符串 (可选)。
- `stageThreshold` – 在转换出错之前可能在其中发生的最大错误数（可选）。默认值为 0。
- `totalThreshold` – 在处理出错之前可能全面发生的最大错误数（可选）。默认值为 0。

返回不包含指定字段的新 DynamicFrame。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

DropNullFields 类

删除 DynamicFrame 中类型为 NullType 的所有 null 字段。即 DynamicFrame 数据集内每个记录中缺少值或值为 null 的字段。

示例

此示例用 DropNullFields 创建新的 DynamicFrame，其中类型为 NullType 的字段已删除。为了演示如何使用 DropNullFields，我们将类型为 null 的新列 empty_column 添加到已加载的 persons 数据集。

> **Note**
>
> 要访问本示例中使用的数据集，请参阅 代码示例：对数据进行联接和关系化 并按照 步骤 1：爬取 Amazon S3 存储桶中的数据 中的说明进行操作。

```
# 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()
```

输出

```
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)

删除 DynamicFrame 中类型为 NullType 的所有 null 字段。即 DynamicFrame 数据集内每个记录中缺少值或值为 null 的字段。

- frame – 要在其中删除 null 字段的 DynamicFrame（必需）。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与转换中的错误关联的字符串 (可选)。
- stageThreshold – 在转换出错之前可能在其中发生的最大错误数（可选）。默认值为 0。
- totalThreshold – 在处理出错之前可能全面发生的最大错误数（可选）。默认值为 0。

返回一个不包含 null 字段的新 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 类

返回 `DynamicFrame`，其中包含源 `DynamicFrame` 创建时所发生错误的嵌套记录。

示例

我们建议您使用 `DynamicFrame.errorsAsDynamicFrame()` 方法检索和查看错误记录。要查看代码示例，请参阅 示例：使用 errorsAsDynamicFrame 查看错误记录。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

__call__(frame)

返回 `DynamicFrame`，其中包含与源 `DynamicFrame` 相关的嵌套错误记录。

- `frame` – 源 `DynamicFrame` (必需)。

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 类

根据 DynamicFrame 中的数据评估数据质量规则集，并返回一个包含数据质量评估结果的新 DynamicFrame。

示例

以下示例代码演示了如何评估 DynamicFrame 的数据质量，然后查看数据质量结果。

```
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()
```

输出

```
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` – 表示您想要评估其数据质量的 `DynamicFrame`。
- `ruleset` – 字符串格式的数据质量定义语言（DQDL）规则集。要了解有关 DQDL 的更多信息，请参阅数据质量定义语言（DQDL）引用指南。
- `publishing_options` – 一个字典，用于为发布评估结果和指标指定以下选项：
  - `dataQualityEvaluationContext` – 一个字符串，用于指定 AWS Glue 应在哪个命名空间下发布 Amazon CloudWatch 指标和数据质量结果。汇总指标显示在 CloudWatch 中，而完整结果显示在 AWS Glue Studio 界面中。
    - 必需：否
    - 默认值：`default_context`
  - `enableDataQualityCloudWatchMetrics` – 指定是否应将数据质量评估结果发布到 CloudWatch。您可以使用 `dataQualityEvaluationContext` 选项为指标指定命名空间。
    - 必需：否
    - 默认值：False
  - `enableDataQualityResultsPublishing` – 指定是否应在 AWS Glue Studio 界面的 Data Quality（数据质量）选项卡上显示数据质量结果。
    - 必需：否
    - 默认值：true
  - `resultsS3Prefix` – 指定 AWS Glue 可以写入数据质量评估结果的 Amazon S3 位置。

- 必需：否
- 默认值："（空字符串）

apply(cls, *args, **kwargs)

继承自 `GlueTransform` apply。

name(cls)

继承自 `GlueTransform` name。

describeArgs(cls)

继承自 `GlueTransform` describeArgs。

describeReturn(cls)

继承自 `GlueTransform` describeReturn。

describeTransform(cls)

继承自 `GlueTransform` describeTransform。

describeErrors(cls)

继承自 `GlueTransform` describeErrors。

describe(cls)

继承自 `GlueTransform` describe。

FillMissingValues 类

`FillMissingValues` 类使用机器学习方法（如线性回归和随机森林）填充指定 `DynamicFrame` 列中的 null 值和空字符串，以预测缺失值。ETL 任务使用输入数据集中的值来训练机器学习模型，然后该模型预测缺失的值应该是什么。

> **Tip**
>
> 如果您使用增量数据集，则每个增量集都会用作机器学习模型的训练数据，因此结果可能不是如此准确。

导入：

```
from awsglueml.transforms import FillMissingValues
```

方法

- Apply

apply(frame, missing_values_column, output_column ="", transformation_ctx ="", info ="", stageThreshold = 0, totalThreshold = 0)

填充指定列中的动态帧缺失值，并在新列中返回包含估计值的新帧。对于没有缺失值的行，指定列的值将复制到新列。

- `frame` – 要填充缺失值的 `DynamicFrame`。必需。
- `missing_values_column` – 包含缺失值（`null` 值和空字符串）的列。必需。
- `output_column` – 新列的名称，该列将包含缺失值的所有行的估计值。可选；默认值是后缀为 `"_filled"` 的 `missing_values_column`。
- `transformation_ctx` – 用于标识状态信息的唯一字符串 (可选)。
- `info` – 与转换中的错误关联的字符串 (可选)。
- `stageThreshold` – 在转换出错之前可能在其中发生的最大错误数 (可选；默认值为零)。
- `totalThreshold` – 在处理出错之前可能全面发生的最大错误数 (可选；默认值为零)。

返回带附加列的新 `DynamicFrame`，该列包含带缺失值的行的估计值和其他行的当前值。

Filter 类

生成新的 `DynamicFrame`，其中包含满足指定谓词函数的输入 `DynamicFrame` 中的记录。

示例

我们建议您使用 `DynamicFrame.filter()` 方法在 `DynamicFrame` 中筛选记录。要查看代码示例，请参阅 示例：使用筛选条件获取筛选后的字段选择。

方法

- __call__
- apply

- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0))

返回新的 DynamicFrame，其通过从满足指定谓词函数的输入 DynamicFrame 中选择记录生成。

- frame – 要将指定筛选函数应用到的源 DynamicFrame (必需)。
- f – 要应用到 DynamicFrame 中的每个 DynamicRecord 的谓词函数。该函数必须将 DynamicRecord 作为实际参数，并且如果 DynamicRecord 满足筛选要求，则返回 True，否则返回 False（必需）。

  一个 DynamicRecord 表示 DynamicFrame 中的一条逻辑记录。其类似于 Spark DataFrame 中的一行，但其具有自描述性，可用于不符合固定架构的数据。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与转换中的错误关联的字符串（可选）。
- stageThreshold – 在转换出错之前可能在其中发生的最大错误数（可选）。默认值为 0。
- totalThreshold – 在处理出错之前可能全面发生的最大错误数（可选）。默认值为 0。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

FindIncrementalMatches 类

标识现有和递增 DynamicFrame 中的匹配记录，并创建一个新的 DynamicFrame，其中包含分配给每组匹配记录的唯一标识符。

导入：

```
from awsglueml.transforms import FindIncrementalMatches
```

方法

- Apply

apply(existingFrame, incrementalFrame, transformId, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0, enforcedMatches = none, computeMatchConfidenceScores = 0)

标识输入 DynamicFrame 中的匹配记录，并创建一个新的 DynamicFrame，其中包含分配给每组匹配记录的唯一标识符。

- existingFrame – 现有和预先匹配的 DynamicFrame，应用 FindIncrementalMatches 转换。必需。
- incrementalFrame – 递增 DynamicFrame，应用 FindIncrementalMatches 转换以匹配 existingFrame。必需。
- transformId – 与 FindIncrementalMatches 转换关联的唯一 ID，在 DynamicFrames 中的记录上应用。必需。
- transformation_ctx – 用于标识状态信息的唯一字符串。可选。
- info – 与转换中的错误关联的字符串。可选。

- stageThreshold – 在转换出错之前可能在其中发生的最大错误数。可选。默认值为 0。
- totalThreshold – 在处理出错之前可能全面发生的最大错误数。可选。默认值为 0。
- enforcedMatches – 用于强制实施匹配的 DynamicFrame。可选。默认值为 None (无)。
- computeMatchConfidenceScores – 布尔值，指示是否为每组匹配记录计算置信度得分。可选。默认值为 false。

返回新 DynamicFrame，并为每组匹配记录分配唯一标识符。

FindMatches 类

标识输入 DynamicFrame 中的匹配记录，并创建一个新的 DynamicFrame，其中包含分配给每组匹配记录的唯一标识符。

导入：

```
from awsglueml.transforms import FindMatches
```

方法

- Apply

apply(frame, transformId, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0, enforcedMatches = none, computeMatchConfidenceScores = 0)

标识输入 DynamicFrame 中的匹配记录，并创建一个新的 DynamicFrame，其中包含分配给每组匹配记录的唯一标识符。

- frame – DynamicFrame，应用 FindMatches 转换。必需。
- transformId – 与 FindMatches 转换关联的唯一 ID，在 DynamicFrame 中的记录上应用。必需。
- transformation_ctx – 用于标识状态信息的唯一字符串。可选。
- info – 与转换中的错误关联的字符串。可选。
- stageThreshold – 在转换出错之前可能在其中发生的最大错误数。可选。默认值为 0。
- totalThreshold – 在处理出错之前可能全面发生的最大错误数。可选。默认值为 0。
- enforcedMatches – 用于强制实施匹配的 DynamicFrame。可选。默认值为 None (无)。
- computeMatchConfidenceScores – 布尔值，指示是否为每组匹配记录计算置信度得分。可选。默认值为 false。

返回新 DynamicFrame，并为每组匹配记录分配唯一标识符。

FlatMap 类

对集合中的每个 DynamicFrame 应用转换。结果不会拼凑成单个 DynamicFrame，而是作为一个集合保存。

FlatMap 的示例

以下示例片段展示了当应用于 FlatMap 时，如何对动态帧集合使用 ResolveChoice 转换。用于输入的数据是位于占位符 Amazon S3 地址 s3://bucket/path-for-data/sample.json 的 JSON，包含以下数据。

JSON 数据示例

```
[{
    "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 将 ResolveChoice 应用于 DynamicFrameCollection，将显示输出。

```
#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**
>
> 调用 `FlatMap.apply` 时，`frame_name` 参数必须为 `"frame"`。目前不接受其他值。

示例输出

```
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__
- Apply
- 名称
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

frame_name __call__ (dfc，BaseTransform transformation_ctx = “”，**，base_kwargs)，

向集合中的每个 DynamicFrame 应用转换，并展平结果。

- dfc – 要展平的 DynamicFrameCollection (必需)。

- BaseTransform – 派生自 GlueTransform、要应用于集合的每个成员的转换 (必需)。
- frame_name – 要将集合元素传递到的参数名称 (必需)。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- base_kwargs – 要传递到基本转换的参数 (必需)。

通过将返回新 DynamicFrameCollection 创建的每个转换 DynamicFrame 在源 DynamicFrameCollection。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

Join 类

对两个 DynamicFrames 执行等式联接。

示例

我们建议您使用 DynamicFrame.join( ) 方法联接 DynamicFrames。要查看代码示例，请参阅 示例：使用联接来组合 DynamicFrames。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

__call__(frame1, frame2, keys1, keys2, transformation_ctx = "")

对两个 DynamicFrames 执行等式联接。

- frame1 – 要联接的第一个 DynamicFrame (必需)。
- frame2 – 要联接的第二个 DynamicFrame (必需)。
- keys1 – 第一个帧要联接的键 (必需)。
- keys2 – 第二个帧要联接的键 (必需)。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。

返回新的 DynamicFrame，其通过联接两个 DynamicFrames 创建。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

Map 类

通过将函数应用于输入 DynamicFrame 中的所有记录来生成新的 DynamicFrame。

示例

我们建议您使用 DynamicFrame.map() 方法将函数应用于 DynamicFrame 中所有记录。要查看代码示例，请参阅 示例：使用映射将函数应用于 DynamicFrame 中的每个记录。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

__call__(frame, f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

返回一个新的 DynamicFrame，它源自将指定函数应用到原始 DynamicFrame 中的所有 DynamicRecords。

- frame – 要对其应用映射函数的原始 DynamicFrame（必需）。
- f – 要应用到 DynamicRecords 中的所有 DynamicFrame 的函数。该函数必须将 DynamicRecord 作为实际参数并返回一个由映射生成的新的 DynamicRecord（必需）。

  一个 DynamicRecord 表示 DynamicFrame 中的一条逻辑记录。它类似于 Apache Spark DataFrame 中的一行，但其具有自描述性，可用于不符合固定架构的数据。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与转换中的错误关联的字符串 (可选)。
- stageThreshold – 在转换出错之前可能在其中发生的最大错误数（可选）。默认值为 0。
- totalThreshold – 在处理出错之前可能全面发生的最大错误数（可选）。默认值为 0。

返回一个新的 DynamicFrame，它源自将指定函数应用到原始 DynamicFrame 中的所有 DynamicRecords。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

MapToCollection 类

对指定 DynamicFrameCollection 中的每个 DynamicFrame 应用转换。

方法

- __call__
- Apply
- 名称
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

frame_name __call__ (dfc，BaseTransform transformation_ctx = “”，**，base_kwargs)，

对指定 DynamicFrameCollection 中的每个 DynamicFrame 应用转换函数。

- dfc – 要对其应用转换函数的原始 DynamicFrameCollection (必需)。
- callable – 要应用于集合的每个成员的可调用转换函数 (必需)。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。

通过将返回新 DynamicFrameCollection 创建的每个转换 DynamicFrame 在源 DynamicFrameCollection。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

Relationalize 类

在 DynamicFrame 中展平嵌套架构，并从展平帧透视数组列。

示例

我们建议您使用 DynamicFrame.relationalize() 方法关系化 DynamicFrame。要查看代码示例，请参阅 示例：使用 relatialize 在 DynamicFrame 中展平嵌套架构。

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)

关系化 DynamicFrame 并生成一个帧列表，这些帧是通过取消嵌套列的嵌套并透视数组列生成的。可使用在取消嵌套阶段生成的联接键将透视数组列联接到根表。

- frame – 要关系化的 DynamicFrame (必需)。
- staging_path – 要将 CSV 格式的透视表分区存储到的路径（可选）。从该路径读取透视表。
- name – 根表的名称 (可选)。
- options – 可选参数的词典。目前未使用。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与转换中的错误关联的字符串 (可选)。
- stageThreshold – 在转换出错之前可能在其中发生的最大错误数（可选）。默认值为 0。
- totalThreshold – 在处理出错之前可能全面发生的最大错误数（可选）。默认值为 0。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

RenameField 类

在 DynamicFrame 内重命名一个节点。

示例

我们建议您使用 DynamicFrame.rename_field( ) 方法在 DynamicFrame 中重命名字段。要查看代码示例，请参阅 示例：使用 rename_field 重命名 DynamicFrame 中的字段。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, old_name, new_name, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

在 DynamicFrame 内重命名一个节点。

- frame – 在其中重命名节点的 DynamicFrame (必需)。
- old_name – 要重命名的节点的完整路径（必需）。

  如果旧名称中包含点，则 RenameField 将不起作用，除非使用反引号（``）将其引起来。例如，要将 this.old.name 替换为 thisNewName，您需要调用 RenameField，如下所示：

  ```
  newDyF = RenameField(oldDyF, "`this.old.name`", "thisNewName")
  ```

- new_name – 新名称，其中包括完整路径（必需）。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。

- info – 与转换中的错误关联的字符串 (可选)。
- stageThreshold – 在转换出错之前可能在其中发生的最大错误数（可选）。默认值为 0。
- totalThreshold – 在处理出错之前可能全面发生的最大错误数（可选）。默认值为 0。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

ResolveChoice 类

解析 DynamicFrame 内的选择类型。

示例

我们建议您使用 DynamicFrame.resolveChoice() 方法来处理 DynamicFrame 中包含多种类型的字段。要查看代码示例，请参阅 示例：使用 resolveChoice 处理包含多种类型的列。

方法

- __call__

- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, specs = none, choice = "", transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

提供有关解析 DynamicFrame 内歧义类型的信息。它返回生成的 DynamicFrame。

- frame – 在其中解析选择类型的 DynamicFrame (必需)。
- specs – 要解析的特定歧义列表，每个歧义均采用元组形式：(path, action)。path 值标识特定歧义元素，action 值标识相应解析。

  只能使用 spec 和 choice 参数之一。如果 spec 参数不为 None，则 choice 参数必须为空字符串。反过来，如果 choice 不为空字符串，则 spec 参数必须为 None。如果两个参数均未提供，则 AWS Glue 尝试解析架构并用它来解析歧义。

  specs 元组的 action 部分可以指定以下解析策略之一：

  - cast – 允许指定一种要强制转换到的类型（例如，cast:int）。
  - make_cols – 通过展平数据来解析潜在的歧义。例如，如果 columnA 是 int 或 string，则解析就是在结果 DynamicFrame 中生成名为 columnA_int 和 columnA_string 的两个列。
  - make_struct – 通过用一种结构表示数据来解析潜在的歧义。例如，如果某个列中的数据是 int 或 string，则使用 make_struct 操作会在结果 DynamicFrame 中生成结构列，而每个结构都同时包含 int 和 string。
  - project – 通过仅在结果 DynamicFrame 中保留指定类型的值来解析潜在的歧义。例如，如果 ChoiceType 列中的数据可以是 int 或 string，则指定 project:string 操作会从结果 DynamicFrame 中删除非 string 类型的值。

  如果 path 识别到数组，则在数组名称后放置一个空的方括号可避免歧义。例如，假设您正在使用结构如下的数据：

```
"myList": [
```

```
  { "price": 100.00 },
  { "price": "$100.00" }
]
```

可以通过将 path 设置为 "myList[].price"、将 action 设置为 "cast:double" 来选择价格的数字而非字符串版本。

- choice – specs 参数为 None 时的默认解析操作。如果 specs 参数不为 None，则只能将此项设置为空字符串，不能设置为其他值。

  除了先前描述的 specs 操作外，此参数还支持以下操作：

  - MATCH_CATALOG – 尝试将每个 ChoiceType 转换为指定数据目录表中的对应类型。
- database – 用于 MATCH_CATALOG 选择的 AWS Glue Data Catalog 数据库（对于 MATCH_CATALOG 是必需的）。
- table_name – 用于 MATCH_CATALOG 操作的 AWS Glue Data Catalog 表名称（对于 MATCH_CATALOG 是必需的）。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与转换中的错误关联的字符串 (可选)。
- stageThreshold – 在转换出错之前可能在其中发生的最大错误数（可选）。默认值为 0。
- totalThreshold – 在处理出错之前可能全面发生的最大错误数（可选）。默认值为 0。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

SelectFields 类

SelectFields 类在现有的 DynamicFrame 中创建新 DynamicFrame，并且只保留您指定的字段。SelectFields 提供与 SQL SELECT 语句类似的功能。

示例

我们建议您使用 DynamicFrame.select_fields() 方法从 DynamicFrame 中选择字段。要查看代码示例，请参阅 示例：使用 select_fields 以利用选定字段创建新 DynamicFrame。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 描述

__call__(frame, paths, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

获取 DynamicFrame 中的字段 (节点)。

- frame – 要在其中选择字段的 DynamicFrame（必需）。
- paths – 要选择的字段的完整路径的列表 (必需)。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与转换中的错误关联的字符串（可选）。

- stageThreshold – 在转换出错之前可能在其中发生的最大错误数（可选）。默认值为 0。
- totalThreshold – 在处理出错之前可能全面发生的最大错误数（可选）。默认值为 0。

返回仅包含指定字段的新 DynamicFrame。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

SelectFromCollection 类

选择 DynamicFrameCollection 中的一个 DynamicFrame。

示例

此示例使用 SelectFromCollection 从 DynamicFrameCollection 中选择 DynamicFrame。

示例数据集

该示例从名为 split_rows_collection 的 DynamicFrameCollection 中选择了两个 DynamicFrames。以下是 split_rows_collection 中的键列表。

```
dict_keys(['high', 'low'])
```

示例代码

```
# 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()
```

输出

```
+---+-----+-----------------------+-------------------------+
| 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 = "")

获取 DynamicFrameCollection 中的一个 DynamicFrame。

- dfc – 应从中选择 DynamicFrame 的 DynamicFrameCollection (必需) 。
- key – 要选择的 DynamicFrame 的密钥 (必需)。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

Simplify_ddb_json 类

简化 DynamicFrame 中的嵌套列，其具体位于 DynamoDB JSON 结构中，并返回一个新的简化 DynamicFrame。

示例

我们建议使用 DynamicFrame.simplify_ddb_json() 方法来简化 DynamicFrame 中的嵌套列，该列特别位于 DynamoDB JSON 结构中。要查看代码示例，请参阅 示例：使用 simplify_ddb_json 调用 DynamoDB JSON simplify 命令。

Spigot 类

将示例记录写入指定的目标，以帮助您验证 AWS Glue 作业执行的转换。

示例

我们建议您使用 DynamicFrame.spigot() 方法将 DynamicFrame 中的记录子集写入指定的目标。要查看代码示例，请参阅 示例：使用 spigot 将 DynamicFrame 中的示例字段写入 Amazon S3。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, path, options, transformation_ctx = "")

在转换期间将示例记录写入指定的目标。

- frame – 要执行 spigot 操作的 DynamicFrame (必需)。
- path – 要写入的目标的路径（必需）。

- options – 指定选项的 JSON 键值对（可选）。"topk" 选项指定应写入的第一个 k 记录。"prob" 选项指定选择任何给定记录的可能性（以十进制数字形式表示）。您可以在选择要写入的记录时使用它。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply

name(cls)

继承自 GlueTransform name

describeArgs(cls)

继承自 GlueTransform describeArgs

describeReturn(cls)

继承自 GlueTransform describeReturn

describeTransform(cls)

继承自 GlueTransform describeTransform

describeErrors(cls)

继承自 GlueTransform describeErrors

describe(cls)

继承自 GlueTransform describe

SplitFields 类

按指定字段将一个 DynamicFrame 拆分成两个新项。

示例

我们建议您使用 DynamicFrame.split_fields() 方法拆分 DynamicFrame 中的字段。要查看代码示例，请参阅 示例：使用 split_fields 将选定字段拆分为单独的 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)

将 DynamicFrame 中的一个或多个字段拆分成新的 DynamicFrame 并创建另一个包含保留字段的新 DynamicFrame。

- frame – 要拆分到新 DynamicFrame 中的源 DynamicFrame (必需)。
- paths – 要拆分的字段的完整路径的列表 (必需)。
- name1 – 要为将包含要拆分的字段的 DynamicFrame 分配的名称（可选）。如果没有提供名称，则会使用源框架的名称并附加“1”。
- name2 – 要为将包含在指定字段拆分后仍保留的字段的 DynamicFrame 分配的名称（可选）。如果没有提供名称，则会使用源框架的名称并附加“2”。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与转换中的错误关联的字符串 (可选)。
- stageThreshold – 在转换出错之前可能在其中发生的最大错误数（可选）。默认值为 0。
- totalThreshold – 在处理出错之前可能全面发生的最大错误数（可选）。默认值为 0。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

SplitRows 类

创建一个包含两个 DynamicFrames 的 DynamicFrameCollection。一个 DynamicFrame 仅包含要拆分的指定行，另一个包含所有剩余行。

示例

我们建议您使用 DynamicFrame.split_rows() 方法拆分 DynamicFrame 中的行。要查看代码示例，请参阅 示例：使用 split_rows 拆分 DynamicFrame 中的行。

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)

将 DynamicFrame 中的一个或多个行拆分到新的 DynamicFrame 中。

- frame – 要拆分到新 DynamicFrame 中的源 DynamicFrame (必需)。
- comparison_dict – 一个词典，其中的键是到一个列的完整路径，值是另一个词典，用于将比较器映射到与该列值所比较的值。例如，{"age": {">": 10, "<": 20}} 拆分“age”值介于 10 和 20 之间的行，不包括“age”在范围之外的行 (必需)。
- name1 – 要为将包含要拆分的行的 DynamicFrame 分配的名称 (可选)。
- name2 – 要为将包含在指定行拆分后仍保留的行的 DynamicFrame 分配的名称 (可选)。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与转换中的错误关联的字符串 (可选)。
- stageThreshold – 在转换出错之前可能在其中发生的最大错误数（可选）。默认值为 0。
- totalThreshold – 在处理出错之前可能全面发生的最大错误数（可选）。默认值为 0。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

Unbox 类

取消装箱（重新格式化）DynamicFrame 中的字符串字段。

示例

我们建议您使用 DynamicFrame.unbox() 方法取消装箱 DynamicFrame 中的字段。要查看代码示例，请参阅 示例：使用 unbox 将字符串字段拆开为结构。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, path, format, transformation_ctx = "", info="", stageThreshold=0, totalThreshold=0, **options)

拆开 DynamicFrame 中的字符串字段。

- frame – 要在其中拆开字段的 DynamicFrame（必需）。
- path – 要拆开的 StringNode 的完整路径 (必需)。
- format – 格式规范（可选）。这用于 Amazon S3 或支持多种格式的 AWS Glue 连接。相关受支持的格式，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。

- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与转换中的错误关联的字符串 (可选)。
- stageThreshold – 在转换出错之前可能在其中发生的最大错误数（可选）。默认值为 0。
- totalThreshold – 在处理出错之前可能全面发生的最大错误数（可选）。默认值为 0。
- separator – 分隔符令牌 (可选)。
- escaper – 转义令牌 (可选)。
- skipFirst – 如果应该跳过第一行数据，则为 True，如果不应跳过，则为 False (可选)。
- withSchema – 一个字符串，其中包含要拆开的数据的架构（可选）。应始终使用 StructType.json 创建它。
- withHeader – 如果要解包的数据包含标头，则为 True，否则为 False (可选)。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

UnnestFrame 类

取消嵌套 DynamicFrame，将嵌套的对象展平到顶级元素，并为数组对象生成联接键。

示例

我们建议您使用 DynamicFrame.unnest() 方法展平 DynamicFrame 中的嵌套结构。要查看代码示例，请参阅 示例：使用 unnest 将嵌套字段转换为顶级字段。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__(frame, transformation_ctx = "", info="", stageThreshold=0, totalThreshold=0)

取消嵌套 DynamicFrame，将嵌套的对象展平到顶级元素，并为数组对象生成联接键。

- frame – 要取消嵌套的 DynamicFrame (必需)。
- transformation_ctx – 用于标识状态信息的唯一字符串 (可选)。
- info – 与转换中的错误关联的字符串 (可选)。
- stageThreshold – 在转换出错之前可能在其中发生的最大错误数（可选）。默认值为 0。
- totalThreshold – 在处理出错之前可能全面发生的最大错误数（可选）。默认值为 0。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

FlagDuplicatesInColumn 班级

FlagDuplicatesInColumn转换会返回一个新列，每行都有指定值，用于指示该行的源列中的值是否与源列前一行中的值匹配。找到匹配项后，它们会被标记为重复项。初始出现的次数不会被标记，因为它与前面的行不匹配。

示例

```
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
```

输出

FlagDuplicatesInColumn转换会在 “df_output” 中添加一个新列 “flag_col”。 DataFrame此列将包含一个字符串值，表示相应行在 “city” 列中是否有重复的值。如果某行具有重复的 “城市” 值，则 “flag_col” 将包含 “true_string” 值 “True”。如果一行具有唯一的 “城市” 值，则 “flag_col” 将包含 “false_string” 值 “False”。

生成的 “df_output” DataFrame 将包含原始 “datasource1” 中的所有列，以及表示重复的 “城市” 值的额外 “flag_col” DataFrame 列。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__ (spark_context、data_frame、source_column、target_column、true_string=default_true_string=def

FlagDuplicatesInColumn转换会返回一个新列，每行都有指定值，用于指示该行的源列中的值是否与源列前一行中的值匹配。找到匹配项后，它们会被标记为重复项。初始出现的次数不会被标记，因为它与前面的行不匹配。

- source_column-源列的名称。
- target_column-目标列的名称。
- true_string— 当源列的值与该列中较早的值重复时，要在目标列中插入的字符串。

- `false_string`— 当源列的值与目标列中较早的值不同时，要在目标列中插入的字符串。

apply(cls, *args, **kwargs)

继承自 `GlueTransform` apply。

name(cls)

继承自 `GlueTransform` name。

describeArgs(cls)

继承自 `GlueTransform` describeArgs。

describeReturn(cls)

继承自 `GlueTransform` describeReturn。

describeTransform(cls)

继承自 `GlueTransform` describeTransform。

describeErrors(cls)

继承自 `GlueTransform` describeErrors。

describe(cls)

继承自 `GlueTransform` describe。

FormatPhoneNumber 班级

`FormatPhoneNumber`转换会返回一列，其中电话号码字符串将转换为格式化值。

示例

```
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
```

输出

输出将是：

```
```
+---------------+
| phone|
+---------------+
|(408) 341-5669|
|(408) 341-5669|
+---------------+
```
```

FormatPhoneNumber转换将 “source_column” 作为 “手机”，将 “default_region” 视为 “US”。

转换成功地将两个电话号码（无论其初始格式如何）格式化为标准的美国格式 `(408) 341-5669`。

方法

- __call__
- apply
- name
- describeArgs

- describeReturn
- describeTransform
- describeErrors
- describe

__call__（spark_context、data_frame、source_column、phone_number_format=None、default_region_colu

FormatPhoneNumber转换会返回一列，其中电话号码字符串将转换为格式化值。

- source_column – 现有列的名称。
- phone_number_format— 将电话号码转换为的格式。如果未指定格式E.164，则默认为国际公认的标准电话号码格式。有效值包括：
  - E164（省略 E 之后的句点）
- default_region— 由两个或三个大写字母组成的有效区域代码，当电话号码本身没有国家代码时，它指定电话号码所在的地区。最多defaultRegionColumn只能提供其中一个defaultRegion或。
- default_region_column— 高级数据类型的列的名称Country。当电话号码本身不存在国家/地区代码时，指定列中的区域代码用于确定电话号码的国家/地区代码。最多defaultRegionColumn只能提供其中一个defaultRegion或。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

FormatCase 班级

FormatCase转换会将列中的每个字符串更改为指定的大小写类型。

示例

```
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
```

输出

FormatCase转换会根据 `case_type= “Lower” 参数将 “城市” 列中的值转换为小写字母。生成的 “df_output” DataFrame 将包含原始 “datasource1” 中的所有列，但 “city” DataFrame 列的值为小写。

方法

- __call__

- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__ (火花上下文、data_frame、source_column、case_type)

FormatCase转换会将列中的每个字符串更改为指定的大小写类型。

- source_column – 现有列的名称。
- case_type— 支持的案例类型是CAPITAL、LOWER、UPPER、SENTENCE。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

FillWithMode 班级

FillWithMode转换会根据您指定的电话号码格式设置列的格式。您也可以指定平局逻辑，其中一些值是相同的。例如，考虑以下值：1 2 2 3 3 4

一个 modeTypeFillWithMode，其中包含返回 2 作为模式值MINIMUM的原因。如果 modeType 为MAXIMUM，则模式为 3。对于AVERAGE，模式为 2.5。

示例

```
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
```

输出

给定代码的输出将是：

```
```
+---------------+---------------+
|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
+---------------+---------------+
```
```

“awsglue.data_quality” 模块的FillWithMode转换应用于 “input_df”。 DataFrame它将列中的 “空” 值替换为该source_column_1列中非空值中的最大值（ `mode_type= “Maximum” ` ）。

在本例中，该source_column_1列中的最大值为 `1055.123`。因此，输出 “df_output” 中的 “空” 值将替换source_column_1为 “1055.123”。 DataFrame

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（火花上下文、数据框架、源列、模式类型）

FillWithMode转换格式化列中字符串的大小写。

- source_column – 现有列的名称。
- mode_type— 如何解析数据中的平局值。此值必须是MINIMUM、NONEAVERAGE、或之一MAXIMUM。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

FlagDuplicateRows 班级

FlagDuplicateRows转换会返回一个新列，每行都有指定值，用于指示该行是否与数据集中的前一行完全匹配。找到匹配项后，它们会被标记为重复项。初始出现的次数不会被标记，因为它与前面的行不匹配。

示例

```
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
```

输出

输出将是 PySpark DataFrame 带有附加列的`flag_row`，该列根据该`source_column_1`列指示某行是否重复。生成的 “df_output” DataFrame 将包含以下行：

```
```
+---------------+---------------+--------+
|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|
+---------------+---------------+--------+
```

该flag_row列指示某行是否重复。“true_string” 设置为 “True”，“false_string” 设置为 “False”。“target_index` 设置为 1，这意味着该flag_row列将被插入到输出的第二个位置（索引 1）。DataFrame

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（spark_context、data_frame、target_column、true_string=default_true_string、target_index=none

FlagDuplicateRows转换会返回一个新列，每行都有指定值，用于指示该行是否与数据集中的前一行完全匹配。找到匹配项后，它们会被标记为重复项。初始出现的次数不会被标记，因为它与前面的行不匹配。

- true_string— 如果该行与前一行匹配，则要插入的值。
- false_string— 如果行是唯一的，则要插入的值。
- target_column— 插入到数据集中的新列的名称。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

RemoveDuplicates 班级

如果在选定的源列中遇到重复的值，则RemoveDuplicates转换会删除整行。

示例

```
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
```

输出

输出将是 a， PySpark DataFrame 并根据该source_column_1列删除了重复项。生成的 “df_output” DataFrame 将包含以下行：

```
```
+---------------+---------------+
|source_column_1|source_column_2|
+---------------+---------------+
105.111	13.12
13.12	13.12
null	13.12
+---------------+---------------+
```
```

请注意，source_column_1值为 “13.12” 和 “null” 的行在输出中仅出现一次 DataFrame，因为已根据该列删除了重复项。source_column_1

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（火花上下文、数据框架、源列）

如果在选定的源列中遇到重复的值，则RemoveDuplicates转换会删除整行。

- source_column – 现有列的名称。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

MonthName 班级

MonthName转换使用表示日期的字符串创建一个包含月份名称的新列。

示例

```
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
```

输出

输出将是：

```
+------------+------------+
| 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转换将 “source_column” 视为 “column_1”，将 “target_column” 视为 “target_column”。它尝试从 “column_1” 列中的日期/时间字符串中提取月份名称，并将其放在 “target_column” 列中。如果日期/时间字符串的格式无法识别或无法解析，则 “target_column” 的值将设置为 “null”。

转换成功地从各种日期/时间格式中提取月份名称，例如 “20-12-2018”、“12/20/2018”、“02/02/2009”、“2013-02-21 06:35:45.658 505” 和 “2009 年 8 月 2 日”。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors

- describe

__call__ (spark_context、data_frame、target_column、source_column=None、valu

MonthName转换使用表示日期的字符串创建一个包含月份名称的新列。

- source_column – 现有列的名称。
- value— 要评估的字符串...
- target_column-新创建的列的名称。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

IsEven 班级

IsEven转换在新列中返回一个布尔值，该值指示源列或值是否为偶数。如果源列或值为十进制，则结果为 false。

示例

```
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
```

输出

输出将是：

```
```
+------------+------------+
|source_column|target_column|
+------------+------------+
5	Not even
0	Even
-1	Not even
2	Even
null	null
```
```

```
+------------+------------+
```
```

IsEven转换将 “源列” 作为 “源列”，将 “目标列” 视为 “目标列”。它检查 “source_column” 中的值是否为偶数。如果该值为偶数，则将 “target_column” 的值设置为 `true_string` “Even”。如果该值为奇数，则将 “target_column” 的值设置为 `false_string` “非偶数”。如果 “source_column” 的值为 “null”，则 “target_column” 的值将设置为 “null”。

转换可以正确识别偶数（0 和 2），并将 “target_column” 的值设置为 “偶数”。对于奇数（5 和 -1），它将 “target_column” 的值设置为 “非偶数”。对于 “source_column” 中的 “null” 值，“target_column” 的值设置为 “null”。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（spark_context、data_frame、target_column=none、true_string=default_true_string=default_false_s

IsEven转换在新列中返回一个布尔值，该值指示源列或值是否为偶数。如果源列或值为十进制，则结果为 false。

- source_column – 现有列的名称。
- target_column-要创建的新列的名称。
- true_string— 表示该值是否为偶数的字符串。
- false_string— 表示该值是否为偶数的字符串。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。
```

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

CryptographicHash 班级

CryptographicHash转换将算法应用于列中的哈希值。

示例

```
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
```

输出

输出将是：

```
+---+------------+--------------------+--------------------+
| 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... |
+---+------------+--------------------+--------------------+
```

转换使用指定的算法和密钥计算 “id” 和 “phone” 列中值的加密哈希值，并以 Base64 格式对哈希进行编码。生成的 “df_output” DataFrame 包含原始 “input_df” 中的所有列，以及带有计算哈希值的额外 “id_has DataFrame hed” 和 “phone_hashed” 列。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__ (spark_context、data_frame、source_columns、secret_if、secret_format=none、entity_type_filter=

CryptographicHash转换将算法应用于列中的哈希值。

- source_columns-现有列的数组。
- secret_id— Secrets Manager 密钥的 ARN。基于哈希的消息身份验证码 (HMAC) 前缀算法中用于对源列进行哈希处理的密钥。
- secret_version：可选。默认为最新的密钥版本。
- entity_type_filter— 可选的实体类型数组。可用于仅加密自由文本列中检测到的 PII。
- create_secret_if_missing— 可选布尔值。如果为 true，将尝试代表呼叫者创建密钥。
- algorithm— 用于对数据进行哈希处理的算法。有效的枚举值：MD5、SHA1、SHA256、SHA512、HMAC_MD5、HMAC_SHA1、HMAC_SHA256、HMAC_SHA512、HI

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

解密类

Decrypt变换会在 Glu AWS e 内部解密。您也可以使用 AWS 加密 SDK 在 Glue 之外 AWS 解密您的数据。如果提供的 KMS 密钥 ARN 与用于加密列的密钥不匹配，则解密操作将失败。

示例

```
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
```

输出

输出将是 PySpark DataFrame 带有原始 “id” 列和解密后的 “phone” 列的：

```
```
+---+------------+
| id| phone|
+---+------------+
1	1234560000
2	1234560001
3	1234560002
4	1234560003
5	1234560004
6	1234560005
7	1234560006
8	1234560007
9	1234560008
```
```

```
| 10| 1234560009|
+---+------------+
```

Encrypt转换将 “source_columns” 作为 `["电话"]` ，将 `kms_key_arn` 作为 `$ {KMS} `环境变量的值。转换使用指定的 KMS 密钥对 “电话” 列中的值进行加密。然后，加密后的 DataFrame “df_encrypt” 从 “awsglue.pii” 模块传递给Decrypt变换。它将 `source_columns` 作为 `["phone"]` ，将 `kms_key_arn` 作为 `$ {KMS} `环境变量的值。转换使用相同的 KMS 密钥解密 “手机” 列中的加密值。生成的 “df_decrypt” DataFrame 包含原始的 “id” 列和解密后的 “电话” 列。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__ (Spark_context、data_frame、source_columns、kms_key_arn

Decrypt变换会在 Glu AWS e 内部解密。您也可以使用 AWS 加密 SDK 在 Glue 之外 AWS 解密您的数据。如果提供的 KMS 密钥 ARN 与用于加密列的密钥不匹配，则解密操作将失败。

- source_columns-现有列的数组。
- kms_key_arn— 用于解密源列的 AWS 密钥管理服务密钥的密钥 ARN。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

加密类

Encrypt转换使用密 AWS 钥管理服务密钥对源列进行加密。该Encrypt转换最多可以加密每个信元 128 MiB。它将在解密时尝试保留格式。要保留数据类型，数据类型元数据必须序列化为小于 1KB。否则，必须将该preserve_data_type参数设置为 false。数据类型元数据将以纯文本形式存储在加密环境中。

示例

```
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
```

输出

输出将是 PySpark DataFrame 带有原始 “id” 列的，另外一列包含 “phone” 列的加密值。

```
+---+------------+-------------------------+
| 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转换将 “source_columns” 作为 `["电话"]` ，将 `kms_key_arn` 作为 `$ {KMS} `环境变量的值。转换使用指定的 KMS 密钥对 “电话” 列中的值进行加密。生成的 “df_encrypted” DataFrame 包含原始的 “id” 列、原始的 “电话” 列，以及一个名为 “phone_encrypted” 的附加列，其中包含 “电话” 列的加密值。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__ (spark_context、data_frame、source_columns、kms_key_arn、entity_type_filter=none、preserve_

Encrypt转换使用密 AWS 钥管理服务密钥对源列进行加密。

- source_columns-现有列的数组。
- kms_key_arn— 用于加密源列的 AWS 密钥管理服务密钥的密钥 ARN。
- entity_type_filter— 可选的实体类型数组。可用于仅加密自由文本列中检测到的 PII。
- preserve_data_type— 可选布尔值。默认值为 true。如果为 false，则不会存储数据类型。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 GlueTransform describeTransform。

describeErrors(cls)

继承自 GlueTransform describeErrors。

describe(cls)

继承自 GlueTransform describe。

IntToIp 班级

IntToIp转换将源列或其他值的整数值转换为目标列中相应的 IPv4 值，并在新列中返回结果。

示例

```
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
```

输出

输出将是：

```
+---------------+---------------+
|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 |
+---------------+---------------+
```

`IntToIp.apply`转换将 “source_column_int” 作为 “source_column_int”，将 “target_column” 作为 “target_column_int” 列，然后将 “source_column_int” 列中的整数值转换为相应的 IPv4 地址表示形式，并将结果存储在 “target_column” 列中。

对于 IPv4 地址范围（0 到 4294967295）内的有效整数值，转换成功地将其转换为其 IPv4 地址表示形式（例如 192.0.0.1、0.0.0.0、10.0.0.10、255.255.255.255）。

对于超出有效范围的整数值（例如 4294967296、-1），“target_column” 值设置为 “null”。对于 “source_column_int” 列中的 “空” 值，“target_column” 值也设置为 “null”。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（spark_context、data_frame、target_column、source_column=None、valu

IntToIp转换将源列或其他值的整数值转换为目标列中相应的 IPv4 值，并在新列中返回结果。

- sourceColumn – 现有列的名称。
- value— 要计算的字符串。
- targetColumn-要创建的新列的名称。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 `GlueTransform` describeTransform。

describeErrors(cls)

继承自 `GlueTransform` describeErrors。

describe(cls)

继承自 `GlueTransform` describe。

IpToInt 班级

`IpToInt`转换将源列或其他值的 Internet 协议版本 4 (IPv4) 值转换为目标列中相应的整数值，并在新列中返回结果。

示例

对于 AWS Glue 4.0 及更高版本，使用创建或更新任务参数 `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()
```

输出

输出将是：

```
```
+----------------+---------------+
|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
+----------------+---------------+
```
```

`IpToInt`转换将 “source_column_ip” 作为 “source_column_ip”，将 “target_column” 作为 “target_column_ip” 列中的有效 IPv4 地址字符串转换为相应的 32 位整数表示形式，并将结果存储在 “target_column” 列中。

对于有效的 IPv4 地址字符串（例如 “192.0.0.1”、“10.10.10.10”、“1.2.3.4”），转换成功地将其转换为整数表示形式（例如 3221225473、168427722、16909060）。对于不是有效 IPv4 地址的字符串（例

如 URL、非 IP 字符串（例如 “abc”）、无效 IP 格式（例如 “abc.abc.abc.abc”），“target_column” 值设置为 “null”。对于 “source_column_ip” 列中的 “空” 值，“target_column” 值也设置为 “null”。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- describe

__call__（spark_context、data_frame、target_column、source_column=None、valu

IpToInt转换将源列或其他值的 Internet 协议版本 4 (IPv4) 值转换为目标列中相应的整数值，并在新列中返回结果。

- sourceColumn – 现有列的名称。
- value— 要计算的字符串。
- targetColumn-要创建的新列的名称。

apply(cls, *args, **kwargs)

继承自 GlueTransform apply。

name(cls)

继承自 GlueTransform name。

describeArgs(cls)

继承自 GlueTransform describeArgs。

describeReturn(cls)

继承自 GlueTransform describeReturn。

describeTransform(cls)

继承自 `GlueTransform` describeTransform。

describeErrors(cls)

继承自 `GlueTransform` describeErrors。

describe(cls)

继承自 `GlueTransform` describe。

数据集成转型

对于 AWS Glue 4.0 及更高版本，使用创建或更新任务参数`key: --enable-glue-di-transforms, value: true`。

作业脚本示例：

```
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
```

使用笔记本的示例会话

```
%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
```

使用示例会话 AWS CLI

```
aws glue create-session --default-arguments "--enable-glue-di-transforms=true"
```

DI 变换：

- FlagDuplicatesInColumn 班级
- FormatPhoneNumber 班级
- FormatCase 班级
- FillWithMode 班级
- FlagDuplicateRows 班级
- RemoveDuplicates 班级
- MonthName 班级
- IsEven 班级
- CryptographicHash 班级
- 解密类
- 加密类
- IntToIp 班级
- IpToInt 班级

Maven：将插件与你的 Spark 应用程序捆绑在一起

在本地开发 Spark 应用程序`pom.xml`时，您可以在 Maven 中添加插件依赖关系，从而将转换依赖项与 Spark 应用程序和 Spark 发行版（3.3 版）捆绑在一起。

```
<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>
```

或者，你可以直接从 AWS Glue Maven 工件下载二进制文件，并将它们包含在你的 Spark 应用程序中，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 中编写 AWS Glue ETL 脚本

您可以在 GitHub 网站上的 AWS Glue 示例存储库中找到 AWS Glue 的 Scala 代码示例和实用程序。

AWS Glue 支持使用 PySpark Scala 方言的扩展来编写提取、转换和加载（ETL）任务脚本。下面几节介绍如何在 ETL 脚本中使用 AWS Glue Scala 库和 AWS Glue API，并提供了用于库的参考文档。

目录

使用 Scala 编写 AWS Glue ETL 脚本

您可以使用 AWS Glue 控制台自动生成 Scala 提取、转换和加载 (ETL) 程序，并根据需要对其进行修改，然后将其分配给作业。或者，您可以从头编写自己的程序。有关更多信息，请参阅在中配置 Spark 作业的作业属性 AWS Glue。然后，AWS Glue 会在服务器上编译您的 Scala 程序，再运行关联的作业。

为了确保您的程序编译时不会出现错误并可按预期运行，请务必在 REPL（读取-求值-输出-循环）或 Jupyter Notebook 的开发端点中加载程序并在此处进行测试，然后再在作业中运行。由于编译过程在服务器上进行，因此您无法清楚地看到服务器上出现的任何问题。

在 Jupyter Notebook 中的开发端点上测试 Scala ETL 程序

要在 AWS Glue 开发终端节点上测试 Scala 程序，请设置开发终端节点（如 添加开发终端节点 中所述）。

接下来，将其连接到在您的计算机上本地运行或在 Amazon EC2 笔记本服务器上远程运行的 Jupyter Notebook。要安装 Jupyter Notebook 的本地版本，请按照 教程：JupyterLab 中的 Jupyter notebook 中的说明操作。

在您的笔记本上运行 Scala 代码与运行 PySpark 代码之间的唯一差别在于，在笔记本上每个段落的开头必须为：

```
%spark
```

这将防止 Notebook 服务器默认使用 Spark 解释器的 PySpark 风格。

在 Scala REPL 中测试 Scala ETL 程序

您可以使用 AWS Glue Scala REPL 在开发终端节点上测试 Scala 程序。按照教程：使用 SageMaker 笔记本中的说明操作，除了 SSH-to-REPL 命令末尾，将 `-t gluepyspark` 替换为 `-t glue-spark-shell`。这将调用 AWS Glue Scala REPL。

要在完成时关闭 REPL，请键入 `sys.exit`。

Scala 脚本示例 - 流式处理 ETL

Example

以下示例脚本连接到 Amazon Kinesis Data Streams，使用来自数据目录的架构解析数据流，将流联接到 Amazon S3 上的静态数据集，并以 parquet 格式将联接结果输出到 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 库中的 API

AWS Glue 支持使用 PySpark Scala 方言的扩展来编写提取、转换和加载 (ETL) 作业脚本。以下部分描述了 AWS Glue Scala 库中的 API。

com.amazonaws.services.glue

Scala 库中的 com.amazonaws.services.glueAWS Glue 程序包包含以下 API：

- ChoiceOption
- DataSink
- 数据源特性
- DynamicFrame
- DynamicRecord
- GlueContext
- MappingSpec
- ResolveSpec

com.amazonaws.services.glue.ml

AWS Glue Scala 库中的 com.amazonaws.services.glue.ml 程序包包含以下 API：

- FillMissingValues
- FindIncrementalMatches
- FindMatches

com.amazonaws.services.glue.dq

AWS Glue Scala 库中的 com.amazonaws.services.glue.dq 程序包包含以下 API：

- EvaluateDataQuality

com.amazonaws.services.glue.types

Scala 库中的 com.amazonaws.services.glue.typesAWS Glue 程序包包含以下 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

Scala 库中的 com.amazonaws.services.glue.utilAWS Glue 程序包包含以下 API：

- GlueArgParser
- 作业

AWS Glue Scala ChoiceOption API

主题

- ChoiceOption 特性
- ChoiceOption 对象
- Case 类 ChoiceOptionWithResolver
- Case 类 MatchCatalogSchemaChoiceOption

程序包：com.amazonaws.services.glue

ChoiceOption 特性

```
trait ChoiceOption extends Serializable
```

ChoiceOption 对象

ChoiceOption

```
object ChoiceOption
```

用于解决适用于 DynamicFrame 中所有 ChoiceType 节点的选项问题的一般策略。

- `val CAST`
- `val MAKE_COLS`
- `val MAKE_STRUCT`
- `val MATCH_CATALOG`

- `val PROJECT`

Def apply

```
def apply(choice: String): ChoiceOption
```

Case 类 ChoiceOptionWithResolver

```
case class ChoiceOptionWithResolver(name: String, choiceResolver: ChoiceResolver)
 extends ChoiceOption {}
```

Case 类 MatchCatalogSchemaChoiceOption

```
case class MatchCatalogSchemaChoiceOption() extends ChoiceOption {}
```

抽象 DataSink 类

主题

- Def writeDynamicFrame
- Def pyWriteDynamicFrame
- Def writeDataFrame
- Def pyWriteDataFrame
- Def setCatalogInfo
- Def supportsFormat
- Def setFormat
- Def withFormat
- Def setAccumulableSize
- Def getOutputErrorRecordsAccumulable
- Def errorsAsDynamicFrame
- DataSink 对象

程序包：com.amazonaws.services.glue

```
abstract class DataSink
```

DataSource 的写入器模拟。DataSink 封装可将 DynamicFrame 写入到的目标和格式。

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 对象

```
object DataSink
```

Def recordMetrics

```
def recordMetrics( frame : DynamicFrame,
                   ctxt : String
                 ) : DynamicFrame
```

AWS Glue Scala 数据源特性

程序包：com.amazonaws.services.glue

用于生成 DynamicFrame 的高级接口。

```
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

程序包：com.amazonaws.services.glue

目录

- Def repartition
- Def resolveChoice
- Def schema
- Def selectField
- Def selectFields
- Def show
- Def simplifyDDBJson
- Def spigot
- Def splitFields
- Def splitRows
- Def stageErrorsCount
- Def toDF
- Def unbox
- Def unnest
- Def unnestDDBJson
- Def withFrameSchema
- Def withName
- Def withTransformationContext

- DynamicFrame 对象
  - Def apply
  - Def emptyDynamicFrame
  - Def fromPythonRDD
  - Def ignoreErrors
  - Def inlineErrors
  - Def newFrameWithErrors

AWS Glue Scala DynamicFrame 类

程序包：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 是自描述的 DynamicRecord 对象的分布式集合。

DynamicFrame 旨在为 ETL（提取、转换和加载）操作提供灵活的数据模型。它们不需要创建架构，可用于读取和转换具有杂乱或不一致的值和类型的数据。可以按需为需要架构的那些操作计算架构。

DynamicFrame 为数据清理和 ETL 提供了一系列转换。它们还支持转换为 SparkSQL DataFrame 和从其转换以与现有代码和 DataFrame 提供的许多分析操作集成。

跨构造 DynamicFrame 的许多 AWS Glue 转换共享以下参数：

- transformationContext — 此 DynamicFrame 的标识符。transformationContext 用作跨运行保存的作业书签状态的密钥。
- callSite – 为错误报告提供上下文信息。在从 Python 调用时，会自动设置这些值。
- stageThreshold – 在引发异常之前允许的来自此 DynamicFrame 计算的最大错误记录数，不包括以前的 DynamicFrame 中存在的记录。
- totalThreshold – 引发异常之前的最大错误记录总数，包括以前的帧中的记录。

Val errorsCount

```
val errorsCount
```

此 DynamicFrame 中的错误记录数。这包括来自以前的操作的错误。

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 – 用于构造新 DynamicFrame 的映射序列。
- caseSensitive – 是否将源列视为区分大小写。在与不区分大小写的存储（如 AWS Glue 数据目录）集成时，将此项设置为 false 可能很有帮助。

基于一系列的映射选择、投影和转换列。

每个映射由源列和类型以及目标列和类型构成。映射可指定为四元组 (source_path、source_type、 target_path、target_type) 或包含相同信息的 MappingSpec 对象。

除了将映射用于简单的投影和转换外，还可以通过使用“.”（句点）分隔路径的组件来用于嵌套或取消嵌套字段。

例如，假设您有一个包含以下架构的 DynamicFrame。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- address: struct
  |    |-- state: string
  |    |-- zip: int
}}}
```

您可以进行以下调用来取消嵌套 state 和 zip 字段。

```
{{{
  df.applyMapping(
   Seq(("name", "string", "name", "string"),
       ("age", "int", "age", "int"),
       ("address.state", "string", "state", "string"),
       ("address.zip", "int", "zip", "int")))
}}}
```

生成的架构如下所示。

```
{{{
  root
```

```
  |-- name: string
  |-- age: int
  |-- state: string
  |-- zip: int
}}}
```

您还可以使用 applyMapping 来重新嵌套列。例如，以下代码反转以前的转换，并在目标中创建一个名为 address 的结构。

```
{{{
  df.applyMapping(
   Seq(("name", "string", "name", "string"),
       ("age", "int", "age", "int"),
       ("state", "string", "address.state", "string"),
       ("zip", "int", "address.zip", "int")))
}}}
```

包含“.”（句点）字符的字段名称可以使用反引号 (``) 括起来。

> **Note**
>
> 目前，您不能使用 applyMapping 方法映射嵌套在数组下的列。

Def assertErrorThreshold

```
def assertErrorThreshold : Unit
```

强制计算并验证错误记录数是否低于 stageThreshold 和 totalThreshold 的操作。如果任一条件失败，则引发异常。

Def count

```
lazy
def count
```

返回 DynamicFrame 中的元素数量。

Def dropField

```
def dropField( path : String,
```

```
              transformationContext : String = "",
              callSite : CallSite = CallSite("Not provided", ""),
              stageThreshold : Long = 0,
              totalThreshold : Long = 0
            ) : DynamicFrame
```

返回已删除指定列的新 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
```

返回已删除指定列的新 DynamicFrame。

您可以使用此方法删除嵌套列（包括数组中的列），但不能用于删除特定数组元素。

Def dropNulls

```
def dropNulls( transformationContext : String = "",
               callSite : CallSite = CallSite("Not provided", ""),
               stageThreshold : Long = 0,
               totalThreshold : Long = 0 )
```

返回已删除所有空列的新 DynamicFrame。

> **Note**
>
> 这只删除类型为 NullType 的列。不删除或修改其他列中的单个空值。

Def errorsAsDynamicFrame

```
def errorsAsDynamicFrame
```

返回包含此 DynamicFrame 中的错误记录的新 DynamicFrame。

Def filter

```
def filter( f : DynamicRecord => Boolean,
            errorMsg : String = "",
            transformationContext : String = "",
            callSite : CallSite = CallSite("Not provided"),
            stageThreshold : Long = 0,
            totalThreshold : Long = 0
          ) : DynamicFrame
```

构造只包含函数“f”为其返回 `true` 的那些记录的新 `DynamicFrame`。筛选器函数“f”不应转变输入记录。

Def getName

```
def getName : String
```

返回此 `DynamicFrame` 的名称。

Def getNumPartitions

```
def getNumPartitions
```

返回 `DynamicFrame` 中的分区数量。

Def getSchemaIfComputed

```
def getSchemaIfComputed : Option[Schema]
```

如果架构已经过计算，则返回该架构。如果架构尚未经过计算，则不扫描数据。

Def isSchemaComputed

```
def isSchemaComputed : Boolean
```

如果已为此 `DynamicFrame` 计算架构，则返回 `true`；否则返回 `false`。如果此方法返回 false，则调用 `schema` 方法将需要再次扫描此 `DynamicFrame` 中的记录。

Def 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 中用于联接的列。
- keys2 – frame2 中用于联接的列。必须与 keys1 的长度相同。
- frame2 – 要联接的 DynamicFrame。

返回使用指定的键对 frame2 执行 equijoin 的结果。

Def map

```
def map( f : DynamicRecord => DynamicRecord,
         errorMsg : String = "",
         transformationContext : String = "",
         callSite : CallSite = CallSite("Not provided", ""),
         stageThreshold : Long = 0,
         totalThreshold : Long = 0
       ) : DynamicFrame
```

返回通过对此 DynamicFrame 中的每个记录应用指定函数“f”构造的新 DynamicFrame。

此方法先复制每个记录，然后再应用指定函数，因此可以安全地转变记录。如果映射函数在给定记录上引发异常，则该记录将标记为错误，并且堆栈跟踪将另存为错误记录中的一个列。

Def 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 – 要合并的暂存 DynamicFrame。
- primaryKeys – 要匹配源和暂存 DynamicFrame 中的记录的主键字段列表。
- transformationContext – 用于检索有关当前转换的元数据的唯一字符串（可选）。
- options – 为此转换提供其他信息的 JSON 名称-值对的字符串。
- callSite – 用于为错误报告提供上下文信息。
- stageThreshold — 一个 Long。给定转换中处理需要排除的错误的数目。
- totalThreshold — 一个 Long。此转换中处理需要排除的错误的总数。

基于指定主键的将此 DynamicFrame 与暂存 DynamicFrame 合并以标识记录。不会对重复记录（具有相同主键的记录）去除重复。如果暂存帧中没有匹配的记录，则从源中保留所有记录（包括重复记录）。如果暂存帧具有匹配的记录，则暂存帧中的记录将覆盖 AWS Glue 中的源中的记录。

在以下情况下，返回的 DynamicFrame 将包含记录 A：

1. 如果 A 在源帧和暂存帧中都存在，则返回暂存帧中的 A。
2. 如果 A 在源表中，且A.primaryKeys 不在 stagingDynamicFrame 中（这意味着，未在暂存表中更新 A）。

源帧和暂存帧不需要具有相同的架构。

Example

```
val mergedFrame: DynamicFrame = srcFrame.mergeDynamicFrames(stageFrame, Seq("id1",
 "id2"))
```

Def printSchema

```
def printSchema : Unit
```

以人类可读的格式将此 DynamicFrame 的架构输出到 stdout。

Def recomputeSchema

```
def recomputeSchema : Schema
```

强制重新计算架构。这需要扫描数据，但如果数据中不存在当前架构中的一些字段，则可能会“压缩”架构。

返回经过重新计算的架构。

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` – 在输出中用于基本 `DynamicFrame` 的名称。通过透视数组创建的 `DynamicFrame` 以此作为前缀开头。
- `stagingPath` – 用于写入中间数据的 Amazon Simple Storage Service（Amazon S3）路径。
- `options` – 关系化选项和配置。目前未使用。

展平所有嵌套的结构并将数组透视为单独的表。

您可以使用此操作准备深度嵌套的数据以提取到关系数据库中。使用与 Unnest 转换相同的方式展平嵌套结构。此外，将数组透视为单独的表，其中每个数组元素成为一行。例如，假设您有一个包含以下数据的 `DynamicFrame`。

```
 {"name": "Nancy", "age": 47, "friends": ["Fred", "Lakshmi"]}
 {"name": "Stephanie", "age": 28, "friends": ["Yao", "Phil", "Alvin"]}
 {"name": "Nathan", "age": 54, "friends": ["Nicolai", "Karen"]}
```

运行以下代码。

```
{{{
  df.relationalize("people", "s3:/my_bucket/my_path", JsonOptions.empty)
}}}
```

这会生成两个表。第一个表名为“people”，其中包含以下内容。

```
{{{
  {"name": "Nancy", "age": 47, "friends": 1}
```

```
  {"name": "Stephanie", "age": 28, "friends": 2}
  {"name": "Nathan", "age": 54, "friends": 3)
}}}
```

在这里，friends 数组已被替换为自动生成的联接键。创建了名为 people.friends 的单独表，其中包含以下内容。

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

在此表中，“id”是标识数组元素来自哪个记录的联接键，“index”是指在原始数组中的位置，“val”是实际的数组条目。

relationalize 方法返回通过以递归方式对所有数组应用此过程创建的 DynamicFrame 序列。

> **Note**
>
> AWS Glue 库自动为新表生成联接键。为确保联接键在作业运行中是唯一的，您必须启用作业书签。

Def renameField

```
def renameField( oldName : String,
                 newName : String,
                 transformationContext : String = "",
                 callSite : CallSite = CallSite("Not provided", ""),
                 stageThreshold : Long = 0,
                 totalThreshold : Long = 0
               ) : DynamicFrame
```

- oldName – 列的原始名称。
- newName – 列的新名称。

返回新 DynamicFrame，其中的指定字段进行了重命名。

您可以使用此方法重命名嵌套字段。例如，以下代码在 address 结构中将 state 重命名为 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
```

返回具有 numPartitions 分区的新 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 – 应用于 specs 序列中未列出的所有 ChoiceType 列的操作。
- database – 与 match_catalog 操作一起使用的数据目录数据库。
- tableName – 与 match_catalog 操作一起使用的数据目录表。

通过将一个或多个 ChoiceType 替换为更具体的类型来返回新 DynamicFrame。

可通过两种方式使用 resolveChoice。第一种是指定一系列特定列以及如何解析它们。这些指定为由 (列, 操作) 对组成的元组。

可能的操作如下：

- cast:type – 尝试将所有值转换为指定的类型。
- make_cols – 将每个不同的类型转换为名为 columnName_type 的列。
- make_struct – 将列转换为具有每个不同类型的键的结构。
- project:type – 仅保留指定类型的值。

resolveChoice 的另一种模式是为所有 ChoiceType 指定单个解析方法。这可以在执行前不知道 ChoiceType 的完整列表的情况下使用。除了上面列出的操作外，此模式还支持以下操作：

- match_catalogChoiceType – 尝试将每个 转换为指定目录表中的对应类型。

示例：

通过转换为 int 来解析 user.id 列，并使 address 字段仅保留结构。

```
{{{
  df.resolveChoice(specs = Seq(("user.id", "cast:int"), ("address", "project:struct")))
}}}
```

通过将每个选择转换为单独的列来解析所有 ChoiceType。

```
{{{
  df.resolveChoice(choiceOption = Some(ChoiceOption("make_cols")))
}}}
```

通过转换为指定目录表中的类型来解析所有 ChoiceType。

```
{{{
  df.resolveChoice(choiceOption = Some(ChoiceOption("match_catalog")),
                   database = Some("my_database"),
                   tableName = Some("my_table"))
}}}
```

Def schema

```
def schema : Schema
```

返回此 DynamicFrame 的架构。

保证返回的架构包含此 DynamicFrame 中的记录中存在的每个字段。但在少数情况下，它可能还包含其他字段。您可以使用 Unnest 方法基于此 DynamicFrame 中的记录来“压缩”架构。

Def selectField

```
def selectField( fieldName : String,
                 transformationContext : String = "",
                 callSite : CallSite = CallSite("Not provided", ""),
                 stageThreshold : Long = 0,
                 totalThreshold : Long = 0
               ) : DynamicFrame
```

返回作为 DynamicFrame 的单个字段。

Def selectFields

```
def selectFields( paths : Seq[String],
                  transformationContext : String = "",
                  callSite : CallSite = CallSite("Not provided", ""),
                  stageThreshold : Long = 0,
                  totalThreshold : Long = 0
                ) : DynamicFrame
```

- paths – 要选择的列名称序列。

返回包含指定列的新 DynamicFrame。

> **Note**
>
> 您只能使用 selectFields 方法选择顶级列。您可以使用 applyMapping 方法选择嵌套列。

Def show

```
def show( numRows : Int = 20 ) : Unit
```

- numRows – 要输出的行数。

以 JSON 格式输出此 DynamicFrame 中的行。

Def simplifyDDBJson

使用 AWS Glue DynamoDB 导出连接器进行 DynamoDB 导出时会生成具有特定嵌套结构的 JSON 文件。有关更多信息，请参阅数据对象。simplifyDDBJson简化此数据类型的 DynamicFrame 中的嵌套列，并返回新的简化 DynamicFrame。如果 List 类型中包含多种类型或 Map 类型，则不会简化 List 中的元素。此方法仅支持 DynamoDB 导出 JSON 格式的数据。考虑使用 unnest 来对其他类型的数据进行类似更改。

```
def simplifyDDBJson() : DynamicFrame
```

此方法不使用任何参数。

示例输入

考虑 DynamoDB 导出生成的以下架构：

```
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
```

示例代码

```
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()
  }

}
```

示例输出

simplifyDDBJson 转换会将其简化为：

```
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
```

返回相同记录但写出一部分记录作为副作用的传递转换。

- path – 将输出写入的 Amazon S3 中的路径，格式为 s3://bucket//path。
- options – 描述取样行为的可选的 JsonOptions 映射。

返回包含与这一个相同的记录的 DynamicFrame。

默认情况下，将 100 个任意记录写入通过 path 指定的位置。您可以使用 options 映射自定义此行为。有效键包括：

- topk – 指定写出的记录的总数。默认值为 100。
- prob – 指定包含单个记录的概率（小数形式）。默认值为 1。

例如，以下调用将对数据集进行取样，方法是以 20% 的概率选择每个记录并在写入 200 个记录后停止。

```
{{{
```

```
  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 — 包括在第一个 DynamicFrame 中的路径。

返回两个 DynamicFrame 的序列。第一个 DynamicFrame 包含指定路径，第二个包含所有其他列。

示例

此示例采用从 AWS Glue 数据目录中的 legislators 数据库中的 persons 表创建的 DynamicFrame，并将 DynamicFrame 拆分为两个，其中指定的字段进入第一个 DynamicFrame，其余字段进入第二个 DynamicFrame。然后，示例将从结果中选择第一个 DynamicFrame。

```
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]
```

基于将列与常量比较的谓词来拆分行。

- paths – 用于比较的列。
- values – 用于比较的常量值。
- operators – 用于比较的运算符。

返回两个 DynamicFrame 的序列。第一个包含其谓词为 true 的行，第二个包含其谓词为 false 的行。

使用三个序列指定谓词：“paths”包含（可能嵌套的）列名称，“values”包含要与其进行比较的常量值，“operators”包含用于比较的运算符。所有这三个序列必须长度相同：第 n 个运算符将用于将第 n 个列与第 n 个值进行比较。

每个运算符必须是以下运算符之一：“!=”、“=”、“<=”、“<”、“>=”或“>”。

例如，以下调用将拆分 DynamicFrame，以便第一个输出帧将包含美国 65 岁以上的人员记录，第二个将包含所有其他记录。

```
{{{
  df.splitRows(Seq("age", "address.country"), Seq(65, "USA"), Seq("&gt;=", "="))
}}}
```

Def stageErrorsCount

```
def stageErrorsCount
```

返回在计算此 DynamicFrame 时创建的错误记录数。这不包括作为输入传递给此 DynamicFrame 的以前操作中的错误。

Def toDF

```
def toDF( specs : Seq[ResolveSpec] = Seq.empty[ResolveSpec] ) : DataFrame
```

将此 DynamicFrame 转换为具有相同架构和记录的 Apache Spark SQL DataFrame。

> **Note**
>
> 由于 DataFrame 不支持 ChoiceType，因此此方法自动将 ChoiceType 列转换为 StructType。有关更多信息和用于解析选择的选项，请参阅 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 – 要分析的列。必须是字符串或二进制。
- format – 用于分析的格式。
- optionString – 传递到格式的选项，如 CSV 分隔符。

根据指定的格式分析嵌入式字符串或二进制列。已分析的列嵌套在具有原始列名称的结构下。

例如，假设您具有包含一个嵌入式 JSON 列的 CSV 文件。

```
name, age, address
Sally, 36, {"state": "NE", "city": "Omaha"}
...
```

在初始分析后，您将获取具有以下架构的 DynamicFrame。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- address: string
}}}
```

您可以对地址列调用 unbox 来分析特定组件。

```
{{{
  df.unbox("address", "json")
}}}
```

这为我们提供了具有以下架构的 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
```

返回平展了所有嵌套结构的新 DynamicFrame。构造名称时使用“.”（句点）字符。

例如，假设您有一个包含以下架构的 DynamicFrame。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- address: struct
  |    |-- state: string
  |    |-- city: string
}}}
```

以下调用取消嵌套 address 结构。

```
{{{
  df.unnest()
}}}
```

生成的架构如下所示。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- address.state: string
  |-- address.city: string
}}}
```

此方法还取消嵌套数组中的嵌套结构。但由于历史原因，此类字段的名称前附加了括起来的数组的名称和“.val”。

Def unnestDDBJson

```
unnestDDBJson(transformationContext : String = "",
            callSite : CallSite = CallSite("Not Provided"),
            stageThreshold : Long = 0,
            totalThreshold : Long = 0): DynamicFrame
```

解除 DynamicFrame 中的嵌套列，具体位于 DynamoDB JSON 结构中，并返回一个新的非嵌套 DynamicFrame。属于结构类型数组的列将不会被解除嵌套。请注意，这是一种特定类型的非嵌套转换，其行为与常规 unnest 转换不同，并且要求数据已存在于 DynamoDB JSON 结构中。有关更多信息，请参阅 DynamoDB JSON。

例如，使用 DynamoDB JSON 结构读取导出的架构可能如下所示：

```
root
|-- Item: struct
|    |-- ColA: struct
|    |    |-- S: string
|    |-- ColB: struct
|    |    |-- S: string
|    |-- ColC: struct
|    |    |-- N: string
|    |-- ColD: struct
|    |    |-- L: array
|    |    |    |-- element: null
```

unnestDDBJson() 转换会将此转换为：

```
root
```

```
|-- ColA: string
|-- ColB: string
|-- ColC: string
|-- ColD: array
|    |-- element: null
```

以下代码示例演示了如何使用 AWS Glue DynamoDB 导出连接器、调用 DynamoDB JSON 解除嵌套命令，以及打印分区数量：

```
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()
  }

}
```

Def withFrameSchema

```
def withFrameSchema( getSchema : () => Schema ) : DynamicFrame
```

- getSchema – 返回要使用的架构的函数。指定为零参数函数来推迟可能昂贵的计算。

将此 DynamicFrame 的架构设置为指定值。这主要在内部使用以避免成本高昂的架构重新计算。传入的架构必须包含数据中存在的所有列。

Def withName

```
def withName( name : String ) : DynamicFrame
```

- name – 要使用的新名称。

返回使用新名称的此 DynamicFrame 的副本。

Def withTransformationContext

```
def withTransformationContext( ctx : String ) : DynamicFrame
```

返回具有指定转换上下文的此 DynamicFrame 的副本。

DynamicFrame 对象

程序包：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 类

主题

- Def write
- Def readFields
- Def clone
- Def schema
- Def getRoot
- Def toJson
- Def getFieldNode
- Def getField
- Def hashCode
- Def equals
- DynamicRecord 对象
- RecordTraverser 特性

程序包：com.amazonaws.services.glue

```
class DynamicRecord extends Serializable with Writable with Cloneable
```

DynamicRecord 是一个自描述数据结构，它表示被处理的数据集中的一行数据。自描述的意义在于，您可以通过检查记录本身来获取 DynamicRecord 所表示行的架构。DynamicRecord 类似于 Apache Spark 中的 Row。

Def addField

```
def addField( path : String,
              dynamicNode : DynamicNode
            ) : Unit
```

将 DynamicNode 添加至指定路径。

- path – 要添加的字段的路径。
- dynamicNode – 要在指定路径处添加的 DynamicNode。

Def dropField

```
 def dropField(path: String, underRename: Boolean = false): Option[DynamicNode]
```

从指定路径删除 DynamicNode 并在指定路径中不存在阵列时返回已删除的节点。

- path – 要删除的字段的路径。
- underRenamedropField – 如果作为重命名转换的一部分调用 ，则为 True，否则为 false (默认为 false)。

返回 scala.Option Option (DynamicNode)。

Def setError

```
def setError( error : Error )
```

按 error 参数指定，将此记录设置为错误记录。

返回 DynamicRecord。

Def isError

```
def isError
```

检查此记录是否为错误记录。

Def getError

```
def getError
```

如果记录为错误记录，则获取 Error。如果此记录为错误记录，则返回 scala.Some Some (错误)；否则返回 scala.None。

Def clearError

```
def clearError
```

将 Error 设置为 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
```

将此记录克隆到新 DynamicRecord 并将其返回。

Def schema

```
def schema
```

通过检查记录获取 Schema。

Def getRoot

```
def getRoot : ObjectNode
```

获取记录的根 ObjectNode。

Def toJson

```
def toJson : String
```

获取记录的 JSON 字符串。

Def getFieldNode

```
def getFieldNode( path : String ) : Option[DynamicNode]
```

获取指定 path 处的字段值，作为 DynamicNode 的 Option。

如果字段存在，则返回 scala.Some Some (DynamicNode)，否则返回 scala.None.None。

Def getField

```
def getField( path : String ) : Option[Any]
```

获取指定 path 处的字段值，作为 DynamicNode 的 Option。

返回 scala.Some Some (值)。

Def hashCode

```
override def hashCode : Int
```

Def equals

```
override def equals( other : Any )
```

DynamicRecord 对象

```
object DynamicRecord
```

Def apply

```
def apply( row : Row,
           schema : SparkStructType )
```

应用方法以将 Apache Spark SQL Row 转换为 DynamicRecord。

- row – Spark SQL Row。
- schema – 该行的 Schema。

返回 DynamicRecord。

RecordTraverser 特性

```
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 GlueScala API GlueContext

程序包：com.amazonaws.services.glue

```
class GlueContext extends SQLContext(sc) (
           @transient val sc : SparkContext,
           val defaultSourcePartitioner : PartitioningStrategy )
```

GlueContext 是在以下位置读取和写入 DynamicFrame 的入口点：Amazon Simple Storage Service（Amazon S3）、 AWS Glue 数据目录、JDBC 等。此类提供实用程序函数，用于创建可用来读取和写入 DynamicFrame 的 数据源特性 和 DataSink 对象。

如果从源创建的分区数小于分区的最小阈值（默认值为 10），您还可以使用 GlueContext 设置 DynamicFrame 中的目标分区数量（默认值为 20）。

def 列 addIngestionTime

```
def addIngestionTimeColumns(
         df : DataFrame,
         timeGranularity : String = "") : dataFrame
```

将提取时间列（例如 ingest_year、ingest_month、ingest_day、ingest_hour、ingest_minute）附加到输入 DataFrame。当您指定以 Amazon S3 为目标的数据目录表时，AWS Glue 生成的脚本中会自动生成

此函数。此函数使用输出表上的提取时间列自动更新分区。这允许根据提取时间自动对输出数据进行分区，而不需要在输入数据中显示提取时间列。

- dataFrame – 提取时间列要附加到的 dataFrame。
- timeGranularity – 时间列的粒度。有效值为“day”、“hour”和“minute”。例如，如果“hour”传递到函数，原始 dataFrame 将附加“ingest_year”、“ingest_month”、“ingest_day”和“ingest_hour”时间列。

在附加时间粒度列后返回数据框。

例如：

```
glueContext.addIngestionTimeColumns(dataFrame, "hour")
```

def createDataFrame FromOptions

```
def createDataFrameFromOptions( connectionType : String,
                          connectionOptions : JsonOptions,
                          transformationContext : String = "",
                          format : String = null,
                          formatOptions : JsonOptions = JsonOptions.empty
                        ) : DataSource
```

返回一个使用指定连接和格式创建的 DataFrame。此功能只能用于 AWS Glue 直播源。

- connectionType – 流式传输连接类型。有效值包括 kinesis 和 kafka。
- connectionOptions – 连接选项，不同于 Kinesis 和 Kafka。您可以在 AWS Glue for Spark 中适用于 ETL 的连接类型和选项 找到每个流式传输数据源的所有连接选项列表。请注意流式传输连接选项的以下差异：
  - Kinesis 流式传输源需要 streamARN、startingPosition、inferSchema 和 classification。
  - Kinesis 流式传输源需要 connectionName、topicName、startingOffsets、inferSchema 和 classification。
- transformationContext – 要使用的转换上下文（可选）。
- format – 格式规范（可选）。这用于 Amazon S3 或支持多种格式的 AWS Glue 连接。有关所支持格式的信息，请参阅 AWS Glue for Spark 中的输入和输出的数据格式选项。

- formatOptions – 指定格式的格式选项。有关所支持的格式选项的信息，请参阅数据格式选项。

Amazon Kinesis 流式传输源示例：

```
val data_frame_datasource0 =
glueContext.createDataFrameFromOptions(transformationContext = "datasource0",
 connectionType = "kinesis",
connectionOptions = JsonOptions("""{"streamName": "example_stream", "startingPosition":
 "TRIM_HORIZON", "inferSchema": "true", "classification": "json"}}""""))
```

Kafka 流式传输源示例：

```
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)

将传入的 batch_function 应用于从流式传输源读取的每个微批处理。

- frame— DataFrame 包含当前微批次的。
- batch_function – 应用于每个微处理的函数。
- options – 键值对集合，其中包含有关如何处理微批处理的信息。以下选项为必填：
  - windowSize – 处理每个批处理所花费的时间量。
  - checkpointLocation – 为流式传输 ETL 任务存储检查点的位置。
  - batchMaxRetries – 在该批处理失败时重试的最大次数。默认值为 3。此选项仅适用于 Glue 版本 2.0 及更高版本。

示例：

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

def getCatalogSink

```
def getCatalogSink( database : String,
        tableName : String,
        redshiftTmpDir : String = "",
        transformationContext : String = ""
        additionalOptions: JsonOptions = JsonOptions.empty,
        catalogId: String = null
) : DataSink
```

创建一个可向在数据目录中定义的表中指定的位置写入的 DataSink。

- `database` – 数据目录中的数据库名称。
- `tableName` – 数据目录中的表名称。
- `redshiftTmpDir` – 用于某些数据接收器的临时暂存目录。设置为 默认情况下为空。

- transformationContext – 与任务书签要使用的接收器关联的转换上下文。设置为 默认情况下为空。
- additionalOptions 提供给 AWS Glue 的额外选项。
- catalogId – 正在访问的数据目录 ID（账户 ID）。当为空时，将使用调用方的默认账户 ID。

返回 DataSink。

def getCatalogSource

```
def getCatalogSource( database : String,
                      tableName : String,
                      redshiftTmpDir : String = "",
                      transformationContext : String = ""
                      pushDownPredicate : String = " "
                      additionalOptions: JsonOptions = JsonOptions.empty,
                      catalogId: String = null
                    ) : DataSource
```

创建可从数据目录中的表定义读取数据的 数据源特性。

- database – 数据目录中的数据库名称。
- tableName – 数据目录中的表名称。
- redshiftTmpDir – 用于某些数据接收器的临时暂存目录。设置为 默认情况下为空。
- transformationContext – 与任务书签要使用的接收器关联的转换上下文。设置为 默认情况下为空。
- pushDownPredicate – 筛选分区，而不必列出并读取数据集中的所有文件。有关更多信息，请参阅 使用下推谓词进行预筛选。
- additionalOptions – 可选名称/值对的集合。可能选项包括 AWS Glue for Spark 中适用于 ETL 的连接类型和选项 中列出的选项，但 endpointUrl、streamName、bootstrap.servers、security.protocol、topicName、classi 和 delimiter 除外。另一个支持的选项是 catalogPartitionPredicate：

  catalogPartitionPredicate – 要传递目录表达式以根据索引列进行筛选。这样会将筛选下推到服务器端。有关更多信息，请参阅 AWS Glue 分区数据。请注意，push_down_predicate 和 catalogPartitionPredicate 使用不同的语法。前者使用 Spark SQL 标准语法，后者使用 JSQL 解析器。

- catalogId – 正在访问的数据目录 ID（账户 ID）。当为空时，将使用调用方的默认账户 ID。

返回 DataSource。

流式传输源示例

```
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
```

创建一个可向在数据目录中的 Connection 对象指定的 JDBC 数据库写入的 DataSink。Connection 对象具有用于连接 JDBC 接收器的信息，包括 URL、用户名、密码、VPC、子网和安全组。

- catalogConnection – 数据目录中包含要写入的 JDBC URL 的连接名称。
- options – JSON 名称-值对的字符串，提供写入 JDBC 数据存储所需的附加信息。这包括：
  - dbtable（必填）- JDBC 表名称。对于在数据库中支持架构的 JDBC 数据存储，指定 schema.table-name。如果未提供架构，则使用默认的“public”架构。以下示例显示一个指向名为 test 的架构的选项参数和数据库 test_db 中一个名为 test_table 的表。

    ```
    options = JsonOptions("""{"dbtable": "test.test_table", "database": "test_db"}""")
    ```

  - database（必填）- JDBC 数据库名称。
  - 任何其他选项直接传递到 SparkSQL JDBC 写入器。有关更多信息，请参阅 Spark 的 Redshift 数据源。

- redshiftTmpDir – 用于某些数据接收器的临时目录。设置为 默认情况下为空。
- transformationContext – 与任务书签要使用的接收器关联的转换上下文。设置为 默认情况下为空。
- catalogId – 正在访问的数据目录 ID（账户 ID）。当为空时，将使用调用方的默认账户 ID。

示例代码：

```
getJDBCSink(catalogConnection = "my-connection-name", options =
 JsonOptions("""{"dbtable": "my-jdbc-table", "database": "my-jdbc-db"}"""),
 redshiftTmpDir = "", transformationContext = "datasink4")
```

返回 DataSink。

def getSink

```
def getSink( connectionType : String,
             connectionOptions : JsonOptions,
             transformationContext : String = ""
           ) : DataSink
```

创建将数据写入目标的DataSink，例如亚马逊简单存储服务 (Amazon S3)、JDBC 或 Glue 数据目录，或者是 AWS Apache Kafka 或 Amazon Kinesis 数据流。

- connectionType - 连接的类型。请参阅 the section called “连接参数”。
- connectionOptions – JSON 名称-值对的字符串，提供与数据接收器建立连接所需的附加信息。请参阅 the section called “连接参数”。
- transformationContext – 与任务书签要使用的接收器关联的转换上下文。设置为 默认情况下为空。

返回 DataSink。

def 格式 getSinkWith

```
def getSinkWithFormat( connectionType : String,
                       options : JsonOptions,
                       transformationContext : String = "",
                       format : String = null,
```

```
                        formatOptions : JsonOptions = JsonOptions.empty
                      ) : DataSink
```

创建 DataSink，将数据写入 Amazon S3、JDBC、数据目录、Apache Kafka 或 Amazon Kinesis 数据流等目标写入数据。此外还将设置要写出到目标的数据格式。

- connectionType - 连接的类型。请参阅 the section called “连接参数”。
- options – JSON 名称-值对的字符串，提供与数据接收器建立连接所需的附加信息。请参阅 the section called “连接参数”。
- transformationContext – 与任务书签要使用的接收器关联的转换上下文。设置为 默认情况下为空。
- format – 要写出到目标的数据的格式。
- formatOptions – JSON 名称-值对的字符串，提供用于设置目标处的数据格式的附加选项。请参阅 数据格式选项。

返回 DataSink。

def getSource

```
def getSource( connectionType : String,
               connectionOptions : JsonOptions,
               transformationContext : String = ""
               pushDownPredicate
             ) : DataSource
```

创建从 Amazon S3、JDBC 或 Glue 数据目录等来源读取数据 AWS 的。数据源特性还支持 Kafka 和 Kinesis 流式传输数据源。

- connectionType – 数据源的类型。请参阅 the section called “连接参数”。
- connectionOptions – JSON 名称-值对的字符串，提供与数据源建立连接所需的附加信息。有关更多信息，请参阅 the section called “连接参数”。

  Kinesis 流式传输源需要以下连接选项：streamARN、startingPosition、inferSchema 和 classification。

  Kinesis 流式传输源需要以下连接选项：connectionName、topicName、startingOffsets、inferSchema 和 classification。

- transformationContext – 与任务书签要使用的接收器关联的转换上下文。设置为 默认情况下为空。
- pushDownPredicate – 预测分区列。

返回 DataSource。

Amazon Kinesis 流式传输源示例：

```
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 流式传输源示例：

```
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)
 }
```

def 格式 getSourceWith

```
def getSourceWithFormat( connectionType : String,
                         options : JsonOptions,
                         transformationContext : String = "",
```

```
                        format : String = null,
                        formatOptions : JsonOptions = JsonOptions.empty
                      ) : DataSource
```

创建一个数据源特性，用于从 Amazon S3、JDBC 或 Glu AWS e 数据目录等来源读取数据，并设置存储在源中的数据的格式。

- `connectionType` – 数据源的类型。请参阅 the section called “连接参数”。
- `options` – JSON 名称-值对的字符串，提供与数据源建立连接所需的附加信息。请参阅 the section called “连接参数”。
- `transformationContext` – 与任务书签要使用的接收器关联的转换上下文。设置为 默认情况下为空。
- `format` – 源中所存储数据的格式。当 `connectionType` 为“s3”时，您也可以指定 `format`。可以是以下值之一：“avro”、“csv”、“grokLog”、“ion”、“json”、“xml”、“parquet”或“orc”。
- `formatOptions` – JSON 名称-值对的字符串，提供用于在源中分析数据的附加选项。请参阅 数据格式选项。

返回 `DataSource`。

示例

在 A DynamicFrame mazon S3 上使用逗号分隔值 (CSV) 文件的数据源创建：

```
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()
```

使用 JDBC 连接 DynamicFrame 从 PostgreSQL 数据源创建：

```
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 连接 DynamicFrame 从 MySQL 的数据源创建：

```
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()
```

def getSparkSession

```
def getSparkSession : SparkSession
```

获取与此 GlueContext 关联的 SparkSession 对象。使用此 SparkSession 对象注册表和 UDF，以便在DataFrame创建时 DynamicFrames使用。

返回 SparkSession。

def startTransaction

```
def startTransaction(readOnly: Boolean):String
```

开启新事务。内部调用 Lake Formation startTransaction API。

- readOnly – (布尔值) 指示此事务应为只读还是读写。使用只读事务 ID 进行的写入将被拒绝。只读事务不需要提交。

返回事务 ID。

def commitTransaction

```
def commitTransaction(transactionId: String, waitForCommit: Boolean): Boolean
```

尝试提交指定的事务。可能在事务完成提交之前返回 commitTransaction。内部调用 Lake Formation commitTransaction API。

- transactionId – (字符串) 要提交的事务。
- waitForCommit – (布尔值) 确定是否立即返回 commitTransaction。默认值为 true。如果为假，则轮询 commitTransaction 并等待事务提交。等待时间限制为 1 分钟使用指数回退，最多重试 6 次。

返回布尔值，以指示是否完成提交。

def cancelTransaction

```
def cancelTransaction(transactionId: String): Unit
```

尝试取消指定的事务。在内部调用 Lake Format CancelTransactionion API。

- transactionId – (字符串) 要取消的事务。

如果事务以前已提交，则返回 TransactionCommittedException 异常。

def this

```
def this( sc : SparkContext,
          minPartitions : Int,
          targetPartitions : Int )
```

使用指定的 SparkContext、最小分区数和目标分区数创建 GlueContext 对象。

- sc — 这些区域有： SparkContext。
- minPartitions – 最小分区数。
- targetPartitions – 目标分区数。

返回 GlueContext。

def this

```
def this( sc : SparkContext )
```

使用提供的 GlueContext 创建一个 SparkContext 对象。将最小分区数设置为 10，将目标分区数设置为 20。

- sc — 这些区域有： SparkContext。

返回 GlueContext。

def this

```
def this( sparkContext : JavaSparkContext )
```

使用提供的 GlueContext 创建一个 JavaSparkContext 对象。将最小分区数设置为 10，将目标分区数设置为 20。

- sparkContext — 这些区域有： JavaSparkContext。

返回 GlueContext。

MappingSpec

程序包：com.amazonaws.services.glue

MappingSpec case 类

```
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 - 源字段的 SchemaPath。
- sourceType - 源字段的 DataType。
- targetPath - 目标字段的 SchemaPath。
- targetType - 目标字段的 DataType。

MappingSpec 指定从源路径和源数据类型到目标路径和目标数据类型的映射。源路径的源帧中的值会显示在目标路径的目标帧中。源数据类型将强制转换为目标数据类型。

它从 Product4 扩展而来，因此您可在 applyMapping 接口中处理任何 Product4。

MappingSpec 对象

```
object MappingSpec
```

MappingSpec 对象具有以下成员：

Val orderingByTarget

```
val orderingByTarget: Ordering[MappingSpec]
```

Def apply

```
def apply( sourcePath : String,
           sourceType : DataType,
           targetPath : String,
           targetType : DataType
         ) : MappingSpec
```

创建 MappingSpec。

- sourcePath – 源路径的字符串表示形式。
- sourceType - 源 DataType。
- targetPath – 目标路径的字符串表示形式。
- targetType - 目标 DataType。

返回 MappingSpec。

Def apply

```
def apply( sourcePath : String,
           sourceTypeString : String,
           targetPath : String,
           targetTypeString : String
```

```
        ) : MappingSpec
```

创建 MappingSpec。

- sourcePath – 源路径的字符串表示形式。
- sourceType – 源数据类型的字符串表示形式。
- targetPath – 目标路径的字符串表示形式。
- targetType – 目标数据类型的字符串表示形式。

返回 MappingSpec。

Def apply

```
def apply( product : Product4[String, String, String, String] ) : MappingSpec
```

创建 MappingSpec。

- product – 源路径、源数据类型、目标路径和目标数据类型的 Product4。

返回 MappingSpec。

AWS Glue Scala ResolveSpec API

主题

- ResolveSpec 对象
- ResolveSpec case 类

程序包：com.amazonaws.services.glue

ResolveSpec 对象

ResolveSpec

```
object ResolveSpec
```

Def

```
def apply( path : String,
```

```
          action : String
        ) : ResolveSpec
```

创建 ResolveSpec。

- path – 需要解析的选择字段的字符串表示形式。
- action – 解析操作。操作可以是以下项之一：Project、KeepAsStruct 或 Cast。

返回 ResolveSpec。

Def

```
def apply( product : Product2[String, String] ) : ResolveSpec
```

创建 ResolveSpec。

- product - Product2：源路径、解析操作。

返回 ResolveSpec。

ResolveSpec case 类

```
case class ResolveSpec extends Product2[String, String]  (
           path : SchemaPath,
           action : String )
```

创建 ResolveSpec。

- path – 需要解析的选择字段的 SchemaPath。
- action – 解析操作。操作可以是以下项之一：Project、KeepAsStruct 或 Cast。

ResolveSpec def 方法

```
def _1 : String
```

```
def _2 : String
```

AWS Glue Scala ArrayNode API

程序包：com.amazonaws.services.glue.types

ArrayNode case 类

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

程序包：com.amazonaws.services.glue.types

BinaryNode case 类

BinaryNode

```
case class BinaryNode extends ScalarNode(value, TypeCode.BINARY)  (
           value : Array[Byte] )
```

BinaryNode val 字段

- `ordering`

BinaryNode def 方法

```
def clone
```

```
def equals( other : Any )
```

```
def hashCode : Int
```

AWS Glue Scala BooleanNode API

程序包：com.amazonaws.services.glue.types

BooleanNode case 类

BooleanNode

```
case class BooleanNode extends ScalarNode(value, TypeCode.BOOLEAN)  (
           value : Boolean )
```

BooleanNode val 字段

- ordering

BooleanNode def 方法

```
def equals( other : Any )
```

AWS Glue Scala ByteNode API

程序包：com.amazonaws.services.glue.types

ByteNode case 类

ByteNode

```
case class ByteNode extends ScalarNode(value, TypeCode.BYTE)  (
           value : Byte )
```

ByteNode val 字段

- ordering

ByteNode def 方法

```
def equals( other : Any )
```

AWS Glue Scala DateNode API

程序包：com.amazonaws.services.glue.types

DateNode case 类

DateNode

```
case class DateNode extends ScalarNode(value, TypeCode.DATE)  (
           value : Date )
```

DateNode val 字段

- ordering

DateNode def 方法

```
def equals( other : Any )
```

```
def this( value : Int )
```

AWS Glue Scala DecimalNode API

程序包：com.amazonaws.services.glue.types

DecimalNode case 类

DecimalNode

```
case class DecimalNode extends ScalarNode(value, TypeCode.DECIMAL)  (
           value : BigDecimal )
```

DecimalNode val 字段

- `ordering`

DecimalNode def 方法

```
def equals( other : Any )
```

```
def this( value : Decimal )
```

AWS Glue Scala DoubleNode API

程序包：com.amazonaws.services.glue.types

DoubleNode case 类

DoubleNode

```
case class DoubleNode extends ScalarNode(value, TypeCode.DOUBLE)  (
           value : Double )
```

DoubleNode val 字段

- `ordering`

DoubleNode def 方法

```
def equals( other : Any )
```

AWS Glue Scala DynamicNode API

主题

- DynamicNode 类
- DynamicNode 对象

程序包：com.amazonaws.services.glue.types

DynamicNode 类

DynamicNode

```
class DynamicNode extends Serializable with Cloneable
```

DynamicNode def 方法

```
def getValue : Any
```

获取无格式值并绑定到当前记录：

```
def nodeType : TypeCode
```

```
def toJson : String
```

调试方法：

```
def toRow( schema : Schema,
           options : Map[String, ResolveOption]
         ) : Row
```

```
def typeName : String
```

DynamicNode 对象

DynamicNode

```
object DynamicNode
```

DynamicNode def 方法

```
def quote( field : String,
           useQuotes : Boolean
         ) : String
```

```
def quote( node : DynamicNode,
           useQuotes : Boolean
         ) : String
```

EvaluateDataQuality 类

AWS Glue Data Quality 目前为 AWS Glue 的预览版，可能会发生变化。

程序包：com.amazonaws.services.glue.dq

```
object EvaluateDataQuality
```

Def apply

```
def apply(frame: DynamicFrame,
            ruleset: String,
            publishingOptions: JsonOptions = JsonOptions.empty): DynamicFrame
```

根据 DynamicFrame 评估数据质量规则集，并返回一个包含评估结果的新 DynamicFrame。要了解有关 AWS Glue 数据质量的更多信息，请参阅AWS Glue 数据质量。

- frame 表示您想要评估的数据质量的 DynamicFrame。
- ruleset – 字符串格式的数据质量定义语言（DQDL）规则集。要了解有关 DQDL 的更多信息，请参阅数据质量定义语言（DQDL）引用指南。

- publishingOptions – 一个字典，用于为发布评估结果和指标指定以下选项：
  - dataQualityEvaluationContext – 一个字符串，用于指定 AWS Glue 应在哪个命名空间下发布 Amazon CloudWatch 指标和数据质量结果。汇总指标显示在 CloudWatch 中，而完整结果显示在 AWS Glue Studio 界面中。
    - 必需：否
    - 默认值：default_context
  - enableDataQualityCloudWatchMetrics – 指定是否应将数据质量评估结果发布到 CloudWatch。您可以使用 dataQualityEvaluationContext 选项为指标指定命名空间。
    - 必需：否
    - 默认值：False
  - enableDataQualityResultsPublishing – 指定是否应在 AWS Glue Studio 界面的 Data Quality（数据质量）选项卡上显示数据质量结果。
    - 必需：否
    - 默认值：true
  - resultsS3Prefix – 指定 AWS Glue 可以写入数据质量评估结果的 Amazon S3 位置。
    - 必需：否
    - 默认值：""（空字符串）

示例

以下示例代码演示了在执行 SelectFields 转换之前如何评估 DynamicFrame 数据质量。该脚本在尝试转换之前会验证所有数据质量规则是否均已通过。

```
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

程序包：com.amazonaws.services.glue.types

FloatNode case 类

FloatNode

```
case class FloatNode extends ScalarNode(value, TypeCode.FLOAT)  (
           value : Float )
```

FloatNode val 字段

- ordering

FloatNode def 方法

```
def equals( other : Any )
```

FillMissingValues 类

程序包：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
```

填充指定列中的动态帧缺失值，并在新列中返回包含估计值的新帧。对于没有缺失值的行，指定列的值将复制到新列。

- `frame` – 要填充缺失值的 DynamicFrame。必需。
- `missingValuesColumn` – 包含缺少值的列（`null` 值和空字符串）。必需。
- `outputColumn` – 新列的名称，该列将包含值缺失的所有行的估计值。可选；默认为后缀为 `"_filled"` 的 `missingValuesColumn` 值。
- `transformationContext` – 用于标识状态信息的唯一字符串（可选）。
- `callSite` – 用于为错误报告提供上下文信息（可选）。
- `stageThreshold` – 在转换出错之前可能在其中发生的最大错误数（可选；默认值为零）。
- `totalThreshold` – 在处理出错之前可能全面发生的最大错误数（可选；默认值为零）。

返回带附加列的新动态帧，该列包含带缺失值的行的估计值和其他行的当前值。

FindMatches 类

程序包：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
```

在输入框架中查找匹配项并返回一个新框架，其中包含每个匹配组唯一 ID 的新列。

- `frame` – 要在其中查找匹配项的 DynamicFrame。必需。
- `transformId` – 与要应用于输入框架的 FindMatches 转换关联的唯一 ID。必需。
- `transformationContext` – 此 DynamicFrame 的标识符。`transformationContext` 用作跨运行保存的任务书签状态的密钥。可选。
- `callSite` – 用于为错误报告提供上下文信息。在从 Python 调用时，会自动设置这些值。可选。
- `stageThreshold` – 在引发异常之前允许的来自此 DynamicFrame 计算的最大错误记录数，不包括以前的 DynamicFrame 中存在的记录。可选。默认值为 0。
- `totalThreshold` – 引发异常之前的最大错误记录总数，包括以前的帧中的记录。可选。默认值为 0。
- `enforcedMatches` – 强制匹配的帧。可选。默认为 `null`。
- `computeMatchConfidenceScores` — 布尔值，指示是否为每组匹配记录计算置信度得分。可选。默认值为 false。

返回一个新的动态框架，该框架具有分配给每组匹配记录的唯一标识符。

FindIncrementalMatches 类

程序包：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
```

查找现有帧和增量帧之间的匹配项，并返回一个新帧，其中包含每个匹配组唯一 ID 的新列。

- existingframe – 已为每个组分配了匹配 ID 的现有帧。必需。
- incrementalframe – 用于查找与现有帧匹配的增量帧。必需。
- transformId – 与要应用于输入帧的 FindIncrementalMatches 转换相关联的唯一 ID。必需。
- transformationContext – 此 DynamicFrame 的标识符。transformationContext 用作跨运行保存的任务书签状态的密钥。可选。
- callSite – 用于为错误报告提供上下文信息。在从 Python 调用时，会自动设置这些值。可选。
- stageThreshold – 在引发异常之前允许的来自此 DynamicFrame 计算的最大错误记录数，不包括以前的 DynamicFrame 中存在的记录。可选。默认值为 0。
- totalThreshold – 引发异常之前的最大错误记录总数，包括以前的帧中的记录。可选。默认值为 0。
- enforcedMatches – 强制匹配的帧。可选。默认为 null。
- computeMatchConfidenceScores — 布尔值，指示是否为每组匹配记录计算置信度得分。可选。默认值为 false。

返回一个新的动态框架，该框架具有分配给每组匹配记录的唯一标识符。

AWS Glue Scala IntegerNode API

程序包：com.amazonaws.services.glue.types

IntegerNode case 类

IntegerNode

```
case class IntegerNode extends ScalarNode(value, TypeCode.INT)  (
           value : Int )
```

IntegerNode val 字段

- `ordering`

IntegerNode def 方法

```
def equals( other : Any )
```

AWS Glue Scala LongNode API

程序包：com.amazonaws.services.glue.types

LongNode case 类

LongNode

```
case class LongNode extends ScalarNode(value, TypeCode.LONG)  (
           value : Long )
```

LongNode val 字段

- `ordering`

LongNode def 方法

```
def equals( other : Any )
```

AWS Glue Scala MapLikeNode API

程序包：com.amazonaws.services.glue.types

MapLikeNode 类

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
```

示例：鉴于此 JSON：

```
{"foo": "bar"}
```

如果 `useQuotes == true`，则 `toJson` 会生成 `{"foo": "bar"}`。如果 `useQuotes == false`，则 `toJson` 会生成 `{foo: bar}` @return。

AWS Glue Scala MapNode API

程序包：com.amazonaws.services.glue.types

MapNode case 类

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

主题

- NullNode 类
- NullNode case 对象

程序包：com.amazonaws.services.glue.types

NullNode 类

NullNode

```
class NullNode
```

NullNode case 对象

NullNode

```
case object NullNode extends NullNode
```

AWS Glue Scala ObjectNode API

主题

- ObjectNode 对象
- ObjectNode case 类

程序包：com.amazonaws.services.glue.types

ObjectNode 对象

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 case 类

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

主题

- ScalarNode 类
- ScalarNode 对象

程序包：com.amazonaws.services.glue.types

ScalarNode 类

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 对象

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

程序包：com.amazonaws.services.glue.types

ShortNode case 类

ShortNode

```
case class ShortNode extends ScalarNode(value, TypeCode.SHORT)  (
           value : Short )
```

ShortNode val 字段

- `ordering`

ShortNode def 方法

```
def equals( other : Any )
```

AWS Glue Scala StringNode API

程序包：com.amazonaws.services.glue.types

StringNode case 类

StringNode

```
case class StringNode extends ScalarNode(value, TypeCode.STRING)  (
           value : String )
```

StringNode val 字段

- `ordering`

StringNode def 方法

```
def equals( other : Any )
```

```
def this( value : UTF8String )
```

AWS Glue Scala TimestampNode API

程序包：com.amazonaws.services.glue.types

TimestampNode case 类

TimestampNode

```
case class TimestampNode extends ScalarNode(value, TypeCode.TIMESTAMP)  (
           value : Timestamp )
```

TimestampNode val 字段

- `ordering`

TimestampNode def 方法

```
def equals( other : Any )
```

```
def this( value : Long )
```

AWS Glue Scala GlueArgParser API

程序包：com.amazonaws.services.glue.util

GlueArgParser 对象

GlueArgParser

```
object GlueArgParser
```

这与 AWSGlueDataplanePython 程序包中的 utils.getResolvedOptions 的 Python 版本严格一致。

GlueArgParser def 方法

```
def getResolvedOptions( args : Array[String],
                        options : Array[String]
                      ) : Map[String, String]
```

```
def initParser( userOptionsSet : mutable.Set[String] ) : ArgumentParser
```

Example 检索传递到作业的参数

要检索作业参数，您可以使用 getResolvedOptions 方法。考虑以下示例，它检索名为 aws_region 的作业参数。

```
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 作业 API

程序包：com.amazonaws.services.glue.util

Job 对象

任务

```
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 编程的功能和优化

以下各部分介绍通常适用于任何语言中的 AWS Glue for Spark ETL（提取、转换和加载）编程的技术和值。

主题

- AWS Glue for Spark 中适用于 ETL 的连接类型和选项
- AWS Glue for Spark 中的输入和输出的数据格式选项
- 适用于 Spark SQL 作业的 AWS Glue 数据目录支持
- 使用作业书签

- 在 AWS Glue Studio 外部使用敏感数据检测
- AWS Glue 可视化任务 API

AWS Glue for Spark 中适用于 ETL 的连接类型和选项

在 AWS Glue for Spark 中，各种 PysPark 和 Scala 方法和转换使用 `connectionType` 参数指定连接类型。它们使用 `connectionOptions` 或 `options` 参数指定连接选项。

`connectionType` 参数可以采用下表中显示的值。下面几个部分介绍每种类型所关联的 `connectionOptions`（或 `options`）参数值。除非另有说明，否则这些参数会在连接用作源或接收器时使用。

有关展示如何设置和使用连接选项的代码示例，请参阅每种连接类型的主页。

connectionType	连接到
dynamodb	Amazon DynamoDB 数据库
kinesis	Amazon Kinesis Data Streams
S3	Amazon S3
documentdb	Amazon DocumentDB (with MongoDB compatibility) 数据库
opensearch	Amazon OpenSearch Service。
redshift	Amazon Redshift 数据库
kafka	Kafka 或 Amazon Managed Streaming for Apache Kafka
azurecosmos	Azure Cosmos for NoSQL。
azuresql	Azure SQL。
bigquery	Google BigQuery。
mongodb	MongoDB 数据库，包括 MongoDB Atlas。
sqlserver	Microsoft SQL Server 数据库（请参阅JDBC 连接）
mysql	MySQL 数据库（请参阅JDBC 连接）

connectionType	连接到
oracle	Oracle 数据库（请参阅JDBC 连接）
postgresql	PostgreSQL 数据库（请参阅JDBC 连接）
saphana	SAP HANA。
snowflake	Snowflake 数据湖
teradata	Teradata Vantage。
vertica	Vertica。
custom.*	Spark、Athena 或 JDBC 数据存储（请参阅自定义和 AWS Marketplace connectionType 值）
marketplace.*	Spark、Athena 或 JDBC 数据存储（请参阅自定义和 AWS Marketplace connectionType 值）

DynamoDB 连接

您可以使用 AWS Glue for Spark 在 AWS Glue 中读取和写入 DynamoDB 中的表。您可以使用附加到您的 AWS Glue 任务的 IAM 权限连接到 DynamoDB。AWS Glue 支持将数据写入其他 AWS 账户的 DynamoDB 表。有关更多信息，请参阅 the section called “跨账户、跨区域访问 DynamoDB 表”。

除了 AWS Glue DynamoDB ETL 连接器外，您还可以使用 DynamoDB 导出连接器从 DynamoDB 读取数据，该连接器调用 DynamoDB `ExportTableToPointInTime` 请求并将其存储在您提供的 AmazonS3 位置，格式为 DynamoDB JSON。AWS Glue然后，通过从 AmazonS3 导出位置读取数据来创建 DynamicFrame 对象。

DynamoDB 写入器在 AWS Glue 版本 1.0 或更高版本中可用。AWS Glue DynamoDB 导出连接器在 AWS Glue 版本 2.0 或更高版本中可用。

有关 DynamoDB 的更多信息，请参阅 Amazon DynamoDB 文档。

> **Note**
>
> DynamoDB ETL 读取器不支持筛选条件或下推谓词。

配置 DynamoDB 连接

要从 AWS Glue 连接到 DynamoDB，请授予与您的 AWS Glue 任务关联的 IAM 角色与 DynamoDB 交互的权限。有关从 DynamoDB 读取或写入所需权限的更多信息，请参阅 IAM 文档中的 DynamoDB 的操作、资源和条件键。

在以下情况下，您可能需要额外的配置：

- 使用 DynamoDB 导出连接器时，您需要配置 IAM，以便您的任务可以请求 DynamoDB 表导出。此外，您还需要为导出标识一个 AmazonS3 桶，并在 IAM 中提供适当的权限，以便 DynamoDB 向其写入，以及 AWS Glue 任务从中读取。有关更多信息，请参阅在 DynamoDB 中请求表导出。
- 如果您的 AWS Glue 任务有特定的 Amazon VPC 连接要求，请使用 NETWORK AWS Glue 连接类型提供网络选项。由于对 DynamoDB 的访问由 IAM 授权，因此无需使用 AWS Glue DynamoDB 连接类型。

从 DynamoDB 读取和写入 DynamoDB

以下代码示例演示了如何从 DynamoDB 表中读取（通过 ETL 连接器）以及向其写入数据。它们演示了如何从一个表读取数据并将数据写入其他表。

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 导出连接器

在 DynamoDB 表大小超过 80 GB 时，导出连接器的性能优于 ETL 连接器。此外，鉴于导出请求在 AWS Glue 任务中的 Spark 进程之外执行，您可以启用 AWS Glue 任务的弹性伸缩以节省导出请求期间的 DPU 使用量。借助导出连接器，您也无需为 Spark 执行程序并行度或 DynamoDB 吞吐量读取百分比配置拆分数。

> **Note**
>
> DynamoDB 对调用 `ExportTableToPointInTime` 请求有特定的要求。有关更多信息，请参阅在 DynamoDB 中请求表导出。例如，表需要启用时间点恢复 (PITR) 才能使用此连接器。DynamoDB 连接器还支持在将 DynamoDB 导出到 Amazon S3 时进行 AWS KMS 加密。在 AWS Glue 任务配置中指定安全性配置，将为 DynamoDB 导出启用 AWS KMS 加密。KMS 密钥必须与 Simple Storage Service (Amazon S3) 存储桶位于同一区域。
> 请注意，您需要支付 DynamoDB 导出的额外费用和 Simple Storage Service (Amazon S3) 存储成本。任务运行完成后，Simple Storage Service (Amazon S3) 中的导出数据仍然存在，因此您无需其他 DynamoDB 导出即可重复使用这些数据。使用此连接器的一个要求是该表启用了时间点恢复 (PITR)。
> DynamoDB ETL 连接器或导出连接器不支持在 DynamoDB 源应用筛选条件或下推谓词。

以下代码示例演示如何进行读取（通过导出连接器）以及打印分区数量。

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()
  }

}
```

以下示例演示如何从具有 dynamodb 分类的 AWS Glue 数据目录表进行读取（通过导出连接器）以及打印分区数量：

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())
)
```

简化 DynamoDB 导出 JSON 的使用

使用 AWS Glue DynamoDB 导出连接器进行 DynamoDB 导出时会生成具有特定嵌套结构的 JSON 文件。有关更多信息，请参阅数据对象。AWS Glue 提供 DynamicFrame 转换，可以将这些结构解除嵌套，成为易于下游应用程序使用的形式。

您可以通过以下两种方式之一调用该转换。在调用要从 DynamoDB 读取的方法时，您可以使用值 `"true"` 来设置连接选项 `"dynamodb.simplifyDDBJson"`。您也可以将转换作为 AWS Glue 库中独立提供的方法进行调用。

考虑 DynamoDB 导出生成的以下架构：

```
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` 转换会将其简化为：

```
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**
>
> simplifyDDBJson 在 AWS Glue 3.0 及更高版本中可用。unnestDDBJson 转换还可用于简化 DynamoDB 导出 JSON。我们鼓励用户从 unnestDDBJson 转换到 simplifyDDBJson。

在 DynamoDB 操作中配置并行性

为了提高性能，可以调整可用于 DynamoDB 连接器的某些参数。在调整并行性参数时，您的目标是最大限度地使用预配置的 AWS Glue 工作线程。然后，如果您需要更高的性能，我们建议您增加 DPU 数量以横向扩展任务。

使用 ETL 连接器时，可以使用 dynamodb.splits 参数更改 DynamoDB 读取操作中的并行性。使用导出连接器进行读取时，不需要为 Spark 执行程序并行度配置拆分数。您可以使用 dynamodb.output.numParallelTasks 更改 DynamoDB 写入操作中的并行度。

使用 DynamoDB ETL 连接器读取

我们建议您根据任务配置中设置的最大工作线程数和以下 numSlots 计算来计算 dynamodb.splits。如果自动扩缩，则在该上限之下，实际可用的工作线程数量可能会发生变化。有关设置最大工作线程数量的更多信息，请参阅 the section called “配置 Spark 作业属性” 中的工作线程数 (NumberOfWorkers)。

- numExecutors = NumberOfWorkers - 1

  就上下文而言，为 Spark 驱动程序保留了一个执行程序；其他执行程序用于处理数据。
- numSlotsPerExecutor =

  AWS Glue 3.0 and later versions
  - 4，如果 WorkerType 为 G.1X
  - 8，如果 WorkerType 为 G.2X
  - 16，如果 WorkerType 为 G.4X
  - 32，如果 WorkerType 为 G.8X

  AWS Glue 2.0 and legacy versions
  - 8，如果 WorkerType 为 G.1X
  - 16，如果 WorkerType 为 G.2X

- numSlots = numSlotsPerExecutor * numExecutors

我们建议您将 dynamodb.splits 设置为可用插槽数量 numSlots。

写入 DynamoDB

dynamodb.output.numParallelTasks 参数用于通过以下计算确定每个 Spark 任务的 WCU：

```
permittedWcuPerTask = ( TableWCU * dynamodb.throughput.write.percent ) / dynamodb.output.numParallelTasks
```

如果配置准确地表示写入 DynamoDB 的 Spark 任务的数量，则 DynamoDB 写入器将发挥最佳作用。在某些情况下，您可能需要覆盖默认计算以提高写入性能。如果不指定此参数，则每个 Spark 任务允许的 WCU 将自动通过以下公式计算：

-
  - numPartitions = dynamicframe.getNumPartitions()
  - numSlots（如本节前面所定义）
  - numParallelTasks = min(numPartitions, numSlots)
- 示例 1。DPU=10，WorkerType=Standard。输入 DynamicFrame 具有 100 个 RDD 分区。
  - numPartitions = 100
  - numExecutors = (10 - 1) * 2 - 1 = 17
  - numSlots = 4 * 17 = 68
  - numParallelTasks = min(100, 68) = 68
- 示例 2。DPU=10，WorkerType=Standard。输入 DynamicFrame 具有 20 个 RDD 分区。
  - numPartitions = 20
  - numExecutors = (10 - 1) * 2 - 1 = 17
  - numSlots = 4 * 17 = 68
  - numParallelTasks = min(20, 68) = 20

> **Note**
>
> 旧版 AWS Glue 上的任务和使用标准工作线程的任务需要不同的方法来计算插槽数量。如果您需要对这些任务进行性能调整，我们建议您转换到支持的 AWS Glue 版本。

DynamoDB 连接选项参考

指定与 Amazon DynamoDB 的连接。

源连接和接收器连接的连接选项不同。

"connectionType": "dynamodb" with the ETL connector as Source

在使用 AWS Glue DynamoDB ETL 连接器时，请使用以下连接选项并将 `"connectionType": "dynamodb"` 作为源：

- `"dynamodb.input.tableName"`：（必需）要从中读取数据的 DynamoDB 表格。
- `"dynamodb.throughput.read.percent"`：（可选）要使用的读取容量单位 (RCU) 的百分比。默认设置为“0.5”。可接受的值从“0.1”到“1.5”，包含这两个值。
  - `0.5` 表示默认读取速率，这意味着 AWS Glue 将尝试占用表的一半的读取容量。如果增加值超过 `0.5`，AWS Glue 将增加请求速率；将值降低到 `0.5` 以下将降低读取请求速率。（实际读取速率取决于 DynamoDB 表中是否存在统一键分配的等因素。）
  - 当 DynamoDB 表处于按需模式时，AWS Glue 处理表的读取容量为 40000。要导出大型表，我们建议您将 DynamoDB 表切换为按需模式。
- `"dynamodb.splits"`：（可选）定义在读取时将此 DynamoDB 表分成多少个部分。默认设置为“1”。可接受的值从“1”到“1,000,000”，包含这两个值。

  1 表示没有并行度。我们强烈建议您使用以下公式指定更大的值以获得更好的性能。有关适当设置值的更多信息，请参阅 the section called “DynamoDB 并行性”。
- `"dynamodb.sts.roleArn"`：（可选）用于跨账户访问的 IAM 角色 ARN。此参数适用于 AWS Glue 1.0 或更高版本。
- `"dynamodb.sts.roleSessionName"`：（可选）STS 会话名称。默认设置为“glue-dynamodb-read-sts-session”。此参数适用于 AWS Glue 1.0 或更高版本。

"connectionType": "dynamodb" with the AWS Glue DynamoDB export connector as Source

在使用 AWS Glue DynamoDB 导出连接器（仅适用于 AWS Glue 版本 2.0 以上）时，使用以下连接选项并将 "connectionType": "dynamodb"用作源：

- `"dynamodb.export"`：（必需）字符串值：
  - 如果设置为 `ddb`，将启用 AWS Glue DynamoDB 导出连接器，其中在 AWS Glue 任务期间将调用新的 `ExportTableToPointInTimeRequest`。新的导出将通过从 `dynamodb.s3.bucket` 和 `dynamodb.s3.prefix` 传递的位置生成。

  - 如果设置为 s3，将启用 AWS Glue DynamoDB 导出连接器但会跳过创建新的 DynamoDB 导出，而使用 dynamodb.s3.bucket 和 dynamodb.s3.prefix 作为该表以前导出的 Simple Storage Service (Amazon S3) 位置。
- "dynamodb.tableArn"：（必需）要从中读取数据的 DynamoDB 表格。
- "dynamodb.unnestDDBJson"：（可选）默认值：false。有效值：布尔值。如果设置为 true（真），则对导出中存在的 DynamoDB JSON 结构执行解除嵌套转换。同时将 "dynamodb.unnestDDBJson" 和 "dynamodb.simplifyDDBJson" 设置为 true 是错误的。在 AWS Glue 3.0 及更高版本中，我们建议您在简化 DynamoDB Map 类型时使用 "dynamodb.simplifyDDBJson" 以获得更好的行为。有关更多信息，请参阅 the section called “简化 DynamoDB 导出 JSON 的使用”。
- "dynamodb.simplifyDDBJson"：（可选）默认值：false。有效值：布尔值。如果设置为 true，则执行转换以简化导出中存在的 DynamoDB JSON 结构的架构。这与 "dynamodb.unnestDDBJson" 选项的目的相同，但为 DynamoDB 表中的 DynamoDB Map 类型甚至嵌套 Map 类型提供了更好的支持。此选项在 AWS Glue 3.0 及更高版本中可用。同时将 "dynamodb.unnestDDBJson" 和 "dynamodb.simplifyDDBJson" 设置为 true 是错误的。有关更多信息，请参阅 the section called “简化 DynamoDB 导出 JSON 的使用”。
- "dynamodb.s3.bucket"：（可选）指示将会执行 DynamoDB ExportTableToPointInTime 进程的 Amazon S3 存储桶位置。导出的文件格式为 DynamoDB JSON。
  - "dynamodb.s3.prefix": (可选）指示将用于存储 DynamoDB ExportTableToPointInTime 负载的 Amazon S3 存储桶内的 Amazon S3 前缀位置。如果既未指定 dynamodb.s3.prefix，也未指定 dynamodb.s3.bucket，则这些值将默认为 AWS Glue 任务配置中指定的临时目录位置。有关更多信息，请参阅 AWS Glue 使用的特殊参数。
  - "dynamodb.s3.bucketOwner"：指示跨账户 Amazon S3 访问所需的存储桶拥有者。
- "dynamodb.sts.roleArn"：（可选）跨账户访问和/或跨区域访问 DynamoDB 表时将会代入的 IAM 角色 ARN。注意：相同的 IAM 角色 ARN 将用于访问为 ExportTableToPointInTime 请求指定的 Amazon S3 位置。
- "dynamodb.sts.roleSessionName"：（可选）STS 会话名称。默认设置为“glue-dynamodb-read-sts-session”。
- "dynamodb.exportTime"（可选）有效值：表示 ISO-8601 瞬时的字符串。应进行导出的时间点。
- "dynamodb.sts.region"：（如果使用区域端点进行跨区域调用，则为必填项）托管要读取的 DynamoDB 表的区域。

"connectionType": "dynamodb" with the ETL connector as Sink

将 "connectionType": "dynamodb" 用作连接器时可使用以下连接选项：

- "dynamodb.output.tableName"：（必需）要写入的 DynamoDB 表。
- "dynamodb.throughput.write.percent"：（可选）要使用的写入容量单位（WCU）的百分比。默认设置为“0.5”。可接受的值从“0.1”到“1.5”，包含这两个值。
  - 0.5 表示默认写入速率，这意味着 AWS Glue 将尝试占用表的一半的写入容量。如果增加值超过 0.5，AWS Glue 将增加请求速率；将值降低到 0.5 以下将降低写入请求速率。（实际写入速率取决于 DynamoDB 表中是否存在统一键分配等因素）。
  - 当 DynamoDB 表处于按需模式时，AWS Glue 处理表的写入容量为 40000。要导入大型表，我们建议您将 DynamoDB 表切换为按需模式。
- "dynamodb.output.numParallelTasks"：（可选）定义同时向 DynamoDB 写入数据的并行任务数。用于计算每个 Spark 任务的允许 WCU。在大多数情况下，AWS Glue 将为此值计算合理的默认值。有关更多信息，请参阅 the section called “DynamoDB 并行性”。
- "dynamodb.output.retry"：（可选）定义存在 DynamoDB 中的 ProvisionedThroughputExceededException 时我们执行的重试次数。默认设置为“10”。
- "dynamodb.sts.roleArn"：（可选）用于跨账户访问的 IAM 角色 ARN。
- "dynamodb.sts.roleSessionName"：（可选）STS 会话名称。默认设置为“glue-dynamodb-write-sts-session”。

跨账户、跨区域访问 DynamoDB 表

AWS Glue ETL 任务支持跨账户、跨区域访问 DynamoDB 表。AWS GlueETL 任务支持从其他 AWS 账户的 DynamoDB 表读取数据，并将数据写入其他 AWS 账户的 DynamoDB 表。AWS Glue 还支持从其他区域的 DynamoDB 表读取数据并将数据写入其他区域的 DynamoDB 表。本部分提供关于设置访问的说明以及示例脚本。

本部分的过程参考了用于创建 IAM 角色和授予角色访问权限的 IAM 教程。本教程还讨论了担任角色，但在此处，您将使用任务脚本来担任 AWS Glue 中的角色。本教程还包含有关常规跨账户实践的信息。有关更多信息，请参阅《IAM 用户指南》中的教程：使用 IAM 角色委派跨 AWS 账户的访问权限。

创建角色

按照教程中的步骤 1 在账户 A 中创建 IAM 角色。在定义角色的权限时，您可以选择附加现有策略（例如 AmazonDynamoDBReadOnlyAccess 或 AmazonDynamoDBFullAccess）以允许角色读

取/写入 DynamoDB。以下示例演示了如何使用权限策略 AmazonDynamoDBFullAccess 创建名为 DynamoDBCrossAccessRole 的角色。

向角色授予访问权限

遵循《IAM 用户指南》中的教程中的步骤 2，允许账户 B 切换到新创建的角色。以下示例使用以下语句创建新策略：

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": "<DynamoDBCrossAccessRole's ARN>"
    }
}
```

然后，您可以将此策略附加到要用于访问 DynamoDB 的组/角色/用户。

担任 AWS Glue 作业脚本中的角色

现在，您可以登录账户 B 并创建 AWS Glue 任务。要创建任务，请参阅在中配置 Spark 作业的作业属性 AWS Glue中的说明。

在任务脚本中，您需要使用 dynamodb.sts.roleArn 参数担任 DynamoDBCrossAccessRole 角色。假设此角色允许您获取临时凭证，这些凭证需要用于访问账户 B 中的 DynamoDB。请查看这些示例脚本。

对于跨区域跨账户读取（ETL 连接器）：

```
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()
```

对于跨区域跨账户读取（ELT 连接器）：

```
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()
```

对于跨区域的读取和跨账户写入：

```
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 连接

您可以使用存储在 Data Catalog 表中的信息或通过提供信息直接访问数据流来读取 Amazon Kinesis Data Streams 或向其中写入数据。你可以从 Kinesis 读取信息到 Spark 中 DataFrame，然后将其转换为 G AWS lue。 DynamicFrame你可以用 JSON 格式写入 DynamicFrames Kinesis。如果直接访问数据流，请使用这些选项提供有关如何访问数据流的信息。

如果您使用 `getCatalogSource` 或 `create_data_frame_from_catalog` 使用来自 Kinesis 流式处理源的记录，则任务具有数据目录数据库和表名称信息，将其用于获取一些基本参数，以便从 Kinesis 流式处理源读取数据。如果使用 `getSource`、`getSourceWithFormat`、`createDataFrameFromOptions` 或 `create_data_frame_from_options`，则必须使用此处描述的连接选项指定这些基本参数。

您可以使用 `GlueContext` 类中指定方法的以下参数为 Kinesis 指定连接选项。

- Scala
  - `connectionOptions`：与 `getSource`、`createDataFrameFromOptions`、`getSink` 结合使用

  - `additionalOptions`：与 `getCatalogSource`、`getCatalogSink` 结合使用
  - `options`：与 `getSourceWithFormat`、`getSinkWithFormat` 结合使用
- Python
  - `connection_options`：与 `create_data_frame_from_options`、`write_dynamic_frame_from_options` 结合使用
  - `additional_options`：与 `create_data_frame_from_catalog`、`write_dynamic_frame_from_catalog` 结合使用
  - `options`：与 `getSource`、`getSink` 结合使用

有关流式处理 ETL 作业的注意事项和限制，请参阅 the section called “串流 ETL 注释和限制”。

配置 Kinesis

要在 G AWS lue Spark 作业中连接到 Kinesis 数据流，你需要一些先决条件：

- 如果读取， AWS Glue 作业必须具有对 Kinesis 数据流的读取访问权限级别 IAM 权限。
- 如果要写入，则 AWS Glue 作业必须对 Kinesis 数据流具有写入访问级别 IAM 权限。

在某些情况下，您需要配置其他先决条件：

- 如果您的 AWS Glue 任务配置了其他网络连接（通常用于连接到其他数据集），并且其中一个连接提供了 Amazon VPC 网络选项，则这将引导您的任务通过 Amazon VPC 进行通信。在这种情况下，您还需要将 Kinesis 数据流配置为通过 Amazon VPC 进行通信。您可以通过在 Amazon VPC 和 Kinesis 数据流之间创建接口 VPC 端点实现此目的。有关更多信息，请参阅 Using Kinesis Data Streams with Interface VPC Endpoints。
- 在另一个账户中指定 Amazon Kinesis Data Streams 时，您必须设置角色和策略，从而允许跨账户访问。有关更多信息，请参阅示例：从不同账户的 Kinesis 串流中读取。

有关流式处理 ETL 作业先决条件的更多信息，请参阅 the section called “流式处理 ETL 作业”。

示例：从 Kinesis 流读取

示例：从 Kinesis 流读取

与 the section called “forEachBatch” 结合使用。

Amazon Kinesis 流式处理源示例：

```
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)
```

示例：写入 Kinesis 流

示例：从 Kinesis 流读取

与 the section called “forEachBatch” 结合使用。

Amazon Kinesis 流式处理源示例：

```
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 连接选项参考

为 Amazon Kinesis Data Streams 指定连接选项。

为 Kinesis 流式处理数据源使用以下连接选项：

- `"streamARN"`（必填）用于读/写。Kinesis 数据流的 ARN。
- `"classification"`（读取所必填）用于读取。记录中数据使用的文件格式。除非 Data Catalog 提供，否则为必需。
- `"streamName"` –（可选）用于读取。要从中读取的 Kinesis 数据流的名称。与 `endpointUrl` 一起使用。
- `"endpointUrl"` –（可选）用于读取。默认：“https://kinesis.us-east-1.amazonaws.com”。Kinesis 直播的 AWS 端点。除非要连接到特殊区域，否则您无需对此进行更改。

- "partitionKey" – (可选) 用于写入。生成记录时所使用的 Kinesis 分区键。
- "delimiter" (可选) 用于读取。当 classification 为 CSV 时使用的值分隔符。默认为“,”。
- "startingPosition" : (可选) 用于读取。要从中读取数据的 Kinesis 数据流中的起始位置。可能的值是 "latest"、"trim_horizon"、"earliest" 或以模式 yyyy-mm-ddTHH:MM:SSZ 采用 UTC 格式的时间戳字符串 (其中 Z 表示带有 +/-的 UTC 时区偏移量。例如 : “2023-04-04T08:00:00-04:00”) 。默认值为 "latest"。注意 : 只有 G "startingPosition" AWS lue 版本 4.0 或更高版本支持 UTC 格式的时间戳字符串。
- "failOnDataLoss" : (可选) 如果有任何活动分片丢失或已过期，则作业失败。默认值为 "false"。
- "awsSTSRoleARN" : (可选) 用于读取/写入。使用 () 代入角色的亚马逊资源名称 AWS Security Token Service (AWS STS ARN)。此角色必须拥有针对 Kinesis 数据流执行描述或读取记录操作的权限。在访问其他账户中的数据流时，必须使用此参数。与 "awsSTSSessionName" 结合使用。
- "awsSTSSessionName" : (可选) 用于读取/写入。使用 AWS STS代入角色的会话的标识符。在访问其他账户中的数据流时，必须使用此参数。与 "awsSTSRoleARN" 结合使用。
- "awsSTSEndpoint": (可选) 使用代入角色连接到 Kinesis 时要使用的 AWS STS 端点。这允许在 VPC 中使用区域 AWS STS 终端节点，而默认的全局终端节点是不可能的。
- "maxFetchTimeInMs" : (可选) 用于读取。作业执行程序从 Kinesis 数据流中读取当前批处理记录所花费的最长时间，以毫秒为单位指定。在这段时间内可以进行多次 GetRecords API 调用。默认值为 1000。
- "maxFetchRecordsPerShard" : (可选) 用于读取。每个微批次将从 Kinesis 数据流中的每个分片获取的最大记录数。注意 : 如果流式传输作业已经从 Kinesis 读取了额外的记录 (在同一个 get-records 调用中) ，则客户端可以超过此限制。如果 maxFetchRecordsPerShard 需要严格，则必须是 maxRecordPerRead 的整数倍。默认值为 100000。
- "maxRecordPerRead" : (可选) 用于读取。每项 getRecords 操作中要从 Kinesis 数据流获取的最大记录数。默认值为 10000。
- "addIdleTimeBetweenReads" : (可选) 用于读取。在两项连续 getRecords 操作之间添加时间延迟。默认值为 "False"。此选项仅适用于 Glue 版本 2.0 及更高版本。
- "idleTimeBetweenReadsInMs" : (可选) 用于读取。两项连续 getRecords 操作之间的最短时间延迟，以毫秒为单位。默认值为 1000。此选项仅适用于 Glue 版本 2.0 及更高版本。
- "describeShardInterval" : (可选) 用于读取。两个 ListShards API 调用之间的最短时间间隔，供您的脚本考虑重新分区。有关更多信息，请参阅《Amazon Kinesis Data Streams 开发人员指南》中的重新分区策略。默认值为 1s。
- "numRetries" : (可选) 用于读取。Kinesis Data Streams API 请求的最大重试次数。默认值为 3。

- "retryIntervalMs"：（可选）用于读取。重试 Kinesis Data Streams API 调用之前的冷却时间（以毫秒为单位指定）。默认值为 1000。
- "maxRetryIntervalMs"：（可选）用于读取。Kinesis Data Streams API 调用的两次重试之间的最长冷却时间（以毫秒为单位指定）。默认值为 10000。
- "avoidEmptyBatches"：（可选）用于读取。在批处理开始之前检查 Kinesis 数据流中是否有未读数据，避免创建空白微批处理任务。默认值为 "False"。
- "schema"：（当 inferSchema 设为 false 时为必填）用于读取。用于处理有效负载的架构。如果分类为 avro，则提供的架构必须采用 Avro 架构格式。如果分类不是 avro，则提供的架构必须采用 DDL 架构格式。

  以下是一些架构示例。

  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"：（可选）用于读取。默认值为‘false’。如果设置为“true”，则会在运行时检测到 foreachbatch 内的有效工作负载中的架构。
- "avroSchema"：（已弃用）用于读取。用于指定 Avro 数据架构（使用 Avro 格式时）的参数。此参数现已被弃用。使用 schema 参数。
- "addRecordTimestamp"：（可选）用于读取。当选项设置为 'true' 时，数据输出将包含一个名为 "__src_timestamp" 的附加列，表示数据流收到相应记录的时间。默认值为‘false’。4.0 或更高 AWS Glue 版本支持此选项。
- "emitConsumerLagMetrics"：（可选）用于读取。当该选项设置为 “true” 时，对于每个批次，它将发出从直播收到的最旧记录到它到达的时间之间的持续时间内的AWS Glue指标。 CloudWatch 该指标的名字是 “glue.driver.streaming”。 maxConsumerLagInMs”。默认值为‘false’。4.0 或更高 AWS Glue 版本支持此选项。
- "fanoutConsumerARN"：（可选）用于读取。streamARN 中指定的流的 Kinesis 流用户的 ARN。用于为您的 Kinesis 连接启用增强型扇出功能模式。有关使用增强型扇出功能的 Kinesis 流的更多信息，请参阅 the section called “在 Kinesis 流作业中使用增强型扇出功能”。
- "recordMaxBufferedTime" –（可选）用于写入。默认值：1000（ms）。记录在等待写入时缓冲的最长时间。
- "aggregationEnabled" –（可选）用于写入。默认值：真。指定是否应在将记录发送到 Kinesis 之前对其进行汇总。
- "aggregationMaxSize" –（可选）用于写入。默认值：51200（字节）。如果记录超过了此限制，则它将绕过聚合器。注意：Kinesis 将记录大小限制为 50KB。如果您将其设置为 50KB 以上，Kinesis 将拒绝过大的记录。
- "aggregationMaxCount" –（可选）用于写入。默认值：4294967295。要打包至汇总记录的最大项目数量。
- "producerRateLimit" –（可选）用于写入。默认值：150（%）。限制从单个生产者（例如您的作业）发送的每个分片的吞吐量，以占后端限制的百分比表示。
- "collectionMaxCount" –（可选）用于写入。默认值：500。一个 PutRecords 请求中要打包的最大物品数量。
- "collectionMaxSize" –（可选）用于写入。默认值：5242880（字节）。与 PutRecords 请求一起发送的最大数据量。

在 Kinesis 流作业中使用增强型扇出功能

使用增强型扇出功能的用户能够接收来自 Kinesis 流的记录，其专用吞吐量可能高于普通用户。这是通过优化用于向 Kinesis 用户（例如您的作业）提供数据的传输协议来完成的。有关 Kinesis 增强型扇出功能的更多信息，请参阅 Kinesis 文档。

在增强型扇出功能模式下，`maxRecordPerRead` 和 `idleTimeBetweenReadsInMs` 连接选项不再适用，因为使用增强型扇出功能时这些参数不可配置。重试的配置选项按所述执行。

使用以下步骤为您的流作业启用和禁用增强型扇出功能。应为每项将消耗流中数据的作业注册一个流用户。

要在作业中启用增强型扇出功能使用，请执行以下操作：

1. 使用 Kinesis API 为作业注册流用户。按照说明，使用 Kinesis 文档中的使用 Kinesis Data Streams API 为用户注册增强型扇出功能。只需要按照第一步进行操作，即调用 RegisterStreamConsumer。您的请求应返回 ARN *consumerARN*。
2. 在连接方法参数中将连接选项 `fanoutConsumerARN` 设置为 *consumerARN*。
3. 重新启动作业。

要在作业中禁用增强型扇出功能使用，请执行以下操作：

1. 从方法调用中移除 `fanoutConsumerARN` 连接选项。
2. 重新启动作业。
3. 按照 Kinesis 文档中的说明，取消注册用户。这些说明适用于控制台，但也可以通过 Kinesis API 来实现。有关通过 Kinesis API 取消注册流用户的更多信息，请查阅 Kinesis 文档中的 DeregisterStreamConsumer。

Amazon S3 连接

您可以使用 AWS Glue for Spark 在 Amazon S3 中读取和写入文件。AWSGlue for Spark 支持许多开箱即用的存储在 Amazon S3 中的常见数据格式，包括 CSV、Avro、JSON、Orc 和 Parquet。有关支持的数据格式的更多信息，请参阅 the section called “数据格式选项”。每种数据格式可能支持不同的 AWS Glue 功能集。有关功能支持的细节，请查阅您的数据格式页面。此外，您可以读取和写入存储在 Hudi、Iceberg 和 Delta Lake 数据湖框架中的版本控制文件。有关数据湖框架的更多信息，请参阅 the section called “数据湖框架”。

使用 AWS Glue，您可以在写入时将 Amazon S3 对象分成文件夹结构，然后使用简单的配置按分区检索以提高性能。您还可以设置配置，以便在转换数据时将小文件分组在一起以提高性能。您可以在 Amazon S3 中进行读写 `bzip2` 和 `gzip` 存档。

主题

配置 S3 连接

要在 AWS Glue with Spark 作业中连接到 Amazon S3，需要具备一些先决条件：

- AWS Glue 作业必须拥有对相关 Amazon S3 存储桶的 IAM 权限。

在某些情况下，您需要配置其他先决条件：

- 配置跨账户访问时，需要对 Amazon S3 存储桶进行适当的访问控制。
- 出于安全考虑，您可以选择通过 Amazon VPC 路由您的 Amazon S3 请求。此方法可能会带来带宽和可用性方面的难题。有关更多信息，请参阅 the section called “适用于 Amazon S3 的 Amazon VPC 终端节点”。

Amazon S3 连接选项参考

指定与 Amazon S3 的连接。

由于 Amazon S3 管理文件而不是表，因此除了指定本文档中提供的连接属性外，您还需要指定有关文件类型的额外配置。您可以通过数据格式选项来指定此信息。有关格式选项的更多信息，请参阅 the section called “数据格式选项”。您也可以通过与 AWS Glue Data Catalog 集成来指定此信息。

有关连接选项和格式选项之间区别的示例，请考虑 the section called “create_dynamic_frame_from_options” 方法如何采用

connection_type、connection_options、format 和 format_options。本节专门讨论提供给 connection_options 的参数。

"connectionType": "s3" 可使用以下连接选项：

- "paths"：（必需）要从中读取数据的 Amazon S3 路径的列表。
- "exclusions"：（可选）包含要排除的 Unix 样式 glob 模式的 JSON 列表的字符串。例如，"[\"**.pdf\"]" 会排除所有 PDF 文件。有关 AWS Glue 支持的 glob 语法的更多信息，请参阅包含和排除模式。
- "compressionType" 或 "compression"：（可选）指定数据压缩方式。使用适用于 Amazon S3 源的 "compressionType" 以及适用于 Amazon S3 目标的 "compression"。通常，如果数据有标准文件扩展名，则不需要指定。可能的值为 "gzip" 和 "bzip2"。特定格式可能支持其他压缩格式。有关功能支持的细节，请查阅数据格式页面。
- "groupFiles"：（可选）当输入包含超过 50,000 个文件时，默认启用文件分组。当少于 50,000 个文件时，若要启用分组，请将此参数设置为 "inPartition"。当超过 50,000 个文件时，若要禁用分组，请将此参数设置为 "none"。
- "groupSize"：（可选）目标组大小（以字节为单位）。默认值根据输入数据大小和群集大小进行计算。当少于 50,000 个输入文件时，"groupFiles" 必须设置为 "inPartition"，此选项才能生效。
- "recurse"：（可选）如果设置为 true，则以递归方式读取指定路径下的所有子目录中的文件。
- "maxBand"：（可选，高级）此选项控制 s3 列表可能保持一致的持续时间（以毫秒为单位）。当使用 JobBookmarks 来表明 Amazon S3 最终一致性时，将专门跟踪修改时间戳在最后 maxBand 毫秒内的文件。大多数用户不需要设置此选项。默认值为 900000 毫秒或 15 分钟。
- "maxFilesInBand"：（可选，高级）此选项指定在最后 maxBand 秒内可保存的最大文件数量。如果超过此值，额外的文件将会跳过，且只能在下一次作业运行中处理。大多数用户不需要设置此选项。
- "isFailFast"：（可选）此选项用于确定 AWS Glue ETL 任务是否导致读取器解析异常。如果设置为 true，并且 Spark 任务的四次重试无法正确解析数据，则任务会快速失败。
- "catalogPartitionPredicate"：（可选）用于读取。SQL WHERE 子句的内容。从具有大量分区的 Data Catalog 表中读取时使用。从 Data Catalog 索引中检索匹配的分区。与 push_down_predicate 一起使用，the section called “create_dynamic_frame_from_catalog” 方法（以及其他类似方法）上的一个选项。有关更多信息，请参阅 the section called “目录分区谓词”。
- "partitionKeys"：（可选）用于写入。列标签字符串数组。AWSGlue 将按照此配置的指定对您的数据进行分区。有关更多信息，请参阅 the section called “写入分区”。

- "excludeStorageClasses"：（可选）用于读取。指定 Amazon S3 存储类的字符串数组。AWSGlue 将根据此配置排除 Amazon S3 对象。有关更多信息，请参阅 the section called “排除 Amazon S3 存储类”。

已弃用的数据格式连接语法

某些数据格式可以使用特定的连接类型语法进行访问。此语法已被弃用。我们建议您改用 the section called “数据格式选项” 中提供的 s3 连接类型和格式选项来指定格式。

"connectionType": "orc"

指定与 Amazon S3 中以 Apache Hive 优化的行列式（ORC）文件格式存储的文件的连接。

"connectionType": "orc" 可使用以下连接选项：

- paths：（必需）要从中读取数据的 Amazon S3 路径的列表。
- （其他选项名称/值对）：任何其他选项（包括格式化选项）将直接传递给 SparkSQL DataSource。

"connectionType": "parquet"

指定与 Amazon S3 中以 Apache Parquet 文件格式存储的文件的连接。

"connectionType": "parquet" 可使用以下连接选项：

- paths：（必需）要从中读取数据的 Amazon S3 路径的列表。
- （其他选项名称/值对）：任何其他选项（包括格式化选项）将直接传递给 SparkSQL DataSource。

排除 Amazon S3 存储类

如果您要运行从 Amazon Simple Storage Service（Amazon S3）读取文件或分区的 AWS Glue ETL 任务，则可以排除某些 Amazon S3 存储类类型。

Amazon S3 中提供以下存储类：

- STANDARD – 用于频繁访问数据的通用存储。
- INTELLIGENT_TIERING – 用于具有未知或访问模式不断变化的数据。

- STANDARD_IA 和 ONEZONE_IA – 用于长期存在但不常访问的数据。
- GLACIER、DEEP_ARCHIVE 和 REDUCED_REDUNDANCY – 用于长期归档和数字化保存。

有关更多信息，请参阅《Amazon S3 开发人员指南》中的 Amazon S3 存储类。

本节中的示例向您展示了如何排除 GLACIER 和 DEEP_ARCHIVE 存储类。这些类允许您列出文件，但除非文件已还原，否则它们不会让您读取文件。（有关更多信息，请参阅《Amazon S3 开发人员指南》中的还原存档对象。

通过使用存储类排除，您可以确保您的 AWS Glue 作业可以在具有跨这些存储类层的分区的表上工作。如果没有排除，则从这些层读取数据的作业将失败，并显示以下错误：AmazonS3Exception：操作对于此对象的存储类无效。

有多种不同的方法可供您在 AWS Glue 中筛选 Amazon S3 存储类。

主题

- 创建动态帧时排除 Amazon S3 存储类
- 在数据目录表上排除 Amazon S3 存储类

创建动态帧时排除 Amazon S3 存储类

要在创建动态帧时排除 Amazon S3 存储类，请使用 additionalOptions 中的 excludeStorageClasses。 AWS Glue 会自动使用自己的 Amazon S3 Lister 实施列出和排除与指定存储类对应的文件。

以下 Python 和 Scala 示例显示了在创建动态帧时如何排除 GLACIER 和 DEEP_ARCHIVE 存储类。

Python 示例：

```
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 示例：

```
val* *df = glueContext.getCatalogSource(
    nameSpace, tableName, "", "my_transformation_context",
    additionalOptions = JsonOptions(
        Map("excludeStorageClasses" -> List("GLACIER", "DEEP_ARCHIVE"))
    )
).getDynamicFrame()
```

在数据目录表上排除 Amazon S3 存储类

您可以将 AWS Glue ETL 任务使用的存储类排除指定为 AWS Glue 数据目录中的表参数。您可以使用 AWS Command Line Interface (AWS CLI) 或以编程方式使用 API 在 `CreateTable` 操作中包含此参数。有关更多信息，请参阅 Table 结构和 CreateTable。

您还可以在 AWS Glue 控制台上指定排除的存储类。

排除 Amazon S3 存储类（控制台）

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在左侧的导航窗格中，选择 Tables (表)。
3. 选择列表中的表名称，然后选择 Edit table (编辑表)。
4. 在 Table properties (表属性) 中，添加 **`excludeStorageClasses`** 作为键，添加 **`[\"GLACIER\",\"DEEP_ARCHIVE\"]`** 作为值。
5. 选择 应用。

管理 AWS Glue 中用于 ETL 输出的分区

分区是用于组织数据集以便高效查询数据集的重要技术。它根据一个或多个列的不同值用分层目录结构来组织数据。

例如，您可以决定按日期（包括年、月和日）对 Amazon Simple Storage Service（Amazon S3）中的应用程序日志分区。然后，与某一天的数据相对应的文件将放置在前缀下，例如 `s3://my_bucket/logs/year=2018/month=01/day=23/`。Amazon Athena、Amazon Redshift Spectrum 和 AWS Glue 等系统可以使用这些分区按分区值筛选数据，而无需读取 Amazon S3 中的所有底层数据。

爬网程序不仅推断文件类型和架构，它们还会在填充 AWS Glue 数据目录时自动标识数据集的分区结构。生成的分区列可用于在 AWS Glue ETL 任务或 Amazon Athena 之类的查询引擎中进行查询。

对表进行网络爬取后，您可以查看爬网程序创建的分区。在 AWS Glue 控制台的左侧导航窗格中，选择 Tables (表)。选择爬网程序创建的表，然后选择 View Partitions (查看分区)。

对于采用 key=val 样式的 Apache Hive 风格分区路径，爬网程序会使用键名自动填充列名称。否则，它使用默认名称，如 partition_0、partition_1 等。您可以在控制台上更改默认名称。为此，请导航至该表格。检查索引选项卡下是否存在索引。如果是这样的话，您需要删除它们才能继续（之后您可以使用新的列名重新创建它们）。然后，选择编辑架构，并在那里修改分区列的名称。

然后，在您的 ETL 脚本中，便可以筛选分区列。因为分区信息存储于数据目录，所以使用 from_catalog API 调用包含 DynamicFrame 中的分区列。例如，使用 create_dynamic_frame.from_catalog 而不是 create_dynamic_frame.from_options。

分区是一种可减少数据扫描量的优化技术。要详细了解确定何时适合使用这种技术的过程，请参阅《AWS 规范性指南》中“优化 AWS Glue for Apache Spark 作业性能的最佳实践”指南中的 减少数据扫描量。

使用下推谓词进行预筛选

在许多情况下，您可以使用下推谓词来筛选分区，而不必列出并读取数据集中的所有文件。您可以直接对数据目录中的分区元数据应用筛选，而不是读取整个数据集，然后在 DynamicFrame 中筛选。这样，只需将您实际需要的内容列出和读取到 DynamicFrame 中即可。

例如，在 Python 中，您可以写入以下内容。

```
glue_context.create_dynamic_frame.from_catalog(
    database = "my_S3_data_set",
    table_name = "catalog_data_table",
    push_down_predicate = my_partition_predicate)
```

这会创建一个 DynamicFrame，它仅在数据目录中加载满足谓词表达式的分区。根据您要加载的数据子集的规模，这样可以节省大量处理时间。

谓词表达式可以是 Spark SQL 支持的任何布尔表达式。您可以在 Spark SQL 查询的 WHERE 子句中放置的任何内容都可以使用。例如，谓词表达式 pushDownPredicate = "(year=='2017' and month=='04')" 仅加载数据目录中 year 等于 2017 并且 month 等于 04 的分区。有关更多信息，请参阅 Apache Spark SQL 文档，尤其是 Scala SQL 函数参考。

使用目录分区谓词进行服务器端筛选

push_down_predicate 选项将在列出目录中的所有分区之后以及列出 Amazon S3 中针对这些分区的文件之前应用。如果您有大量表分区，则目录分区列表仍然会产生额外的时间开销。为解决这一开

销，您可以结合使用服务器端分区修剪和 `catalogPartitionPredicate` 选项，该选项使用 AWS Glue 数据目录中的分区索引。当您在一个表中有数百万个分区时，这样可以提高分区筛选速度。如果您的 `catalogPartitionPredicate` 需要目录分区索引尚不支持的谓词语法，您可以结合使用 `push_down_predicate` 与 `additional_options` 中的 `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` 使用不同的语法。前者使用 Spark SQL 标准语法，后者使用 JSQL 解析器。

写入分区

默认情况下，DynamicFrame 在写入时不分区。所有输出文件都写入指定输出路径的顶级。直到最近，将 DynamicFrame 写入分区的唯一途径是在写入之前将其转换为 Spark SQL DataFrame。

但是，DynamicFrames 现在支持您在创建接收器时使用 `partitionKeys` 选项通过密钥序列进行本机分区。例如，以下 Python 代码将数据集以 Parquet 格式写出到 Amazon S3 中，写入到类型字段分区的目录中。然后，您可以使用其他系统（如 Amazon Athena）处理这些分区。

```
glue_context.write_dynamic_frame.from_options(
    frame = projectedEvents,
    connection_type = "s3",
    connection_options = {"path": "$outpath", "partitionKeys": ["type"]},
    format = "parquet")
```

以较大的组读取输入文件

您可以设置表的属性，以使 AWS Glue ETL 任务能够在从 Amazon S3 数据存储中读取文件时对文件分组。这些属性使每个 ETL 任务可将一组输入文件读取到单个内存分区中，当 Amazon S3 数据存储中存在大量小型文件时，此功能特别有用。当您设置某些属性时，您会指示 AWS Glue 对 Amazon S3 数据分区中的文件分组并设置要读取的组的大小。您还可以在从 Amazon S3 数据存储中读取时用 `create_dynamic_frame.from_options` 方法设置这些选项。

要对表启用文件分组，可在表结构的参数字段中设置键值对。使用 JSON 表示法为表的参数字段设置值。有关编辑表属性的更多信息，请参阅 查看和编辑表详细信息。

您可以使用此方法对数据目录中使用 Amazon S3 数据存储的表启用分组。

groupFiles

将 groupFiles 设置为 `inPartition`，以在 Amazon S3 数据分区中启用文件分组。如果输入文件数多于 50000 个，AWS Glue 会自动启用分组，如以下示例中所示。

```
'groupFiles': 'inPartition'
```

groupSize

将 groupSize 设置为组的目标大小 (以字节为单位)。groupSize 属性为可选属性，如果未提供，AWS Glue 将计算一个大小以使用集群中的所有 CPU 内核，同时仍会减少 ETL 任务和内存分区总数。

例如，以下内容将组大小设置为 1 MB。

```
'groupSize': '1048576'
```

请注意，应使用计算的结果设置 groupsize。例如，1024 * 1024 = 1048576。

recurse

将递归设置为 True，这样在将 paths 指定为路径数组时，会以递归方式读取所有子目录中的文件。如果 paths 是 Amazon S3 中的一组对象密钥，或者输入格式为 parquet/orc，您不需要将其设置为 recurse，如以下示例所示。

```
'recurse':True
```

如果您使用 create_dynamic_frame.from_options 方法直接从 Amazon S3 中读取，请添加这些连接选项。例如，下面尝试将文件分组到 1 MB 组中。

```
df = glueContext.create_dynamic_frame.from_options("s3", {'paths': ["s3://s3path/"],
 'recurse':True, 'groupFiles': 'inPartition', 'groupSize': '1048576'}, format="json")
```

> **Note**
>
> 从以下数据格式创建的 DynamicFrames 支持 groupFiles：csv、ion、grokLog、json 和 xml。avro、parquet 和 orc 不支持此选项。

适用于 Amazon S3 的 Amazon VPC 终端节点

出于安全原因，许多AWS客户在 Amazon Virtual Private Cloud 环境（Amazon VPC）中运行其应用程序。利用 Amazon VPC，您可以在 Virtual Private Cloud 中启动 Amazon EC2 实例，Virtual Private Cloud 在逻辑上与其他网络（包括公共互联网）隔离。利用 Amazon VPC，您可以控制该网络的 IP 地址范围、子网、路由表、网络网关和安全设置。

> **Note**
>
> 如果您的AWS账户是在 2013 年 12 月 4 日之后创建的，则您在每个AWS区域都已经有一个默认 VPC。您无需任何额外配置即能立即开始使用您的默认 VPC。
> 详情请参阅《Amazon VPC 用户指南》中的您的默认 VPC 和子网。

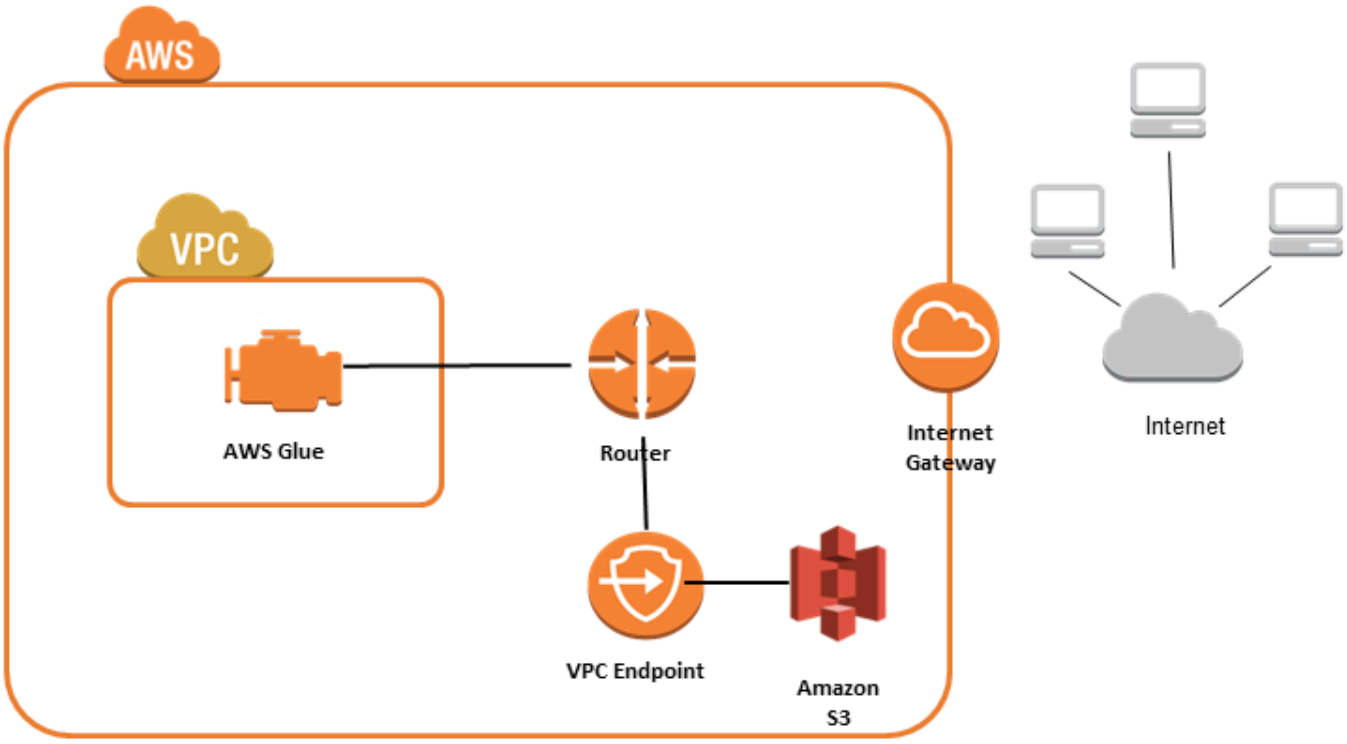
许多客户对跨公共 Internet 发送和接收数据存在合理的私密性和安全性担心。客户可以利用 virtual private network (VPN)，通过其企业网络基础设施路由所有 Amazon S3 网络流量，从而消除这些担心。不过，此方法可能会带来带宽和可用性方面的难题。

Amazon S3 的 VPC 终端节点可以克服这些难题。Amazon S3 的 VPC 终端节点使 AWS Glue 可以使用私有 IP 地址访问 Amazon S3，而无需接触公共互联网。AWS Glue 不需要公有 IP 地址，因此您的 VPC 中不需要有互联网网关、NAT 设备或虚拟私有网关。您使用终端节点策略控制对 Amazon S3 的访问。您的 VPC 和AWS服务之间的流量不会脱离 Amazon 网络。

在为 Amazon S3 创建 VPC 终端节点时，发送到区域（如 s3.us-west-2.amazonaws.com）内的 Amazon S3 终端节点的任何请求都被路由到亚马逊网络中的私有 Amazon S3 终端节点。您不需要修改正在 VPC 中的 Amazon EC2 实例上运行的应用程序 – 终端节点名称保持不变，但到 Amazon S3 的路由会完全保留在亚马逊网络中，不会访问公共互联网。

有关 VPC 终端节点的更多信息，请参阅《Amazon VPC 用户指南》中的 VPC 终端节点。

下图说明 AWS Glue 如何使用 VPC 终端节点访问 Amazon S3。



设置 Amazon S3 访问权限

1. 登录到 AWS Management Console，然后通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
2. 在左侧导航窗格中，选择终端节点。
3. 选择 Create Endpoint (创建终端节点)，然后按照步骤创建网关类型的 Amazon S3 VPC 终端节点。

Amazon DocumentDB 连接

您可以使用 AWS Glue for Spark 读取和写入 Amazon DocumentDB 中的表。您可以通过 AWS Glue 连接使用存储在 AWS Secrets Manager 中的凭证连接到 Amazon DocumentDB。

有关 Amazon DocumentDB 的更多信息，请参阅 Amazon DocumentDB 文档。

> **Note**
>
> 使用 AWS Glue 连接器时，目前不支持 Amazon DocumentDB 弹性集群。有关弹性集群的更多信息，请参阅 Using Amazon DocumentDB elastic clusters。

读取和写入 Amazon DocumentDB 集合

> **Note**
>
> 当您创建连接到 Amazon DocumentDB 的 ETL 任务时，对于 `Connections` 任务属性，您必须指定一个连接对象，用于指定在其中运行 Amazon DocumentDB 的 Virtual Private Cloud（VPC）。对于该连接对象，连接类型必须为 `JDBC`，且 `JDBC URL` 必须为 `mongo://<DocumentDB_host>:27017`。

> **Note**
>
> 这些代码示例是为 AWS Glue 3.0 开发的。要迁移到 AWS Glue 4.0，请参阅 the section called "MongoDB"。`uri` 参数已更改。

> **Note**
>
> 使用 Amazon DocumentDB 时，在某些情况下必须将 `retryWrites` 设置为 false，例如编写的文档指定 `_id` 时。有关更多信息，请参阅 Amazon DocumentDB 文档中的 与 MongoDB 之间的功能差异。

以下 Python 脚本展示了如何使用连接类型和连接选项来读写 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()
```

以下 Scala 脚本展示了如何使用连接类型和连接选项来读写 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 连接选项参考

指定与 Amazon DocumentDB (with MongoDB compatibility) 的连接。

源连接和接收器连接的连接选项不同。

"connectionType": "documentdb" as Source

将 `"connectionType": "documentdb"` 用作源时可使用以下连接选项：

- `"uri"`：（必需）要从中读取数据的 Amazon DocumentDB 主机，格式为 `mongodb://<host>:<port>`。
- `"database"`：（必需）要从中读取数据的 Amazon DocumentDB 数据库。
- `"collection"`：（必需）要从中读取数据的 Amazon DocumentDB 连接。
- `"username"`：（必需）Amazon DocumentDB 用户名。
- `"password"`：（必需）Amazon DocumentDB 密码。
- `"ssl"`：（如果使用 SSL，则必需）如果您的连接使用 SSL，则必须包含此选项且值为 `"true"`。
- `"ssl.domain_match"`：（如果使用 SSL，则必需）如果您的连接使用 SSL，则必须包含此选项且值为 `"false"`。
- `"batchSize"`：（可选）每个批处理返回的文档数量，在内部批处理的游标中使用。
- `"partitioner"`：（可选）从 Amazon DocumentDB 中读取输入数据的分区器的类名称。该连接器提供以下分区器：
  - `MongoDefaultPartitioner`（默认）（AWS Glue 4.0 不支持）
  - `MongoSamplePartitioner`（AWS Glue 4.0 不支持）
  - `MongoShardedPartitioner`

  - MongoSplitVectorPartitioner
  - MongoPaginateByCountPartitioner
  - MongoPaginateBySizePartitioner（AWS Glue 4.0 不支持）
- "partitionerOptions"：（可选）指定分区器的选项。各个分区器支持的选项如下：
  - MongoSamplePartitioner: partitionKey, partitionSizeMB, samplesPerPartition
  - MongoShardedPartitioner: shardkey
  - MongoSplitVectorPartitioner：partitionKey、partitionSizeMB
  - MongoPaginateByCountPartitioner: partitionKey, numberOfPartitions
  - MongoPaginateBySizePartitioner：partitionKey、partitionSizeMB

  有关这些选项的更多信息，请参阅 MongoDB 文档中的分区器配置。

"connectionType": "documentdb" as Sink

将 "connectionType": "documentdb" 用作连接器时可使用以下连接选项：

- "uri"：（必需）要在其中写入数据的 Amazon DocumentDB 主机，格式为 mongodb://<host>:<port>。
- "database"：（必需）要在其中写入数据的 Amazon DocumentDB 数据库。
- "collection"：（必需）要在其中写入数据的 Amazon DocumentDB 连接。
- "username"：（必需）Amazon DocumentDB 用户名。
- "password"：（必需）Amazon DocumentDB 密码。
- "extendedBsonTypes"：（可选）如果为 true，则在 Amazon DocumentDB 中写入数据时会启用扩展 BSON 类型。默认为 true。
- "replaceDocument"：（可选）如果为 true，则在保存包含 _id 字段的数据集时会替换整个文档。如果为 false，则只会更新文档中与数据集中的字段匹配的字段。默认为 true。
- "maxBatchSize"：（可选）保存数据时的批量操作的最大批次大小。默认值为 512。
- "retryWrites"：（可选）：如果 AWS Glue 遇到网络错误，则会自动重试某些写入操作一次。

OpenSearch Service 连接

在 AWS Glue 4.0 及更高版本中，您可以使用 AWS Glue for Spark 来读取和写入 OpenSearch Service 中的表。您可以使用 OpenSearch 查询来定义要从 OpenSearch Service 中读取的内容。您可以使

用存储在 AWS Secrets Manager 中的 HTTP 基本身份验证凭证，并通过 AWS Glue 连接来连接到 OpenSearch Service。此功能与 OpenSearch Service 无服务器不兼容。

有关 Amazon OpenSearch Service 的更多信息，请参阅 Amazon OpenSearch Service 文档。

配置 OpenSearch Service 连接

要从 AWS Glue 连接到 OpenSearch Service，您需要创建 OpenSearch Service 凭证并将其存储在 AWS Secrets Manager 密钥中，然后将该密钥关联到某个 OpenSearch Service AWS Glue 连接。

先决条件：

- 确定您要从中读取的域端点 *aosEndpoint* 和端口 *aosPort*，或者按照 Amazon OpenSearch Service 文档中的说明创建资源。有关更多信息，请参阅《Amazon OpenSearch Service 开发人员指南》中的 Creating and managing Amazon OpenSearch Service domains。

  Amazon OpenSearch Service 域端点的默认格式为 https://search-*domainName*-*unstructuredIdContent*.*region*.es.amazonaws.com。有关如何确定域端点的更多信息，请参阅《Amazon OpenSearch Service 开发人员指南》中的 Creating and managing Amazon OpenSearch Service domains。

  确定或生成域的 HTTP 基本身份验证凭证（*aosUser* 和 *aosPassword*）。

配置 OpenSearch Service 连接：

1. 在 AWS Secrets Manager 中，使用您的 OpenSearch Service 凭证创建一个密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *secretName*，以供下一步使用。
   - 在选择键/值对时，请使用键 `opensearch.net.http.auth.user` 和值 *aosUser* 创建一个键值对。
   - 在选择键/值对时，请使用键 `opensearch.net.http.auth.pass` 和值 *aosPassword* 创建一个键值对。
2. 在 AWS Glue 控制台中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建一个连接。创建连接后，保留连接名为 *connectionName*，以供未来在 AWS Glue 中使用。
   - 选择连接类型时，请选择 OpenSearch Service。
   - 选择域端点时，请提供 *aosEndpoint*。
   - 选择端口时，请提供 *aosPort*。

- 选择 AWS 密钥时，请提供 *secretName*。

创建 AWS Glue OpenSearch Service 连接后，您需要完成以下操作，然后才能运行 AWS Glue 作业：

- 向与您的 AWS Glue 作业关联的 IAM 角色授予读取 *secretName* 的权限。
- 在 AWS Glue 作业配置中，提供 *connectionName* 作为附加网络连接。

读取 OpenSearch Service 索引

先决条件：

- 您要读取的 OpenSearch Service 索引 *aosIndex*。
- 为了提供身份验证和网络位置信息而配置的 AWS Glue OpenSearch Service 连接。要获得此信息，请完成前面“配置 OpenSearch Service 连接”中的步骤。您需要 AWS Glue 连接的名称 *connectionName*。

此示例将从 Amazon OpenSearch Service 读取索引。您需要提供 pushdown 参数。

例如：

```
opensearch_read = glueContext.create_dynamic_frame.from_options(
    connection_type="opensearch",
    connection_options={
        "connectionName": "connectionName",
        "opensearch.resource": "aosIndex",
        "pushdown": "true",
    }
)
```

您还可以提供查询字符串来筛选返回到 DynamicFrame 的结果。您将需要配置 opensearch.query。

opensearch.query 可以为 URL 查询参数字符串 *queryString*，也可以为查询 DSL JSON 对象 *queryObject*。有关查询 DSL 的更多信息，请参阅 OpenSearch 文档中的 Query DSL。要提供 URL 查询参数字符串，请在查询前面加上 ?q=，这与完全限定 URL 中一样。要提供查询 DSL 对象，请首先对 JSON 对象进行字符串转义，然后再提供该对象。

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",
    }
)
```

有关如何构建不符合特定语法的查询的更多信息，请参阅 OpenSearch 文档中的 Query string syntax。

从包含数组类型数据的 OpenSearch 集合中读取时，必须使用 `opensearch.read.field.as.array.include` 参数在方法调用中指定哪些字段是数组类型。

例如，在阅读以下文档时，您会遇到 `genre` 和 `actor` 数组字段：

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

在这种情况下，您将在方法调用中包含这些字段名称。例如：

```
"opensearch.read.field.as.array.include": "genre,actor"
```

如果您的数组字段嵌套在文档结构内部，请使用点表示法引用它："genre,actor,foo.bar.baz"。这将通过包含嵌入式文档 bar 的嵌入式文档 foo 指定源文档中包含的数组 baz。

写入 OpenSearch Service 表

此示例会将来自现有 DynamicFrame *dynamicFrame* 的信息写入 OpenSearch Service。如果索引中已经含有信息，AWS Glue 会将来自 DynamicFrame 的数据附加到现有信息之后。您需要提供 pushdown 参数。

先决条件：

- 您要写入的 OpenSearch Service 表。您将需要该表的标识信息。我们称之为 *tableName*。
- 为了提供身份验证和网络位置信息而配置的 AWS Glue OpenSearch Service 连接。要获得此信息，请完成前面“配置 OpenSearch Service 连接”中的步骤。您需要 AWS Glue 连接的名称 *connectionName*。

例如：

```
glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="opensearch",
    connection_options={
      "connectionName": "connectionName",
      "opensearch.resource": "aosIndex",
    },
)
```

OpenSearch Service 连接选项参考

- connectionName – 必需。用于读/写。为了向您的连接方法提供身份验证和网络位置信息而配置的 AWS Glue OpenSearch Service 连接的名称。
- opensearch.resource – 必需。用于读/写。有效值：OpenSearch 索引名。您的连接方法将与之交互的索引的名称。
- opensearch.query – 用于读取。有效值：字符串转义后的 JSON，如果此字符串以？开头，则为 URL 的搜索部分。用于在读取时筛选检索范围的 OpenSearch 查询。有关此参数用法的更多信息，请参阅上一节“the section called “从 OpenSearch Service 读取””。
- pushdown – 对于下列情况为必填项。用于读取。有效值：布尔值。指示 Spark 将读取查询传递给 OpenSearch，以便数据库仅返回相关文档。
- opensearch.read.field.as.array.include – 如果读取数组类型数据，则为必填项。用于读取。有效值：以逗号分隔的字段名称列表。指定要从 OpenSearch 文档中作为数组读取的字段。有关此参数用法的更多信息，请参阅上一节“the section called “从 OpenSearch Service 读取””。

Redshift 连接

你可以使用 AWS Glue for Spark 读取和写入亚马逊 Redshift 数据库中的表。连接到亚马逊 Redshift 数据库时，G AWS lue 使用亚马逊 Red COPY shift SQL 和命令通过亚马逊 S3 移动数据以实现最大吞吐量。UNLOAD在 AWS Glue 4.0 及更高版本中，您可以使用适用于 Apache Spark 的 Amazon Redshift 集成进行读写，除了通过先前版本连接时可用的优化和功能外，还具有针对亚马逊 Redshift 的优化和功能。

了解 Glue 如何 AWS 让亚马逊 Redshift 用户比以往任何时候都更轻松地迁移到 G AWS lue 进行无服务器数据集成和 ETL。

配置 Redshift 连接

要在 G AWS lue 中使用 Amazon Redshift 集群，你需要一些先决条件：

- 读取和写入数据库时用于临时存储的 Amazon S3 目录。
- 一种亚马逊 VPC，允许在您的亚马逊 Redshift 集群、Glu AWS e 任务和亚马逊 S3 目录之间进行通信。
- AWS Glue 任务和 Amazon Redshift 集群的相应的 IAM 权限。

配置 IAM 角色

为 Amazon Redshift 集群设置角色

您的 Amazon Redshift 集群需要能够读取和写入亚马逊 S3 才能与 Glue 任务集成 AWS 。为此，您可以将 IAM 角色与要连接的新 Amazon Redshift 集群关联。您的角色应具有允许读取和写入 Amazon S3 临时目录的策略。您的角色应该有信任关系，允许 `redshift.amazonaws.com` 服务连接到 `AssumeRole`。

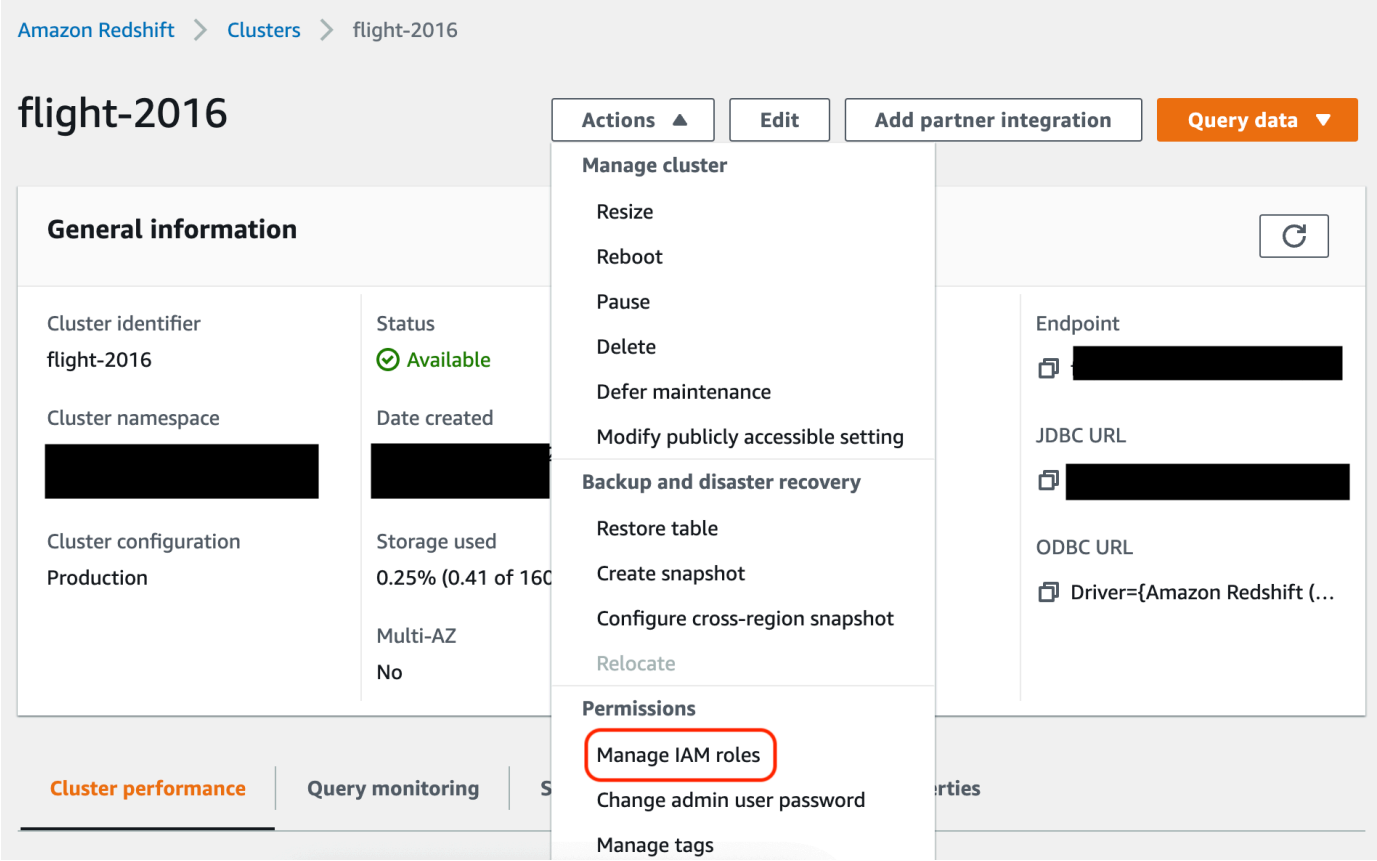
将 IAM 角色与 Amazon Redshift 关联

1. 先决条件：用于临时存储文件的 Amazon S3 存储桶或目录。
2. 确定您的 Amazon Redshift 集群需要哪些 Amazon S3 权限。在将数据移入和移出亚马逊 Redshift 集群时，G AWS lue 作业会针对亚马逊 Redshift 发出 COPY 和 UNLOAD 语句。如果你的任务修改了 Amazon Redshift 中的表， AWS Glue 还会发出 CREATE LIBRARY 语句。有关亚马逊 Redshift 执行这些语句所需的特定 Amazon S3 权限的信息，请参阅亚马逊 Redshift 文档：Amazon R edshift：访问其他资源的权限。 AWS
3. 在 IAM 控制台中，创建具有必要权限的 IAM policy。有关创建策略的更多信息，请参阅 Creating IAM policies。
4. 在 IAM 控制台中，创建角色和信任关系，允许 Amazon Redshift 担任该角色。按照 IAM 文档中的说明为 AWS 服务创建角色（控制台）
   - 当系统要求选择 AWS 服务用例时，选择 “Redshift-可自定义”。
   - 当系统要求附加策略时，请选择您之前定义的策略。

> **Note**
>
> 有关为亚马逊 Redshift 配置角色的更多信息，请参阅亚马逊 R edshift 文档中的授权亚马逊 Redshift 代表您访问其他 AWS 服务。

5. 在 Amazon Redshift 控制台中，将角色与您的 Amazon Redshift 集群关联。按照 Amazon Redshift 文档中的说明操作。

   在 Amazon Redshift 控制台中选择突出显示的选项，配置此设置：



> **Note**
>
> 默认情况下， AWS Glue 任务会通过 Amazon Redshift 临时证书，这些证书是使用您为运行任务而指定的角色创建的。我们建议不使用这些凭证。出于安全考虑，这些凭证将在 1 小时后过期。

为 Glue 作业 AWS 设置角色

AWS Glue 任务需要一个角色才能访问 Amazon S3 存储桶。您不需要 Amazon Redshift 集群的 IAM 权限，您的访问权限由 Amazon VPC 中的连接和您的数据库凭证控制。

设置 Amazon VPC

设置 Amazon Redshift 数据存储访问权限

1. 登录 AWS Management Console 并打开亚马逊 Redshift 控制台，网址为 https://console.aws.amazon.com/redshiftv2/。
2. 在左侧导航窗格中，选择集群。
3. 选择要从 AWS Glue 访问的集群名称。
4. 在 Cluster Properties (集群属性) 部分，从 VPC security groups (VPC 安全组) 中选择一个安全组以允许 AWS Glue 使用。记录下所选的安全组名称以供将来参考。选择安全组将打开 Amazon EC2 控制台 Security Groups (安全组) 列表。
5. 选择要修改的安全组并导航到入站选项卡。
6. 添加一个自引用规则，以允许 AWS Glue 组件进行通信。具体来讲，添加或确认有一条类型为 `All TCP` 的规则，协议为 `TCP`，端口范围包括所有端口，其源具有与组 ID相同的安全组名。

   入站规则类似如下：

类型	协议	端口范围	来源
所有 TCP	TCP	0–65535	database-security-group

   例如：

7. 同时也为出站流量添加一条规则。打开到所有端口的出站流量，例如：

类型	协议	端口范围	目标位置
所有流量	ALL	ALL	0.0.0.0/0

或创建一条 Type (类型) 为 `All TCP`、Protocol (协议) 为 TCP、Port Range (端口范围) 包括所有端口及其 Destination (目标) 具有与 Group ID (组 ID) 相同的安全组名称的自引用规则。如果使用 Amazon S3 VPC 终端节点，还可以添加 HTTPS 规则以访问 Amazon S3。为了允许从 V prefix-list-id PC *`# Amazon S3 VPC ################## s3-`*。

例如：

类型	协议	端口范围	目标位置
所有 TCP	TCP	0–65535	*`security-group`*
HTTPS	TCP	443	*`s3-prefix-list-id`*

设置 AWS Glue

您需要创建一个提供 Amazon VPC 连接信息的 AWS Glue 数据目录连接。

要在控制台中配置 Amazon Redshift 亚马逊 VPC 与 GI AWS ue 的连接

1. 按照步骤创建 Data Catalog 连接：the section called “添加 AWS Glue 连接”。创建连接后，保留连接名称 *`connectionName`*，以供下一步使用。

   - 选择连接类型时，请选择 Amazon Redshift。
   - 选择 Redshift 集群时，按名称选择集群。
   - 为集群上的 Amazon Redshift 用户提供默认连接信息。
   - 您的 Amazon VPC 设置将自动配置。

   > **Note**
   >
   > 通过 AWS SDK 创建 Amazon Redshift 连接时，您需要手动为 Amazon VPC 提供 `PhysicalConnectionRequirements`。

2. 在你的 AWS Glue 作业配置中，提供 *`ConnectionName`* 作为附加网络连接。

示例：从 Amazon Redshift 表中读取

您可以从 Amazon Redshift 集群和 Amazon Redshift Serverless 环境中读取。

先决条件：您想读取的 Amazon Redshift 表。按照上一节中的步骤进行操作，the section called “配置 Redshift”之后您应该拥有临时目录、t *emp-s3-dir # IAM ######## Amazon S* 3 URI。*rs-role-namerole-account-id*

Using the Data Catalog

其他先决条件：您要从中读取 Amazon Redshift 表的 Data Catalog 数据库和表。有关 Data Catalog 的更多信息，请参阅 数据发现和编目。为您的 Amazon Redshift 表创建条目后，您将识别出您与和的*redshift-dc-database-name*关联。*redshift-table-name*

配置：在函数选项中，您将使用 `database` 和 `table_name` 参数标识数据目录表。您将使用 `redshift_tmp_dir` 标识 Amazon S3 临时目录。您还将*rs-role-name*使用`additional_options`参数中的`aws_iam_role`密钥提供。

```
 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

其他先决条件：你需要你的 Amazon Redshift 表的名称（. *redshift-table-name* 您将需要存储该表的 Amazon Redshift 集群的 JDBC 连接信息。您将提供连接信息，包括*##*、*##redshift-database-name*、*###*和*##*。

使用 Amazon Redshift 集群时，您可以从 Amazon Redshift 控制台检索连接信息。使用 Amazon Redshift Serverless 时，请参阅 Amazon Redshift 文档中的 Connecting to Amazon Redshift Serverless。

配置：在函数选项中，您将使用 `url`、`dbtable`、`user` 和 `password` 参数标识连接参数。您将使用 `redshift_tmp_dir` 标识 Amazon S3 临时目录。您可以在使用 `aws_iam_role` 时使用 `from_options` 来指定 IAM 角色。语法类似于通过 Data Catalog 进行连接，但您可以将参数放在 `connection_options` 地图中。

将密码硬编码为 AWS Glue 脚本是不好的做法。考虑使用适用于 AWS Secrets Manager Python 的 SDK (Boto3) 将密码存储在脚本中，然后将其检索到脚本中。

```
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)
```

示例：写入 Amazon Redshift 表

您可以写入 Amazon Redshift 集群和 Amazon Redshift Serverless 环境。

先决条件：一个 Amazon Redshift 集群并按照上一节中的步骤进行操作，the section called “配置 Redshift”之后您应该拥有临时目录、t *emp-s3-dir # IAM ### Amazon S* 3 URI（在账户中 *rs-role-name* ）。*role-account-id*您还需要想写入数据库的内容的 DynamicFrame。

Using the Data Catalog

其他先决条件：您要对其写入的 Amazon Redshift 集群和表的 Data Catalog 数据库。有关 Data Catalog 的更多信息，请参阅 数据发现和编目。您将标识与的连接*redshift-dc-database-name*以及与的目标表*redshift-table-name*。

配置：在函数选项中，您将使用 database 参数标识 Data Catalog 数据库，并为表提供 table_name。您将使用 redshift_tmp_dir 标识 Amazon S3 临时目录。您还将*rs-role-name*使用additional_options参数中的aws_iam_role密钥提供。

```
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` 方法直接连接到 Amazon Redshift。但是，与其将连接详细信息直接插入脚本，不如使用 `from_jdbc_conf` 方法引用存储在 Data Catalog 连接中的连接详细信息。无需为数据库爬取或创建 Data Catalog 表，即可执行此操作。有关 Data Catalog 连接的更多信息，请参阅 连接到数据。

其他先决条件：您要从中读取 Amazon Redshift 表的数据库的 Data Catalog 连接。

配置：您将识别与您的数据目录连接*dc-connection-name*。您将使用和来标识您的 Amazon Redshift 数据库和表。*redshift-table-nameredshift-database-name*您将使用 `catalog_connection` 提供 Data Catalog 连接信息，使用 `dbtable` 和 `database` 提供 Amazon Redshift 信息。语法类似于通过 Data Catalog 进行连接，但您可以将参数放在 `connection_options` 地图中。

```
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 连接选项参考

用于所有 AWS Glue JDBC 连接的基本连接选项在所有 JDBC 类型中`password`都是一致的`url`，用于设置`user`和之类的信息。有关标准 JDBC 参数的更多信息，请参阅 the section called “JDBC 连接参数”。

Amazon Redshift 连接类型需要一些额外的连接选项：

- `"redshiftTmpDir"`：（必需）从数据库中复制时，可以用于暂存临时数据的 Amazon S3 路径。
- `"aws_iam_role"`：（可选）IAM 角色的 ARN。 AWS Glue 任务会将此角色传递给 Amazon Redshift 集群，以授予完成任务指令所需的集群权限。

AWS Glue 4.0+ 版本中提供了其他连接选项

你也可以通过 Glue 连接选项传递新的 Amazon Redshift 连接 AWS 器的选项。有关支持的连接器选项的完整列表，请参阅适用于 Apache Spark 的 Amazon Redshift 集成中的 Spark SQL 参数部分。

为方便起见，我们在此重申某些新选项：

名称	必需	默认值	描述
autopushdown	否	TRUE	通过捕获和分析 SQL 操作的 Spark 逻辑计划，应用谓词和查询下推。这些操作转换为 SQL 查询，然后在 Amazon Redshift 中运行以提高性能。
autopushdown.s3_result_cache	否	FALSE	缓存 SQL 查询以将 Amazon S3 路径映射的数据卸载到内存中，以便同一查询不需要在同一 Spark 会话中再次运行。仅在启用 `autopushdown` 时支持。
unload_s3_format	否	PARQUET	PARQUET — 以 Parquet 格式卸载查询结果。 TEXT — 以竖线分隔的文本格式卸载查询结果。

名称	必需	默认值	描述
sse_kms_key	否	不适用	在UNLOAD操作期间用于加密的 AWS SSE-KMS 密钥，而不是默认的加密密钥。 AWS
extracopyoptions	否	不适用	加载数据时附加到 Amazon Redshift COPY命令的额外选项列表，例如 TRUNCATECOLUMNS 或 MAXERROR n（有关其他选项，请参阅 COPY: Optional 参数）。 请注意，由于这些选项附加到 COPY 命令的末尾，因此只能使用在命令末尾有意义的选项。这应该涵盖最有可能的使用案例。
cvsnullstring（实验）	否	NULL	使用 CSV tempformat 时要为空值写入的字符串值。这应该是一个不会出现在实际数据中的值。

这些新参数可以通过以下方式使用。

提高性能的新选项

新的连接器推出了一些新的性能改进选项：

- autopushdown：默认情况下启用。

- autopushdown.s3_result_cache：默认情况下禁用。
- unload_s3_format：默认为 PARQUET。

有关使用这些选项的信息，请参阅适用于 Apache Spark 的 Amazon Redshift 集成。我们建议您在进行混合读取和写入操作时不要开启 autopushdown.s3_result_cache，因为缓存的结果可能包含过时的信息。UNLOAD 命令的默认选项 unload_s3_format 设置为 PARQUET，以提高性能并降低存储成本。要使用 UNLOAD 命令的默认行为，请将该选项重置为 TEXT。

新的读取加密选项

默认情况下，从 Amazon Redshift 表读取数据时，AWS Glue 使用的临时文件夹中的数据使用 SSE-S3 加密进行加密。要使用 AWS Key Management Service (AWS KMS) 中的客户托管密钥来加密您的数据，您可以设置密钥 ID 来自("sse_kms_key" # kmsKey)哪里 ksmKey AWS KMS，而不是AWS Glue版本 3.0 ("extraunloadoptions" # s"ENCRYPTED KMS_KEY_ID '$kmsKey'") 中的传统设置选项。

```
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"
)
```

支持基于 IAM 的 JDBC URL

新的连接器支持基于 IAM 的 JDBC URL，因此您无需传递用户/密码或密钥。通过基于 IAM 的 JDBC URL，连接器可使用作业运行时角色访问 Amazon Redshift 数据源。

步骤 1：将以下最低要求策略附加到您的 AWS Glue 作业运行时角色。

```
{
    "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": "*"
        }
    ]
}
```

步骤 2：使用基于 IAM 的 JDBC URL，如下所示。使用您连接的 Amazon Redshift 用户名指定一个新选项 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 目前在 GlueContext.create_dynamic_frame.from_options 工作流程中仅支持对 DbUser 使用基于 IAM 的 JDBC URL。

从 AWS Glue 版本 3.0 迁移到版本 4.0

在 AWS Glue 4.0 中，ETL 作业可以访问新的 Amazon Redshift Spark 连接器和具有不同选项和配置的新 JDBC 驱动程序。全新 Amazon Redshift 连接器和驱动程序在编写时充分考虑了性能，可保持数据的交易一致性。这些产品记录在 Amazon Redshift 文档中。有关更多信息，请参阅：

- 适用于 Apache Spark 的 Amazon Redshift 集成
- Amazon Redshift JDBC 驱动程序版本 2.1

表/列名和标识符限制

新的 Amazon Redshift Spark 连接器和驱动程序对 Redshift 表名称的要求更为严格。有关更多信息，请参阅名称和标识符以定义您的 Amazon Redshift 表名称。作业书签工作流程可能不适用于不符合规则的表名和某些字符（例如空格）。

如果您的旧表的名称不符合名称和标识符规则，并且发现书签存在问题（作业正在重新处理旧的 Amazon Redshift 表数据），我们建议您重命名表名。有关更多信息，请参阅修改表示例。

数据帧中的默认临时格式更改

在写入到 Amazon Redshift 时，AWS Glue 版本 3.0 Spark 连接器会将 `tempformat` 的默认值设置为 CSV。为了保持一致，在 AWS Glue 3.0 版中， `DynamicFrame` 仍会将 `tempformat` 的默认值设置为使用 `CSV`。如果您之前曾在 Amazon Redshift Spark 连接器上直接使用过 Spark Dataframe API，则可以在 `DataframeReader/Writer` 选项中将 `tempformat` 明确设置为 CSV。否则，`tempformat` 在新 Spark 连接器中默认为 `AVRO`。

行为更改：将 Amazon Redshift 数据类型 REAL 映射到 Spark 数据类型 FLOAT，而不是 DOUBLE

在 AWS Glue 3.0 版本中，Amazon Redshift `REAL` 被转换为 Spark `DOUBLE` 类型。新的 Amazon Redshift Spark 连接器更新了行为，因此 Amazon Redshift `REAL` 类型可以转换为 Spark `FLOAT` 类型，然后从这一类型转换回来。如果您的旧用例仍希望将 Amazon Redshift `REAL` 类型映射到 Spark `DOUBLE` 类型，则可以使用以下解决方法：

- 对于 `DynamicFrame`，使用 `DynamicFrame.ApplyMapping` 将 `Float` 类型映射到 `Double` 类型。对于 `Dataframe`，需要使用 `cast`。

代码示例：

```
dyf_cast = dyf.apply_mapping([('a', 'long', 'a', 'long'), ('b', 'float', 'b',
 'double')])
```

Kafka 连接

指定与 Kafka 集群或 Amazon Managed Streaming for Apache Kafka 集群的连接。

您可以使用存储在 Data Catalog 表中的信息或通过提供信息直接访问数据流来从 Kafka 数据流读取或向其中写入数据。你可以从 Kafka 读取信息到 Spark 中 DataFrame，然后将其转换为 AWS Glue。DynamicFrame你可以用 JSON 格式写入 DynamicFrames Kafka。如果直接访问数据流，请使用这些选项提供有关如何访问数据流的信息。

如果您通过 `getCatalogSource` 或 `create_data_frame_from_catalog` 来使用来自 Kafka 流式处理源的记录，或通过 `getCatalogSink` 或 `write_dynamic_frame_from_catalog` 将记录写入 Kafka，则作业具有 Data Catalog 数据库和表名称信息，并且可以使用该信息来获取一些基本参数，以便从 Kafka 流式处理源读取数据。如果使用 `getSource`、`getCatalogSink`、`getSourceWithFormat`、`getSinkWithFormat`、`createDataFrame` 或 `write_dynamic_frame_from_catalog`，则必须使用此处描述的连接选项指定这些基本参数。

您可以使用 `GlueContext` 类中指定方法的以下参数为 Kafka 指定连接选项。

- Scala
  - `connectionOptions`：与 `getSource`、`createDataFrameFromOptions`、`getSink` 结合使用
  - `additionalOptions`：与 `getCatalogSource`、`getCatalogSink` 结合使用
  - `options`：与 `getSourceWithFormat`、`getSinkWithFormat` 结合使用
- Python
  - `connection_options`：与 `create_data_frame_from_options`、`write_dynamic_frame_from_options` 结合使用
  - `additional_options`：与 `create_data_frame_from_catalog`、`write_dynamic_frame_from_catalog` 结合使用
  - `options`：与 `getSource`、`getSink` 结合使用

有关流式处理 ETL 作业的注意事项和限制，请参阅 the section called “串流 ETL 注释和限制”。

配置 Kafka

连接可通过互联网访问的 Kafka 直播没有任何 AWS 先决条件。

您可以创建 AWS Glue Kafka 连接来管理您的连接凭证。有关更多信息，请参阅 the section called “为 Kafka 数据流创建连接”。在你的 AWS Glue 作业配置中，提供 c *onnectionName* 作为附加网络连接，然后在方法调用中为参数提供 c *onnectionName*。connectionName

在某些情况下，您需要配置其他先决条件：

- 如果使用 Amazon Managed Streaming for Apache Kafka 搭配 IAM 身份验证，则需要适当的 IAM 配置。
- 如果在 Amazon VPC 内使用 Amazon Managed Streaming for Apache Kafka，则需要适当的 Amazon VPC 配置。您需要创建一个提供 Amazon VPC 连接信息的 AWS Glue 连接。您需要在作业配置中将 Glue AWS 连接作为附加网络连接包括在内。

有关流式处理 ETL 作业先决条件的更多信息，请参阅 the section called “流式处理 ETL 作业”。

示例：从 Kafka 流读取

与 the section called “forEachBatch” 结合使用。

Kafka 流式处理源示例：

```
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)
```

示例：写入 Kafka 流

写入 Kafka 的示例：

getSink 方法示例：

```
data_frame_datasource0 =
glueContext.getSink(
 connectionType="kafka",
 connectionOptions={
```

```
  JsonOptions("""{
   "connectionName": "ConfluentKafka",
   "classification": "json",
   "topic": "kafka-auth-topic",
   "typeOfData": "kafka"}
  """)},
 transformationContext="dataframe_ApacheKafka_node1711729173428")
 .getDataFrame()
```

write_dynamic_frame.from_options 方法示例：

```
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 连接选项参考

在读取时，可以使用以下连接选项和 "connectionType": "kafka"：

- "bootstrap.servers"（必需）引导服务器 URL 的列表，例如，作为 b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094。此选项必须在 API 调用中指定，或在数据目录的表元数据中定义。
- "security.protocol"（必填）用于与代理通信的协议。可能的值为 "SSL" 或 "PLAINTEXT"。
- "topicName"（必填）要订阅的以逗号分隔的主题列表。您必须指定"topicName"、"assign" 或 "subscribePattern" 中的其中一个，且只能指定一个。
- "assign"：（必填）用于指定要使用的 TopicPartitions 的 JSON 字符串。您必须指定"topicName"、"assign" 或 "subscribePattern" 中的其中一个，且只能指定一个。

  例如：“{"topicA":[0,1],"topicB":[2,4]}”
- "subscribePattern"：（必需）标识要订阅的主题列表的 Java 正则表达式字符串。您必须指定"topicName"、"assign" 或 "subscribePattern" 中的其中一个，且只能指定一个。

  示例：“topic.*”
- "classification"（必需）记录中数据使用的文件格式。除非 Data Catalog 提供，否则为必需。

- "delimiter"（可选）当 classification 为 CSV 时使用的值分隔符。默认为“,”。
- "startingOffsets"：（可选）Kafka 主题中数据读取的起始位置。可能的值为 "earliest" 或 "latest"。默认值为 "latest"。
- "startingTimestamp": (可选，仅支持 AWS Glue 4.0 或更高版本）Kafka 主题中要从中读取数据的记录的时间戳。可能的值是以模式 yyyy-mm-ddTHH:MM:SSZ 采用 UTC 格式的时间戳字符串（其中 Z 表示带有 +/-的 UTC 时区偏移量。例如：“2023-04-04T08:00:00-04:00”）。

  注意：GI AWS ue 直播脚本的 “连接选项” 列表中只能出现 “startingOffsets” 或 “startingTimeStamp” 中的一个，包括这两个属性都会导致任务失败。
- "endingOffsets"：（可选）批处理查询结束时的终点。可能值为 "latest"，或者为每个 TopicPartition 指定结束偏移的 JSON 字符串。

  对于 JSON 字符串，格式为 {"topicA":{"0":23,"1":-1},"topicB":{"0":-1}}。偏移值 -1 表示 "latest"。
- "pollTimeoutMs"：（可选）Spark 任务执行程序中，从 Kafka 轮询数据的超时时间（以毫秒为单位）。默认值为 512。
- "numRetries"：（可选）无法获取 Kafka 偏移时的重试次数。默认值为 3。
- "retryIntervalMs"：（可选）重试获取 Kafka 偏移时的等待时间（以毫秒为单位）。默认值为 10。
- "maxOffsetsPerTrigger"：（可选）每个触发间隔处理的最大偏移数的速率限制。指定的总偏移数跨不同卷的 topicPartitions 按比例分割。默认值为 null，这意味着使用者读取所有偏移，直到已知的最新偏移。
- "minPartitions"：（可选）从 Kafka 读取数据的必需最小分区数。默认值为 null，这意味着 Spark 分区数等于 Kafka 分区数。
- "includeHeaders"：（可选）是否包含 Kafka 标头。当选项设置为“true”时，数据输出将包含一个名为“glue_streaming_kafka_headers”的附加列，类型为 Array[Struct(key: String, value: String)]。默认值为“false”。此选项仅适用于 AWS Glue 版本 3.0 或更高版本。
- "schema"：（当 inferSchema 设为 false 时为必填）用于处理有效工作负载的架构。如果分类为 avro，则提供的架构必须采用 Avro 架构格式。如果分类不是 avro，则提供的架构必须采用 DDL 架构格式。

  以下是一些架构示例。

  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"：（可选）默认值为“false”。如果设置为“true”，则会在运行时检测到 foreachbatch 内的有效工作负载中的架构。
- "avroSchema"：（已弃用）用于指定 Avro 数据架构（使用 Avro 格式时）的参数。此参数现已被弃用。使用 schema 参数。
- "addRecordTimestamp"：（可选）当选项设置为 'true' 时，数据输出将包含一个名为 "__src_timestamp" 的附加列，表示主题收到相应记录的时间。默认值为‘false’。4.0 或更高 AWS Glue 版本支持此选项。
- "emitConsumerLagMetrics": （可选）当该选项设置为 “true” 时，对于每个批次，它将发布从主题收到的最旧记录到该记录到达的时间之间的持续时间内的AWS Glue指标。 CloudWatch该指标的名字是 “glue.driver.streaming”。 maxConsumerLagInMs”。默认值为‘false’。4.0 或更高 AWS Glue 版本支持此选项。

在写入时，可以使用以下连接选项和 "connectionType": "kafka"：

- "connectionName"（必填）用于连接到 Kafka 集群的 AWS Glue 连接的名称（类似于 Kafka 源代码）。
- "topic"（必填项）如果存在主题列，则在将给定行写入 Kafka 时，除非设置了主题配置选项，否则其值将用作主题。也就是说，topic 配置选项会覆盖主题列。
- "partition"（可选）如果指定了有效的分区号，则 partition 将在发送记录时使用该分区号。

  如果未指定分区但存在 key，则将使用该键的哈希值来选择分区。

  如果 key 和 partition 都不存在，则将根据在向分区至少生成了 batch.size 字节的数据时，对这些更改进行粘性分区来选择分区。
- "key"（可选）如果 partition 为空，则用于分区。
- "classification"（可选）记录中数据使用的文件格式。我们只支持 JSON、CSV 和 Avro。

  对于 Avro 格式，我们可以提供自定义 avroSchema 来进行序列化，但请注意，还需要在源中提供该格式以进行反序列化。否则，默认情况下，它使用 Apache AvroSchema 进行序列化。

此外，您可以在需要时通过更新 Kafka 生产者配置参数来微调 Kafka 接收器。请注意，不存在有关连接选项的允许列表，所有键值对都按原样保存在接收器上。

虽然存在一个很小的选项拒绝列表，不过不会生效。有关更多信息，请参阅 Kafka specific configurations。

Azure Cosmos DB 连接

借助 NoSQL API，您可以在 AWS Glue 4.0 及更高版本中使用 AWS Glue for Spark 读取和写入 Azure Cosmos DB 中现有的容器。您可以使用 SQL 查询来定义要从 Azure Cosmos DB 中读取的信息。您可以通过 AWS Glue 连接，使用存储在 AWS Secrets Manager 中的 Azure Cosmos DB 密钥连接到 Azure Cosmos DB。

有关 Azure Cosmos DB for NoSQL 的更多信息，请参阅 Azure 文档。

配置 Azure Cosmos DB 连接

要从 AWS Glue 连接到 Azure Cosmos DB，您需要创建一个 Azure Cosmos DB 密钥并将其存储在一个 AWS Secrets Manager 密钥中，然后将该密钥关联到某个 Azure Cosmos DB AWS Glue 连接。

先决条件：

- 在 Azure 中，您需要确定或生成一个 Azure Cosmos DB 密钥 `cosmosKey`，以供 AWS Glue 使用。有关更多信息，请参阅 Azure 文档中的 保护对 Azure Cosmos DB 中数据的访问。

配置 Azure Cosmos DB 连接：

1. 在 AWS Secrets Manager 中，使用您的 Azure Cosmos DB 密钥创建一个密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *`secretName`*，以供下一步使用。
   - 在选择键/值对时，请使用键 `spark.cosmos.accountKey` 和值 *`cosmosKey`* 创建一个键值对。
2. 在 AWS Glue 控制台中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建一个连接。创建连接后，保留连接名为 *`connectionName`*，以供未来在 AWS Glue 中使用。
   - 选择连接类型时，请选择 Azure Cosmos DB。
   - 选择 AWS 密钥时，请提供 *`secretName`*。

创建 AWS Glue Azure Cosmos DB 连接后，您需要完成以下操作，然后才能运行 AWS Glue 作业：

- 向与您的 AWS Glue 作业关联的 IAM 角色授予读取 *`secretName`* 的权限。
- 在 AWS Glue 作业配置中，提供 *`connectionName`* 作为附加网络连接。

读取 Azure Cosmos DB for NoSQL 容器

先决条件：

- 您要读取的 Azure Cosmos DB for NoSQL 容器。您将需要该容器的标识信息。

  Azure Cosmos DB for NoSQL 容器由其数据库和容器来标识。在连接到 Azure Cosmos for NoSQL API 时，您必须提供数据库 *`cosmosDBName`* 和容器 *`cosmosContainerName`* 的名称。
- 为了提供身份验证和网络位置信息而配置的 AWS Glue Azure Cosmos DB 连接。要获得此信息，请完成前面“配置 Azure Cosmos DB 连接”中的步骤。您需要 AWS Glue 连接的名称 *`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,
    }
)
```

您还可以提供 SELECT SQL 查询来筛选返回到 DynamicFrame 的结果。您将需要配置 `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"
    }
)
```

写入 Azure Cosmos DB for NoSQL 容器

此示例会将来自现有 DynamicFrame *dynamicFrame* 的信息写入 Azure Cosmos DB。如果容器中已经含有信息，AWS Glue 会将来自 DynamicFrame 的数据附加到现有信息之后。如果容器中的信息与您写入的信息具有不同的 Schema，则会出现错误。

先决条件：

- 您要写入的 Azure Cosmos DB 表。您将需要该容器的标识信息。必须首先创建容器，然后才能调用连接方法。

  Azure Cosmos DB for NoSQL 容器由其数据库和容器来标识。在连接到 Azure Cosmos for NoSQL API 时，您必须提供数据库 *cosmosDBName* 和容器 *cosmosContainerName* 的名称。
- 为了提供身份验证和网络位置信息而配置的 AWS Glue Azure Cosmos DB 连接。要获得此信息，请完成前面“配置 Azure Cosmos DB 连接”中的步骤。您需要 AWS Glue 连接的名称 *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 连接选项参考

- `connectionName` – 必需。用于读/写。为了向您的连接方法提供身份验证和网络位置信息而配置的 AWS Glue Azure Cosmos DB 连接的名称。
- `spark.cosmos.database` – 必需。用于读/写。有效值：数据库名。Azure Cosmos DB for NoSQL 数据库名。
- `spark.cosmos.container` – 必需。用于读/写。有效值：容器名。Azure Cosmos DB for NoSQL 容器名。
- `spark.cosmos.read.customQuery` – 用于读取。有效值：SELECT SQL 查询。用于选择要读取的文档的自定义查询。

Azure SQL 连接

在 AWS Glue 4.0 及更高版本中，您可以使用 AWS Glue for Spark 来读取和写入 Azure SQL 托管实例中的表。您可以使用 SQL 查询来定义要从 Azure SQL 中读取的信息。您可以通过 AWS Glue 连接并使用存储在 AWS Secrets Manager 中的用户名和密码凭证连接到 Azure SQL。

有关 Azure SQL 的更多信息，请参阅 Azure SQL 文档。

配置 Azure SQL 连接

要从 AWS Glue 连接到 Azure SQL，您需要创建 Azure SQL 凭证并将其存储在一个 AWS Secrets Manager 密钥中，然后将该密钥关联到某个 Azure SQL AWS Glue 连接。

配置 Azure SQL 连接：

1. 在 AWS Secrets Manager 中，使用您的 Azure SQL 凭证创建密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *secretName*，以供下一步使用。

   - 在选择键/值对时，请使用键 `user` 和值 *azuresqlUsername* 创建一个键值对。

- 在选择键/值对时，请使用键 password 和值 *azuresqlPassword* 创建一个键值对。

2. 在 AWS Glue 控制台中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建一个连接。创建连接后，保留连接名为 *connectionName*，以供未来在 AWS Glue 中使用。

   - 选择连接类型时，请选择 Azure SQL。
   - 在提供 Azure SQL URL 时，请提供 JDBC 端点的 URL。

     该 URL 必须为以下格式：jdbc:sqlserver://*databaseServerName*:*databasePort*;databaseName=*azuresqlDB*

     AWS Glue 需要以下 URL 属性：

     - databaseName – Azure SQL 中要连接的默认数据库。

     有关 Azure SQL 托管实例的 JDBC URL 的更多信息，请参阅 Microsoft 文档。
   - 选择 AWS 密钥时，请提供 *secretName*。

创建 AWS Glue Azure SQL 连接后，您需要完成以下操作，然后才能运行 AWS Glue 作业：

- 向与您的 AWS Glue 作业关联的 IAM 角色授予读取 *secretName* 的权限。
- 在 AWS Glue 作业配置中，提供 *connectionName* 作为附加网络连接。

读取 Azure SQL 表

先决条件：

- 您要读取的 Azure SQL 表。您将需要表的标识信息 *databaseName* 和 *tableIdentifier*。

  Azure SQL 表由其数据库、Schema 和表名来标识。连接到 Azure SQL 时，必须提供数据库名和表名。如果 Schema 不是默认值“public”，则还必须提供 Schema。数据库通过 *connectionName* 中的 URL 属性来提供，Schema 和表名通过 dbtable 来提供。
- 为了提供身份验证信息而配置的 AWS Glue Azure SQL 连接。完成上一节“配置 Azure SQL 连接”中的步骤，以配置您的身份验证信息。您需要 AWS Glue 连接的名称 *connectionName*。

例如：

```
azuresql_read_table = glueContext.create_dynamic_frame.from_options(
    connection_type="azuresql",
```

```
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableIdentifier"
    }
)
```

您还可以提供 SELECT SQL 查询来筛选返回到 DynamicFrame 的结果。您将需要配置 `query`。

例如：

```
azuresql_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="azuresql",
    connection_options={
        "connectionName": "connectionName",
        "query": "query"
    }
)
```

写入 Azure SQL 表

此示例会将来自现有 DynamicFrame *dynamicFrame* 的信息写入 Azure SQL。如果表中已经含有信息，AWS Glue 会将来自 DynamicFrame 的数据附加到现有信息之后。

先决条件：

- 您要写入的 Azure SQL 表。您将需要表的标识信息 *databaseName* 和 *tableIdentifier*。

  Azure SQL 表由其数据库、Schema 和表名来标识。连接到 Azure SQL 时，必须提供数据库名和表名。如果 Schema 不是默认值“public”，则还必须提供 Schema。数据库通过 *connectionName* 中的 URL 属性来提供，Schema 和表名通过 `dbtable` 来提供。
- Azure SQL 身份验证信息。完成上一节“配置 Azure SQL 连接”中的步骤，以配置您的身份验证信息。您需要 AWS Glue 连接的名称 *connectionName*。

例如：

```
azuresql_write = glueContext.write_dynamic_frame.from_options(
    connection_type="azuresql",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableIdentifier"
```

```
    }
)
```

Azure SQL 连接选项参考

- `connectionName` – 必需。用于读/写。为了向您的连接方法提供身份验证信息而配置的 AWS Glue Azure SQL 连接的名称。
- `databaseName` - 用于读/写。有效值：Azure SQL 数据库名。要连接的 Azure SQL 数据库的名称。
- `dbtable` – 对于写入为必填项，对于读取也为必填项，但提供了 `query` 时除外。用于读/写。有效值：Azure SQL 表名或用句点分隔的 Schema 名/表名组合。用于指定表和 Schema，以标识要连接的表。默认 Schema 为“public”。如果您的表并非使用默认 Schema，请在表单 *`schemaName`*`.`*`tableName`* 中提供此信息。
- `query` – 用于读取。用来定义从 Azure SQL 读取数据时要检索的内容的 Transact-SQL SELECT 查询。有关更多信息，请参阅 Microsoft 文档。

BigQuery 连接

在 AWS Glue 4.0 及更高版本中，您可以使用 AWS Glue for Spark 在 Google BigQuery 中读取和写入表。您可以通过 Google SQL 查询从 BigQuery 中读取。您可以通过 AWS Glue 连接使用存储在 AWS Secrets Manager 中的凭证连接到 BigQuery。

有关 Google BigQuery 的更多信息，请参见 Google Cloud BigQuery 网站。

配置 BigQuery 连接

要从 AWS Glue 中连接到 Google BigQuery，您需要创建 Google Cloud Platform 凭证并将其存储在 AWS Secrets Manager 密钥中，然后将该密钥与 Google BigQuery AWS Glue 连接关联。

要配置与 BigQuery 的连接：

1. 在 Google Cloud Platform，创建并识别相关资源：
   - 创建或标识包含您想要连接的 BigQuery 表的 GCP 项目。
   - 启用 BigQuery API。有关更多信息，请参阅使用 BigQuery 存储读取 API 读取表数据。
2. 在 Google Cloud Platform 中，创建和导出服务账户凭证：

   您可以使用 BigQuery 凭证向导来加快此步骤：创建凭证。

   要在 GCP 中创建服务账户，请按照创建服务账户中的教程进行操作。

- 选择项目时，请选择包含您的 BigQuery 表的项目。
- 为您的服务账户选择 GCP IAM 角色时，请添加或创建一个角色，该角色将授予运行 BigQuery 作业的相应权限，以读取、写入或创建 BigQuery 表。

要为您的服务账户创建凭证，请按照创建服务账户密钥中的教程进行操作。

- 在选择密钥类型时，请选择 JSON。

现在，您应该已经下载了包含服务账户凭证的 JSON 文件。如下所示：

```
{
  "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. base64 对您下载的凭证文件进行编码。在 AWS CloudShell 会话或类似会话中，您可以通过运行 `cat credentialsFile.json | base64 -w 0` 从命令行执行此操作。保留此命令的输出 *credentialString*。
4. 在 AWS Secrets Manager 中，使用您的 Google Cloud Platform 凭证创建密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *secretName*，以供下一步使用。

   - 在选择键/值对时，请使用值 *credentialString* 为键 `credentials` 创建对。
5. 在 AWS Glue Data Catalog 中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建连接。创建连接后，保留连接名称 *connectionName*，以供下一步使用。

   - 在选择连接类型时，请选择 Google BigQuery。
   - 选择 AWS 密钥时，请提供 *secretName*。
6. 向与您的 AWS Glue 作业关联的 IAM 角色授予读取 *secretName* 的权限。

7. 在 AWS Glue 作业配置中，提供 *connectionName* 作为附加网络连接。

从 BigQuery 表格中读取

先决条件：

- 您想从中读取的 BigQuery 表。您需要 [dataset].[table] 形式的 BigQuery 表和数据集名称。我们称之为 *tableName*。
- BigQuery 表的计费项目。您将需要项目的名称 *parentProject*。如果没有计费父项目，请使用包含该表的项目。
- BigQuery 身份验证信息。完成使用 AWS Glue 管理连接凭证的步骤，以配置您的身份验证信息。您需要 AWS Glue 连接的名称 *connectionName*。

例如：

```
bigquery_read = glueContext.create_dynamic_frame.from_options(
    connection_type="bigquery",
    connection_options={
        "connectionName": "connectionName",
        "parentProject": "parentProject",
        "sourceType": "table",
        "table": "tableName",
    }
```

您还可以提供查询，以筛选返回到您的 DynamicFrame 的结果。您将需要配置 query、sourceType、viewsEnabled 和 materializationDataset。

例如：

其他先决条件：

您需要创建或识别一个 BigQuery 数据集 *materializationDataset*，BigQuery 可以在其中为您的查询编写实体化视图。

您需要向您的服务账户授予相应的 GCP IAM 权限，才能在 *materializationDataset* 中创建表。

```
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"
            }
        )
```

写入 BigQuery 表

此示例直接写入 BigQuery 服务。BigQuery 还支持“间接”写入方法。有关配置间接写入的更多信息，请参阅 the section called “在 Google BigQuery 中使用间接写入”。

先决条件：

- 您想向其中写入的 BigQuery 表。您需要 [dataset].[table] 形式的 BigQuery 表和数据集名称。您也可以提供一个将自动创建的新表名。我们称之为 *tableName*。
- BigQuery 表的计费项目。您将需要项目的名称 *parentProject*。如果没有计费父项目，请使用包含该表的项目。
- BigQuery 身份验证信息。完成使用 AWS Glue 管理连接凭证的步骤，以配置您的身份验证信息。您需要 AWS Glue 连接的名称 *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 连接选项参考

- `project` – 默认：Google Cloud 服务账户默认。用于读/写。与您的表格关联的 Google Cloud 项目的名称。

- `table` - (必需) 用于读/写。格式为 `[[project:]dataset.]` 的 BigQuery 表的名称。
- `dataset` – 如果未通过 `table` 选项进行定义，则为必填项。用于读/写。包含您的 BigQuery 表的数据集的名称。
- `parentProject` – 默认：Google Cloud 服务账户默认。用于读/写。与用于计费的 `project` 关联的 Google Cloud 项目的名称。
- `sourceType` – 用于读取。读取时为必填项。有效值：`table`，`query` 告知 AWS Glue 是按表读取还是按查询读取。
- `materializationDataset` – 用于读取。有效值：字符串。用于存储视图实例化的 BigQuery 数据集的名称。
- `viewsEnabled` – 用于读取。默认值：false。有效值：true、false。配置 BigQuery 是否使用视图。
- `query` – 用于读取。在 `viewsEnabled` 为真时使用。GoogleSQL DQL 查询。
- `temporaryGcsBucket` – 用于写入。在 `writeMethod` 设置为默认值 (`indirect`) 时为必填。写入 BigQuery 时用于存储中间形式数据的 Google Cloud Storage 存储桶的名称。
- `writeMethod` - 默认值：`indirect`。有效值：`direct`、`indirect`。用于写入。指定用于写入数据的方法。
  - 如果设置为 `direct`，则您的连接器将使用 BigQuery 存储写入 API 进行写入。
  - 如果设置为 `indirect`，则您的连接器将 Google Cloud Storage，然后使用加载操作将其传输到 BigQuery。您的 Google Cloud 服务账户将需要相应的 GCS 权限。

在 Google BigQuery 中使用间接写入

此示例使用间接写入，将数据写入 Google Cloud Storage 并将其复制到 Google BigQuery。

先决条件：

您需要一个临时的 Google Cloud Storage 存储桶 *`temporaryBucket`*。

AWS Glue 的 GCP 服务账户的 GCP IAM 角色需要适当的 GCS 权限才能访问 *`temporaryBucket`*。

其他配置：

要使用 BigQuery 配置间接写入，请执行以下操作：

1. 评测 the section called “配置 BigQuery” 并找到或重新下载您的 GCP 凭证 JSON 文件。识别 *`SecretName`*，这是您的作业中使用的 Google BigQuery AWS Glue 连接的 AWS Secrets Manager 密钥。

2. 将您的凭证 JSON 文件上传到适当安全的 Amazon S3 位置。保留文件路径*s3secretpath* 以备将来的步骤使用。
3. 编辑 *secretName*，添加 `spark.hadoop.google.cloud.auth.service.account.json.keyfile` 密钥。将值设置为 *s3secretpath*。
4. 向您的 AWS Glue 作业 Amazon S3 授予访问 *s3secretpath* 的 IAM 权限。

现在，您可以将临时 GCS 存储桶位置提供给您的写入方法。您无需提供 `writeMethod`，因为 `indirect` 历史上是默认设置。

```
bigquery_write = glueContext.write_dynamic_frame.from_options(
    frame=frameToWrite,
    connection_type="bigquery",
    connection_options={
        "connectionName": "connectionName",
        "parentProject": "parentProject",
        "temporaryGcsBucket": "temporaryBucket",
        "table": "tableName",
    }
)
```

JDBC 连接

某些（通常是关系型）数据库类型支持通过 JDBC 标准进行连接。有关 JDBC 的更多信息，请参阅 Java JDBC API 文档。AWSGlue 原生支持通过 JDBC 连接器连接到某些数据库，JDBC 库在 AWS Glue Spark 作业中提供。使用 AWS Glue 库连接到这些数据库类型时，您可以访问一组标准选项。

JDBC connectionType 值包括：

- `"connectionType": "sqlserver"`：指定与 Microsoft SQL Server 数据库的连接。
- `"connectionType": "mysql"`：指定与 MySQL 数据库的连接。
- `"connectionType": "oracle"`：指定与 Oracle 数据库的连接。
- `"connectionType": "postgresql"`：指定与 PostgreSQL 数据库的连接。
- `"connectionType": "redshift"`：指定与 Amazon Redshift 数据库的连接。有关更多信息，请参阅 the section called “Redshift 连接”。

下表列出了 AWS Glue 支持的 JDBC 驱动程序版本。

产品	Glue 4.0 的 JDBC 驱动程序版本	Glue 3.0 的 JDBC 驱动程序版本	Glue 0.9、1.0、2.0 的 JDBC 驱动程序版本
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

* 如果是 Amazon Redshift 连接类型，用于 JDBC 连接的连接选项中包含的所有其他选项名称/值对（包括格式选项）将直接传递到底层 SparkSQL DataSource。在 AWS Glue 4.0 及更高版本的 AWS Glue with Spark 作业中，Amazon Redshift 的 AWS Glue 原生连接器使用适用于 Apache Spark 的 Amazon Redshift 集成。有关更多信息，请参阅 Amazon Redshift integration for Apache Spark。在之前的版本中，请参阅 Amazon Redshift data source for Spark。

要将您的 Amazon VPC 配置为使用 JDBC 连接到 Amazon RDS 数据存储，请参阅 the section called “设置 Amazon VPC 以连接到 Amazon RDS 数据存储”。

> **Note**
>
> AWS Glue 作业在一次运行期间仅与一个子网关联。这可能会影响您使用同一作业连接到多个数据来源。此行为不仅限于 JDBC 源。

主题

- JDBC 连接选项参考
- 使用 sampleQuery
- 使用自定义 JDBC 驱动程序
- 从 JDBC 表并行读取
- 设置 Amazon VPC 以通过建立从 AWS Glue 到 Amazon RDS 数据存储的 JDBC 连接

JDBC 连接选项参考

如果您已经定义了 JDBC AWS Glue 连接，则可以重复使用其中定义的配置属性，例如：url、用户和密码；不必在代码中将它们指定为连接选项。此功能只在 AWS Glue 3.0 及更高版本中提供。为此，请使用以下连接属性：

- `"useConnectionProperties"`：设置为 "true" 以表示您要使用连接中的配置。
- `"connectionName"`：输入要从中检索配置的连接名称，必须在与作业相同的区域中定义连接。

将这些连接选项与 JDBC 连接结合使用：

- `"url"`：（必填）数据库的 JDBC URL。
- `"dbtable"`：（必需）要从中读取数据的数据库表。对于在数据库中支持架构的 JDBC 数据存储，指定 `schema.table-name`。如果未提供架构，则使用默认的“public”架构。
- `"user"`：（必需）在连接时使用的用户名。
- `"password"`：（必填）在连接时使用的密码。
- （可选）以下选项允许您提供自定义 JDBC 驱动程序。如果必须使用 AWS Glue 本身不支持的驱动程序，请使用这些选项。

  ETL 作业可以为数据源和目标使用不同的 JDBC 驱动程序版本，即使源和目标是相同的数据库产品也是如此。这允许您在不同版本的源数据库和目标数据库之间迁移数据。要使用这些选项，您必须首先将 JDBC 驱动程序的 JAR 文件上传到 Amazon S3。

  - `"customJdbcDriverS3Path"`：自定义 JDBC 驱动程序的 Amazon S3 路径。
  - `"customJdbcDriverClassName"`：JDBC 驱动程序的类名。
- `"bulkSize"`：（可选）用于配置并行插入以加速批量加载到 JDBC 目标。为写入或插入数据时要使用的并行度指定整数值。此选项有助于提高写入数据库（如 Arch User Repository (AUR)）的性能。
- `"hashfield"`（可选）一个字符串，用于指定 JDBC 表中一列的名称，用于在并行读取 JDBC 表时将数据划分为多个分区。提供“哈希字段”或“哈希表达式”。有关更多信息，请参阅 the section called “从 JDBC 并行读取”。
- `"hashexpression"`（可选）返回整数的 SQL 选择子句。用于在并行读取 JDBC 表时将 JDBC 表中的数据划分为多个分区。提供“哈希字段”或“哈希表达式”。有关更多信息，请参阅 the section called “从 JDBC 并行读取”。
- `"hashpartitions"`（可选）正整数。用于指定并行读取 JDBC 表时对 JDBC 表进行并行读取的次数。默认值：7。有关更多信息，请参阅 the section called “从 JDBC 并行读取”。

- "sampleQuery"：（可选）自定义 SQL 查询语句。用于指定表中信息的子集以检索表格内容的样本。如果不考虑您的数据进行配置，则其效率可能低于 DynamicFrame 方法，从而导致超时或内存不足错误。有关更多信息，请参阅 the section called “使用 sampleQuery”。
- "enablePartitioningForSampleQuery"：（可选）布尔值。默认值：false。用于在指定 sampleQuery 时并行读取 JDBC 表。如果设置为 true，**sampleQuery** 必须以“where”或“and”结尾，以便于 AWS Glue 追加分区条件。有关更多信息，请参阅 the section called “使用 sampleQuery”。
- "sampleSize"：（可选）正整数。限制示例查询返回的行数。仅当 enablePartitioningForSampleQuery 为 true 时有用。如果未启用分区，则应直接在 sampleQuery 中添加 "limit x" 以限制大小。有关更多信息，请参阅 the section called “使用 sampleQuery”。

使用 sampleQuery

本部分介绍如何设置 sampleQuery、sampleSize 和 enablePartitioningForSampleQuery。

sampleQuery 是对数据集的几行进行采样的有效方法。默认情况下，查询由单个执行程序执行。如果不考虑您的数据进行配置，则其效率可能低于 DynamicFrame 方法，从而导致超时或内存不足错误。通常仅出于性能考虑，才需要在底层数据库上运行 SQL 作为 ETL 管道的一部分。如果试图预览数据集的几行，请考虑使用 the section called “show”。如果您试图使用 SQL 转换数据集，请考虑使用 the section called “toDF”，在 DataFrame 表单中针对您的数据定义 SparkSQL 转换。

虽然您的查询可能会处理各种表，但 dbtable 仍然是必需的。

使用 sampleQuery 检索表的样本

当使用默认 sampleQuery 行为检索数据样本时，AWS Glue 预计吞吐量不会很大，因此它会在单个执行程序上运行查询。为了限制您提供的数据，以便不会导致性能问题，我们建议您为 SQL 提供一个 LIMIT 子句。

Example 在不进行分区的情况下使用 samplQuery

以下代码示例演示了如何在不进行分区的情况下使用 sampleQuery。

```
//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()
```

对较大的数据集使用 sampleQuery

如果您读取的是大型数据集，则可能需要启用 JDBC 分区才能并行查询表。有关更多信息，请参阅 the section called “从 JDBC 并行读取”。要将 `sampleQuery` 与 JDBC 分区一同使用，请将 `enablePartitioningForSampleQuery` 设置为 true。启用此功能需要您对自己的 `sampleQuery` 进行一些更改。

将 JDBC 分区与 `sampleQuery` 一起使用时，查询必须以“where”或“and”结尾，以便于 AWS Glue 附加分区条件。

如果您想在并行读取 JDBC 表时限制 sampleQuery 的结果，请设置 `"sampleSize"` 参数而不是指定 `LIMIT` 子句。

Example 将 sampleQuery 与 JDBC 分区一起使用

以下代码示例演示了如何将 `sampleQuery` 与 JDBC 分区一起使用。

```
//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()
```

注释和限制：

示例查询不可以与作业书签一起使用。提供两者的配置时，书签状态将会被忽略。

使用自定义 JDBC 驱动程序

以下代码示例演示了如何使用自定义 JDBC 驱动程序读取和写入 JDBC 数据库。这些示例演示了如何从一个版本的数据库产品读取和写入同一产品的更高版本。

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)
  }
}
```

从 JDBC 表并行读取

您可以设置 JDBC 表的属性，以使 AWS Glue 并行读取数据。当您设置某些属性时，您可以指示 AWS Glue 对数据的逻辑分区运行并行 SQL 查询。您可以通过设置哈希字段或哈希表达式来控制分区。您还可以控制用于访问您的数据的并行读取的数量。

并行读取 JDBC 表是一种可以提高性能的优化技术。要详细了解确定何时适合使用这种技术的过程，请参阅《AWS 规范性指南》中“优化 AWS Glue for Apache Spark 作业性能的最佳实践”指南中的 减少数据扫描量。

要启用并行读取，可在表结构的参数字段中设置键值对。使用 JSON 表示法为表的参数字段设置值。有关编辑表属性的更多信息，请参阅 查看和编辑表详细信息。当您调用 ETL（提取、转换和加载）方法 `create_dynamic_frame_from_options` 和 `create_dynamic_frame_from_catalog` 时，您也可以启用并行读取。有关在这些方法中指定选项的更多信息，请参阅 from_options 和 from_catalog。

您可以将此方法用于 JDBC 表，也即，其基本数据是 JDBC 数据存储的大多数表。当读取 Amazon Redshift 和 Amazon S3 表时，这些属性将被忽略。

hashfield

将 `hashfield` 设置为 JDBC 表中要用来将数据划分成分区的列的名称。为了获得最佳效果，此列应具有值的偶数分布以便在各分区之间分布数据。此列可以是任意数据类型。AWS Glue 生成非重叠的查询，这些查询并行运行以读取通过此列进行分区的数据。例如，如果您的数据按月份均匀地分布，您可以使用 `month` 列并行读取每月的数据。

```
'hashfield': 'month'
```

AWS Glue 创建一个查询以便将字段值哈希处理成分区编号，并针对这些分区并行运行该查询。要使用您自己的查询对表读取进行分区，请提供 hashexpression 而不是 hashfield。

hashexpression

将 hashexpression 设置为会返回一个整数的 SQL 表达式（符合 JDBC 数据库引擎语法）。简单表达式是表中任何数字列的名称。AWS Glue 生成 SQL 查询以并行读取 JDBC 数据（通过使用 WHERE 子句中的 hashexpression 对数据分区）。

例如，使用数字列 customerID 来读取按客户编号分区的数据。

```
'hashexpression': 'customerID'
```

要让 AWS Glue 控制分区，请提供 hashfield 而不是 hashexpression。

hashpartitions

将 hashpartitions 设置为 JDBC 表的并行读取数目。如果未设置该属性，默认值为 7。

例如，将并行读取数目设置为 5，以便 AWS Glue 通过五个（或更少）查询读取您的数据。

```
'hashpartitions': '5'
```

设置 Amazon VPC 以通过建立从 AWS Glue 到 Amazon RDS 数据存储的 JDBC 连接

使用 JDBC 连接到 Amazon RDS 中的数据库时，您需要执行额外的设置。要允许 AWS Glue 组件与 Amazon RDS 进行通信，您必须在 Amazon VPC 设置对 Amazon RDS 数据存储的访问权限。要支持 AWS Glue 在其组件之间通信，请为所有 TCP 端口指定一个具有自引用入站规则的安全组。通过创建自引用规则，您可以将源限制为 VPC 中的同一安全组。自引用规则不会向所有网络开放 VPC。VPC 的默认安全组可能已经为所有流量设置了自引用入站规则。

在 AWS Glue 和 Amazon RDS 数据存储之间设置访问权限

1. 登录 AWS Management Console 并打开 Amazon RDS 控制台，网址为 https://console.aws.amazon.com/rds/。

2. 在 Amazon RDS 控制台中，找到用于控制 Amazon RDS 数据库访问权限的安全组。

   在左侧导航窗格中，选择数据库，然后从主窗格的列表中选择要连接的实例。

   在数据库详细信息页面中，在连接和安全选项卡上找到 VPC 安全组。

3. 根据您的网络架构，确定哪个关联安全组最适合修改以允许 Glue AWS 服务访问。保存其名称，*database-security-group*以备将来参考。如果没有合适的安全组，请按照 Amazon RDS 文档中通过创建安全组提供对 VPC 中的数据库实例的访问部分的说明操作。

4. 登录 AWS Management Console 并打开亚马逊 VPC 控制台，网址为 https://console.aws.amazon.com/vpc/。

5. 在 Amazon VPC 控制台中，确定如何更新*database-security-group*。

   在左侧导航窗格中，选择安全组，然后*database-security-group*从主窗格的列表中进行选择。

6. 识别*database-security-group*、的安全组 ID *database-sg-id*。保存以备将来参考。

   在安全组详细信息页面中，找到安全组 ID。

7. 更改的入站规则 *database-security-group*，添加自引用规则以允许AWS Glue组件进行通信。具体而言，请添加或确认有一条规则，其中 “类型” 为All TCP，“协议” 为TCP，“端口范围” 包括所有端口，“源” 为*database-sg-id*。确认您为源输入的安全组正是您正在编辑的安全组。

   在安全组详细信息页面中，选择编辑入站规则。

   入站规则类似于以下内容：

类型	协议	端口范围	来源
所有 TCP	TCP	0–65535	*database-sg-id*

8. 添加出站流量规则。

   在安全组详细信息页面中，选择编辑出站规则。

   如果您的安全组允许所有出站流量，则不需要单独的规则。例如：

类型	协议	端口范围	目标位置
所有流量	ALL	ALL	0.0.0.0/0

如果您的网络架构旨在限制出站流量，请创建以下出站规则：

创建自引用规则，其中“类型”为All TCP，“协议”为TCP，“端口范围”包括所有端口，“目标”为。*database-sg-id*确认您为目标输入的安全组正是您正在编辑的安全组。

如果使用 Amazon S3 VPC 端点，添加 HTTPS 规则以允许从该 VPC 到 Amazon S3 的流量。创建一个规则，其中类型为HTTPS，协议为，端口范围为TCP，目标是 443 Amazon S3 网关终端节点 s *3-* 的托管前缀列表的 ID prefix-list-id。有关前缀列表和 Amazon S3 网关端点的更多信息，请参阅 Amazon VPC 文档中的 Amazon S3 网关端点。

例如：

类型	协议	端口范围	目标位置
所有 TCP	TCP	0–65535	*database-sg-id*
HTTPS	TCP	443	*s3-prefix-list-id*

MongoDB 连接

在 AWS Glue 4.0 及更高版本中，您可以使用 AWS Glue for Spark 来读取和写入 MongoDB 和 MongoDB Atlas 中的表。您可以通过 AWS Glue 连接并使用存储在 AWS Secrets Manager 中的用户名和密码凭证连接到 MongoDB。

有关 MongoDB 的更多信息，请参阅 MongoDB 文档。

配置 MongoDB 连接

要从 AWS Glue 连接到 MongoDB，您需要拥有 MongoDB 凭证 *mongodbUser* 和 *mongodbPass*。

要从 AWS Glue 连接到 MongoDB，您可能需要满足一些先决条件：

- 如果您的 MongoDB 实例位于某个 Amazon VPC 中，请确保您的 Amazon VPC 配置允许您的 AWS Glue 作业与 MongoDB 实例进行通信，并且无需通过公共互联网路由流量。

  在 Amazon VPC 中，确定或创建将在执行 AWS Glue 作业时使用的 VPC、子网和安全组。此外，您的 Amazon VPC 配置需要允许您的 MongoDB 实例与该位置之间的网络流量。根据您的网络布局，这可能需要更改安全组规则、网络 ACL、NAT 网关和对等连接。

然后您可以继续配置 AWS Glue 以便与 MongoDB 配合使用。

配置 MongoDB 连接：

1. 您还可以在 AWS Secrets Manager 中使用您的 MongoDB 凭证创建密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *secretName*，以供下一步使用。

   - 在选择键/值对时，请使用键 `username` 和值 *mongodbUser* 创建一个键值对。

     在选择键/值对时，请使用键 `password` 和值 *mongodbPass* 创建一个键值对。

2. 在 AWS Glue 控制台中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建一个连接。创建连接后，保留连接名为 *connectionName*，以供未来在 AWS Glue 中使用。

   - 选择连接类型时，请选择 MongoDB 或 MongoDB Atlas。
   - 选择 MongoDB URL 或 MongoDB Atlas URL 时，请提供 MongoDB 实例的主机名。

     MongoDB URL 的格式为 `mongodb://`*mongoHost*`:`*mongoPort*`/`*mongoDBname*。

     MongoDB Atlas URL 的格式为 `mongodb+srv://`*mongoHost*`:`*mongoPort*`/`*mongoDBname*。

     此外还可以选择提供连接的默认数据库 *mongoDBName*。
   - 如果您选择创建 Secrets Manager 密钥，请选择 AWS Secrets Manager 凭证类型。

     然后在 AWS密钥中提供 *secretName*。
   - *########## mongodbUser # mongodbPass*。

3. 对于下列情况，您可能需要添加额外的配置：

   - 对于通过 Amazon VPC 在 AWS 云端托管的 MongoDB 实例
     - 您需要向 AWS Glue 连接提供用于定义 MongoDB 安全凭证的 Amazon VPC 连接信息。创建或更新连接时，请在网络选项中设置 VPC、子网和安全组。

创建 AWS Glue MongoDB 连接后，您需要执行以下操作，然后才能调用您的连接方法：

- 如果您选择创建 Secrets Manager 密钥，请向与您的 AWS Glue 作业关联的 IAM 角色授予读取 *secretName* 的权限。
- 在 AWS Glue 作业配置中，提供 *connectionName* 作为附加网络连接。

要在 AWS Glue for Spark 中使用 AWS Glue MongoDB 连接，请在您连接方法调用中提供 `connectionName` 选项。您还可以按照 the section called “与 MongoDB 集成” 中的步骤操作，将该连接与 AWS Glue Data Catalog 结合使用。

使用 AWS Glue 连接从 MongoDB 读取

先决条件：

- 您要读取的 MongoDB 集合。您将需要该集合的标识信息。

  MongoDB 集合由数据库名 *mongodbName* 和集合名 *mongodbCollection* 来标识。
- 为了提供身份验证信息而配置的 AWS Glue MongoDB 连接。完成上一节“配置 MongoDB 连接”中的步骤，以配置您的身份验证信息。您需要 AWS Glue 连接的名称 *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",
    }
)
```

写入 MongoDB 表

此示例会将来自现有 DynamicFrame *dynamicFrame* 的信息写入 MongoDB。

先决条件：

- 您要写入的 MongoDB 集合。您将需要该集合的标识信息。

  MongoDB 集合由数据库名 *mongodbName* 和集合名 *mongodbCollection* 来标识。
- 为了提供身份验证信息而配置的 AWS Glue MongoDB 连接。完成上一节“配置 MongoDB 连接”中的步骤，以配置您的身份验证信息。您需要 AWS Glue 连接的名称 *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",
    },
)
```

读取和写入 MongoDB 表

此示例会将来自现有 DynamicFrame *dynamicFrame* 的信息写入 MongoDB。

先决条件：

- 您要读取的 MongoDB 集合。您将需要该集合的标识信息。

  您要写入的 MongoDB 集合。您将需要该集合的标识信息。

  MongoDB 集合由数据库名 *mongodbName* 和集合名 *mongodbCollection* 来标识。
- MongoDB 身份验证信息 *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 连接选项参考

指定与 MongoDB 的连接。源连接和接收器连接的连接选项不同。

源连接和接收器连接之间会共享以下连接属性：

- connectionName - 用于读/写。为了向您的连接方法提供身份验证和网络信息而配置的 AWS Glue MongoDB 连接的名称。如果在按照上一节“the section called “配置 MongoDB””所述配置 AWS Glue 连接时提供 connectionName，则不再需要提供 "uri"、"username" 和 "password" 连接选项。
- "uri"：（必需）要从中读取数据的 MongoDB 主机，格式为 mongodb://<host>:<port>。适用于 AWS Glue 4.0 之前的 AWS Glue 版本。
- "connection.uri"：（必需）要从中读取数据的 MongoDB 主机，格式为 mongodb://<host>:<port>。适用于 AWS Glue 4.0 及更高版本。
- "username"：（必需）MongoDB 用户名。
- "password"：（必需）MongoDB 密码。
- "database"：（必需）要从中读取数据的 MongoDB 数据库。当在您的任务脚本中调用 glue_context.create_dynamic_frame_from_catalog 时，此选项还可以在 additional_options 中传递。
- "collection"：（必需）要从中读取数据的 MongoDB 集合。当在您的任务脚本中调用 glue_context.create_dynamic_frame_from_catalog 时，此选项还可以在 additional_options 中传递。

"connectionType": "mongodb" as Source

将 "connectionType": "mongodb" 用作源时可使用以下连接选项：

- "ssl"：（可选）如果为 true，则启动 SSL 连接。默认为 false。
- "ssl.domain_match"：（可选）如果为 true，且 ssl 为 true，则执行域匹配检查。默认为 true。
- "batchSize"：（可选）每个批处理返回的文档数量，在内部批处理的游标中使用。
- "partitioner"：（可选）从 MongoDB 中读取输入数据的分区器的类名称。该连接器提供以下分区器：
  - MongoDefaultPartitioner（默认）（AWS Glue 4.0 不支持）
  - MongoSamplePartitioner（需要 MongoDB 3.2 或更高版本）（但 AWS Glue 4.0 不支持）
  - MongoShardedPartitioner（AWS Glue 4.0 不支持）
  - MongoSplitVectorPartitioner（AWS Glue 4.0 不支持）
  - MongoPaginateByCountPartitioner（AWS Glue 4.0 不支持）
  - MongoPaginateBySizePartitioner（AWS Glue 4.0 不支持）
  - 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"：（可选）指定分区器的选项。各个分区器支持的选项如下：
  - MongoSamplePartitioner: partitionKey, partitionSizeMB, samplesPerPartition
  - MongoShardedPartitioner: shardkey
  - MongoSplitVectorPartitioner: partitionKey, partitionSizeMB
  - MongoPaginateByCountPartitioner: partitionKey, numberOfPartitions
  - MongoPaginateBySizePartitioner: partitionKey, partitionSizeMB

  有关这些选项的更多信息，请参阅 MongoDB 文档中的分区器配置。

"connectionType": "mongodb" as Sink

将 "connectionType": "mongodb" 用作连接器时可使用以下连接选项：

- "ssl"：（可选）如果为 true，则启动 SSL 连接。默认为 false。
- "ssl.domain_match"：（可选）如果为 true，且 ssl 为 true，则执行域匹配检查。默认为 true。
- "extendedBsonTypes"：（可选）如果为 true，则在 MongoDB 中写入数据时会允许扩展 BSON 类型。默认为 true。

- `"replaceDocument"`：（可选）如果为 `true`，则在保存包含 `_id` 字段的数据集时会替换整个文档。如果为 `false`，则只会更新文档中与数据集中的字段匹配的字段。默认为 `true`。
- `"maxBatchSize"`：（可选）保存数据时的批量操作的最大批次大小。默认值为 512。
- `"retryWrites"`：（可选）：如果 AWS Glue 遇到网络错误，则会自动重试某些写入操作一次。

SAP HANA 连接

在 AWS Glue 4.0 及更高版本中，您可以使用 AWS Glue for Spark 来读取和写入 SAP HANA 中的表。您可以使用 SQL 查询来定义要从 SAP HANA 中读取的信息。您可以通过 AWS Glue SAP HANA 连接并使用存储在 AWS Secrets Manager 中的 JDBC 凭证连接到 SAP HANA。

有关 SAP HANA JDBC 的更多信息，请参阅 SAP HANA 文档。

配置 SAP HANA 连接

要从 AWS Glue 连接到 SAP HANA，您需要创建 SAP HANA 凭证并将其存储在某个 AWS Secrets Manager 密钥中，然后将该密钥关联到某个 SAP HANA AWS Glue 连接。您需要配置 SAP HANA 服务与 AWS Glue 之间的网络连接。

要连接到 SAP HANA，您可能需要满足一些先决条件：

- 如果您的 SAP HANA 服务位于某个 Amazon VPC 中，请确保您的 Amazon VPC 配置允许您的 AWS Glue 作业与 SAP HANA 服务进行通信，并且无需通过公共互联网路由流量。

  在 Amazon VPC 中，确定或创建将在执行 AWS Glue 作业时使用的 VPC、子网和安全组。此外，您的 Amazon VPC 配置需要允许您的 SAP HANA 端点与该位置之间的网络流量。您的作业需要与您的 SAP HANA JDBC 端口建立 TCP 连接。有关 SAP HANA 端口的更多信息，请参阅 SAP HANA 文档。根据您的网络布局，这可能需要更改安全组规则、网络 ACL、NAT 网关和对等连接。
- 如果您的 SAP HANA 端点可以访问互联网，则无需满足其他先决条件。

配置 SAP HANA 连接：

1. 在 AWS Secrets Manager 中，使用您的 SAP HANA 凭证创建一个密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *`secretName`*，以供下一步使用。
   - 在选择键/值对时，请使用键 `user` 和值 *`saphanaUsername`* 创建一个键值对。
   - 在选择键/值对时，请使用键 `password` 和值 *`saphanaPassword`* 创建一个键值对。

2. 在 AWS Glue 控制台中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建一个连接。创建连接后，保留连接名为 *connectionName*，以供未来在 AWS Glue 中使用。

   - 选择连接类型时，请选择 SAP HANA。
   - 在提供 SAP HANA URL 时，请提供您的实例的 URL。

     SAP HANA JDBC URL 的格式为 `jdbc:sap://`*saphanaHostname*`:`*saphanaPort*`/?`*databaseName*`=`*saphanaDBname*`,`*Paramete*

     AWS Glue 需要以下 JDBC URL 参数：

     - `databaseName` – SAP HANA 中要连接的默认数据库。
   - 选择 AWS 密钥时，请提供 *secretName*。

创建 AWS Glue SAP HANA 连接后，您需要完成以下操作，然后才能运行 AWS Glue 作业：

- 向与您的 AWS Glue 作业关联的 IAM 角色授予读取 *secretName* 的权限。
- 在 AWS Glue 作业配置中，提供 *connectionName* 作为附加网络连接。

读取 SAP HANA 表

先决条件：

- 您要读取的 SAP HANA 表。您将需要该表的标识信息。

  可以使用 SAP HANA 表名和 Schema 名（格式为 *schemaName*`.`*tableName*）来指定表。如果表的 Schema 为默认的“public”，则不需要指定 Schema 名和“.”分隔符。调用此 *tableIdentifier* 操作。请注意，数据库在 `connectionName` 中是以 JDBC URL 参数的形式提供的。
- 为了提供身份验证信息而配置的 AWS Glue SAP HANA 连接。完成上一节“配置 SAP HANA 连接”中的步骤，以配置您的身份验证信息。您需要 AWS Glue 连接的名称 *connectionName*。

例如：

```
saphana_read_table = glueContext.create_dynamic_frame.from_options(
    connection_type="saphana",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableIdentifier",
    }
```

```
)
```

您还可以提供 SELECT SQL 查询来筛选返回到 DynamicFrame 的结果。您将需要配置 `query`。

例如：

```
saphana_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="saphana",
    connection_options={
        "connectionName": "connectionName",
        "query": "query"
    }
)
```

写入 SAP HANA 表

此示例会将来自现有 DynamicFrame *dynamicFrame* 的信息写入 SAP HANA。如果表中已经含有信息，则 AWS Glue 会出现错误。

先决条件：

- 您要写入的 SAP HANA 表。

  可以使用 SAP HANA 表名和 Schema 名（格式为 *schemaName*.*tableName*）来指定表。如果表的 Schema 为默认的“public”，则不需要指定 Schema 名和“.”分隔符。调用此 *tableIdentifier* 操作。请注意，数据库在 `connectionName` 中是以 JDBC URL 参数的形式提供的。
- SAP HANA 身份验证信息。完成上一节“配置 SAP HANA 连接”中的步骤，以配置您的身份验证信息。您需要 AWS Glue 连接的名称 *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 连接选项参考

- `connectionName` – 必需。用于读/写。为了向您的连接方法提供身份验证和网络信息而配置的 AWS Glue SAP HANA 连接的名称。
- `databaseName` - 用于读/写。有效值：SAP HANA 中数据库的名称。要连接的数据库的名称。
- `dbtable` – 对于写入为必填项，对于读取也为必填项，但提供了 `query` 时除外。用于读/写。有效值：SAP HANA SQL FROM 子句的内容。标识要连接的 SAP HANA 中的表。除表名之外，您还可以提供其他 SQL，例如子查询。有关更多信息，请参阅 SAP HANA 文档中的 From clause。
- `query` – 用于读取。用来定义从 SAP HANA 读取数据时要检索的内容的 SAP HANA SQL SELECT 查询。

Snowflake 连接

在 AWS Glue 4.0 及更高版本中，您可以使用 AWS Glue for Spark 在 Snowflake 中读取和写入表。您可以通过 SQL 查询从 Snowflake 中读取。您可以使用用户名和密码连接到 Snowflake。您可以通过 AWS Glue Data Catalog 引用存储在 AWS Secrets Manager 中的 Snowflake 凭证。用于 AWS Glue for Spark 的 Data Catalog Snowflake 凭证与用于爬网程序的 Data Catalog Snowflake 凭证分开存储。必须选择 `SNOWFLAKE` 类型连接，而不是配置为连接到 Snowflake 的 `JDBC` 类型连接。

有关 Snowflake 的更多信息，请参阅 Snowflake 网站。有关 AWS 上 Snowflake 的更多信息，请参阅 Snowflake Data Warehouse on Amazon Web Services。

配置 Snowflake 连接

连接到可通过互联网访问的 Snowflake 数据库没有任何 AWS 先决条件。

或者，您可以执行以下配置，使用 AWS Glue 管理连接凭证。

使用 AWS Glue 管理连接凭证

1. 在 Snowflake 中，生成一个用户 *`snowflakeUser`* 和密码 *`snowflakePassword`*。
2. 在 AWS Secrets Manager 中，使用您的 Snowflake 凭证创建密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *`secretName`*，以供下一步使用。
   - 选择键/值对时，请使用键 `sfUser` 为 *`snowflakeUser`* 创建一对。
   - 选择键/值对时，请使用键 `sfPassword` 为 *`snowflakePassword`* 创建一对。
   - 选择键/值对时，您可以向 Snowflake 仓库提供键 `sfWarehouse`。

3. 在 AWS Glue Data Catalog 中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建连接。创建连接后，保留连接名称 *connectionName*，以供下一步使用。

   - 选择连接类型时，请选择 Snowflake。
   - 选择 Snowflake URL 时，请提供您的 Snowflake 实例的 URL。URL 将使用表单 *account_identifier*.snowflakecomputing.com 中的主机名。
   - 选择 AWS 密钥时，请提供 *secretName*。

4. 在 AWS Glue 作业配置中，提供 *connectionName* 作为附加网络连接。

在以下情况下，您可能需要以下内容：

- 适用于托管在 AWS 中 Amazon VPC 上的 Snowflake
  - 您需要对 Snowflake 进行适当的 Amazon VPC 配置。有关如何配置 Amazon VPC 的更多信息，请参阅 Snowflake 文档中的 AWS PrivateLink & Snowflake。
  - 您需要对 AWS Glue 进行适当的 Amazon VPC 配置。the section called “VPC 端点 (AWS PrivateLink)”
  - 您需要创建一个提供 Amazon VPC 连接信息的 AWS Glue Data Catalog 连接（以及定义您的 Snowflake 安全凭证的 AWS Secrets Manager 密钥 ID）。使用 AWS PrivateLink 时您的 URL 将发生变化，如前一项目中链接的 Snowflake 文档中所述。
  - 您需要在作业配置中将 Data Catalog 连接作为附加网络连接包括在内。

从 Snowflake 表中读取

先决条件：您想读取的 Snowflake 表。您需要使用 Snowflake 表名 *tableName*。您需要 Snowflake url *snowflakeUrl*、用户名 *snowflakeUser* 和密码 *snowflakePassword*。如果您的 Snowflake 用户没有设置默认命名空间，则需要 Snowflake 数据库名称 *databaseName* 和架构名称 *schemaName*。此外，如果您的 Snowflake 用户没有设置默认仓库，则需要仓库名称 *warehouseName*。

例如：

其他先决条件：完成使用 AWS Glue 管理连接凭证的步骤，配置 *snowflakeUrl*、*snowflakeUsername* 和 *snowflakePassword*。要查看这些步骤，请参阅 the section called “配置 Snowflake” 上一节。为选择要连接的附加网络连接，我们将使用 connectionName 参数。

```
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，您还可以使用 autopushdown 和 query 参数来读取 Snowflake 表的一部分。这可能比在结果加载到 Spark 后对其进行筛选要有效得多。举一个例子，其中所有销售额都存储在同一个表中，但您只需要分析假日期间某家商店的销售额即可。如果这些信息存储在表中，则可以使用谓词下推来检索结果，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",
    }
)
```

写入 Snowflake 表

先决条件：您要写入的 Snowflake 数据库。您将需要一个最新或所需的表名，如 *tableName*。您需要 Snowflake url *snowflakeUrl*、用户名 *snowflakeUser* 和密码 *snowflakePassword*。如果您的 Snowflake 用户没有设置默认命名空间，则需要 Snowflake 数据库名称 *databaseName* 和架构名称 *schemaName*。此外，如果您的 Snowflake 用户没有设置默认仓库，则需要仓库名称 *warehouseName*。

例如：

其他先决条件：完成使用 AWS Glue 管理连接凭证的步骤，配置 *snowflakeUrl*、*snowflakeUsername* 和 *snowflakePassword*。要查看这些步骤，请参阅 the section called “配置 Snowflake” 上一节。为选择要连接的附加网络连接，我们将使用 connectionName 参数。

```
glueContext.write_dynamic_frame.from_options(
    connection_type="snowflake",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName",
        "sfDatabase": "databaseName",
        "sfSchema": "schemaName",
        "sfWarehouse": "warehouseName",
    },
)
```

Snowflake 连接选项参考

Snowflake 连接类型采用以下连接选项：

您可以从 Data Catalog 连接（sfUrl、sfUser、sfPassword）中检索本节中的某些参数，在这种情况下，您无需提供这些参数。您可以提供参数 connectionName 来实现。

您可以从 AWS Secrets Manager 密钥（sfUser、sfPassword）中检索本节中的某些参数，在这种情况下，您无需提供这些参数。密钥必须提供 sfUser 和 sfPassword 键下的内容。您可以提供参数 secretId 来实现。

连接到 Snowflake 时通常使用以下参数。

- sfDatabase - 如果未在 Snowflake 中设置用户默认值，则为必填项。用于读/写。连接后用于会话的数据库。
- sfSchema - 如果未在 Snowflake 中设置用户默认值，则为必填项。用于读/写。连接后用于会话的架构。
- sfWarehouse - 如果未在 Snowflake 中设置用户默认值，则为必填项。用于读/写。连接后用于会话的默认虚拟仓库。
- sfRole - 如果未在 Snowflake 中设置用户默认值，则为必填项。用于读/写。连接后用于会话的默认安全角色。
- sfUrl -（必需）用于读/写。采用以下格式指定账户的主机名：*account_identifier*.snowflakecomputing.com。有关账户标识符的更多信息，请参阅 Snowflake 文档中的 Account Identifiers。
- sfUser -（必需）用于读/写。Snowflake 用户的登录名。
- sfPassword —（除非提供 pem_private_key，否则为必填项）用于读/写。Snowflake 用户的密码。

- dbtable - 处理完整表格时为必填项。用于读/写。要读取的表的名称或要写入数据的表的名称。读取时，将检索所有列和记录。
- pem_private_key - 用于读/写。未加密的 b64 编码私钥字符串。Snowflake 用户的私钥。通常将其从 PEM 文件中复制出来。有关更多信息，请参阅 Snowflake 文档中的密钥对身份验证和密钥对轮换。
- query - 使用查询读取时为必填项。用于读取。要运行的确切查询（SELECT 语句）

以下选项用于配置连接到 Snowflake 过程中的特定行为。

- preactions - 用于读/写。有效值：以分号分隔的 SQL 语句列表作为字符串。SQL 语句在 AWS Glue 和 Snowflake 之间传输数据之前运行。如果语句包含 %s，则 %s 将替换为操作所引用的表名。
- postactions - 用于读/写。SQL 语句在 AWS Glue 和 Snowflake 之间传输数据之后运行。如果语句包含 %s，则 %s 将替换为操作所引用的表名。
- autopushdown - 默认值："on"。有效值："on"、"off"。此参数控制是否启用自动查询下推。如果启用了下推，那么当在 Spark 上运行查询时，如果可以将部分查询“下推”到 Snowflake 服务器，则会将其下推。这提高了某些查询的性能。有关是否可以下推查询的信息，请参阅 Snowflake 文档中的 Pushdown。

此外，AWS Glue 可能支持 Snowflake Spark 连接器上可用的某些选项。有关 Snowflake Spark 连接器上可用选项的更多信息，请参阅 Snowflake 文档中的 Setting Configuration Options for the Connector。

Snowflake 连接器限制

使用 AWS Glue for Spark 连接到 Snowflake 需要遵守以下限制。

- 此连接器不支持作业书签。有关作业书签的更多信息，请参阅 the section called “使用作业书签跟踪已处理的数据”。
- 此连接器不支持 Snowflake 使用 create_dynamic_frame.from_catalog 和 write_dynamic_frame.from_catalog 方法对 AWS Glue Data Catalog 中的表进行读取和写入。
- 此连接器不支持使用除用户名和密码之外的凭证连接到 Snowflake。
- 流作业中不支持此连接器。
- 此连接器在检索信息（例如使用 query 参数）时支持基于 SELECT 语句的查询。不支持其他类型的查询（例如 SHOW、DESC 或 DML 语句）。

- Snowflake 将通过 Snowflake 客户端提交的查询文本（即 SQL 语句）的大小限制为每条语句 1MB。有关更多详细信息，请参阅 Limits on Query Text Size。

Teradata Vantage 连接

在 AWS Glue 4.0 及更高版本中，您可以使用 AWS Glue for Spark 来读取和写入 Teradata Vantage 中的表。您可以使用 SQL 查询来定义要从 Teradata 中读取的信息。您可以通过 AWS Glue 连接并使用存储在 AWS Secrets Manager 中的用户名和密码凭证连接到 Teradata。

有关 Teradata 的更多信息，请参阅 Teradata 文档

配置 Teradata 连接

要从 AWS Glue 连接到 Teradata，您需要创建 Teradata 凭证并将其存储在某个 AWS Secrets Manager 密钥中，然后将该密钥关联到某个 AWS Glue Teradata 连接。如果您的 Teradata 实例位于某个 Amazon VPC 中，则还需要提供 AWS Glue Teradata 连接的联网选项。

要从 AWS Glue 连接到 Teradata，您可能需要满足一些先决条件：

- 如果您通过 Amazon VPC 访问您的 Teradata 环境，您的 Amazon VPC 配置应允许您的 AWS Glue 作业与 Teradata 环境进行通信。我们不建议通过公共互联网访问 Teradata 环境。

  在 Amazon VPC 中，确定或创建将在执行 AWS Glue 作业时使用的 VPC、子网和安全组。此外，您的 Amazon VPC 配置需要允许您的 Teradata 实例与该位置之间的网络流量。您的作业需要与您的 Teradata 客户端端口建立 TCP 连接。有关 Teradata 端口的更多信息，请参阅 Teradata 文档。

  根据您的网络布局，可能需要更改 Amazon VPC 和其他联网服务以建立安全的 VPC 连接。有关 AWS 连接的更多信息，请参阅 Teradata 文档中的 AWS Connectivity Options。

配置 AWS Glue Teradata 连接：

1. 在您的 Teradata 配置中，确定或创建 AWS Glue 将用于连接的用户 *teradataUser* 和密码 *teradataPassword*。有关更多信息，请参阅 Teradata 文档中的 Vantage Security Overview。
2. 在 AWS Secrets Manager 中，使用您的 Teradata 凭证创建一个密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *secretName*，以供下一步使用。
   - 在选择键/值对时，请使用键 `user` 和值 *teradataUsername* 创建一个键值对。
   - 在选择键/值对时，请使用键 `password` 和值 *teradataPassword* 创建一个键值对。

3. 在 AWS Glue 控制台中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建一个连接。创建连接后，保留连接名称 *connectionName*，以供下一步使用。

   - 选择连接类型时，请选择 Teradata。
   - 在提供 JDBC URL 时，请提供您的实例的 URL。您还可以在 JDBC URL 中将某些连接参数进行硬编码，并以逗号分隔。该 URL 必须符合以下格式：`jdbc:teradata://`*teradataHostname*`/`*ParameterName*`=`*ParameterValue*`,`*ParameterN*

     支持的 URL 参数包括：

     - DATABASE – 主机上默认要访问的数据库的名称。
     - DBS_PORT – 在非标准端口上运行时要使用的数据库端口。
   - 选择凭证类型时，请选择 AWS Secrets Manager，然后将 AWS 密钥设置为 *secretName*。
4. 对于下列情况，您可能需要添加额外的配置：

   - 对于通过 Amazon VPC 在 AWS 云端托管的 Teradata 实例
     - 您需要向 AWS Glue 连接提供用于定义 Teradata 安全凭证的 Amazon VPC 连接信息。创建或更新连接时，请在网络选项中设置 VPC、子网和安全组。

创建 AWS Glue Teradata 连接后，您需要执行以下操作，然后才能调用您的连接方法：

- 向与您的 AWS Glue 作业关联的 IAM 角色授予读取 *secretName* 的权限。
- 在 AWS Glue 作业配置中，提供 *connectionName* 作为附加网络连接。

从 Teradata 读取

先决条件：

- 您要读取的 Teradata 表。您需要表名 *tableName*。
- 为了提供身份验证信息而配置的 AWS Glue Teradata 连接。完成“配置 Teradata 连接”部分的步骤以配置您的身份验证信息。您需要 AWS Glue 连接的名称 *connectionName*。

例如：

```
teradata_read_table = glueContext.create_dynamic_frame.from_options(
    connection_type="teradata",
    connection_options={
        "connectionName": "connectionName",
```

```
        "dbtable": "tableName"
    }
)
```

您还可以提供 SELECT SQL 查询来筛选返回到 DynamicFrame 的结果。您将需要配置 `query`。

例如：

```
teradata_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="teradata",
    connection_options={
        "connectionName": "connectionName",
        "query": "query"
    }
)
```

写入 Teradata 表

先决条件：您要写入的 Teradata 表 *tableName*。必须首先创建表，然后才能调用连接方法。

例如：

```
teradata_write = glueContext.write_dynamic_frame.from_options(
    connection_type="teradata",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName"
    }
)
```

Teradata 连接选项参考

- `connectionName` – 必需。用于读/写。为了向您的连接方法提供身份验证和网络信息而配置的 AWS Glue Teradata 连接的名称。
- `dbtable` – 对于写入为必填项，对于读取也为必填项，但提供了 `query` 时除外。用于读/写。您的连接方法将与之交互的表的名称。
- `query` – 用于读取。用来定义从 Teradata 读取数据时要检索的内容的 SQL SELECT 查询。

Vertica 连接

在 AWS Glue 4.0 及更高版本中，您可以使用 AWS Glue for Spark 来读取和写入 Vertica 中的表。您可以使用 SQL 查询来定义要从 Vertica 中读取的信息。您可以通过 AWS Glue 连接并使用存储在 AWS Secrets Manager 中的用户名和密码凭证连接到 Vertica。

有关 Vertica 的更多信息，请参阅 Vertica 文档。

配置 Vertica 连接

要从 AWS Glue 连接到 Vertica，您需要创建 Vertica 凭证并将其存储在某个 AWS Secrets Manager 密钥中，然后将该密钥关联到某个 Vertica AWS Glue 连接。如果您的 Vertica 实例位于某个 Amazon VPC 中，则还需要提供 AWS Glue Vertica 连接的联网选项。您需要提供读取和写入数据库时用于临时存储的 Amazon S3 存储桶或文件夹。

要从 AWS Glue 连接到 Vertica，您将需要满足一些先决条件：

- 读取和写入数据库时用于临时存储的 Amazon S3 存储桶或文件夹，也称为 *tempS3Path*。

> **Note**
>
> 在 AWS Glue 任务数据预览中使用 Vertica 时，临时文件可能不会自动从 *tempS3Path* 中删除。为确保删除临时文件，请在数据预览窗格中选择结束会话，以直接结束数据预览会话。如果无法保证数据预览会话直接结束，请考虑将 Amazon S3 生命周期配置设置为删除旧数据。我们建议根据最大作业运行时间加一定的裕度移除已存在超过 49 小时的数据。有关配置 Amazon S3 生命周期的更多信息，请参阅 Amazon S3 文档中的 管理存储生命周期。

- 对您的 Amazon S3 路径具有适当权限，并且您可以将其关联到您的 AWS Glue 作业角色的 IAM policy。
- 如果您的 Vertica 实例位于某个 Amazon VPC 中，请确保您的 Amazon VPC 配置允许您的 AWS Glue 作业与 Vertica 实例进行通信，并且无需通过公共互联网路由流量。

  在 Amazon VPC 中，确定或创建将在执行 AWS Glue 作业时使用的 VPC、子网和安全组。此外，您的 Amazon VPC 配置需要允许您的 Vertica 实例与该位置之间的网络流量。您的作业需要与您的 Vertica 客户端端口（默认为 5433）建立 TCP 连接。根据您的网络布局，这可能需要更改安全组规则、网络 ACL、NAT 网关和对等连接。

然后您可以继续配置 AWS Glue 以便与 Vertica 配合使用。

配置 Vertica 连接：

1. 在 AWS Secrets Manager 中，使用您的 Vertica 凭证 *verticaUsername* 和 *verticaPassword* 创建一个密钥。要在 Secrets Manager 中创建密钥，请按照 AWS Secrets Manager 文档中创建 AWS Secrets Manager 密钥中的教程进行操作。创建密钥后，保留密钥名称 *secretName*，以供下一步使用。
   - 在选择键/值对时，请使用键 `user` 和值 *verticaUsername* 创建一个键值对。
   - 在选择键/值对时，请使用键 `password` 和值 *verticaPassword* 创建一个键值对。
2. 在 AWS Glue 控制台中，按照 the section called “添加 AWS Glue 连接” 中的步骤创建一个连接。创建连接后，保留连接名称 *connectionName*，以供下一步使用。
   - 选择连接类型时，请选择 Vertica。
   - 选择 Vertica 主机时，请提供您安装了 Vertica 的主机名。
   - 选择 Vertica 端口时，请提供可用于访问 Vertica 安装的端口。
   - 选择 AWS 密钥时，请提供 *secretName*。
3. 对于下列情况，您可能需要添加额外的配置：
   - 对于通过 Amazon VPC 在 AWS 云端托管的 Vertica 实例
     - 向 AWS Glue 连接提供用于定义 Vertica 安全凭证的 Amazon VPC 连接信息。创建或更新连接时，请在网络选项中设置 VPC、子网和安全组。

创建 AWS Glue Vertica 连接后，您需要执行以下操作，然后才能调用您的连接方法：

- 向与您的 AWS Glue 作业关联的 IAM 角色授予对 *tempS3Path* 的权限。
- 向与您的 AWS Glue 作业关联的 IAM 角色授予读取 *secretName* 的权限。
- 在 AWS Glue 作业配置中，提供 *connectionName* 作为附加网络连接。

从 Vertica 读取

先决条件：

- 您要读取的 Vertica 表。您需要 Vertica 数据库名 *dbName* 和表名 *tableName*。
- 为了提供身份验证信息而配置的 AWS Glue Vertica 连接。完成上一节“配置 Vertica 连接”中的步骤，以配置您的身份验证信息。您需要 AWS Glue 连接的名称 *connectionName*。

- 前面提到的用作临时存储的 Amazon S3 存储桶或文件夹。您将需要名称 *tempS3Path*。您将需要使用 s3a 协议连接到此位置。

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",
    }
)
```

您还可以提供 SELECT SQL 查询来筛选返回到 DynamicFrame 的结果，或者访问来自多个表的数据集。

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",
    },
)
```

写入 Vertica 表

此示例会将来自现有 DynamicFrame *dynamicFrame* 的信息写入 Vertica。如果表中已经含有信息，AWS Glue 会将来自 DynamicFrame 的数据附加到现有信息之后。

先决条件：

- 您要写入的当前已有或需要的表名 *tableName*。您还需要相应的 Vertica 数据库名 *dbName*。
- 为了提供身份验证信息而配置的 AWS Glue Vertica 连接。完成上一节“配置 Vertica 连接”中的步骤，以配置您的身份验证信息。您需要 AWS Glue 连接的名称 *connectionName*。

- 前面提到的用作临时存储的 Amazon S3 存储桶或文件夹。您将需要名称 *tempS3Path*。您将需要使用 s3a 协议连接到此位置。

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 连接选项参考

- connectionName – 必需。用于读/写。为了向您的连接方法提供身份验证和网络信息而配置的 AWS Glue Vertica 连接的名称。
- db – 必需。用于读/写。您的连接方法将与之交互的 Vertica 中数据库的名称。
- dbSchema – 如果需要标识您的表，则为必填项。用于读/写。默认值：public。您的连接方法将与之交互的 Schema 的名称。
- table – 对于写入为必填项，对于读取也为必填项，但提供了 query 时除外。用于读/写。您的连接方法将与之交互的表的名称。
- query – 用于读取。用来定义从 Teradata 读取数据时要检索的内容的 SQL SELECT 查询。
- staging_fs_url – 必需。用于读/写。有效值：s3a URL。用作临时存储的 Amazon S3 存储桶或文件夹的 URL。

自定义和 AWS Marketplace connectionType 值

这些功能包括：

- "connectionType": "marketplace.athena"：指定与 Amazon Athena 数据存储的连接。连接使用来自 AWS Marketplace 的连接器。
- "connectionType": "marketplace.spark"：指定与 Apache Spark 数据存储的连接。连接使用来自 AWS Marketplace 的连接器。

- `"connectionType": "marketplace.jdbc"`：指定与 JDBC 数据存储的连接。连接使用来自 AWS Marketplace 的连接器。
- `"connectionType": "custom.athena"`：指定与 Amazon Athena 数据存储的连接。连接使用您上传到 AWS Glue Studio 的自定义连接器。
- `"connectionType": "custom.spark"`：指定与 Apache Spark 数据存储的连接。连接使用您上传到 AWS Glue Studio 的自定义连接器。
- `"connectionType": "custom.jdbc"`：指定与 JDBC 数据存储的连接。连接使用您上传到 AWS Glue Studio 的自定义连接器。

适用于类型 custom.jdbc 或 marketplace.jdbc 的连接选项

- `className` – 字符串，必需，驱动程序类名称。
- `connectionName` – 字符串，必需，与连接器关联的连接的名称。
- `url` – 字符串，必需，用于建立与数据源的连接且带占位符（`${}`）的 JDBC URL。占位符 `${secretKey}` 替换为 AWS Secrets Manager 中同名的密钥。有关构建 URL 的详细信息，请参阅数据存储文档。
- `secretId` 或 `user/password` – 字符串，必需，用于检索 URL 的凭证。
- `dbTable` 或 `query` – 字符串，必需，从中获取数据的表或 SQL 查询。您可以指定 `dbTable` 或 `query`，但不能同时指定两者。
- `partitionColumn` – 字符串，可选，用于分区的整数列的名称。此选项仅在包含 `lowerBound`、`upperBound` 和 `numPartitions` 时有效。此选项的工作方式与 Spark SQL JDBC 阅读器中的工作方式相同。有关更多信息，请参阅《Apache Spark SQL、DataFrame 和 Dataset 指南》中的 JDBC 转换到其他数据库。

  `lowerBound` 和 `upperBound` 值用于确定分区步长，而不是用于筛选表中的行。对表中的所有行进行分区并返回。

> **Note**
>
> 使用查询（而不是表名称）时，您应验证查询是否适用于指定的分区条件。例如：
>
> - 如果您的查询格式为 `"SELECT col1 FROM table1"`，则在使用分区列的查询结尾附加 `WHERE` 子句，以测试查询。
> - 如果您的查询格式为 `SELECT col1 FROM table1 WHERE col2=val"`，则通过 `AND` 和使用分区列的表达式扩展 `WHERE` 子句，以测试查询。

- lowerBound – 整数，可选，用于确定分区步长的最小 partitionColumn 值。
- upperBound – 整数，可选，用于确定分区步长的最大 partitionColumn 值。
- numPartitions – 整数，可选，分区数。此值以及 lowerBound（包含）和 upperBound（排除）为用于拆分 partitionColumn 而生成的 WHERE 子句表达式构成分区步长。

> **Important**
>
> 请注意分区的数量，因为分区过多可能会导致外部数据库系统出现问题。

- filterPredicate – 字符串，可选，用于筛选源数据的额外条件子句。例如：

```
BillingCity='Mountain View'
```

  使用查询（而不是表名称）时，您应验证查询是否适用于指定的 filterPredicate。例如：

  - 如果您的查询格式为 "SELECT col1 FROM table1"，则在使用筛选条件谓词的查询结尾附加 WHERE 子句，以测试查询。
  - 如果您的查询格式为 "SELECT col1 FROM table1 WHERE col2=val"，则通过 AND 和使用筛选条件谓词的表达式扩展 WHERE 子句，以测试查询。

- dataTypeMapping – 目录，可选，用于构建从 JDBC 数据类型到 Glue 数据类型的映射的自定义数据类型映射。例如，选项 "dataTypeMapping":{"FLOAT":"STRING"} 会通过调用驱动程序的 ResultSet.getString() 方法，将 JDBC 类型 FLOAT 的数据字段映射到 Java String 类型，并将其用于构建 AWS Glue 记录。ResultSet 对象由每个驱动程序实现，因此行为特定于您使用的驱动程序。请参阅 JDBC 驱动程序的文档，了解驱动程序执行转换的方式。
- 目前受支持的 AWS Glue 数据类型包括：
  - DATE
  - string
  - TIMESTAMP
  - INT
  - FLOAT
  - LONG
  - BIGDECIMAL
  - BYTE
  - SHORT
  - DOUBLE

支持的 JDBC 数据类型为 Java8 java.sql.types。

默认数据类型映射（从 JDBC 到 AWS Glue）如下：

- 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

如果将自定义数据类型映射与选项 `dataTypeMapping` 结合使用，则可以覆盖默认数据类型映射。只有 `dataTypeMapping` 选项中列出的 JDBC 数据类型会受到影响；默认映射适用于所有其他 JDBC 数据类型。如果需要，您可以为其他 JDBC 数据类型添加映射。如果默认映射或自定义映射中均未包含 JDBC 数据类型，则数据类型默认转换为 AWS Glue `STRING` 数据类型。

以下 Python 代码示例演示了如何使用 AWS Marketplace JDBC 驱动程序从 JDBC 数据库读取数据。它演示了如何从数据库读取数据并将数据写入 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()
```

适用于类型 custom.athena 或 marketplace.athena 的连接选项

- `className` – 字符串，必需，驱动程序类名称。当您使用 Athena-CloudWatch 连接器时，此参数值是类名称（例如 `"com.amazonaws.athena.connectors"`）的前缀。Athena-CloudWatch 连接器由两个类组成：元数据处理程序和记录处理程序。如果您在此处提供通用前缀，则 API 会根据该前缀加载正确的类。
- `tableName` – 字符串，必需，要读取的 CloudWatch 日志流的名称。此代码段使用特别视图名称 `all_log_streams`，这意味着返回的动态数据框将包含日志组中所有日志流的数据。
- `schemaName` – 字符串，必需，要从中读取数据的 CloudWatch 日志流的名称。例如，`/aws-glue/jobs/output`。
- `connectionName` – 字符串，必需，与连接器关联的连接的名称。

有关此连接器的其他选项，请参阅 GitHub 上的 Amazon Athena CloudWatch 连接器自述文件。

以下 Python 代码示例演示了如何从使用 AWS Marketplace 连接器的 Athena 数据存储读取数据。它演示了如何从 Athena 读取数据并将数据写入 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()
```

适用于类型 custom.spark 或 marketplace.spark 的连接选项

- `className` – 字符串，必需，连接器类名称。
- `secretId` – 字符串，可选，用于检索连接器连接的凭证。
- `connectionName` – 字符串，必需，与连接器关联的连接的名称。
- 其他选项取决于数据存储。例如，OpenSearch 配置选项以前缀 `es` 开头，正如适用于 Apache Hadoop 的 Elasticsearch 文档中所述。Spark 与 Snowflake 的连接使用 `sfUser` 和 `sfPassword` 等连接，正如《连接 Snowflake》指南中的使用 Spark 连接器所述。

以下 Python 代码示例演示了如何从使用 `marketplace.spark` 连接的 OpenSearch 数据存储读取数据。

```
    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()
```

常规选项

本节中的选项以 connection_options 形式提供，但不是专门适用于一个连接器。

配置书签时通常使用以下参数。它们可能适用于 Amazon S3 或 JDBC 工作流程。有关更多信息，请参阅 the section called “使用作业书签”。

- jobBookmarkKeys - 列名称的数组。

- `jobBookmarkKeysSortOrder` - 定义如何根据排序顺序比较值的字符串。有效值："asc"、"desc"。
- `useS3ListImplementation` - 用于在列出 Amazon S3 存储桶内容时管理内存性能。有关更多信息，请参阅 Optimize memory management in AWS Glue。

AWS Glue for Spark 中的输入和输出的数据格式选项

这些页面提供有关 AWS Glue for Spark 支持的数据格式的功能支持和配置参数的信息。有关这些信息用法和适用性的说明，请参阅以下内容。

AWS Glue 中跨数据格式的支持功能

每种数据格式可能支持不同的 AWS Glue 功能。您的数据格式是否以下常用功能应视其类型而定。请参阅数据格式的相关文档，了解如何利用我们的功能满足您的需求。

读取	AWS Glue 无需额外资源（例如连接器）即可识别和解释此数据格式。
写入	AWS Glue 可以在没有额外资源的情况下以此格式写入数据。您可以在任务中加入第三方库并使用标准 Apache Spark 函数来写入数据，就像在其他 Spark 环境中一样。有关库的更多信息，请参阅 the section called “Python 库”。
流式处理读取	AWS Glue 可以从 Apache Kafka、Amazon Managed Streaming for Apache Kafka 或 Amazon Kinesis 消息流中识别和解释此数据格式。我们期望流以一致的格式呈现数据，因此数据将读入为 `DataFrames` 。
对小文件进行分组	AWS Glue 在执行 AWS Glue 转换时可以将文件合并到发送至每个节点的批处理工作中。这可以显著提高涉及大量小文件的工作负载性能。有关更多信息，请参阅 the section called “对输入文件分组”。
作业书签	AWS Glue 可以跟踪转换的进度，在任务运行期间使用任务书签对相同数据集执行相同的工作。

这可以提高涉及多个数据集且其中只需要处理上次任务运行之后产生的新数据的工作负载性能。有关更多信息，请参阅 the section called “使用作业书签跟踪已处理的数据”。

AWS Glue 中用于与数据格式交互的参数

某些 AWS Glue 连接类型支持多种 `format` 类型，这需要您在使用类似 `GlueContext.write_dynamic_frame.from_options` 的方法时，使用 `format_options` 对象指定关于数据格式的信息。

- `s3` – 有关更多信息，请参阅 AWS Glue 中的 ETL 的连接类型和选项：S3 连接参数。您还可以查看关于支持此连接类型的方法的文档：Python 中的 the section called “create_dynamic_frame_from_options” 和 the section called “write_dynamic_frame_from_options” 以及相应的 Scala 方法 the section called “getSourceWith格式” 和 the section called “getSinkWith格式”。

- `kinesis` – 有关更多信息，请参阅 AWS Glue 中的 ETL 的连接类型和选项：Kinesis 连接参数。您还可以查看关于支持此连接类型的方法的文档：the section called “create_data_frame_from_options” 以及相应的 Scala 方法 the section called “createDataFrameFromOptions”。
- `kafka` – 有关更多信息，请参阅 AWS Glue 中的 ETL 的连接类型和选项：Kafka 连接参数。您还可以查看关于支持此连接类型的方法的文档：the section called “create_data_frame_from_options” 以及相应的 Scala 方法 the section called “createDataFrameFromOptions”。

有些连接类型不需要 `format_options`。例如，在正常使用过程中，连接至关系数据库的 JDBC 连接以一致的表格数据格式检索数据。因此，从 JDBC 连接中进行读取不需要 `format_options`。

某些在 Glue 中读取和写入数据的方法不需要 `format_options`。例如，通过 AWS Glue 爬网程序使用 `GlueContext.create_dynamic_frame.from_catalog`。爬网程序决定数据的形状。使用爬网程序时，AWS Glue 分类器将检查数据，以便就如何表示数据格式做出明智决策。然后，它会将数据表示形式存储在 AWS Glue 数据目录中，此数据目录可以在 AWS Glue ETL 脚本中使用，以通过 `GlueContext.create_dynamic_frame.from_catalog` 方法检索数据。爬网程序无需您手动指定有关数据格式的信息。

对于访问 AWS Lake Formation 受管表的任务，AWS Glue 支持读取和写入 Lake Formation 受管表支持的所有格式。有关当前 AWS Lake Formation 受管表支持的格式列表，请参阅 AWS Lake Formation 开发人员指南中的受管表的注释和限制。

> **Note**
>
> 对于写入 Apache Parquet，AWS Glue ETL 仅支持为针对动态帧进行优化的自定义 Parquet 编写器类型指定选项来写入受管表。使用 `parquet` 格式写入受管表时，应在表参数中添加值为 `true` 的键 `useGlueParquetWriter`。

主题

在 AWS Glue 中使用 CSV 格式

AWS Glue 从源中检索数据，并将数据写入以各种数据格式存储和传输的目标。如果您的数据以 CSV 数据格式存储或传输，本文档将向您介绍供您使用 AWS Glue 中的数据的可用功能。

AWS Glue 支持逗号分隔值（CSV）格式。此格式是一种基于行的最小数据格式。CSV 通常不严格遵守某一标准，但您可以参考 RFC 4180 和 RFC 7111 了解更多信息。

您可以使用 AWS Glue 从 Amazon S3 和流式传输源读取 CSV，以及将 CSV 写入 Amazon S3。您可以读取并写入包含 S3 中的 CSV 文件的 `bzip` 和 `gzip` 存档。请在 S3 连接参数 上而非本页中讨论的配置中配置压缩行为。

下表显示了哪些常用 AWS Glue 功能支持 CSV 格式选项。

读取	写入	流式处理读取	对小文件进行分组	作业书签
支持	支持	支持	支持	支持

示例：从 S3 读取 CSV 文件或文件夹

先决条件：您将需要至您想要读取的 CSV 文件或文件夹的 S3 路径（s3path）。

配置：在函数选项中，请指定 format="csv"。在您的 connection_options 中，请使用 paths 键指定 s3path。您可以在 connection_options 中配置读取器与 S3 的交互方式。有关详细信息，请参阅 AWS Glue 中 ETL 的连接类型和选项：S3 连接参数。您可以配置读取器如何解释 format_options 中的 CSV 文件。有关详细信息，请参阅 CSV 配置参考。

以下 AWS Glue ETL 脚本显示了从 S3 读取 CSV 文件或文件夹的过程。

我们提供自定义的 CSV 读取器，其中包含通过 optimizePerformance 配置键针对常见工作流进行的性能优化。要确定此读取器是否适合您的工作负载，请参阅 the section called “使用优化的 CSV 读取器”。

Python

在本示例中，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,
    },
)
```

您还可以使用脚本（`pyspark.sql.DataFrame`）中的 DataFrames。

```
dataFrame = spark.read\
    .format("csv")\
    .option("header", "true")\
    .load("s3://s3path")
```

Scala

在本示例中，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()
  }
}
```

您还可以使用脚本（`org.apache.spark.sql.DataFrame`）中的 DataFrames。

```
val dataFrame = spark.read
  .option("header","true")
  .format("csv")
```

```
  .load("s3://s3path“)
```

示例：将 CSV 文件和文件夹写入 S3

先决条件：您将需要一个初始化的 DataFrame（`dataFrame`）或 DynamicFrame（`dynamicFrame`）。您还需要预期 S3 输出路径 `s3path`。

配置：在函数选项中，请指定 `format="csv"`。在您的 `connection_options` 中，请使用 `paths` 键指定 `s3path`。您可以在 `connection_options` 中配置编写器与 S3 的交互方式。有关详细信息，请参阅 AWS Glue 中 ETL 的连接类型和选项：S3 连接参数。您可以配置自己的操作在 `format_options` 中写入文件的内容的方式。有关详细信息，请参阅 CSV 配置参考。以下 AWS Glue ETL 脚本显示了将 CSV 文件和文件夹写入 S3 的过程。

Python

在本示例中，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,
    },
)
```

您还可以使用脚本（`pyspark.sql.DataFrame`）中的 DataFrames。

```
dataFrame.write\
```

```
    .format("csv")\
    .option("quote", None)\
    .mode("append")\
    .save("s3://s3path")
```

Scala

在本示例中，请使用 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)
  }
}
```

您还可以使用脚本（`org.apache.spark.sql.DataFrame`）中的 DataFrames。

```
dataFrame.write
    .format("csv")
    .option("quote", null)
    .mode("Append")
    .save("s3://s3path")
```

CSV 配置参考

您可以在 AWS Glue 库指定 `format="csv"` 的任何位置使用以下 `format_options`：

- separator – 指定分隔符。默认值为逗号，但也可以指定任何其他字符。
  - 类型：文本，默认值："，"
- escaper – 指定要用于转义的字符。此选项仅在读取 CSV 文件而非写入时使用。如果启用，则按原样使用紧跟其后的字符，一小组已知的转义符（\n、\r、\t 和 \0）除外。
  - 类型：文本，默认值：无
- quoteChar – 指定要用于引用的字符。默认值为双引号。将这设置为 -1 可完全关闭引用。
  - 类型：文本，默认值：'"'
- multiLine – 指定单个记录能否跨越多行。当字段包含带引号的换行符时，会出现此选项。如果有记录跨越多个行，您必须将此选项设置为 True。启用 multiLine 可能会降低性能，因为它在解析时需要更加谨慎的文件拆分。
  - 类型：布尔值，默认值：false
- withHeader – 指定是否将第一行视为标头。可以在 DynamicFrameReader 类中使用此选项。
  - 类型：布尔值，默认值：false
- writeHeader – 指定是否将标头写入输出。可以在 DynamicFrameWriter 类中使用此选项。
  - 类型：布尔值，默认值：true
- skipFirst – 指定是否跳过第一个数据行。
  - 类型：布尔值，默认值：false
- optimizePerformance – 指定是否使用高级 SIMD CSV 读取器以及基于 Apache Arrow 的列式内存格式。仅适用于 AWS Glue 3.0+。
  - 类型：布尔值，默认值：false
- strictCheckForQuoting - 在编写 CSV 时，Glue 可能会在其解释为字符串的值中添加引号。这样做是为了防止写出的内容出现模棱两可之处。为了节省决定写入什么的时间，Glue 可能会在某些不需要引号的情况下进行引用。启用严格检查将执行更密集的计算，并且只有在绝对必要时才会引用。仅适用于 AWS Glue 3.0+。
  - 类型：布尔值，默认值：false

使用向量化 SIMD CSV 读取器优化读取性能

AWS Glue 3.0 版添加了经过优化的 CSV 读取器，与基于行的 CSV 读取器相比，它可以显著提高整体任务性能。

优化的读取器：

- 使用 CPU SIMD 指令从磁盘读取

- 立即以列格式（Apache Arrow）将记录写入内存
- 将记录分成几批

这样可以节省日后对记录进行批处理或转换为列格式时的处理时间。例如，更改架构或按列检索数据时。

要使用优化的读取器，请将在 `format_options` 或表属性中将 `"optimizePerformance"` 设置为 `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")
```

矢量化 CSV 读取器的限制

请注意向量化 CSV 读取器的以下限制：

- 它不支持 `multiLine` 和 `escaper` 格式选项。它使用默认双引号字符 `'"'` 的 `escaper`。设置这些选项后，AWS Glue 会自动回退使用基于行的 CSV 读取器。
- 它不支持创建具有 ChoiceType 的 DynamicFrame。
- 它不支持创建具有错误记录的 DynamicFrame。
- 它不支持读取带多字节字符（如日语或中文字符）的 CSV 文件。

在 AWS Glue 中使用 Parquet 格式

AWS Glue 从源中检索数据，并将数据写入以各种数据格式存储和传输的目标。如果您的数据以 Parquet 数据格式存储或传输，本文档将向您介绍供您使用 AWS Glue 中的数据的可用功能。

AWS Glue 支持使用 Parquet 格式。此格式是一种以性能为导向、基于列的数据格式。有关标准颁发机构对此格式的简介，请参阅 Apache Parquet Documentation Overview（Apache Parquet 文档概述）。

您可以使用 AWS Glue 从 Amazon S3 和流式处理媒体源读取 Parquet 文件，以及将 Parquet 文件写入 Amazon S3。您可以读取并写入包含 S3 中的 Parquet 文件的 `bzip` 和 `gzip` 存档。请在 S3 连接参数 上而非本页中讨论的配置中配置压缩行为。

下表显示了哪些常用 AWS Glue 功能支持 Parquet 格式选项。

读取	写入	流式处理读取	对小文件进行分组	作业书签
支持	支持	支持	不支持	支持*

* 在 AWS Glue 版本 1.0+ 中受支持

示例：从 S3 读取 Parquet 文件或文件夹

先决条件：您将需要至您想要读取的 Parquet 文件或文件夹的 S3 路径（`s3path`）。

配置：在函数选项中，请指定 `format="parquet"`。在您的 `connection_options` 中，请使用 `paths` 键指定 `s3path`。

您可以在 `connection_options` 中配置读取器与 S3 的交互方式。有关详细信息，请参阅 AWS Glue 中 ETL 的连接类型和选项：S3 连接参数。

您可以配置读取器如何解释 `format_options` 中的 Parquet 文件。有关详细信息，请参阅 Parquet 配置参考。

以下 AWS Glue ETL 脚本显示了从 S3 读取 Parquet 文件或文件夹的过程：

Python

在本示例中，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"
)
```

您还可以使用脚本（`pyspark.sql.DataFrame`）中的 DataFrames。

```
dataFrame = spark.read.parquet("s3://s3path/")
```

Scala

在本示例中，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()
  }
}
```

您还可以使用脚本（`org.apache.spark.sql.DataFrame`）中的 DataFrames。

```
spark.read.parquet("s3://s3path/")
```

示例：将 Parquet 文件和文件夹写入 S3

先决条件：您将需要一个初始化的 DataFrame（`dataFrame`）或 DynamicFrame（`dynamicFrame`）。您还需要预期 S3 输出路径 `s3path`。

配置：在函数选项中，请指定 `format="parquet"`。在您的 `connection_options` 中，请使用 `paths` 键指定 `s3path`。

您可以在 `connection_options` 中进一步修改编写器与 S3 的交互方式。有关详细信息，请参阅 AWS Glue 中 ETL 的连接类型和选项：S3 连接参数。您可以配置自己的操作在 `format_options` 中写入文件的内容的方式。有关详细信息，请参阅 Parquet 配置参考。

以下 AWS Glue ETL 脚本显示了将 Parquet 文件和文件夹写入 S3 的过程。

我们通过 `useGlueParquetWriter` 配置键为自定义 Parquet 编写器提供 DynamicFrames 的性能优化。要确定此编写器是否适合您的工作负载，请参阅 Glue Parquet 编写器。

Python

在本示例中，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,
    },
)
```

您还可以使用脚本（`pyspark.sql.DataFrame`）中的 DataFrames。

```
df.write.parquet("s3://s3path/")
```

Scala

在本示例中，请使用 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)
  }
}
```

您还可以使用脚本（`org.apache.spark.sql.DataFrame`）中的 DataFrames。

```
df.write.parquet("s3://s3path/")
```

Parquet 配置参考

您可以在 AWS Glue 库指定 `format="parquet"` 的任何位置使用以下 `format_options`：

- `useGlueParquetWriter` – 指定使用具有 DynamicFrame 工作流性能优化的自定义 Parquet 编写器。有关使用情况的详细信息，请参阅 Glue Parquet 编写器。
  - 类型：布尔值，默认值：`false`
- `compression` – 指定使用的压缩编解码器。值与 `org.apache.parquet.hadoop.metadata.CompressionCodecName` 完全兼容。
  - 类型：枚举文本，默认值：`"snappy"`
  - 值：`"uncompressed"`、`"snappy"`、`"gzip"` 和 `"lzo"`

- `blockSize` – 指定内存中缓冲的行组的字节大小。您可以用它来调整性能。大小应精确地划分为若干兆字节。
  - 类型：数值，默认值：`134217728`
  - 默认值等于 128MB。
- `pageSize` – 指定页面的大小（以字节为单位）。您可以用它来调整性能。页面是必须完全读取以访问单个记录的最小单位。
  - 类型：数值，默认值：`1048576`
  - 默认值等于 1MB。

> **Note**
>
> 此外，基础 SparkSQL 代码所接受的任何选项均可通过 `connection_options` 映射参数传递给此格式。例如，您可以为 AWS Glue Spark 读取器设置 Spark 配置（如 mergeSchema），以合并所有文件的架构。

使用 AWS Glue Parquet 编写器优化写入性能

> **Note**
>
> AWS Glue Parquet 编写器以前一直通过 `glueparquet` 格式类型访问。这种访问模式已不再提倡。请改用启用了 `useGlueParquetWriter` 的 `parquet` 类型。

AWS Glue Parquet 编写器具有允许更快地写入 Parquet 文件的性能增强功能。传统编写器在写入之前计算架构。Parquet 格式不会以可快速检索的方式存储架构，因此可能需要一些时间。使用 AWS Glue Parquet 编写器时，不需要预计算的架构。在数据传入时，编写器会动态计算和修改架构。

指定 `useGlueParquetWriter` 时请注意以下限制：

- 编写器仅支持架构发展（例如添加或删除列）但不支持更改列类型，例如使用 `ResolveChoice`。
- 写入器不支持写入空 DataFrame，例如，写入纯架构文件。通过设置 `enableUpdateCatalog=True` 实现与 AWS Glue Data Catalog 的集成时，尝试写入空 DataFrame 不会更新数据目录。这将导致在数据目录中创建一个没有架构的表。

如果您的转换不需要这些限制，则开启 AWS Glue Parquet 编写器应该能提高性能。

在 AWS Glue 中使用 XML 格式

AWS Glue 从源中检索数据，并将数据写入以各种数据格式存储和传输的目标。如果您的数据以 XML 数据格式存储或传输，本文档将向您介绍供您使用 AWS Glue 中的数据的可用功能。

AWS Glue 支持使用 XML 格式。此格式表示高度可配置、严格定义的数据结构，这些数据结构不是基于行或列的。XML 是高度标准化格式。有关标准颁发机构对该格式的简介，请参阅 XML Essentials（XML 基础知识）。

您可以使用 AWS Glue 从 Amazon S3 以及含有 XML 文件的 `bzip` 和 `gzip` 存档中读取 XML 文件。请在 S3 连接参数 上而非本页中讨论的配置中配置压缩行为。

下表显示了哪些常用 AWS Glue 功能支持 XML 格式选项。

读取	写入	流式处理读取	对小文件进行分组	作业书签
支持	不支持	不支持	支持	支持

示例：从 S3 读取 XML

XML 读取器采用 XML 标签名称。它检查输入中带有该标签的元素以推断架构，并使用相应的值填充 DynamicFrame。AWS Glue XML 功能的行为类似于 XML Data Source for Apache Spark（Apache Spark 的 XML 数据来源）。通过将此阅读器与该项目的文档进行比较，您也许可以深入了解基本行为。

先决条件：您将需要至您想要读取的 XML 文件或文件夹以及有关您的 XML 文件的一些信息的 S3 路径（`s3path`）。您还需要您想要读取的 XML 元素的标签 `xmlTag`。

配置：在函数选项中，请指定 `format="xml"`。在您的 `connection_options` 中，请使用 `paths` 键指定 `s3path`。您可以在 `connection_options` 中进一步配置读取器与 S3 的交互方式。有关详细信息，请参阅 AWS Glue 中 ETL 的连接类型和选项：S3 连接参数。在您的 `format_options` 中，请使用 `rowTag` 键指定 `xmlTag`。您可以进一步配置读取器如何解释 `format_options` 中的 XML 文件。有关详细信息，请参阅 XML 配置参考。

以下 AWS Glue ETL 脚本显示了从 S3 读取 XML 文件或文件夹的过程。

Python

在本示例中，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"},
)
```

您还可以使用脚本（`pyspark.sql.DataFrame`）中的 DataFrames。

```
dataFrame = spark.read\
    .format("xml")\
    .option("rowTag", "xmlTag")\
    .load("s3://s3path")
```

Scala

在本示例中，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()
}
```

您还可以使用脚本（`org.apache.spark.sql.DataFrame`）中的 DataFrames。

```
val dataFrame = spark.read
  .option("rowTag", "xmlTag")
  .format("xml")
  .load("s3://s3path“)
```

XML 配置参考

您可以在 AWS Glue 库指定 `format="xml"` 的任何位置使用以下 `format_options`：

- `rowTag` – 指定文件中要视为行的 XML 标签。行标签不能自结束。
  - 类型：文本，必填项
- `encoding` – 指定字符编码。它可以是由我们的运行时环境支持的字符集的名称或别名。我们没有对编码支持做出具体的保证，但主编码应该起作用。
  - 类型：文本，默认值："UTF-8"
- `excludeAttribute` – 指定是否要排除元素中的属性。
  - 类型：布尔值，默认值：`false`
- `treatEmptyValuesAsNulls` – 指定是否将空格视为空值。
  - 类型：布尔值，默认值：`false`
- `attributePrefix` – 用于将属性与子元素文本区分开来的属性的前缀。此前缀用于字段名称。
  - 类型：文本，默认值："_"
- `valueTag` – 在元素中具有没有子项的属性时用于值的标签。
  - 类型：文本，默认值："_VALUE"
- `ignoreSurroundingSpaces` – 指定是否应忽略值周围的空格。
  - 类型：布尔值，默认值：`false`
- `withSchema` – 在您想要覆盖推断的架构的情况下，包含预期的架构。如果您不使用此选项，AWS Glue 会推断 XML 数据中的架构。

- 类型：文本，默认值：不适用
- 该值应该是代表 `StructType` 的一个 JSON 对象。

手动指定 XML 架构

手动 XML 架构示例

此示例使用 `withSchema` 格式选项来指定 XML 数据的架构。

```
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 从源中检索数据，并将数据写入以各种数据格式存储和传输的目标。如果您的数据以 Avro 数据格式存储或传输，本文档将向您介绍供您使用 AWS Glue 中的数据的可用功能。

AWS Glue 支持使用 Avro 格式。此格式是一种以性能为导向、基于行的数据格式。有关标准颁发机构对此格式的简介，请参阅 Apache Avro 1.8.2 Documentation（Apache Avro 1.8.2 文档）。

您可以使用 AWS Glue 从 Amazon S3 和流式传输源读取 Avro 文件，以及将 Avro 文件写入 Amazon S3。您可以读取并写入包含 S3 中的 Avro 文件的 `bzip2` 和 `gzip` 存档。此外，您还可以编写包含 Avro 文件的 `deflate`、`snappy` 和 `xz` 存档。请在 S3 连接参数 上而非本页中讨论的配置中配置压缩行为。

下表显示了支持 Avro 格式选项的常用 AWS Glue 功能。

读取	写入	流式处理读取	对小文件进行分组	作业书签
支持	支持	支持*	不支持	支持

*受支持，但有限制。有关更多信息，请参阅 the section called “Avro 串流源的注释和限制”。

示例：从 S3 读取 Avro 文件或文件夹

先决条件：需要待读取的 Avro 文件或文件夹的 S3 路径 (s3path)。

配置：在函数选项中，请指定 format="avro"。在您的 connection_options 中，请使用 paths 键指定 s3path。您可以在 connection_options 中配置读取器与 S3 的交互方式。有关详细信息，请参阅 AWS Glue：the section called “S3 连接参数” 中的“Data format options for ETL inputs and outputs”（ETL 输入和输出的数据格式选项）。您可以配置读取器解释 format_options 中的 Avro 文件的方式。有关详细信息，请参阅 Avro Configuration Reference（Avro 配置参考）。

以下 AWS Glue ETL 脚本显示了从 S3 读取 Avro 文件或文件夹的过程：

Python

在本示例中，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

在本示例中，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()
  }
```

示例：将 Avro 文件和文件夹写入 Amazon S3

先决条件：您将需要一个初始化的 DataFrame（`dataFrame`）或 DynamicFrame（`dynamicFrame`）。您还需要预期 S3 输出路径 `s3path`。

配置：在函数选项中，请指定 `format="avro"`。在您的 `connection_options` 中，请使用 `paths` 键指定 `s3path`。您可以在 `connection_options` 中进一步修改编写器与 S3 的交互方式。有关详细信息，请参阅 AWS Glue：the section called “S3 连接参数” 中的“Data format options for ETL inputs and outputs”（ETL 输入和输出的数据格式选项）。您可以改变写入器在 `format_options` 中解释 Avro 文件的方式。有关详细信息，请参阅 Avro Configuration Reference（Avro 配置参考）。

以下 AWS Glue ETL 脚本显示了将 Avro 文件或文件夹写入 Amazon S3 的过程。

Python

在本示例中，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

在本示例中，请使用 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 配置参考

您可以在 AWS Glue 库指定 `format="avro"` 的任何位置使用以下 `format_options` 值：

- `version` – 指定要支持的 Apache Avro 读取器/写入器格式的版本。默认值为“1.7”。您可以指定 `format_options={"version": “1.8”}` 以启用 Avro 逻辑类型读取和写入。有关更多信息，请参阅 Apache Avro 1.7.7 规范和 Apache Avro 1.8.2 规范。

  Apache Avro 1.8 连接器支持以下逻辑类型转换：

对于读取器：此表显示 Avro 数据类型（逻辑类型和 Avro 基元类型）与 Avro 阅读器 1.7 和 1.8 的 AWS Glue `DynamicFrame` 数据类型之间的转换。

Avro 数据类型： 逻辑类型	Avro 数据类型： Avro 基元类型	GlueDynamicFrame 数据类型： Avro 读取器 1.7	GlueDynamicFrame 数据类型： Avro 读取器 1.8
十进制	bytes	BINARY	十进制
十进制	固定	BINARY	十进制
Date	int	INT	Date
时间（毫秒）	int	INT	INT
时间（微秒）	long	LONG	LONG
时间戳（毫秒）	long	LONG	Timestamp
时间戳（微秒）	long	LONG	LONG
持续时间（不是逻辑类型）	固定为 12	BINARY	BINARY

对于写入器：此表显示 Avro 写入器 1.7 和 1.8 在 AWS Glue `DynamicFrame` 数据类型与 Avro 数据类型之间的转换。

AWS Glue **`DynamicFrame`** 数据类型	Avro 数据类型： Avro 写入器 1.7	Avro 数据类型： Avro 写入器 1.8
十进制	字符串	decimal
Date	字符串	date
Timestamp	字符串	timestamp-micros

Avro Spark DataFrame 支持

要使用 Spark DataFrame API 中的 Avro，您需要为相应的 Spark 版本安装 Spark Avro 插件。任务中可用的 Spark 版本取决于您的 AWS Glue 版本。有关 Spark 版本的更多信息，请参阅 the section called “AWS Glue 版本”。该插件由 Apache 维护，我们不提供具体的支持保证。

在 AWS Glue 2.0 中 – 使用 2.4.3 版本的 Spark Avro 插件。您可以在 Maven Central 上找到该 JAR，请参阅 org.apache.spark:spark-avro_2.12:2.4.3。

在 AWS Glue 3.0 中 – 使用3.1.1 版本的 Spark Avro 插件。您可以在 Maven Central 上找到该 JAR，请参阅 org.apache.spark:spark-avro_2.12:3.1.1。

要在 AWS Glue ETL 任务中加入额外的 JAR，请使用 `--extra-jars` 任务参数。有关任务参数的更多信息，请参阅 the section called “任务参数”。您也可以在 AWS Management Console 中配置此参数。

在 AWS Glue 中使用 grokLog 格式

AWS Glue 从源中检索数据，并将数据写入以各种数据格式存储和传输的目标。如果您的数据以结构松散的纯文本格式存储或传输，本文档将向您介绍供通过 Grok 模式使用 AWS Glue 中的数据时的可用功能。

AWS Glue 支持使用 Grok 模式。Grok 模式类似于正则表达式捕获组。这些组能识别纯文本文件中的字符序列模式，并为其指定类型和用途。在 AWS Glue 中，其主要用途是读取日志。有关作者对 Grok 的说明，请参阅 Logstash Reference: Grok filter plugin（Logstash 参考：Grok 筛选器插件）。

读取	写入	流式处理读取	对小文件进行分组	作业书签
支持	不适用	支持	支持	不支持

grokLog 配置参考

您可以将以下 `format_options` 值与 `format="grokLog"` 结合使用：

- `logFormat` – 指定与日志的格式匹配的 Grok 模式。
- `customPatterns` – 指定在此处使用的其他 Grok 模式。
- `MISSING` – 指定用于标识缺失值的信号。默认为 `'-'`。
- `LineCount` – 指定每个日志记录中的行数。默认值为 `'1'`，并且目前仅支持单行记录。

- `StrictMode` – 指定是否启用严格模式的布尔值。在严格模式下，读取器不会执行自动类型转换或恢复。默认值为 `"false"`。

在 AWS Glue 中使用 Ion 格式

AWS Glue 从源中检索数据，并将数据写入以各种数据格式存储和传输的目标。如果您的数据以 Ion 数据格式存储或传输，本文档将向您介绍供您使用 AWS Glue 中的数据的可用功能。

AWS Glue 支持使用 Ion 格式。此格式以可互换的二进制和纯文本表示形式表示（并非基于行或列的）数据结构。有关作者对此格式的说明，请参阅 Amazon Ion。(有关更多信息，请参阅 Amazon Ion 规范。)

您可以使用 AWS Glue 从 Amazon S3 中读取 Ion 文件。您可以从 S3 中读取包含 Ion 文件的 `bzip` 和 `gzip` 存档。请在 S3 连接参数 上而非本页中讨论的配置中配置压缩行为。

下表显示了支持 Ion 格式选项的常见 AWS Glue 功能。

读取	写入	流式处理读取	对小文件进行分组	作业书签
支持	不支持	不支持	支持	不支持

示例：从 S3 读取 Ion 文件或文件夹

先决条件：需要待读取的 Ion 文件或文件夹的 S3 路径 (`s3path`)。

配置：在函数选项中，请指定 `format="json"`。在您的 `connection_options` 中，请使用 `paths` 键指定 `s3path`。您可以在 `connection_options` 中配置读取器与 S3 的交互方式。有关详细信息，请参阅 AWS Glue 中 ETL 的连接类型和选项：the section called “S3 连接参数”。

以下 AWS Glue ETL 脚本显示了从 S3 读取 Ion 文件或文件夹的过程：

Python

在本示例中，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

在本示例中，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 配置参考

没有适用于 `format="ion"` 的 `format_options` 值。

在 AWS Glue 中使用 JSON 格式

AWS Glue 从源中检索数据，并将数据写入以各种数据格式存储和传输的目标。如果您的数据以 JSON 数据格式存储或传输，则本文档将向您介绍在 Glue 中使用数据的可用 AWS 功能。

AWS Glue 支持使用 JSON 格式。此格式表示形状一致但内容灵活且并非基于行或列的数据结构。JSON 由多个权威机构发布的平行标准定义，其中一项标准便是 ECMA-404。有关常引用的源对该格式的说明，请参阅 Introducing JSON（JSON 简介）。

你可以使用 AWS Glue 读取 Amazon S3 中的 JSON 文件bzip以及gzip压缩的 JSON 文件。请在 S3 连接参数 上而非本页中讨论的配置中配置压缩行为。

读取	写入	流式处理读取	对小文件进行分组	作业书签	
支持	支持	支持	支持	支持	

示例：从 S3 读取 JSON 文件或文件夹

先决条件：需要待读取的 JSON 文件或文件夹的 S3 路径 (s3path)。

配置：在函数选项中，请指定 format="json"。在您的 connection_options 中，请使用 paths 键指定 s3path。您可以在连接选项中进一步更改读取操作遍历 S3 的方式，请参阅 the section called "S3 连接参数"，了解详细信息。您可以配置读取器解释 format_options 中 JSON 文件的方式。有关详细信息，请参阅 JSON Configuration Reference（JSON 配置参考）。

以下 AWS Glue ETL 脚本显示了从 S3 读取 JSON 文件或文件夹的过程：

Python

在本示例中，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
    }
)
```

你也可以在脚本 DataFrames 中使用 (pyspark.sql.DataFrame)。

```
dataFrame = spark.read\
    .option("multiLine", "true")\
    .json("s3://s3path")
```

Scala

在本示例中，使用 “getSourceWith格式化” 操作。

```
// 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()
  }
```

```
}
```

你也可以在脚本 DataFrames 中使用 (`pyspark.sql.DataFrame`)。

```
val dataFrame = spark.read
    .option("multiLine", "true")
    .json("s3://s3path")
```

示例：将 JSON 文件和文件夹写入 Amazon S3

先决条件：你需要一个初始化的 DataFrame (`dataFrame`) 或 DynamicFrame (`dynamicFrame`)。您还需要预期 S3 输出路径 `s3path`。

配置：在函数选项中，请指定 `format="json"`。在您的 `connection_options` 中，请使用 `paths` 键指定 `s3path`。您可以在 `connection_options` 中进一步修改编写器与 S3 的交互方式。有关详细信息，请参阅 GI AWS ue: the section called “S3 连接参数” 中 ETL 输入和输出的数据格式选项。您可以配置读取器解释 `format_options` 中的 JSON 文件的方式。有关详细信息，请参阅 JSON Configuration Reference（JSON 配置参考）。

以下 AWS Glue ETL 脚本显示了从 S3 中编写 JSON 文件或文件夹的过程：

Python

在本示例中，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"
)
```

你也可以在脚本 DataFrames 中使用 (`pyspark.sql.DataFrame`)。

```
df.write.json("s3://s3path/")
```

Scala

对于此示例，请使用 getSinkWithForma t 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)
  }
}
```

你也可以在脚本 DataFrames 中使用 (pyspark.sql.DataFrame)。

```
df.write.json("s3://s3path")
```

JSON 配置参考

您可以将以下 format_options 值与 format="json" 结合使用：

- jsonPath— 标识要读入记录的对象的JsonPath表达式。当文件包含嵌套在外部数组内的记录时，此表达式尤其有用。例如，以下 JsonPath 表达式以 JSON 对象的id字段为目标。

```
format="json", format_options={"jsonPath": "$.id"}
```

- multiLine – 指定单个记录能否跨越多行的布尔值。当字段包含带引号的换行符时，会出现此选项。如果有记录跨越多个行，您必须将此选项设置为 "true"。默认值为 "false"，它允许在分析过程中更积极地拆分文件。

- `optimizePerformance` – 一个布尔值，用于指定是否将高级 SIMD JSON 读取器与基于 Apache Arrow 的列式内存格式结合使用。仅适用于 AWS Glue 3.0。不兼容 `multiLine` 或 `jsonPath`。提供这两个选项中的任何一个都将指示 AWS Glue 回退到标准阅读器。
- `withSchema` – 一个字符串值，以 the section called “指定 XML 架构” 中描述的格式指定表 Schema。仅从非目录连接读取时与 `optimizePerformance` 结合使用。

将矢量化 SIMD JSON 读取器与 Apache Arrow 列式格式结合使用

AWS Glue 版本 3.0 增加了适用于 JSON 数据的矢量化读取器。与标准读取器相比，它在某些条件下的执行速度可提高 2 倍。此读取器存在一些需要用户在使用前注意的限制，详见本节的说明。

要使用优化的读取器，请将在 `format_options` 或表属性中将 `"optimizePerformance"` 设置为 True。除非从目录中读取，否则您还需要提供 `withSchema`。`withSchema` 需要有一个 the section called “指定 XML 架构” 中描述的输入

```
// 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,
        })
```

有关 Glue 库*SchemaString*中建筑物 a AWS 的更多信息，请参阅the section called “类型”。

矢量化 CSV 读取器的限制

请注意以下限制：

- 不支持具有嵌套对象或数组值的 JSON 元素。如果提供，AWS Glue 将回退到标准阅读器。
- 必须从目录或使用 `withSchema` 参数提供一个 Schema。
- 不兼容 `multiLine` 或 `jsonPath`。提供这两个选项中的任何一个都将指示 AWS Glue 回退到标准阅读器。
- 如果提供的输入记录与输入 Schema 不一致，将会导致读取器失败。
- 将不会创建错误记录。
- 不支持具有多字节字符（如日语或中文字符）的 JSON 文件。

在 AWS Glue 中使用 ORC 格式

AWS Glue 从源中检索数据，并将数据写入以各种数据格式存储和传输的目标。如果数据以 ORC 数据格式存储或传输，本文档将向您介绍在 AWS Glue 中使用数据时可用的功能。

AWS Glue 支持使用 ORC 格式。此格式是一种以性能为导向、基于列的数据格式。有关标准颁发机构对该格式的简介，请参阅 Apache Orc。

您可以使用 AWS Glue 从 Amazon S3 和流式传输源读取 ORC 文件，以及将 ORC 文件写入 Amazon S3。您可以读取并写入包含 S3 中的 ORC 文件的 `bzip` 和 `gzip` 存档。请在 S3 连接参数 上而非本页中讨论的配置中配置压缩行为。

下表展示了支持 ORC 格式选项的常见 AWS Glue 操作。

读取	写入	流式处理读取	对小文件进行分组	作业书签
支持	支持	支持	不支持	支持*

* 在 AWS Glue 1.0 以上版本中受支持

示例：从 S3 读取 ORC 文件或文件夹

先决条件：需要待读取的 ORC 文件或文件夹的 S3 路径 (`s3path`)。

配置：在函数选项中，请指定 `format="orc"`。在您的 `connection_options` 中，请使用 `paths` 键指定 `s3path`。您可以在 `connection_options` 中配置读取器与 S3 的交互方式。有关详细信息，请参阅 AWS Glue 中 ETL 的连接类型和选项：the section called “S3 连接参数”。

以下 AWS Glue ETL 脚本展示了从 S3 读取 ORC 文件或文件夹的过程：

Python

在本示例中，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"
)
```

您还可以使用脚本（pyspark.sql.DataFrame）中的 DataFrames。

```
dataFrame = spark.read\
    .orc("s3://s3path")
```

Scala

在本示例中，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()
  }
}
```

您还可以使用脚本（pyspark.sql.DataFrame）中的 DataFrames。

```
val dataFrame = spark.read
    .orc("s3://s3path")
```

示例：将 ORC 文件和文件夹写入 S3

先决条件：您将需要一个初始化的 DataFrame（`dataFrame`）或 DynamicFrame（`dynamicFrame`）。您还需要预期 S3 输出路径 `s3path`。

配置：在函数选项中，请指定 `format="orc"`。在连接选项中，使用 `paths` 密钥指定 `s3path`。您可以在 `connection_options` 中进一步修改编写器与 S3 的交互方式。有关详细信息，请参阅 AWS Glue：the section called “S3 连接参数” 中的“Data format options for ETL inputs and outputs”（ETL 输入和输出的数据格式选项）。以下代码示例展示了这个过程：

Python

在本示例中，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"
    }
)
```

您还可以使用脚本（`pyspark.sql.DataFrame`）中的 DataFrames。

```
df.write.orc("s3://s3path/")
```

Scala

在本示例中，请使用 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)
  }
}
```

您还可以使用脚本（`pyspark.sql.DataFrame`）中的 DataFrames。

```
df.write.orc("s3://s3path/")
```

ORC 配置参考

没有适用于 `format="orc"` 的 `format_options` 值。不过，基础 SparkSQL 代码所接受的任何选项均可通过 `connection_options` 映射参数传递给它。

在 AWS Glue ETL 任务中使用数据湖框架

开源数据湖框架简化了对存储在 Amazon S3 上的数据湖中的文件的增量数据处理。AWS Glue 3.0 及更高版本支持以下开源数据湖框架：

- Apache Hudi
- Linux Foundation Delta Lake
- Apache Iceberg

我们为这些框架提供原生支持，以便您可以以交易一致的方式读取和写入存储在 Amazon S3 中的数据。无需安装单独的连接器或完成额外的配置步骤即可在 AWS Glue ETL 任务中使用这些框架。

通过 AWS Glue Data Catalog 管理数据集时，您可以使用 AWS Glue 方法读取和写入 Spark DataFrames 数据湖表。也可以使用 Spark DataFrame API 读取和写入 Amazon S3 数据。

在这段视频中，您可以了解 Apache Hudi、Apache Iceberg 和 Delta Lake 工作原理的基础知识。您将看到如何在数据湖中插入、更新和删除数据，以及每个框架的工作原理。

主题

限制

在将数据湖框架与一起使用之前，请考虑以下限制 AWS Glue。

- 以下 AWS Glue `GlueContext`方法 DynamicFrame 不支持读取和写入数据湖框架表。改用 DataFrame 或 Spark DataFrame API `GlueContext` 的方法。
  - Lake F DynamicFrame ormation 权限控制不支持以下`GlueContext`方法：
    - `create_dynamic_frame.from_catalog`
    - `write_dynamic_frame.from_catalog`
    - `getDynamicFrame`
    - `writeDynamicFrame`
  - Lake For DataFrame mation 权限控制支持以下`GlueContext`方法：
    - `create_data_frame.from_catalog`
    - `write_data_frame.from_catalog`
    - `getDataFrame`
    - `writeDataFrame`
- 不支持对小文件进行分组。
- 不支持作业书签。
- AWS Glue 3.0 版 Apache Hudi 0.10.1 不支持 Hudi 读时合并 (MoR) 表。
- `ALTER TABLE … RENAME TO`不适用于版本 3.0 的 Apache Iceberg 0.13.1。 AWS Glue

有关由 Lake Formation 权限管理的数据湖格式表的限制

数据湖格式通过 AWS Glue Lake Formation 权限与 ETL 集成。`create_dynamic_frame`不支持创建 DynamicFrame 使用。有关更多信息，请参阅以下示例：

- 示例：读取和写入具有 Lake Formation 权限控制的 Iceberg 表
- 示例：读取和写入具有 Lake Formation 权限控制的 Hudi 表
- 示例：读取和写入具有 Lake Formation 权限控制的 Delta Lake 表

> **Note**
>
> 只有版本 4.0 支持通过 Apache Hudi、Apache Iceberg 和 Delta AWS Glue Lake 的 Lake Formation 权限与 ETL 集成。 AWS Glue

Apache Iceberg 通过 Lake Format AWS Glue ion 权限与 ETL 的集成效果最好。它支持几乎所有操作，包括支持 SQL。

Hudi 支持除管理操作之外的大多数基本操作。这是因为这些选项通常通过写入 DataFrame 来完成，并通过 `additional_options` 指定。由于不支持 sparkSQL，因此您需要使用 AWS Glue API 来 DataFrames 为自己的操作进行创建。

Delta Lake 仅支持读取、附加和覆盖表数据。Delta Lake 需要使用自己的库才能执行更新等各种任务。

由 Lake Formation 权限管理的 Iceberg 表不支持以下功能。

- 使用 ETL 进行 AWS Glue 压实
- 通过 AWS Glue ETL 激发 SQL 支持

由 Lake Formation 权限管理的 Hudi 表存在以下限制：

- 移除孤立文件

由 Lake Formation 权限管理的 Delta Lake 表存在以下限制：

- 除在 Delta Lake 表中插入和读取数据的所有其他功能。

在 AWS Glue 中使用 Hudi 框架

AWS Glue 3.0 及更高版本支持数据湖的 Apache Hudi 框架。Hudi 是一个开源数据湖存储框架，简化增量数据处理和数据管道开发。本主题涵盖了在 Hudi 表中传输或存储数据时，在 AWS Glue 中使用数据的可用功能。要了解有关 Hudi 的更多信息，请参阅 Apache Hudi 官方文档。

您可以使用 AWS Glue 对 Amazon S3 中的 Hudi 表执行读写操作，也可以使用 AWS Glue 数据目录处理 Hudi 表。还支持其他操作，包括插入、更新和所有 Apache Spark 操作。

> **Note**
>
> Apache Hudi 0.10.1 for AWS Glue 3.0 不支持 Read（MoR）表上的 Hudi Merge。

下表列出了 AWS Glue 每个版本中包含的 Hudi 版本。

AWS Glue 版本	支持的 Hudi 版本
4.0	0.12.1
3.0	0.10.1

要了解有关 AWS Glue 支持的数据湖框架的更多信息，请参阅在 AWS Glue ETL 任务中使用数据湖框架。

启用 Hudi

要为 AWS Glue 启用 Hudi，请完成以下任务：

- 指定 `hudi` 作为 `--datalake-formats` 作业参数的值。有关更多信息，请参阅 AWS Glue 作业参数。
- `--conf` 为 Glue 作业创建一个名为 AWS 的密钥，并将其设置为以下值。或者，您可以在脚本中使用 `SparkConf` 设置以下配置。这些设置有助于 Apache Spark 正确处理 Hudi 表。

```
spark.serializer=org.apache.spark.serializer.KryoSerializer --conf
 spark.sql.hive.convertMetastoreParquet=false
```

- AWS Glue 4.0 默认为 Hudi 表启用了 Lake Formation 权限支持。无需额外配置即可读取/写入注册到 Lake Formation 的 Hudi 表。AWS Glue 作业 IAM 角色必须具有 SELECT 权限才能读取已注册的 Hudi 表。AWS Glue 作业 IAM 角色必须具有 SUPER 权限才能写入已注册的 Hudi 表。要了解有关管理 Lake Formation 权限的更多信息，请参阅 Granting and revoking permissions on Data Catalog resources。

使用不同的 Hudi 版本

要使用 AWS Glue 不支持的 Hudi 版本，请使用 `--extra-jars` 作业参数指定您自己的 Hudi JAR 文件。请勿使用 `hudi` 作为 `--datalake-formats` 作业参数的值。

示例：将 Hudi 表写入 Amazon S3 并将其注册到 AWS Glue 数据目录中

以下示例脚本脚本演示了如何将 Hudi 表写入 Amazon S3，并将该表注册到 AWS Glue 数据目录。该示例使用 Hudi Hive 同步工具来注册该表。

> **Note**
>
> 此示例要求您设置 `--enable-glue-datacatalog` 作业参数，才能将 AWS Glue Data Catalog 用作 Apache Spark Hive 元存储。要了解更多信息，请参阅 AWS Glue 作业参数。

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()
```

示例：使用 AWS Glue Data Catalog 从 Amazon S3 读取 Hudi 表

此示例从 Amazon S3 读取您在 示例：将 Hudi 表写入 Amazon S3 并将其注册到 AWS Glue 数据目录中 中创建的 Hudi 表。

> **Note**
>
> 此示例要求您设置 `--enable-glue-datacatalog` 任务参数，才能将 AWS Glue Data Catalog 用作 Apache Spark Hive 元存储。要了解更多信息，请参阅 AWS Glue 作业参数。

Python

在本示例中，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

在本示例中，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()
  }
}
```

示例：更新 **DataFrame** 并将其插入到 Amazon S3 的 Hudi 表中

此示例使用 AWS Glue Data Catalog 将 DataFrame 插入到您在 示例：将 Hudi 表写入 Amazon S3 并将其注册到 AWS Glue 数据目录中 中创建的 Hudi 表中。

> **Note**
>
> 此示例要求您设置 `--enable-glue-datacatalog` 任务参数，才能将 AWS Glue Data Catalog 用作 Apache Spark Hive 元存储。要了解更多信息，请参阅 AWS Glue 作业参数。

Python

在本示例中，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

在本示例中，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)
  }
}
```

示例：使用 Spark 从 Amazon S3 读取 Hudi 表

此示例使用 Spark DataFrame API 从 Amazon S3 读取 Hudi 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/>")
```

示例：使用 Spark 向 Amazon S3 写入 Hudi 表

示例：使用 Spark 向 Amazon S3 写入 Hudi 表

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/>")
```

示例：读取和写入具有 Lake Formation 权限控制的 Hudi 表

此示例将读取和写入一个具有 Lake Formation 权限控制的 Hudi 表。

1. 创建一个 Hudi 表并将其注册到 Lake Formation。

a. 要启用 Lake Formation 权限控制，您首先需要将表的 Amazon S3 路径注册到 Lake Formation。有关更多信息，请参阅 Registering an Amazon S3 location（注册 Amazon S3 位置）。您可以通过 Lake Formation 控制台或使用 AWS CLI 进行注册：

```
aws lakeformation register-resource --resource-arn arn:aws:s3:::<s3-bucket>/<s3-folder> --use-service-linked-role --region <REGION>
```

注册了 Amazon S3 位置后，对于任何指向该位置（或其任何子位置）的 AWS Glue 表，`GetTable` 调用中的 `IsRegisteredWithLakeFormation` 参数都将返回值 true。

b. 创建一个指向通过 Spark dataframe API 注册的 Amazon S3 路径的 Hudi 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. 向 AWS Glue 作业 IAM 角色授予 Lake Formation 权限。您可以通过 Lake Formation 控制台授予权限，也可以使用 AWS CLI 授予权限。有关更多信息，请参阅 Granting table permissions using the Lake Formation console and the named resource method。
3. 读取注册到 Lake Formation 的 Hudi 表。代码与读取未注册的 Hudi 表相同。请注意，AWS Glue 作业 IAM 角色需要具有 SELECT 权限才能成功读取。

```
 val dataFrame = glueContext.getCatalogSource(
      database = "<your_database_name>",
      tableName = "<your_table_name>"
    ).getDataFrame()
```

4. 写入注册到 Lake Formation 的 Hudi 表。代码与写入未注册的 Hudi 表相同。请注意，AWS Glue 作业 IAM 角色需要具有 SUPER 权限才能成功写入。

```
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 框架

AWS Glue 3.0 及更高版本支持 Linux Foundation Delta Lake 框架。Delta Lake 是一个开源数据湖存储框架，可帮助您执行 ACID 交易、扩展元数据处理以及统一流式和批处理数据处理。本主题涵盖了在 Delta Lake 表中传输或存储数据时，在 AWS Glue 中使用数据的可用功能。要了解有关 Delta Lake 的更多信息，请参阅 Delta Lake 官方文档。

您可以使用 AWS Glue 对 Amazon S3 中的 Delta Lake 表执行读写操作，也可以使用 AWS Glue 数据目录处理 Delta Lake 表。还支持插入、更新和表批量读取和写入等其他操作。使用 Delta Lake 表时，也可以选择使用 Delta Lake Python 库中的方法，例如 `DeltaTable.forPath`。有关 Delta Lake Python 库的更多信息，请参阅 Delta Lake 的 Python 文档页面。

下表列出了 AWS Glue 每个版本中包含的 Delta Lake 版本。

AWS Glue 版本	支持的 Delta Lake 版本
4.0	2.1.0
3.0	1.0.0

要了解有关 AWS Glue 支持的数据湖框架的更多信息，请参阅[在 AWS Glue ETL 任务中使用数据湖框架](#)。

为 AWS Glue 启用 Delta Lake

要为 AWS Glue 启用 Delta Lake，请完成以下任务：

- 指定 `delta` 作为 `--datalake-formats` 作业参数的值。有关更多信息，请参阅 [AWS Glue 作业参数](#)。
- `--conf` 为 Glue 作业创建一个名为 AWS 的密钥，并将其设置为以下值。或者，您可以在脚本中使用 `SparkConf` 设置以下配置。这些设置有助于 Apache Spark 正确处理 Delta Lake 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
```

- AWS Glue 4.0 默认为 Delta 表启用了 Lake Formation 权限支持。无需额外配置即可读取/写入注册到 Lake Formation 的 Delta 表。AWS Glue 作业 IAM 角色必须具有 SELECT 权限才能读取已注册的 Delta 表。AWS Glue 作业 IAM 角色必须具有 SUPER 权限才能写入已注册的 Delta 表。要了解有关管理 Lake Formation 权限的更多信息，请参阅 [Granting and revoking permissions on Data Catalog resources](#)。

使用不同的 Delta Lake 版本

要使用 AWS Glue 不支持的 Delta Lake 版本，请使用 `--extra-jars` 作业参数指定您自己的 Delta Lake JAR 文件。请勿包含 `delta` 作为 `--datalake-formats` 作业参数的值。要在这种情况下使用 Delta Lake Python 库，必须使用 `--extra-py-files` 作业参数指定库 JAR 文件。Python 库打包在 Delta Lake JAR 文件中。

示例：将 Delta Lake 表写入 Amazon S3，并将其注册到 AWS Glue 数据目录

以下 AWS Glue ETL 脚本演示了如何将 Delta Lake 表写入 Amazon S3，并将该表注册到 AWS Glue 数据目录。

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>")
```

示例：使用 AWS Glue 数据目录从 Amazon S3 读取 Delta Lake 表

以下 AWS Glue ETL 脚本读取您在 示例：将 Delta Lake 表写入 Amazon S3，并将其注册到 AWS Glue 数据目录 中创建的 Delta Lake 表。

Python

在本示例中，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

在本示例中，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()
  }
}
```

示例：使用 AWS Glue 数据目录在 Amazon S3 中将 **DataFrame** 插入 Delta Lake 表

此示例将数据插入您在 示例：将 Delta Lake 表写入 Amazon S3，并将其注册到 AWS Glue 数据目录 中创建的 Delta Lake 表。

> **Note**
>
> 此示例要求您设置 `--enable-glue-datacatalog` 任务参数，才能将 AWS Glue Data Catalog 用作 Apache Spark Hive 元存储。要了解更多信息，请参阅 AWS Glue 作业参数。

Python

在本示例中，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

在本示例中，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)
  }
```

```
}
```

示例：使用 Spark API 从 Amazon S3 读取 Delta Lake 表

此示例使用 Spark API 从 Amazon S3 读取 Delta Lake 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/>")
```

示例：使用 Spark 向 Amazon S3 写入 Delta Lake 表

此示例使用 Spark 向 Amazon S3 写入 Delta Lake 表。

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/>")
```

示例：读取和写入具有 Lake Formation 权限控制的 Delta Lake 表

此示例将读取和写入一个具有 Lake Formation 权限控制的 Delta Lake 表。

1. 创建一个 Delta 表并将其注册到 Lake Formation

   a. 要启用 Lake Formation 权限控制，您首先需要将表的 Amazon S3 路径注册到 Lake Formation。有关更多信息，请参阅 Registering an Amazon S3 location（注册 Amazon S3 位置）。您可以通过 Lake Formation 控制台或使用 AWS CLI 进行注册：

   ```
   aws lakeformation register-resource --resource-arn arn:aws:s3:::<s3-bucket>/<s3-folder> --use-service-linked-role --region <REGION>
   ```

   注册了 Amazon S3 位置后，对于任何指向该位置（或其任何子位置）的 AWS Glue 表，`GetTable` 调用中的 `IsRegisteredWithLakeFormation` 参数都将返回值 true。

   b. 创建一个指向通过 Spark 注册的 Amazon S3 路径的 Delta 表：

   > **Note**
   >
   > 以下示例属于 Python 示例。

   ```
   dataFrame.write \
    .format("delta") \
    .mode("overwrite") \
    .partitionBy("<your_partitionkey_field>") \
    .save("s3://<the_s3_path>")
   ```

   将数据写入 Amazon S3 后，使用 AWS Glue 爬网程序创建新的 Delta 目录表。有关更多信息，请参阅 Introducing native Delta Lake table support with AWS Glue crawlers。

   您也可以通过 AWS Glue `CreateTable` API 手动创建表。

2. 向 AWS Glue 作业 IAM 角色授予 Lake Formation 权限。您可以通过 Lake Formation 控制台授予权限，也可以使用 AWS CLI 授予权限。有关更多信息，请参阅 Granting table permissions using the Lake Formation console and the named resource method。

3. 读取注册到 Lake Formation 的 Delta 表。代码与读取未注册的 Delta 表相同。请注意，AWS Glue 作业 IAM 角色需要具有 SELECT 权限才能成功读取。

```
# 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. 写入注册到 Lake Formation 的 Delta 表。代码与写入未注册的 Delta 表相同。请注意，AWS Glue 作业 IAM 角色需要具有 SUPER 权限才能成功写入。

   默认情况下，AWS Glue 会将 Append 作为 saveMode 使用。您可以通过设置 `additional_options` 中的 saveMode 选项来对其进行更改。要了解 Delta 表中对 saveMode 的支持，请参阅 Write to a table。

```
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 框架

AWS Glue 3.0 及更高版本支持数据湖的 Apache Iceberg 框架。Iceberg 提供了一种高性能的表格式，其工作原理与 SQL 表类似。本主题涵盖了在 Iceberg 表中传输或存储数据时，在 AWS Glue 中使用数据的可用功能。要了解有关 Iceberg 的更多信息，请参阅 Apache Iceberg 官方文档。

您可以使用 AWS Glue 对 Amazon S3 中的 Iceberg 表执行读写操作，也可以使用 AWS Glue 数据目录处理 Iceberg 表。还支持其他操作，包括插入、更新和所有 Spark 查询 Spark 写入。

> **Note**
>
> `ALTER TABLE … RENAME TO` 不适用于 Apache Iceberg 0.13.1 for AWS Glue 3.0。

下表列出了 AWS Glue 每个版本中包含的 Iceberg 版本。

AWS Glue 版本	支持 Iceberg 版本
4.0	1.0.0
3.0	0.13.1

要了解有关 AWS Glue 支持的数据湖框架的更多信息，请参阅[在 AWS Glue ETL 任务中使用数据湖框架](#)。

启用 Iceberg 框架

要启用 Iceberg for AWS Glue，请完成以下任务：

- 指定 `iceberg` 作为 `--datalake-formats` 作业参数的值。有关更多信息，请参阅 [AWS Glue 作业参数](#)。
- `--conf` 为 Glue 作业创建一个名为 AWS 的密钥，并将其设置为以下值。或者，您可以在脚本中使用 `SparkConf` 设置以下配置。这些设置有助于 Apache Spark 正确处理 Iceberg 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
```

如果您读取或写入注册到 Lake Formation 的 Iceberg 表，请添加以下配置以启用 Lake Formation 支持。请注意，只有 AWS Glue 4.0 才支持注册到 Lake Formation 的 Iceberg 表：

```
--conf spark.sql.catalog.glue_catalog.glue.lakeformation-enabled=true
--conf spark.sql.catalog.glue_catalog.glue.id=<table-catalog-id>
```

如果您将 AWS Glue 3.0 与 Iceberg 0.13.1 一起使用，则必须设置以下附加配置才能使用 Amazon DynamoDB 锁定管理器来确保原子交易。AWSGlue 4.0 默认使用乐观锁。有关更多信息，请参阅 Apache Iceberg 官方文档中的 [Iceberg AWS 集成](#)。

```
--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

要使用 AWS Glue 不支持的 Iceberg 版本，请使用 `--extra-jars` 作业参数指定您自己的 Iceberg JAR 文件。请勿包含 `iceberg` 作为 `--datalake-formats` 参数的值。

为 Iceberg 表启用加密

> **Note**
>
> Iceberg 表有自己的用于启用服务器端加密的机制。除了 AWS Glue 的安全配置外，您还应该启用此配置。

要在 Iceberg 表上启用服务器端加密，请查看 Iceberg 文档中的指南。

示例：将 Iceberg 表写入 Amazon S3 并将其注册到 AWS Glue 数据目录

此示例脚本演示了如何将 Iceberg 表写入 Amazon S3。该示例使用 IcebergAWS 集成将表注册到 AWS Glue 数据目录。

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，您可以使用 Spark 方法将 Iceberg 表写入 Amazon S3 和 Data Catalog。

先决条件：您需要预置目录以供 Iceberg 库使用。使用 AWS Glue Data Catalog 时，AWS Glue 让这一切变得简单明了。AWS Glue Data Catalog 已预先配置为供 Spark 库作为 `glue_catalog` 使用。Data Catalog 表由 *`databaseName`* 和 *`tableName`* 标识。有关 AWS Glue Data Catalog 的更多信息，请参阅 数据发现和编目。

如果您不使用 AWS Glue Data Catalog ，则需要通过 Spark API 配置目录。有关更多信息，请参阅 Iceberg 文档中的 Spark Configuration。

此示例使用 Spark 从将 Iceberg 表写入 Amazon S3 和 Data Catalog 中。

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()
```

示例：使用 AWS Glue 数据目录从 Amazon S3 读取 Iceberg 表

此示例读取您在 示例：将 Iceberg 表写入 Amazon S3 并将其注册到 AWS Glue 数据目录 中创建的 Iceberg 表。

Python

在本示例中，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

在本示例中，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()
  }
}
```

示例：使用 AWS Glue 数据目录在 Amazon S3 将 **DataFrame** 插入 Iceberg 表

此示例将数据插入您在 示例：将 Iceberg 表写入 Amazon S3 并将其注册到 AWS Glue 数据目录 中创建的 Iceberg 表。

> **Note**
>
> 此示例要求您设置 `--enable-glue-datacatalog` 任务参数，才能将 AWS Glue Data Catalog 用作 Apache Spark Hive 元存储。要了解更多信息，请参阅 AWS Glue 作业参数。

Python

在本示例中，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

在本示例中，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)
  }
}
```

示例：使用 Spark 从 Amazon S3 读取 Iceberg 表

先决条件：您需要预置目录以供 Iceberg 库使用。使用 AWS Glue Data Catalog 时，AWS Glue 让这一切变得简单明了。AWS Glue Data Catalog 已预先配置为供 Spark 库作为 `glue_catalog` 使用。Data Catalog 表由 *`databaseName`* 和 *`tableName`* 标识。有关 AWS Glue Data Catalog 的更多信息，请参阅 数据发现和编目。

如果您不使用 AWS Glue Data Catalog ，则需要通过 Spark API 配置目录。有关更多信息，请参阅 Iceberg 文档中的 Spark Configuration。

此示例使用 Spark 从 Data Catalog 读取 Amazon S3 中的 Iceberg 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")
```

示例：读取和写入具有 Lake Formation 权限控制的 Iceberg 表

此示例将读取和写入一个具有 Lake Formation 权限控制的 Iceberg 表。

1. 创建一个 Iceberg 表并将其注册到 Lake Formation：

   a. 要启用 Lake Formation 权限控制，您首先需要将表的 Amazon S3 路径注册到 Lake Formation。有关更多信息，请参阅 Registering an Amazon S3 location（注册 Amazon S3 位置）。您可以通过 Lake Formation 控制台或使用 AWS CLI 进行注册：

   ```
   aws lakeformation register-resource --resource-arn arn:aws:s3:::<s3-bucket>/<s3-folder> --use-service-linked-role --region <REGION>
   ```

   注册了 Amazon S3 位置后，对于任何指向该位置（或其任何子位置）的 AWS Glue 表，`GetTable` 调用中的 `IsRegisteredWithLakeFormation` 参数都将返回值 true。

   b. 创建一个指向通过 Spark SQL 注册的路径的 Iceberg 表：

   > **Note**
   >
   > 以下示例属于 Python 示例。

   ```
   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)
   ```

   您也可以通过 AWS Glue `CreateTable` API 手动创建表。有关更多信息，请参阅 Creating Apache Iceberg tables。

2. 向作业 IAM 角色授予 Lake Formation 权限。您可以通过 Lake Formation 控制台授予权限，也可以使用 AWS CLI 授予权限。有关更多信息，请参阅 https://docs.aws.amazon.com/lake-formation/latest/dg/granting-table-permissions.html

3. 读取注册到 Lake Formation 的 Iceberg 表。代码与读取未注册的 Iceberg 表相同。请注意，您的 AWS Glue 作业 IAM 角色需要具有 SELECT 权限才能成功读取。

   ```
   # 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. 写入注册到 Lake Formation 的 Iceberg 表。代码与写入未注册的 Iceberg 表相同。请注意，您的 AWS Glue 作业 IAM 角色需要具有 SUPER 权限才能成功写入。

```
glueContext.write_data_frame.from_catalog(
    frame=dataFrame,
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

共享配置参考

您可以对任何格式类型使用以下 `format_options` 值：

- `attachFilename` - 适当格式的字符串，用作列名。如果您提供此选项，则记录的源文件名将附加到记录中。参数值将用作列名。
- `attachTimestamp` - 适当格式的字符串，用作列名。如果您提供此选项，则记录的源文件的修改时间将附加到记录中。参数值将用作列名。

适用于 Spark SQL 作业的 AWS Glue 数据目录支持

AWS Glue 是一个与 Apache Hive 元存储兼容的目录。您可以配置 AWS Glue 任务和开发端点以使用数据目录作为外部 Apache Hive 元存储。随后，您可以直接对存储在数据目录中的表运行 Apache Spark SQL 查询。预设情况下，AWS Glue 动态帧与数据目录集成。但是，利用此功能，Spark SQL 任务可以开始使用数据目录作为外部 Hive 元存储。

此功能要求对 AWS Glue API 端点的网络访问权限。对于连接位于私有子网中的 AWS Glue 任务，必须配置 VPC 终端节点或 NAT 网关以提供网络访问。有关配置 VPC 终端节点的信息，请参阅 设置对数据存储的网络访问。要创建 NAT 网关，请参阅 Amazon VPC 用户指南中的 NAT 网关。

您可以通过将 `"--enable-glue-datacatalog": ""` 参数分别添加到作业参数和开发终端节点参数来配置 AWS Glue 作业和开发终端节点。传递此参数将在 Spark 中设置某些配置，使其能够访问数

据目录作为外部 Hive 元存储。它还在 AWS Glue 任务或开发端点中创建的 SparkSession 对象中启用 Hive 支持。

要启用数据目录访问，请在控制台的 Add job (添加任务) 或 Add endpoint (添加端点) 页面上，选中 Catalog options (目录选项) 组中的 Use AWS Glue Data Catalog as the Hive metastore (使用 AWS Glue 数据目录作为 Hive 元存储) 复选框。请注意，用于作业或开发终端节点的 IAM 角色应具有 glue:CreateDatabase 权限。在数据目录中创建一个名为“default”的数据库（如果该数据库不存在）。

让我们看一下如何在 Spark SQL 作业中使用此功能的示例。以下示例假定您已对 s3://awsglue-datasets/examples/us-legislators 中提供的美国议员数据集进行爬网。

要从 AWS Glue 数据目录中定义的表中序列化/反序列化数据，Spark SQL 需要在 Spark 任务的类路径中的 AWS Glue 数据目录中定义的格式的 Hive SerDe 类。

某些常见格式的 SerDes 由 AWS Glue 分发。以下是这些格式的 Amazon S3 链接：

- JSON
- XML
- Grok

将 JSON SerDe 作为额外的 JAR 添加到开发终端节点。对于作业，您可以使用参数字段中的 --extra-jars 参数添加 SerDe。有关更多信息，请参阅 AWS Glue 作业参数。

以下是用于创建开发终端节点的示例输入 JSON，其中已为 Spark SQL 启用数据目录。

```
{
    "EndpointName": "Name",
    "RoleArn": "role_ARN",
    "PublicKey": "public_key_contents",
    "NumberOfNodes": 2,
    "Arguments": {
      "--enable-glue-datacatalog": ""
    },
    "ExtraJarsS3Path": "s3://crawler-public/json/serde/json-serde.jar"
}
```

现在，使用 Spark SQL 查询从美国议员数据集创建的表。

```
>>> 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|
+--------------------+---------+-----------------+
```

如果作业的类路径中没有该格式的 SerDe 类，您将看到与以下内容类似的错误。

```
>>> 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
```

要仅从 memberships 表中查看不同的 organization_ids，请运行以下 SQL 查询。

```
>>> spark.sql("select distinct organization_id from memberships_json").show()
+--------------------+
```

```
|     organization_id|
+--------------------+
|d56acebe-8fdc-47b...|
|8fa6c3d2-71dc-478...|
+--------------------+
```

如果需要对动态帧执行相同操作，请运行以下操作。

```
>>> 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...|
+--------------------+
```

虽然 DynamicFrames 已针对 ETL 操作进行优化，但启用 Spark SQL 以直接访问数据目录提供了一种运行复杂的 SQL 语句或移植现有应用程序的简洁方法。

使用作业书签

AWS Glue for Spark 使用作业书签来跟踪已处理的数据。有关作业书签功能及其支持的内容的摘要，请参阅 the section called “使用作业书签跟踪已处理的数据”。使用书签对 AWS Glue 作业进行编程时，您可以获得可视化作业中无法获得的灵活性。

- 从 JDBC 读取时，您可以指定要用作 AWS Glue 脚本中书签键的列。
- 您可以选择将哪个 `transformation_ctx` 方法应用于每个方法调用。

始终在脚本开头调用 *`job.init`* 并在脚本结尾调用 *`job.commit`*，并恰当配置参数。这两个函数初始化书签服务并更新服务的状态更改。如果没有调用书签，书签将无法正常工作。

指定书签键

对于 JDBC 工作流程，书签通过将关键字段的值与已添加书签的值进行比较来跟踪您的作业已读取的行。这不是必需的，也不适用于 Amazon S3 工作流程。在没有可视化编辑器的情况下编写 AWS Glue 脚本时，您可以指定使用书签跟踪哪一列。您也可以指定多列。指定用户定义的书签键时，允许在值序列中出现间隔。

> **Warning**
>
> 如果使用用户定义的书签键，则它们每个都必须严格单调递增或递减。为复合键选择其他字段时，“次要版本”或“修订编号”等概念的字段不符合此标准，因为它们的值会在整个数据集中重复使用。

您可以通过以下方式指定 `jobBookmarkKeys` 和 `jobBookmarkKeysSortOrder`：

- `create_dynamic_frame.from_catalog` — 使用 `additional_options`。
- `create_dynamic_frame.from_options` — 使用 `connection_options`。

转换上下文

许多 AWS Glue PySpark 动态帧方法都包括一个名为 `transformation_ctx` 的可选参数，这是 ETL 运算符实例的唯一标识符。该 `transformation_ctx` 参数用于在作业书签中标识给定运算符的状态信息。具体来说，AWS Glue 使用 `transformation_ctx` 来为书签状态键建立索引。

> **Warning**
>
> `transformation_ctx` 作为键以搜索脚本中特定源的书签状态。为了使书签正常工作，您应始终让源和相关的 `transformation_ctx` 保持一致。更改源属性或重命名 `transformation_ctx` 可能会使之前的书签无效，并且基于时间戳的筛选条件可能无法产生正确的结果。

为了使作业书签正常使用，请启用作业书签参数并设置 `transformation_ctx` 参数。如果您未传入 `transformation_ctx` 参数，则不会为方法中使用的动态帧或表启用作业书签。例如，如果您有一个用于读取和连接两个 Amazon S3 源的 ETL 任务，您可能会选择仅将 `transformation_ctx` 参数传递给要启用书签的方法。如果您重置作业的作业书签，它会重置与作业相关联的所有转换，而不考虑所使用的 `transformation_ctx`。

有关 `DynamicFrameReader` 类的更多信息，请参阅DynamicFrameReader 班级。有关 PySpark 扩展的更多信息，请参阅 AWS Glue PySpark 扩展参考。

示例

Example

以下是为 Amazon S3 数据源生成的脚本的示例。使用作业书签所需的脚本部分以斜体显示。有关这些元素的更多信息，请参阅 GlueContext 班级 API 和 DynamicFrameWriter 类 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

以下是为 JDBC 源生成的脚本的示例。源表是一个将 empno 列作为主键的员工表。尽管默认情况下，如果未指定书签键，则任务会使用顺序主键作为书签键，但由于 empno 不一定是连续的，值中可能会有间隙，它不能作为默认书签键。因此，脚本会将 empno 显式指定为书签键。代码的此部分以斜体显示。

```
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 外部使用敏感数据检测

AWS Glue Studio 允许您检测敏感数据，但是，您也可以在 AWS Glue Studio 之外使用敏感数据检测功能。

有关托管的敏感数据类型的完整列表，请参阅 Managed data types。

使用 AWS 托管 PII 类型检测敏感数据检测

AWS Glue 在一个 AWS Glue ETL 任务中提供了两个 API。它们是 `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 来识别 AWS 托管 PII 类型和自定义实体类型。将使用检测结果自动创建一个新列。`classifyColumns()` API 将返回一个映射，其中键是列名，值是检测到的实体类型列

表。SampleFraction 表示扫描 PII 实体时要采样的一小部分数据，而 ThresholdFraction 表示为了将列标识为 PII 数据而必须满足的一小部分数据。

行级别检测

在示例中，该任务使用 detect() 和 classifyColumns() API 执行以下操作：

- 从 Amazon S3 存储桶中读取数据并将其变成 DynamicFrame
- 在 dynamicFrame 中检测“电子邮件”和“信用卡”的实例
- 返回一个 dynamicFrame，其中包含原始值和一列，其中包含每行的检测结果
- 将返回的 DynamicFrame 写入另一条路径 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()
   }
 }
```

使用精细操作进行行级别检测

示例中的作业将使用 detect() API 执行以下操作：

- 从 Amazon S3 存储桶读取数据并将其转换为 dynamicFrame
- 检测 dynamicFrame 中的“USA_PTIN”、“BANK_ACCOUNT”、“USA_SSN”、“USA_PASSPORT_NUMBER”和“PHONE_NUMBER”等敏感数据类型
- 返回一个 dynamicFrame，其中包含修改后的遮蔽值以及一个包含每行的检测结果的列
- 将返回的 dynamicFrame 写入另一个 Amazon S3 路径

与上述 detect() API 不同的是，这将使用精细操作来检测实体类型。有关更多信息，请参阅 [使用 detect() 方法所需的检测参数]。

```
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()
  }
}
```

列级别检测

示例中的作业将使用 `classifyColumns()` API 执行以下操作：

- 从 Amazon S3 存储桶读取数据并将其转换为 dynamicFrame
- 在 dynamicFrame 中检测“电子邮件”和“信用卡”的实例
- 将参数设置为对列进行 100% 采样，如果实体位于 10% 的单元格中，并且敏感性为“低”，则将其标记为已检测
- 返回一个以列名为键，以检测到的实体类型列表为值的映射
- 将返回的 dynamicFrame 写入另一个 Amazon S3 路径

```
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 类型检测敏感数据检测

您可以通过 AWS Studio 定义自定义实体。但是，要在 AWS Studio 中使用此功能，必须先定义自定义实体类型，然后将定义的自定义实体类型添加到列表中`entityTypesToDetect`。

如果您的数据中有特定的敏感数据类型（例如“员工 ID”），则可以通过调用 `CreateCustomEntityType()` API 创建自定义实体。以下示例使用请求参数将自定义实体类型“EMPLOYEE_ID”定义到 `CreateCustomEntityType()` API：

```
  {
      "name": "EMPLOYEE_ID",
      "regexString": "\d{4}-\d{3}",
      "contextWords": ["employee"]
  }
```

然后，通过将自定义实体类型 (EMPLOYEE_ID) 添加到 `EntityDetector()` API，修改任务以使用新的自定义敏感数据类型：

```
 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**
>
> 如果定义的自定义敏感数据类型与现有托管实体类型的名称相同，则自定义敏感数据类型将优先并覆盖该托管实体类型的逻辑。

使用 `detect()` 方法所需的检测参数

此方法用于检测中的实体 DynamicFrame。它返回一个 DataFrame 包含原始值的新列和一个包含 PII 检测元数据的附加列 outputColumnName 。自定义屏蔽可以在AWS Glue脚本中返回后完成，也可以改用带有细粒度操作的 DynamicFrame detect () API。

```
detect(frame: DynamicFrame,
       entityTypesToDetect: Seq[String],
       outputColumnName: String = "DetectedEntities",
       detectionSensitivity: String = "LOW"): DynamicFrame
```

参数：

- fram e — (类型：`DynamicFrame`) DynamicFrame 包含待处理数据的输入。
- entityTypesTo检测- (类型：`[Seq[String]`) 要检测的实体类型列表。可以是托管式实体类型，也可以是自定义实体类型。
- outputColumnName— (type:`String` ，默认值：DetectedEntities“”) 存储检测到的实体的列的名称。如果未提供，则默认列名为 “DetectedEntities”。
- detectionSensitivity – (类型：`String` ，选项：“低”或“高” ，默认值：“低”) 指定检测过程的灵敏度。有效选项为“低”或“高”。如果未提供此参数，则默认灵敏度设置为“低”。

`outputColumnName` 设置：

用于存储检测到的实体的列名。如果未提供，则默认列名为 “DetectedEntities”。对于输出列中的每一行，补充列都用以下键值对来包含了列名与检测到的实体元数据的映射：

- entityType – 检测到的实体类型。
- start – 检测到的实体在原始数据中的起始位置。
- end – 检测到的实体在原始数据中的结束位置。
- actionUsed – 对检测到的实体执行的操作 (例如，“DETECT”、“REDACT”、“PARTIAL_REDACT”、“SHA256_HASH”) 。

例如：

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

使用精细操作的 **detect()** 方法的检测参数

此方法用于 DynamicFrame 使用指定的参数检测中的实体。它返回一个新 DataFrame 列，其原始值替换为屏蔽的敏感数据，以及一个包含 PII 检测元数据的附加列`outputColumnName`。

```
detect(frame: DynamicFrame,
       detectionParameters: JsonOptions,
       outputColumnName: String = "DetectedEntities",
       detectionSensitivity: String = "LOW"): DynamicFrame
```

参数：

- fram e —（类型：DynamicFrame）： DynamicFrame 包含待处理数据的输入。
- detectionParameters –（类型：Json0ptions）：为检测进程指定参数的 JSON 选项。
- outputColumnName—（type:String，默认值："DetectedEntities“）：将存储检测到的实体的列的名称。如果未提供，则默认列名为 “DetectedEntities”。
- detectionSensitivity –（类型：String，选项：“低”或“高”，默认值：“低”）：指定检测过程的灵敏度。有效选项为“低”或“高”。如果未提供此参数，则默认灵敏度设置为“低”。

detectionParameters 设置

如果未包含任何设置，则将使用默认值。

- action –（类型：String，选项：“DETECT”、“REDACT”、“PARTIAL_REDACT”、“SHA256_HASH”）指定要对实体执行的操作。必需。请注意，对于执行遮蔽的操作（除“DETECT”之外的所有操作），每列只能执行一个操作。这是一种用于遮蔽合并后实体的预防措施。
- sourceColumns –（类型：List[String]，默认值：[“*”]）要对实体执行检测的源列名列表。如果不存在，则默认为 [“*”]。如果使用的列名无效，则将引发 IllegalArgumentException。
- sourceColumnsTo排除-（类型：List[String]）要对实体执行检测的源列名称列表。使用 sourceColumns 或 sourceColumnsToExclude。如果使用的列名无效，则将引发 IllegalArgumentException。
- actionOptions – 基于指定操作的附加选项：
  - 对于“DETECT”和“SHA256_HASH”，不允许使用任何选项。
  - 对于“REDACT”：
    - redactText –（键入：String，默认值：“*****”）用于替换检测到的实体的文本。
  - 对于“PARTIAL_REDACT”：
    - redactChar –（类型：String，默认值：“*”）用于替换实体中每个检测到的字符的字符。
    - matchPattern –（类型：String）用于部分编辑的正则表达式模式。不能与 numLeftCharsToExclude 或结合使用numRightCharsToExclude。
    - numLeftCharsToExclude—（类型：String, integer）要排除的左侧字符数。不能与 matchPattern 组合使用，但可以与 numRightCharsToExclude 组合使用。
    - numRightCharsToExclude—（类型：String, integer）要排除的正确字符数。不能与 matchPattern 组合使用，但可以与 numRightCharsToExclude 组合使用。

outputColumnName 设置

查看 outputColumnName 设置

`classifyColumns()` 方法的检测参数

此方法用于检测中的实体 DynamicFrame。这将返回一个以列名为键，以检测到的实体类型列表为值的映射。返回此映射后可以在 AWS Glue 脚本中进行自定义遮蔽。

```
classifyColumns(frame: DynamicFrame,
                entityTypesToDetect: Seq[String],
                sampleFraction: Double = 0.1,
                thresholdFraction: Double = 0.1,
                detectionSensitivity: String = "LOW")
```

参数：

- fram e — (类型：DynamicFrame) DynamicFrame 包含待处理数据的输入。
- entityTypesTo检测- (类型：Seq[String]) 要检测的实体类型列表。可以是托管式实体类型，也可以是自定义实体类型。
- sampleFraction – (类型：Double，默认值：10%) 扫描 PII 实体时的数据采样率。
- thresholdFraction – (类型：Double，默认值：10%) ：要将列标识为 PII 数据时必须达到的数据占比。
- detectionSensitivity – (类型：String，选项：“低”或“高”，默认值：“低”) 指定检测过程的灵敏度。有效选项为“低”或“高”。如果未提供此参数，则默认灵敏度设置为“低”。

托管敏感数据类型

全球实体

数据类型	类别	描述
PERSON_NAME	通用	人员的姓名。
EMAIL	个人	电子邮件地址。
IP_ADDRESS	计算机	IP 地址
MAC_ADDRESS	个人	MAC 地址。

美国数据类型

数据类型	描述
BANK_ACCOUNT	银行账号。但是不特定于某个国家或地区，仅检测美国和加拿大的账户格式。
CREDIT_CARD	信用卡号。
PHONE_NUMBER	电话号码。但是不特定于某个国家或地区，目前仅检测美国和加拿大的电话号码。
USA_ATIN	美国国税局颁发的美国个人纳税识别号码。
USA_CPT_CODE	CPT 代码（美国专用）。
USA_DEA_NUMBER	DEA 编号（美国专用）。
USA_DRIVING_LICENSE	驾驶执照号码（美国专用）。
USA_HCPCS_CODE	HCPCS 代码（美国专用）。
USA_HEALTH_INSURANCE_CLAIM_NUMBER	健康保险索赔编号（美国专用）。
USA_ITIN	ITIN（适用于美国人或实体）。
USA_MEDICARE_BENEFICIARY_IDENTIFIER	医疗保险受益人识别码（美国专用）。
USA_NATIONAL_DRUG_CODE	NDC 代码（美国专用）。
USA_NATIONAL_PROVIDER_IDENTIFIER	国家供应商识别码（美国专用）。
USA_PASSPORT_NUMBER	护照号码（适用于美国人）。
USA_PTIN	美国国税局颁发的美国报税人纳税识别号。
USA_SSN	社会保障号码（适用于美国人）。

阿根廷数据类型

数据类型	描述
ARGENTINA_TAX_IDENTIFICATIO N_NUMBER	阿根廷纳税识别号。也被称为 CUIT 或 CUIL。

澳大利亚数据类型

数据类型	描述
AUSTRALIA_BUSINESS_NUMBER	澳大利亚商业号码（ABN）。澳大利亚商务局（ABR）颁发的唯一标识符，用于向政府和社区标识企业。
AUSTRALIA_COMPANY_NUMBER	澳大利亚公司编号（ACN）。澳大利亚证券和投资委员会颁发的唯一标识符。
AUSTRALIA_DRIVING_LICENSE	澳大利亚的驾驶执照号码。
AUSTRALIA_MEDICARE_NUMBER	澳大利亚医疗保险号码。澳大利亚健康保险委员会颁发的个人身份证件。
AUSTRALIA_PASSPORT_NUMBER	澳大利亚护照号码。
AUSTRALIA_TAX_FILE_NUMBER	澳大利亚税务档案号（TFN）。由澳大利亚税务局（ATO）签发给纳税人（个人、公司等）进行税务交易。

奥地利数据类型

数据类型	描述
AUSTRIA_DRIVING_LICENSE	驾驶执照号码（奥地利专用）。
AUSTRIA_PASSPORT_NUMBER	护照号码（奥地利专用）。
AUSTRIA_SSN	社会保障号码（适用于奥地利人）。

数据类型	描述
AUSTRIA_TAX_IDENTIFICATION_NUMBER	纳税识别号（奥地利专用）。
AUSTRIA_VALUE_ADDED_TAX	增值税（奥地利专用）。

巴尔干数据类型

数据类型	描述
BOSNIA_UNIQUE_MASTER_CITIZEN_NUMBER	波斯尼亚–黑塞哥维亚公民的唯一主公民号码（JMBG）。
KOSOVO_UNIQUE_MASTER_CITIZEN_NUMBER	科索沃的唯一公民主号码（JMBG）。
MACEDONIA_UNIQUE_MASTER_CITIZEN_NUMBER	马其顿的唯一主公民号码。
MONTENEGRO_UNIQUE_MASTER_CITIZEN_NUMBER	黑山的唯一公民主号码（JMBG）。
SERBIA_UNIQUE_MASTER_CITIZEN_NUMBER	塞尔维亚的唯一公民主号码（JMBG）。
SERBIA_VALUE_ADDED_TAX	增值税（塞尔维亚专用）。
VOJVODINA_UNIQUE_MASTER_CITIZEN_NUMBER	伏伊伏丁那的唯一公民主号码（JMBG）。

比利时数据类型

数据类型	描述
BELGIUM_DRIVING_LICENSE	驾驶执照号码（比利时专用）

数据类型	描述
BELGIUM_NATIONAL_IDENTIFICATION_NUMBER	比利时国家号码（BNN）。
BELGIUM_PASSPORT_NUMBER	护照号码（比利时专用）。
BELGIUM_TAX_IDENTIFICATION_NUMBER	纳税识别号（比利时专用）。
BELGIUM_VALUE_ADDED_TAX	增值税（比利时专用）。

巴西数据类型

数据类型	描述
BRAZIL_BANK_ACCOUNT	银行账号（巴西专用）。
BRAZIL_NATIONAL_IDENTIFICATION_NUMBER	国民识别码（巴西专用）。
BRAZIL_NATIONAL_REGISTRY_OF_LEGAL_ENTITIES_NUMBER	颁发给公司的识别码（巴西专用），也称为 CNPJ。
BRAZIL_NATURAL_PERSON_REGISTRY_NUMBER	自然人登记号，也称为 CPF。

保加利亚数据类型

数据类型	描述
BULGARIA_DRIVING_LICENSE	驾驶执照号码（保加利亚专用）。
BULGARIA_UNIFORM_CIVIL_NUMBER	统一民用号码（EGN），用作国家识别码。
BULGARIA_VALUE_ADDED_TAX	增值税（保加利亚专用）。

加拿大数据类型

数据类型	描述
CANADA_DRIVING_LICENSE	驾驶执照号码（加拿大专用）。
CANADA_GOVERNMENT_IDENTIFICATION_CARD_NUMBER	国民识别码（加拿大专用）。
CANADA_PASSPORT_NUMBER	护照号码（加拿大专用）。
CANADA_PERMANENT_RESIDENCE_NUMBER	永久居留号码（PR 卡号）。
CANADA_PERSONAL_HEALTH_NUMBER	医疗保健的唯一标识符（PHN 号码）。
CANADA_SOCIAL_INSURANCE_NUMBER	加拿大社会保险号码（SIN）。

智利数据类型

数据类型	描述
CHILE_DRIVING_LICENSE	驾驶执照号码（智利专用）。
CHILE_NATIONAL_IDENTIFICATION_NUMBER	智利国民标识符，也称为 RUT 或 RUN。

中国、香港特别行政区、澳门特别行政区和台湾数据类型

数据类型	描述
CHINA_IDENTIFICATION	中国标识符。
CHINA_LICENSE_PLATE_NUMBER	驾驶执照号码（中国专用）。
CHINA_MAINLAND_TRAVEL_PERMIT_ID_HONG_KONG_MACAU	港澳居民来往内地通行证。
CHINA_MAINLAND_TRAVEL_PERMIT_ID_TAIWAN	中华人民共和国（中国）政府签发的台湾居民来往大陆通行证。

数据类型	描述
CHINA_PASSPORT_NUMBER	护照号码（中国专用）。
CHINA_PHONE_NUMBER	电话号码（中国专用）。
HONG_KONG_IDENTITY_CARD	香港特别行政区入境事务处签发的正式身份证明文件。
MACAU_RESIDENT_IDENTITY_CARD	澳门居民身份证（BIR）是由澳门特别行政区身份证明服务局签发的官方身份证。
TAIWAN_NATIONAL_IDENTIFICATION_NUMBER	国民识别码（台湾专用）。
TAIWAN_PASSPORT_NUMBER	护照号码（台湾专用）。

哥伦比亚数据类型

数据类型	描述
COLOMBIA_PERSONAL_IDENTIFICATION_NUMBER	哥伦比亚人出生时分配的唯一标识符。
COLOMBIA_TAX_IDENTIFICATION_NUMBER	纳税识别号（哥伦比亚专用）。

克罗地亚数据类型

数据类型	描述
CROATIA_DRIVING_LICENSE	驾驶执照号码（克罗地亚专用）。
CROATIA_IDENTITY_NUMBER	国民识别码（克罗地亚专用）。
CROATIA_PASSPORT_NUMBER	护照号码（克罗地亚专用）。
CROATIA_PERSONAL_IDENTIFICATION_NUMBER	个人标识号（OIB）。

塞浦路斯数据类型

数据类型	描述
CYPRUS_DRIVING_LICENSE	驾驶执照号码（塞浦路斯专用）。
CYPRUS_NATIONAL_IDENTIFICATION_NUMBER	塞浦路斯身份证。
CYPRUS_PASSPORT_NUMBER	护照号码（塞浦路斯专用）。
CYPRUS_TAX_IDENTIFICATION_NUMBER	纳税识别号（塞浦路斯专用）。
CYPRUS_VALUE_ADDED_TAX	增值税（塞浦路斯专用）。

捷克数据类型

数据类型	描述
CZECHIA_DRIVING_LICENSE	驾驶执照号码（捷克专用）。
CZECHIA_PERSONAL_IDENTIFICATION_NUMBER	个人标识号（捷克专用）。
CZECHIA_VALUE_ADDED_TAX	增值税（捷克专用）。

丹麦数据类型

数据类型	描述
DENMARK_DRIVING_LICENSE	驾驶执照号码（丹麦专用）。
DENMARK_PERSONAL_IDENTIFICATION_NUMBER	个人标识号（丹麦专用）。
DENMARK_TAX_IDENTIFICATION_NUMBER	纳税识别号（丹麦专用）。
DENMARK_VALUE_ADDED_TAX	增值税（丹麦专用）。

爱沙尼亚数据类型

数据类型	描述
ESTONIA_DRIVING_LICENSE	驾驶执照号码（爱沙尼亚专用）。
ESTONIA_PASSPORT_NUMBER	护照号码（爱沙尼亚专用）。
ESTONIA_PERSONAL_IDENTIFICATION_CODE	个人标识号（爱沙尼亚专用）。
ESTONIA_VALUE_ADDED_TAX	增值税（爱沙尼亚专用）。

芬兰数据类型

数据类型	描述
FINLAND_DRIVING_LICENSE	驾驶执照号码（芬兰专用）。
FINLAND_HEALTH_INSURANCE_NUMBER	健康保险号码（芬兰专用）。
FINLAND_NATIONAL_IDENTIFICATION_NUMBER	国家标识号（芬兰专用）。
FINLAND_PASSPORT_NUMBER	护照号码（芬兰专用）。
FINLAND_VALUE_ADDED_TAX	增值税（芬兰专用）。

法国数据类型

数据类型	描述
FRANCE_BANK_ACCOUNT	银行账号（法国专用）。
FRANCE_DRIVING_LICENSE	驾驶执照号码（法国专用）。
FRANCE_HEALTH_INSURANCE_NUMBER	法国健康保险号码。

数据类型	描述
FRANCE_INSEE_CODE	法国社会保障、SSN 或 NIR 号码。
FRANCE_NATIONAL_IDENTIFICATION_NUMBER	法国国民识别码（CNI）。
FRANCE_PASSPORT_NUMBER	护照号码（法国专用）。
FRANCE_TAX_IDENTIFICATION_NUMBER	纳税识别号（法国专用）。
FRANCE_VALUE_ADDED_TAX	增值税（法国专用）。

德国数据类型

数据类型	描述
GERMANY_BANK_ACCOUNT	银行账号（德国专用）。
GERMANY_DRIVING_LICENSE	驾驶执照号码（德国专用）。
GERMANY_PASSPORT_NUMBER	护照号码（德国专用）。
GERMANY_PERSONAL_IDENTIFICATION_NUMBER	个人标识号（德国专用）。
GERMANY_TAX_IDENTIFICATION_NUMBER	税务标识号（德国专用）。
GERMANY_VALUE_ADDED_TAX	增值税（德国专用）。

希腊数据类型

数据类型	描述
GREECE_DRIVING_LICENSE	驾驶执照号码（希腊专用）。
GREECE_PASSPORT_NUMBER	护照号码（希腊专用）。

数据类型	描述
GREECE_SSN	社会保障号码（适用于希腊人）。
GREECE_TAX_IDENTIFICATION_NUMBER	纳税识别号（希腊专用）。
GREECE_VALUE_ADDED_TAX	增值税（希腊专用）。

匈牙利数据类型

数据类型	描述
HUNGARY_DRIVING_LICENSE	驾驶执照号码（匈牙利专用）。
HUNGARY_PASSPORT_NUMBER	护照号码（匈牙利专用）。
HUNGARY_SSN	社会保障号码（适用于匈牙利人）。
HUNGARY_TAX_IDENTIFICATION_NUMBER	纳税识别号（匈牙利专用）。
HUNGARY_VALUE_ADDED_TAX	增值税（匈牙利专用）。

冰岛数据类型

数据类型	描述
ICELAND_NATIONAL_IDENTIFICATION_NUMBER	国民识别码（冰岛专用）。
ICELAND_PASSPORT_NUMBER	护照号码（冰岛专用）。
ICELAND_VALUE_ADDED_TAX	增值税（冰岛专用）。

印度数据类型

数据类型	描述
INDIA_AADHAAR_NUMBER	由印度唯一身份识别管理局颁发的 Aadhaar 识别号。
INDIA_PERMANENT_ACCOUNT_NUMBER	印度永久编号（PAN）。

印度尼西亚数据类型

数据类型	描述
INDONESIA_IDENTITY_CARD_NUMBER	国民识别码（印度尼西亚专用）。

爱尔兰数据类型

数据类型	描述
IRELAND_DRIVING_LICENSE	驾驶执照号码（爱尔兰专用）。
IRELAND_PASSPORT_NUMBER	护照号码（爱尔兰专用）。
IRELAND_PERSONAL_PUBLIC_SERVICE_NUMBER	爱尔兰个人公共服务号码（PPS）。
IRELAND_TAX_IDENTIFICATION_NUMBER	纳税识别号（爱尔兰专用）。
IRELAND_VALUE_ADDED_TAX	增值税（爱尔兰专用）。

以色列数据类型

数据类型	描述
ISRAEL_IDENTIFICATION_NUMBER	国民识别码（以色列专用）。

意大利数据类型

数据类型	描述
ITALY_BANK_ACCOUNT	银行账号（意大利专用）。
ITALY_DRIVING_LICENSE	驾驶执照号码（意大利专用）。
ITALY_FISCAL_CODE	标识号，也称为意大利 Codice Fiscale。
ITALY_PASSPORT_NUMBER	护照号码（意大利专用）。
ITALY_VALUE_ADDED_TAX	增值税（意大利专用）。

日本数据类型

数据类型	描述
JAPAN_BANK_ACCOUNT	日本银行账户。
JAPAN_DRIVING_LICENSE	日本的驾驶执照号码。
JAPAN_MY_NUMBER	日本公民或公司的唯一识别码，用于税务管理、社会保障管理和灾难响应
JAPAN_PASSPORT_NUMBER	日本护照号码。

韩国数据类型

数据类型	描述
KOREA_PASSPORT_NUMBER	护照号码（韩国专用）。
KOREA_RESIDENCE_REGISTRATION_NUMBER_FOR_CITIZENS	韩国居民身份证号码。
KOREA_RESIDENCE_REGISTRATION_NUMBER_FOR_FOREIGNERS	外国人韩国居留登记号码。

拉脱维亚数据类型

数据类型	描述
LATVIA_DRIVING_LICENSE	驾驶执照号码（拉脱维亚专用）。
LATVIA_PASSPORT_NUMBER	护照号码（拉脱维亚专用）。
LATVIA_PERSONAL_IDENTIFICATION_NUMBER	个人标识号（拉脱维亚专用）。
LATVIA_VALUE_ADDED_TAX	增值税（拉脱维亚专用）。

列支敦士登数据类型

数据类型	描述
LIECHTENSTEIN_NATIONAL_IDENTIFICATION_NUMBER	国民识别码（列支敦士登专用）。
LIECHTENSTEIN_PASSPORT_NUMBER	护照号码（列支敦士登专用）。
LIECHTENSTEIN_TAX_IDENTIFICATION_NUMBER	纳税识别号（列支敦士登专用）。

立陶宛数据类型

数据类型	描述
LITHUANIA_DRIVING_LICENSE	驾驶执照号码（立陶宛专用）。
LITHUANIA_PERSONAL_IDENTIFICATION_NUMBER	个人标识号（立陶宛专用）。
LITHUANIA_TAX_IDENTIFICATION_NUMBER	纳税识别号（立陶宛专用）。
LITHUANIA_VALUE_ADDED_TAX	增值税（立陶宛专用）。

卢森堡数据类型

数据类型	描述
LUXEMBOURG_DRIVING_LICENSE	驾驶执照号码（卢森堡专用）。
LUXEMBOURG_NATIONAL_INDIVIDUAL_NUMBER	国民识别码（卢森堡专用）。
LUXEMBOURG_PASSPORT_NUMBER	护照号码（卢森堡专用）。
LUXEMBOURG_TAX_IDENTIFICATION_NUMBER	纳税识别号（卢森堡专用）。
LUXEMBOURG_VALUE_ADDED_TAX	增值税（卢森堡专用）。

马来西亚数据类型

数据类型	描述
MALAYSIA_MYKAD_NUMBER	国民识别码（马来西亚专用）。
MALAYSIA_PASSPORT_NUMBER	护照号码（马来西亚专用）。

马耳他数据类型

数据类型	描述
MALTA_DRIVING_LICENSE	驾驶执照号码（马耳他专用）。
MALTA_NATIONAL_IDENTIFICATION_NUMBER	国民识别码（马耳他专用）。
MALTA_TAX_IDENTIFICATION_NUMBER	纳税识别号（马耳他专用）。
MALTA_VALUE_ADDED_TAX	增值税（马耳他专用）。

墨西哥数据类型

数据类型	描述
MEXICO_CLABE_NUMBER	墨西哥 CLABE（Clave Bancaria Estandarizada）银行账号。
MEXICO_DRIVING_LICENSE	驾驶执照号码（墨西哥专用）。
MEXICO_PASSPORT_NUMBER	护照号码（墨西哥专用）。
MEXICO_TAX_IDENTIFICATION_NUMBER	纳税识别号（墨西哥专用）。
MEXICO_UNIQUE_POPULATION_REGISTRY_CODE	Clave Única de Registro de Población（CURP）墨西哥唯一标识码。

荷兰数据类型

数据类型	描述
NETHERLANDS_CITIZEN_SERVICE_NUMBER	荷兰公民号码（BSN，burgerserviceNummer）。
NETHERLANDS_DRIVING_LICENSE	驾驶执照号码（荷兰专用）。
NETHERLANDS_PASSPORT_NUMBER	护照号码（荷兰专用）。
NETHERLANDS_TAX_IDENTIFICATION_NUMBER	纳税识别号（荷兰专用）。
NETHERLANDS_VALUE_ADDED_TAX	增值税（荷兰专用）。
NETHERLANDS_BANK_ACCOUNT	银行账号（荷兰专用）。

新西兰数据类型

数据类型	描述
NEW_ZEALAND_DRIVING_LICENSE	驾驶执照号码（新西兰专用）。
NEW_ZEALAND_NATIONAL_HEALTH_INDEX_NUMBER	新西兰国民健康指数编号。
NEW_ZEALAND_TAX_IDENTIFICATION_NUMBER	纳税识别号，也称为内陆税务号码（新西兰专用）。

挪威数据类型

数据类型	描述
NORWAY_BIRTH_NUMBER	挪威国民身份证号码。
NORWAY_DRIVING_LICENSE	驾驶执照号码（挪威专用）。
NORWAY_HEALTH_INSURANCE_NUMBER	挪威健康保险号码。
NORWAY_NATIONAL_IDENTIFICATION_NUMBER	国家标识号（挪威专用）。
NORWAY_VALUE_ADDED_TAX	增值税（挪威专用）。

菲律宾数据类型

数据类型	描述
PHILIPPINES_DRIVING_LICENSE	驾驶执照号码（菲律宾专用）。
PHILIPPINES_PASSPORT_NUMBER	护照号码（菲律宾专用）。

波兰数据类型

数据类型	描述
POLAND_DRIVING_LICENSE	驾驶执照号码（波兰专用）。
POLAND_IDENTIFICATION_NUMBER	波兰标识符。
POLAND_PASSPORT_NUMBER	护照号码（波兰专用）。
POLAND_REGON_NUMBER	REGON 标识号，也称为统计标识号。
POLAND_SSN	社会保障号码（适用于波兰人）。
POLAND_TAX_IDENTIFICATION_NUMBER	纳税识别号（波兰专用）。
POLAND_VALUE_ADDED_TAX	增值税（波兰专用）。

葡萄牙数据类型

数据类型	描述
PORTUGAL_DRIVING_LICENSE	驾驶执照号码（葡萄牙专用）。
PORTUGAL_NATIONAL_IDENTIFICATION_NUMBER	国家标识号（葡萄牙专用）。
PORTUGAL_PASSPORT_NUMBER	护照号码（葡萄牙专用）。
PORTUGAL_TAX_IDENTIFICATION_NUMBER	纳税识别号（葡萄牙专用）。
PORTUGAL_VALUE_ADDED_TAX	增值税（葡萄牙专用）。

罗马尼亚数据类型

数据类型	描述
ROMANIA_DRIVING_LICENSE	驾驶执照号码（罗马尼亚专用）。
ROMANIA_NUMERICAL_PERSONAL_CODE	个人标识号（罗马尼亚专用）。

数据类型	描述
ROMANIA_PASSPORT_NUMBER	护照号码（罗马尼亚专用）。
ROMANIA_VALUE_ADDED_TAX	增值税（罗马尼亚专用）。

新加坡数据类型

数据类型	描述
SINGAPORE_DRIVING_LICENSE	驾驶执照号码（新加坡专用）。
SINGAPORE_NATIONAL_REGISTRY_IDENTIFICATION_NUMBER	新加坡国家注册标识卡。
SINGAPORE_PASSPORT_NUMBER	护照号码（新加坡专用）。
SINGAPORE_UNIQUE_ENTITY_NUMBER	新加坡唯一实体编号。

斯洛伐克数据类型

数据类型	描述
SLOVAKIA_DRIVING_LICENSE	驾驶执照号码（斯洛伐克专用）。
SLOVAKIA_NATIONAL_IDENTIFICATION_NUMBER	国家标识号（斯洛伐克专用）。
SLOVAKIA_PASSPORT_NUMBER	护照号码（斯洛伐克专用）。
SLOVAKIA_VALUE_ADDED_TAX	增值税（斯洛伐克专用）。

斯洛文尼亚数据类型

数据类型	描述
SLOVENIA_DRIVING_LICENSE	驾驶执照号码（斯洛文尼亚专用）。

数据类型	描述
SLOVENIA_PASSPORT_NUMBER	护照号码（斯洛文尼亚专用）。
SLOVENIA_TAX_IDENTIFICATION_NUMBER	纳税识别号（斯洛文尼亚专用）。
SLOVENIA_UNIQUE_MASTER_CITIZEN_NUMBER	斯洛文尼亚公民的唯一主公民号码（JMBG）。
SLOVENIA_VALUE_ADDED_TAX	增值税（斯洛文尼亚专用）。

南非数据类型

数据类型	描述
SOUTH_AFRICA_PERSONAL_IDENTIFICATION_NUMBER	个人标识号（南非专用）。

西班牙数据类型

数据类型	描述
SPAIN_BANK_ACCOUNT	银行账号（西班牙专用）。
SPAIN_DNI	西班牙国民身份证（Documento Nacional de Identidad）。
SPAIN_DRIVING_LICENSE	驾驶执照号码（西班牙专用）。
SPAIN_NIE	外国人身份号码（西班牙专用），也称为 NIE。
SPAIN_NIF	纳税识别号（西班牙专用），也称为 NIF。
SPAIN_PASSPORT_NUMBER	护照号码（西班牙专用）。
SPAIN_SSN	社会保障号码（适用于西班牙人）。
SPAIN_VALUE_ADDED_TAX	增值税（西班牙专用）。

斯里兰卡数据类型

数据类型	描述
SRI_LANKA_NATIONAL_IDENTIFICATION_NUMBER	国民识别码（斯里兰卡专用）。

瑞典数据类型

数据类型	描述
SWEDEN_DRIVING_LICENSE	驾驶执照号码（瑞典专用）。
SWEDEN_PASSPORT_NUMBER	护照号码（瑞典专用）。
SWEDEN_PERSONAL_IDENTIFICATION_NUMBER	国家标识号（瑞典专用）。
SWEDEN_TAX_IDENTIFICATION_NUMBER	瑞典纳税识别号（人员编号）。
SWEDEN_VALUE_ADDED_TAX	增值税（瑞典专用）。

瑞士数据类型

数据类型	描述
SWITZERLAND_AHV	瑞士人的社会保障号码（AHV）。
SWITZERLAND_HEALTH_INSURANCE_NUMBER	瑞士健康保险号码。
SWITZERLAND_PASSPORT_NUMBER	护照号码（瑞士专用）。
SWITZERLAND_VALUE_ADDED_TAX	增值税（瑞士专用）。

泰国数据类型

数据类型	描述
THAILAND_PASSPORT_NUMBER	护照号码（泰国专用）。
THAILAND_PERSONAL_IDENTIFICATION_NUMBER	个人标识号（泰国专用）。

土耳其数据类型

数据类型	描述
TURKEY_NATIONAL_IDENTIFICATION_NUMBER	国家标识号（土耳其专用）。
TURKEY_PASSPORT_NUMBER	护照号码（土耳其专用）。
TURKEY_VALUE_ADDED_TAX	增值税（土耳其专用）。

乌克兰数据类型

数据类型	描述
UKRAINE_INDIVIDUAL_IDENTIFICATION_NUMBER	唯一标识符（乌克兰专用）。
UKRAINE_PASSPORT_NUMBER_DOMESTIC	国内护照号码（乌克兰专用）。
UKRAINE_PASSPORT_NUMBER_INTERNATIONAL	国际护照号码（乌克兰专用）。

阿拉伯联合酋长国（UAE）数据类型

数据类型	描述
UNITED_ARAB_EMIRATES_PERSONAL_NUMBER	个人标识号（UAE 专用）。

英国数据类型

数据类型	描述
UK_BANK_ACCOUNT	英国（UK）银行账户。
UK_BANK_SORT_CODE	英国（UK）银行分类代码。银行识别代码是用于通过各自的清算组织在各自国家或地区的银行之间进行汇款的银行代码
UK_DRIVING_LICENSE	大不列颠及北爱尔兰联合王国的驾照号码（英国专用）
UK_ELECTORAL_ROLL_NUMBER	选民名册编号 (ERN) 是发给个人进行英国选举登记的识别号。该号码的格式由英国内阁办公室的《英国政府标准》规定
UK_NATIONAL_HEALTH_SERVICE_NUMBER	国家健康服务（NHS）号码是分配给英国公共卫生服务注册用户的唯一号码
UK_NATIONAL_INSURANCE_NUMBER	国民保险号码 (NINO) 是英国 (UK) 用于针对国民保险计划或社会保障体系识别个人的号码。该号码有时被称为 NI No 或 NINO。
UK_PASSPORT_NUMBER	英国（UK）护照号码。
UK_UNIQUE_TAXPAYER_REFERENCE_NUMBER	英国 (UK) 唯一纳税人参考号 (UTR)。英国政府用来管理税收系统的识别码
UK_VALUE_ADDED_TAX	增值税是由最终消费者承担的消费税。制造和分销过程中的每笔交易都要支付增值税。对于英国，增值税编号由企业所在地区的增值税办公室发布
UK_PHONE_NUMBER	英国（UK）电话号码。

委内瑞拉数据类型

数据类型	描述
VENEZUELA_DRIVING_LICENSE	驾驶执照号码（委内瑞拉专用）。
VENEZUELA_NATIONAL_IDENTIFICATION_NUMBER	国家标识号（委内瑞拉专用）。
VENEZUELA_VALUE_ADDED_TAX	增值税（委内瑞拉专用）。

使用精细敏感数据检测

> **Note**
>
> 精细操作功能仅在 AWS Glue 3.0 和 4.0 中可用。这包括 AWS Glue Studio 体验。此外，2.0 版本也不支持持久审计日志更改。
> 所有 AWS Glue Studio 3.0 和 4.0 可视化作业都将创建一个会自动使用精细操作 API 的脚本。

借助检测敏感数据转换功能，可以检测、遮蔽或移除您定义的或由 AWS Glue 预定义的实体。您还可以借助精细操作对每个实体应用特定的操作。其他优点包括：

- 可在检测到数据后立即应用操作，从而提高性能。
- 提供了包含或排除特定列的选项。
- 能够使用部分遮蔽功能。从而让您可以部分遮蔽检测到的敏感数据实体，而不是遮蔽整个字符串。支持带有偏移量的简单参数和正则表达式。

以下是敏感数据检测 API 代码片段和下一节中引用的示例作业中使用的精细操作。

检测 API – 精细操作将使用新的 `detectionParameters` 参数：

```
def detect(
    frame: DynamicFrame,
    detectionParameters: JsonOptions,
    outputColumnName: String = "DetectedEntities",
    detectionSensitivity: String = "LOW"
): DynamicFrame = {}
```

将敏感数据检测 API 与精细操作结合使用

敏感数据检测 API 使用 detect 来分析给定的数据，确定行或列是否属于敏感数据实体类型，并且将运行用户为每种实体类型指定的操作。

将 detect API 与精细操作结合使用

使用 detect API 并指定 `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()
      }
```

上面的脚本将从 Amazon S3 中的某个位置创建一个 DataFrame，然后运行 `detect` API。由于 `detect` API 要求字段 `detectionParameters`（实体名称与将用于该实体的所有操作设置列表的映射）由 AWS Glue 的 `JsonOptions` 对象来表示，因此还有利于我们扩展该 API 的功能。

对于为每个实体指定的每个操作，输入要应用该实体/操作组合的所有列名的列表。这让您能够为数据集中的每一列自定义要检测的实体，并跳过您知道特定列中并未包含的实体。此外，这还让您能够不对这些实体执行不必要的检测调用，从而提高作业性能，并且能够为每个列和实体组合所特有的操作。

如果更深入看 detectionParameters，示例作业中有三种实体类型，分别是 Phone Number、USA_PASSPORT_NUMBER 和 USA_DRIVING_LICENSE。AWS Glue 将针对每种实体类型运行不同的操作，分别是 PARTIAL_REDACT、SHA256_HASH、REDACT 和 DETECT。每种实体类型也必须拥有要应用到的 sourceColumns 和/或 sourceColumnsToExclude（如果检测到）。

> **Note**
>
> 每列只能使用一个就地编辑操作（PARTIAL_REDACT、SHA256_HASH 或 REDACT），但 DETECT 操作可以与这些操作中的任何一个结合使用。

detectionParameters 字段的布局如下：

```
    ENTITY_NAME -> List[Actions]
    {
     "ENTITY_NAME": [{
      Action, // required
      ColumnSpecs,
      ActionOptionsMap
        }],
        "ENTITY_NAME2": [{
      ...
        }]
    }
```

actions 和 actionOptions 的类型列举如下：

```
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"
    ]
}
```

脚本运行后，结果将输出到给定的 Amazon S3 位置。您可以在 Amazon S3 中查看数据，但对于选定的实体类型，将根据所选操作进行敏感化处理。在此例中，结果行将如下所示：

```
{
    "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"
}
```

上面的脚本对 Phone Number 使用 # 进行了部分编辑。Passport No 已更改为 SHA256 哈希值。检测到 DL NO# 属于美国驾驶执照号码，并如 detectionParameters 中所述编辑为“USA_DL”。

> **Note**
>
> 由于 classifyColumns API 的性质，无法与精细操作结合使用。此 API 会执行列采样（可由用户调整，不过有默认值）来提高检测速度。由于这一原因，精细操作将需要迭代每个值。

持久审计日志

随精细操作引入的一项新功能（但在使用普通 API 时也可用）是持久审计日志。目前，运行 detect API 会添加一个带有 PII 检测元数据的附加列（默认为 DetectedEntities，但可通过 outputColumnName 进行自定义）参数。现在推出了“actionUsed”元数据键，可以是 DETECT、PARTIAL_REDACT、SHA256_HASH 或 REDACT。

```
"DetectedEntities": {
    "Credit Card Number": [
        {
            "entityType": "CREDIT_CARD",
            "actionUsed": "DETECT",
            "start": 0,
            "end": 19
        }
    ],
    "Phone Number": [
        {
            "entityType": "PHONE_NUMBER",
            "actionUsed": "REDACT",
            "start": 0,
            "end": 14
        }
    ]
}
```

即使客户使用不支持精细操作（例如 `detect(entityTypesToDetect, outputColumnName)`）的 API，也会在生成的数据帧中看到此持久审计日志。

如果客户使用支持精细操作的 API，则将看到所有操作，无论是否经过编辑。例如：

```
+--------------------+---------------
+-------------------------------------------------------------------------------------------------------
+
| 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}]}} |
+--------------------+---------------
+-------------------------------------------------------------------------------------------------------
+
```

如果您不想看到 DetectedEntities 列，则只需在自定义脚本中删除该附加列即可。

AWS Glue 可视化任务 API

AWS Glue 提供了一个 API，允许客户使用 AWS Glue API 从表示可视化步骤工作流的 JSON 对象创建数据集成任务。然后，客户可以使用 AWS Glue Studio 中的可视化编辑器来处理这些任务。

有关可视化任务 API 数据类型的更多信息，请参阅可视化任务 API。

主题

- API 设计和 CRUD API
- 开始使用
- 可视化任务限制

API 设计和 CRUD API

CreateJob 和 UpdateJob API 现在支持额外的可选参数 codeGenConfigurationNodes。若为此字段提供非空 JSON 结构，将导致为创建的任务在 AWS Glue Studio 中注册 DAG，并生成关联代码。任务创建时此字段的空值或空字符串将被忽略。

codeGenConfigurationNodes 字段的更新可通过 UpdateJob AWS Glue API 以类似于 CreateJob 的方式完成。整个字段应在 UpdateJob 中指定，其中 DAG 已根据需要更改。提供的空值将被忽略，并且不会执行 DAG 更新。空结构或字符串将导致 codeGenConfigurationNodes 设置为空，并删除以前的任何 DAG。GetJob API 将返回 DAG（如果存在）。DeleteJob API 还会删除任何关联的 DAG。

开始使用

要创建任务，请使用 CreateJob 操作。CreateJob 请求输入包含一个额外的字段“codeGenConfigurationNodes”，您可以从中获取并指定 JSON 中的 DAG 对象。

请记住以下事项：

- “codeGenConfigurationNodes”字段是 nodeId 到节点的映射。
- 每个节点都以一个标识其节点类型的键开头。
- 由于节点只能是一种类型，因此只能指定一个键。
- 输入字段包含当前节点的父节点。

以下是 CreateJob 输入的 JSON 表示。

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

更新和获取任务

由于 UpdateJob 也包含“codeGenConfigurationNodes”字段，因此输入格式相同。请参阅 UpdateJob 操作。

GetJob 操作也将以相同的格式返回“codeGenConfigurationNodes”字段。请参阅 GetJob 操作。

可视化任务限制

由于“codeGenConfigurationNodes”参数已添加到现有 API 中，因此将继承这些 API 中的任何限制。此外，codeGenConfigurationNodes 和某些节点的大小也将受到限制。有关更多信息，请参阅任务结构。

Ray 脚本编程

AWS Glue 使编写和运行 Ray 脚本变得容易。本节介绍了 AWS Glue for Ray 中支持的 Ray 功能。在 Python 中编写 Ray 脚本

您的自定义脚本必须与作业定义中的 `Runtime` 字段定义的 Ray 版本兼容。有关 Jobs API 中 `Runtime` 的更多信息，请参阅 the section called “作业”。有关每个运行时环境的信息，请参阅 the section called “支持的 Ray 运行时环境”。

主题

- 教程：在 AWS Glue for Ray 中编写 ETL 脚本
- 在 AWS Glue for Ray 中使用 Ray Core 和 Ray Data
- 为 Ray 作业提供文件和 Python 库
- 连接 Ray 作业中的数据

教程：在 AWS Glue for Ray 中编写 ETL 脚本

Ray 让您能够在 Python 中以原生方式编写和扩展分布式任务。AWS Glue for Ray 提供无服务器 Ray 环境，您可以从作业和交互式会话中访问这些环境（Ray 交互式会话以预览形式提供）。AWS Glue 作业系统提供了一种一致的方式来管理和运行任务，可以按计划、通过触发器或 AWS Glue 控制台执行任务。

结合使用这些 AWS Glue 工具可创建功能强大的工具链，可用于提取、转换、加载（ETL）工作负载，这是 AWS Glue 的一种受欢迎的用例。在本教程中，您将学习组合此解决方案的基础知识。

我们还支持将 for Spark AWS Glue 用于您的 ETL 工作负载。有关编写 AWS Glue for Spark 脚本的教程，请参阅 the section called “教程：编写 Spark 脚本”。有关可用引擎的更多信息，请参阅 the section called “AWS Glue for Spark 和 AWS Glue for Ray”。Ray 能够处理分析、机器学习（ML）和应用程序开发中的许多不同类型的任务。

在本教程中，您将提取、转换和加载一个托管在 Amazon Simple Storage Service（Amazon S3）中的 CSV 数据集。您将首先使用纽约市出租车和豪华轿车委员会（TLC）行程记录数据集，该数据集存储在一个公开的 Amazon S3 存储桶中。有关该数据集的更多信息，请参阅 Registry of Open Data on AWS。

您将使用 Ray Data 库中提供的预定义转换来转换数据。Ray Data 是一个由 Ray 设计的数据集准备库，默认包含在 AWS Glue for Ray 环境中。有关默认情况下包含库的更多信息，请参阅 the section called “Ray 作业提供的模块”。然后，将转换后的数据写入由您控制的 Amazon S3 存储桶。

先决条件 - 在本教程中，您需要一个有权访问 AWS Glue 和 Amazon S3 的 AWS 账户。

第 1 步：在 Amazon S3 中创建一个存储桶来保存您的输出数据

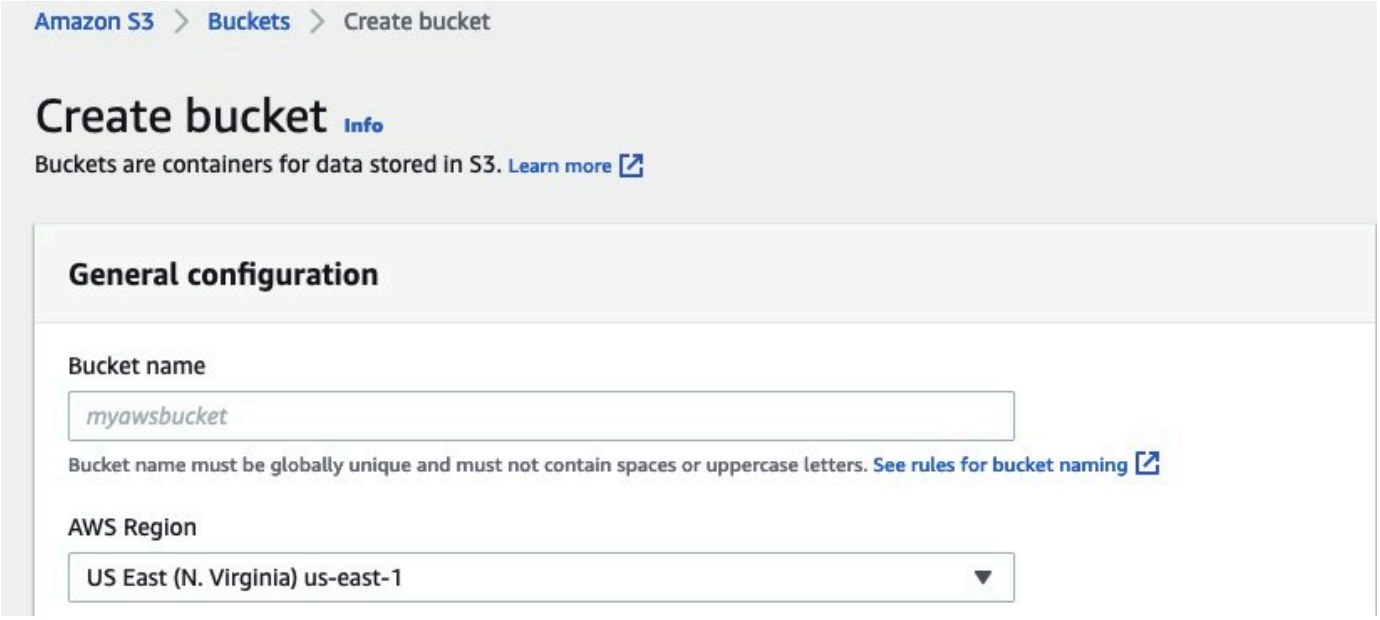
您需要一个由您控制的 Amazon S3 存储桶，以用作本教程中创建的数据的接收器。您可以按照以下步骤创建此存储桶。

> **Note**
>
> 如果您想将数据写入由您控制的现有存储桶，则可以跳过此步骤。记下 *yourBucketName*（现有存储桶的名称），以便在后续步骤中使用。

为 Ray 作业输出创建存储桶

- 按照《Amazon S3 用户指南》中创建存储桶中的步骤创建存储桶。
  - 选择存储桶名称时，请记下 *yourBucketName*，您将在后续步骤中参考该名称。
  - 对于其他配置，Amazon S3 控制台中提供的建议设置在本教程中应该可以正常工作。

  例如，Amazon S3 控制台中的存储桶创建对话框可能如下所示。



第 2 步：为 Ray 作业创建 IAM 角色

您的作业将需要一个具有以下内容的 AWS Identity and Access Management（IAM）角色：

- `AWSGlueServiceRole` 托管式策略授予的权限。这些是运行 AWS Glue 作业所需的基本权限。
- 针对 `nyc-tlc/*` Amazon S3 资源的 `Read` 访问级别权限。
- 针对 *`yourBucketName`*`/*` Amazon S3 资源的 `Write` 访问级别权限。
- 一种允许 `glue.amazonaws.com` 主体担任角色的信任关系。

您可以按照以下步骤创建此角色。

为您的 AWS Glue for Ray 作业创建 IAM 角色

> **Note**
>
> 您可以按照许多不同的步骤创建 IAM 角色。有关如何预置 IAM 资源的更多信息或选项，请参阅 AWS Identity and Access Management 文档。

1. 按照《IAM 用户指南》中使用可视化编辑器创建 IAM 策略（控制台）中的步骤，创建定义前面概述的 Amazon S3 权限的策略。

   - 选择服务时，请选择 Amazon S3。

- 为策略选择权限时，请为以下资源（如前所述）附加以下几组操作：
  - 针对 nyc-tlc/* Amazon S3 资源的读取访问级别权限。
  - 针对 *yourBucketName*/* Amazon S3 资源的写入访问级别权限。
- 选择策略名称时，请记下 *YourPolicyName*，您将在后续步骤中参考该名称。

2. 按照《IAM 用户指南》中为 AWS 服务（控制台）创建角色中的步骤，为您的 AWS Glue for Ray 作业创建角色。

- 选择可信 AWS 服务实体，请选择 Glue。这将自动填充您的作业所需的信任关系。
- 为权限策略选择策略时，请附加以下策略：
  - AWSGlueServiceRole
  - *YourPolicyName*
- 选择角色名称时，请记下 *YourRoleName*，您将在后续步骤中参考该名称。

第 3 步：创建并运行 AWS Glue for Ray 作业

在此步骤中，您将使用 AWS Management Console 创建 AWS Glue 作业，为其提供示例脚本，然后运行该作业。创建作业时，它会在控制台中创建一个位置供您存储、配置和编辑 Ray 脚本。有关创建任务的信息，请参阅 the section called “登录到控制台”。

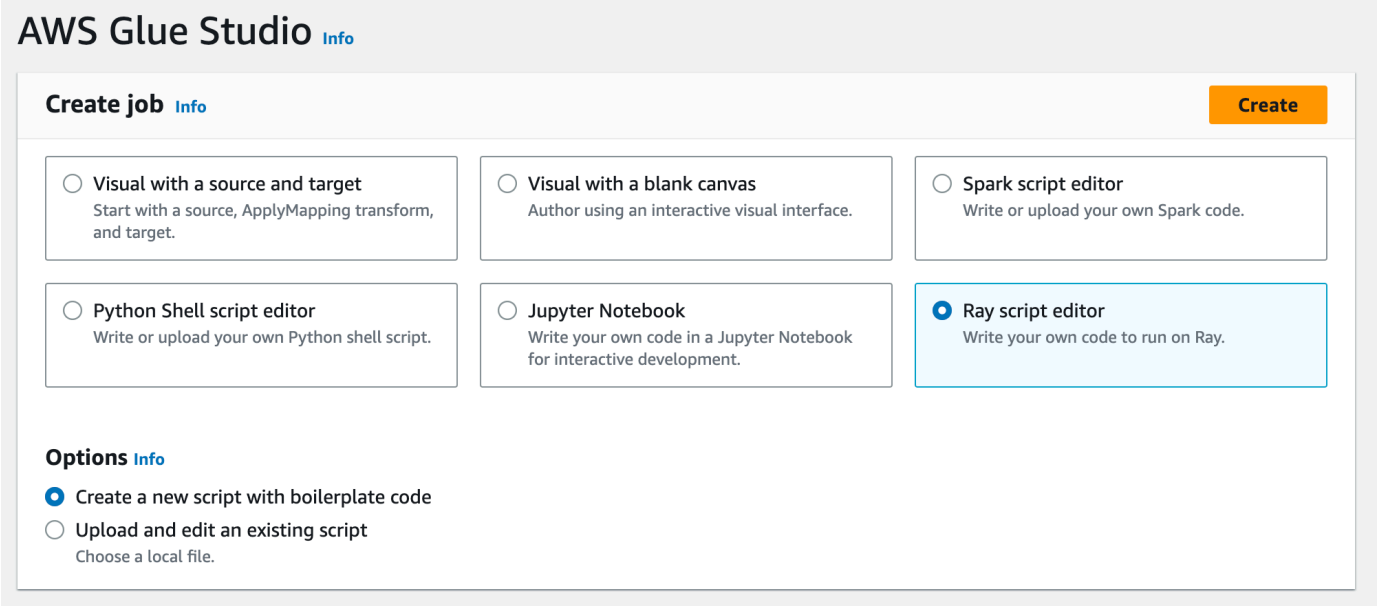
本教程将介绍以下 ETL 场景：您需要从纽约市 TLC行程记录数据集中读取 2022 年 1 月的记录，通过合并现有列中的数据向数据集添加一个新列（tip_rate），然后移除某些当前分析无关的列，最后将结果写入 *yourBucketName*。以下 Ray 脚本执行以下步骤：

```
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/")
```

创建并运行 AWS Glue for Ray 作业

1. 在 AWS Management Console 中，导航到 AWS Glue 登录页面。
2. 在侧面的导航窗格中，选择 ETL 作业。
3. 在创建作业中，选择 Ray 脚本编辑器，然后选择创建，如下图所示。



4. 将脚本的全文粘贴到脚本窗格中，然后替换所有现有文本。
5. 导航到任务详细信息，然后将 IAM 角色属性设置为 *YourRoleName*。
6. 选择保存，然后选择运行。

第 4 步：检查输出

运行 AWS Glue 作业后，您应验证输出是否符合此场景的预期。可以使用以下步骤来做到这一点。

验证您的 Ray 作业是否成功运行

1. 在作业详细信息页面上，导航到运行。
2. 几分钟后，您应该会看到运行状态为成功的运行。

3. 通过 https://console.aws.amazon.com/s3/ 导航到 Amazon S3 控制台，然后检查 *yourBucketName*。您应该会看到写入输出存储桶的文件。
4. 读取 Parquet 文件并验证其内容。您可以用您现有的工具来做到这一点。如果您没有验证 Parquet 文件的流程，则可以在 AWS Glue 控制台中使用 Spark 或 Ray（预览版）通过 AWS Glue 交互式会话完成此操作。

   在交互式会话中，您可以访问 Ray Data、Spark 或 Pandas 库，这些库是默认提供的（取决于您选择的引擎）。要验证文件内容，您可以使用这些库中可用的常用检查方法，例如 `count`、`schema` 和 `show`。有关控制台中交互式会话的更多信息，请参阅 Using notebooks with AWS Glue Studio and AWS Glue。

   由于您已确认文件已写入存储桶，因此可以相对肯定地说，如果您的输出有问题，则它们与 IAM 配置无关。使用 *yourRoleName* 配置您的会话以访问相关文件。

如果您看不到预期结果，请查看本指南中的故障排除内容，找出并修复错误的根源。要解释作业运行错误状态，请参阅 the section called “作业运行状态”。您可以在 故障排除 AWS Glue 章节中找到故障排除的内容。有关与 Ray 作业相关的特定错误，请参阅故障排除章节中的 the section called “Ray 错误故障排除”。

后续步骤

现在，您已经看到并执行了使用 AWS Glue for Ray 的端到端的 ETL 过程。您可以使用以下资源来了解 AWS Glue for Ray 提供了哪些工具来大规模转换和解释数据。

- 有关 Ray 任务模型的更多信息，请参阅 the section called “在 AWS Glue for Ray 中使用 Ray Core 和 Ray Data”。要获得使用 Ray 任务的更多体验，请按照 Ray Core 文档中的示例进行操作。请参阅 Ray 文档中的 Ray Core: Ray Tutorials and Examples (2.4.0)。
- 有关 AWS Glue for Ray 中可用数据管理库的指南，请参阅 the section called “连接到数据”。要获得使用 Ray Data 转换和写入数据集的更多经验，请按照 Ray Data 文档中的示例进行操作。请参阅 Ray Data: Examples (2.4.0)。
- 有关配置 AWS Glue for Ray 作业的更多信息，请查阅 the section called “处理 Ray 作业”。
- 有关编写 AWS Glue for Ray 脚本的更多信息，请继续阅读文档的本节。

在 AWS Glue for Ray 中使用 Ray Core 和 Ray Data

Ray 是一个框架，用于通过在集群中分配工作来纵向扩展 Python 脚本。您可以使用 Ray 来解决多种问题，因此 Ray 提供了用于优化某些任务的库。在 AWS Glue 中，我们专注于使用 Ray 来转换大型数据集。AWS Glue 为 Ray Data 和 Ray Core 的某些部分提供支持，以简化此任务。

什么是 Ray Core？

构建分布式应用程序的第一步是识别和定义可以同时执行的工作。Ray Core 包含 Ray 中用来定义可以同时执行的任务的部分。Ray 提供了参考和快速入门信息，您可以使用这些信息来学习他们提供的工具。有关更多信息，请参阅 What is Ray Core? 以及 Ray Core Quick Start。有关在 Ray 中有效定义并发任务的更多信息，请参阅 Tips for first-time users。

> **Ray 任务和操作者**
>
> 在 AWS Glue for Ray 文档中，我们可能会提及任务和操作者，它们是 Ray 中的核心概念。Ray 使用 Python 函数和类作为分布式计算系统的构建基块。就像 Python 函数和变量在类中使用时变成“方法”和“属性”一样，在 Ray 中使用函数向工作线程发送代码时，函数变成“任务”，类变成“操作者”。您可以通过 `@ray.remote` 注解来识别 Ray 可能使用的函数和类。
> 任务和操作者是可配置的，它们具有生命周期，并且在整个生命周期中都会占用计算资源。当您找到问题的根本原因时，会引发错误的代码可以追溯到任务或操作者。因此，当您学习如何配置、监控或调试 AWS Glue for Ray 作业时，可能会出现这些术语。
> 要开始学习如何有效地使用任务和操作者来构建分布式应用程序，请参阅 Ray 文档中的 Key Concepts。

AWS Glue for Ray 中的 Ray Core

AWS Glue for Ray 环境管理集群的形成和扩展，以及收集和可视化日志。由于我们管理这些问题，因此我们限制了对 Ray Core 中用于解决开源集群中这些问题的 API 的访问和支持。

在托管式 `Ray2.4` 运行时环境中，我们不支持：

- Ray Core CLI
- Ray State CLI
- `ray.util.metrics` Prometheus 指标实用方法：
  - 计数器
  - 计量表

  - 直方图
- 其他调试工具：
  - ray.util.pdb.set_trace
  - ray.util.inspect_serializability
  - ray.timeline

什么是 Ray Data？

当您连接到数据来源和目标、处理数据集和启动常见转换时，Ray Data 是一种使用 Ray 来解决 Ray 数据集转换问题的简单方法。有关使用 Ray Data 的更多信息，请参见 Ray 数据集：分布式数据预处理。

您可以使用 Ray Data 或其他工具来访问您的数据。有关在 Ray 中访问数据的更多信息，请参阅 the section called “连接到数据”。

AWS Glue for Ray 中的 Ray Data

默认情况下，托管式 `Ray2.4` 运行时环境支持和提供 Ray Data。有关提供的模块的更多信息，请参阅 the section called “Ray 作业提供的模块”。

为 Ray 作业提供文件和 Python 库

本节提供了在 AWS Glue Ray 作业中使用 Python 库所需的信息。您可以使用所有 Ray 作业中默认包含的某些公共库。也可以为 Ray 作业提供您自己的 Python 库。

Ray 作业提供的模块

您可以使用以下提供的包在 Ray 作业中执行数据集成工作流程。默认情况下，这些包在 Ray 作业中可用。

AWS Glue version 4.0

在 AWS Glue 4.0 中，Ray（`Ray2.4` 运行时系统）环境提供以下软件包：

- boto3 == 1.26.133
- ray == 2.4.0
- pyarrow == 11.0.0
- pandas==1.5.3

- numpy == 1.24.3
- fsspec == 2023.4.0

此列表包括所有将与 `ray[data] == 2.4.0` 一起安装的软件包。开箱即用支持 Ray Data。

为您的 Ray 作业提供文件

您可以使用 `--working-dir` 参数为 Ray 作业提供文件。为此参数提供指向 Amazon S3 上托管的.zip 文件的路径。在 .zip 文件中，您的文件必须包含在单个顶级目录中。任何其他文件都不应位于顶层。

在脚本开始运行之前，您的文件将分发到每个 Ray 节点。考虑一下这会如何影响每个 Ray 节点的可用磁盘空间。可用磁盘空间由作业配置中设置的 WorkerType 决定。如果您想大规模提供作业数据，那么这种机制不是合适的解决方案。有关为作业系统数据的更多信息，请参阅 the section called “连接到数据”。

可以访问您的文件，就像通过 `working_dir` 参数向 Ray 提供目录一样。例如，要读取 .zip 文件顶级目录中名为 `sample.txt` 的文件，可以调用：

```
@ray.remote
def do_work():
    f = open("sample.txt", "r")
    print(f.read())
```

有关 `working_dir` 的详细信息，请参阅 Ray 文档。此功能的行为与 Ray 的原生功能类似。

用于 Ray 作业的其他 Python 模块

来自 PyPI 的其他模块

Ray 作业使用 Python Package Installer (pip3) 安装 Ray 脚本使用的其他模块。您可以将 `--pip-install` 参数与逗号分隔的 Python 模块列表结合使用，以添加新模块或更改现有模块的版本。

例如，要更新或添加新的 `scikit-learn` 模块，请使用以下键-值对：

`"--pip-install", "scikit-learn==0.21.3"`

如果您有自定义模块或自定义补丁，则可以使用 `--s3-py-modules` 参数从 Amazon S3 分发自己的库。在上传发行版之前，您的发行版可能需要重新打包和重新构建。遵循 the section called “在 Ray 作业中包含 Python 代码” 中的准则。

来自 Amazon S3 的自定义分配

自定义发行版应符合 Ray 依赖项打包要求。您可以在下一节中了解如何构建这些发行版。有关 Ray 如何设置依赖项的更多信息，请参阅 Ray 文档中的 Environment Dependencies（环境依赖项）。

要在评估其内容后添加自定义可分发文件，请将您的可分发文件上传到作业的 IAM 角色可用的存储桶。在参数配置中指定 Python ZIP 存档的 Amazon S3 路径。如果您要提供多个可分发文件，请用逗号分隔它们。例如：

```
"--s3-py-modules", "s3://s3bucket/pythonPackage.zip"
```

限制

Ray 作业不支持在作业环境中编译本机代码。如果您的 Python 依赖项在传递期间依赖于本机编译的代码，则可能会受到此限制。Ray 作业可以运行提供的二进制文件，但必须已针对基于 ARM64 的 Linux 进行编译。这意味着您可以使用 `aarch64manylinux` 仪表盘里的内容。通过将轮子重新打包到 Ray 标准，您可以以编译后的形式提供原生依赖项。通常这意味着删除 `dist-info` 文件夹，从而使存档的根目录中只有一个文件夹。

您无法使用此参数升级 `ray` 或 `ray[data]` 的版本。要使用新版本的 Ray，您需要在我们发布对作业的支持后更改运行时字段。有关支持的 Ray 版本的更多信息，请参阅 the section called “AWS Glue 版本”。

在 Ray 作业中包含 Python 代码

Python 软件基金会为打包 Python 文件以供在不同的运行时使用提供了标准化的行为。请注意，Ray 引入了打包标准的限制。除对 Ray 规定打包标准外，AWS Glue 未指定其他打包标准。以下说明提供了有关打包简单 Python 包的标准指导。

将您的文件打包为 `.zip` 存档。目录应位于存档的根目录处。存档的根级别上应该没有其他文件，否则，这可能会导致意外行为。根目录是软件包，其名称用于在导入时引用您的 Python 代码。

如果您使用 `--s3-py-modules` 将这种形式的分配提供给 Ray 作业，则应该能够在 Ray 脚本中从软件包中导入 Python 代码。

您的软件包可以为单个 Python 模块提供一些 Python 文件，也可以将许多模块打包在一起。重新打包依赖关系（例如 PyPI 中的库）时，请检查这些软件包中是否有隐藏文件和元数据目录。

> **Warning**
>
> 某些操作系统行为使得正确遵循这些打包说明变得困难。
>
> - OSX 可能会将隐藏文件（例如 `__MACOSX` 添加到顶层的 Zip 文件文件中）。

- Windows 可能会自动将您的文件添加到 Zip 内的文件夹，从而无意中创建了一个嵌套文件夹。

以下过程假设您正在 Amazon Linux 2 或提供 Info-ZIP `zip` 分发版和 `zipinfo` 实用程序的类似操作系统中与文件进行交互。我们建议使用这些工具来防止意外行为。

打包 Python 文件以在 Ray 中使用

1. 使用您的软件包名称创建一个临时目录，然后确认您的工作目录是其父目录。您可使用以下命令执行此操作：

```
cd parent_directory
mkdir temp_dir
```

2. 将您的文件复制到临时目录中，然后确认您的目录结构。该目录的内容将作为您的 Python 模块直接访问。您可使用以下 命令执行此操作：

```
ls -AR temp_dir
# my_file_1.py
# my_file_2.py
```

3. 使用 Zip 压缩您的临时文件夹。您可使用以下命令执行此操作：

```
zip -r zip_file.zip temp_dir
```

4. 确认您的文件已正确打包。*zip_file*.zip 现在应该可以在您的工作目录中找到。您可以使用以下命令检查：

```
zipinfo -1 zip_file.zip
# temp_dir/
# temp_dir/my_file_1.py
# temp_dir/my_file_2.py
```

重新打包一个 Python 软件包以便在 Ray 中使用。

1. 使用您的软件包名称创建一个临时目录，然后确认您的工作目录是其父目录。您可使用以下命令执行此操作：

```
cd parent_directory
mkdir temp_dir
```

2. 解压缩您的软件包并将内容复制到您的临时目录中。移除与之前的打包标准相关的文件，只保留模块的内容。使用以下命令确认您的文件结构看起来正确：

```
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 压缩您的临时文件夹。您可使用以下命令执行此操作：

```
zip -r zip_file.zip temp_dir
```

4. 确认您的文件已正确打包。*zip_file*.zip 现在应该可以在您的工作目录中找到。您可以使用以下命令检查：

```
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
```

连接 Ray 作业中的数据

AWS Glue Ray 作业可以使用各种 Python 包，这些包专为快速集成数据而设计。我们提供了一组最少的依赖关系，以免您的环境混乱。有关默认情况下包含内容的更多信息，请参阅 the section called “Ray 作业提供的模块”。

> **Note**
>
> AWS Glue 提取、转换、加载（ETL）提供 DynamicFrame 抽象化来简化 ETL 工作流程，在这些工作流程中，您可以解决数据集各行之间的架构差异。AWS GlueETL 还提供其他功能，例如作业书签和分组输入文件。我们目前不在 Ray 作业中提供相应的功能。
> AWS Glue for Spark 直接支持连接到某些数据格式、源和接收器。在 Ray 中，适用于 Pandas 的 AWS 开发工具包和最新的第三方库在很大程度上满足了这一需求。您需要查阅这些库，以了解有哪些功能可用。

AWS Glue for Ray 目前无法与 Amazon VPC 集成。如果没有公共路由，就无法访问 Amazon VPC 中的资源。有关将 AWS Glue 与 Amazon VPC 结合使用的更多信息，请参阅 the section called “VPC 端点 (AWS PrivateLink)”。

在 Ray 中处理数据的常用库

Ray Data – Ray Data 提供了处理常见数据格式、源和接收器的方法。有关 Ray Data 中支持的格式和源的更多信息，请参阅 Ray Data 文档中的 Input/Output。Ray Data 是一个坚持己见的库，而不是一个用于处理数据集的通用库。

Ray 围绕 Ray Data 可能是您工作的最佳解决方案的用例提供了某些指导。有关更多信息，请参阅 Ray 文档中的 Ray use cases。

适用于 Pandas 的 AWS 开发工具包（awswrangler）– 适用于 Pandas 的 AWS 开发工具包是一款 AWS 产品，当您使用 pandas DataFrames 进行转换管理数据时，该产品提供经过测试的简洁解决方案，用于读取和写入 AWS 服务。有关适用于 Pandas 的 AWS 开发工具包中支持的格式和来源的更多信息，请参阅AWS适用于 Pandas 的开发工具包文档中的 API Reference。

有关如何使用适用于 Pandas 的 AWS 开发工具包读取和写入数据的示例，请参阅适用于 Pandas 的 AWS 开发工具包文档中的 Quick Start。适用于 Pandas 的 AWS 开发工具包不为您的数据提供转换。它仅支持从源读取和写入。

Modin – Modin 是一个 Python 库，它以可分发的方式实现了常见的 Pandas 操作。有关 Modin 的更多信息，请参阅 Modin 文档。Modin 本身不提供从源读取和写入的支持。它提供常见转换的分布式实现。适用于 Pandas 的 AWS 开发工具包支持 Modin。

当您在 Ray 环境中同时运行 Modin 和适用于 Pandas 的 AWS 开发工具包时，您可以执行常见的 ETL 任务并获得高性能结果。有关将 Modin 与适用于 Pandas 的 AWS 开发工具包一起使用的更多信息，请参阅适用于 Pandas 的 AWS 开发工具包文档中的 At scale。

其他框架 - 有关 Ray 支持的框架的更多信息，请参阅 Ray 文档中的 The Ray Ecosystem。我们不为 AWS Glue for Ray 中的其他框架提供支持。

通过 Data Catalog 连接

适用于 Pandas 的 AWS 开发工具包支持通过 Data Catalog 与 Ray 作业一起管理数据。有关更多信息，请参阅适用于 Pandas 的 AWS 开发工具包网站上的 Glue 目录。

将此服务与 AWS SDK 配合使用

AWS 软件开发套件 (SDK) 可用于许多流行的编程语言。每个软件开发工具包都提供 API、代码示例和文档，使开发人员能够更轻松地以其首选语言构建应用程序。

SDK 文档	代码示例
AWS SDK for C++	AWS SDK for C++ 代码示例
AWS CLI	AWS CLI 代码示例
AWS SDK for Go	AWS SDK for Go 代码示例
AWS SDK for Java	AWS SDK for Java 代码示例
AWS SDK for JavaScript	AWS SDK for JavaScript 代码示例
AWS SDK for Kotlin	AWS SDK for Kotlin 代码示例
AWS SDK for .NET	AWS SDK for .NET 代码示例
AWS SDK for PHP	AWS SDK for PHP 代码示例
AWS Tools for PowerShell	PowerShell 代码示例工具
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 代码示例
AWS SDK for Ruby	AWS SDK for Ruby 代码示例
AWS SDK for Rust	AWS SDK for Rust 代码示例
适用于 SAP ABAP 的 AWS SDK	适用于 SAP ABAP 的 AWS SDK 代码示例
AWS SDK for Swift	AWS SDK for Swift 代码示例

有关特定于此服务的示例，请参阅AWS Glue 使用 AWS 软件开发工具包的 API 代码示例。

示例可用性

找不到所需的内容？ 通过使用此页面底部的提供反馈链接请求代码示例。

AWS Glue API

本节介绍了 AWS Glue SDK 和工具使用的数据类型和基元。在之外有三种 AWS Glue 以编程方式与之交互的通用方式 AWS Management Console，每种方式都有自己的文档：

- 语言软件开发工具包 (SDK) 库允许您访问来自常用编程语言的 AWS 资源。请在用于在 AWS上进行构建的工具查找更多信息。
- AWS CLI 允许您从命令行访问 AWS 资源。请在 AWS CLI 命令参考查找更多信息。
- AWS CloudFormation 允许您定义一组要一致配置的 AWS 资源。有关更多信息，请访问 AWS CloudFormation： AWS Glue 资源类型参考。

本节记录了独立于这些软件开发工具包 (SDK) 和工具的共享原语。工具使用 AWS Glue Web API 参考进行通信 AWS。

目录

- SkewedInfo 结构
- TableVersion 结构
- TableError 结构
- TableVersionError 结构
- SortCriterion 结构
- TableIdentifier 结构
- KeySchemaElement 结构
- PartitionIndex 结构
- PartitionIndexDescriptor 结构
- BackfillError 结构
- IcebergInput 结构
- OpenTableFormatInput 结构
- ViewDefinition 结构
- ViewDefinitionInput 结构
- ViewRepresentation 结构
- ViewRepresentationInput 结构
- 操作
- CreateTable 动作（Python：create_table）
- UpdateTable 动作（Python：update_table）
- DeleteTable 操作（Python：delete_table）
- BatchDeleteTable 操作（Python：batch_delete_table）
- GetTable 动作（Python：get_table）
- GetTables 动作（Python：get_tables）
- GetTableVersion 操作（Python：获取表格版本）
- GetTableVersions 操作（Python：获取表格版本）
- DeleteTableVersion 操作（Python：删除表格版本）
- BatchDeleteTableVersion 操作（Python：batch_delete_table_version）
- SearchTables 操作（Python：搜索表）
- GetPartitionIndexes 操作（Python：获取分区索引）
- CreatePartitionIndex 操作（Python：create_partition_index）

- GrokClassifier 结构
- XMLClassifier 结构
- JsonClassifier 结构
- CsvClassifier 结构
- CreateGrokClassifierRequest 结构
- UpdateGrokClassifierRequest 结构
- CreateXMLClassifierRequest 结构
- UpdateXMLClassifierRequest 结构
- CreateJsonClassifierRequest 结构
- UpdateJsonClassifierRequest 结构
- CreateCsvClassifierRequest 结构
- UpdateCsvClassifierRequest 结构
- 操作
- CreateClassifier 操作（Python：create_classifier）
- DeleteClassifier 操作（Python：delete_classifier）
- GetClassifier 操作（Python：get_classifier）
- GetClassifiers 操作（Python：get_classifiers）
- UpdateClassifier 操作（Python：update_classifier）

- 爬网程序 API
  - 数据类型
  - Crawler 结构
  - Schedule 结构
  - CrawlerTargets 结构
  - S3Target 结构
  - S3 DeltaCatalogTarget 结构
  - S3 DeltaDirectTarget 结构
  - JdbcTarget 结构
  - MongoDBTarget 结构
  - DynamoDBTarget 结构
  - DeltaTarget 结构

- AthenaConnectorSource 结构
- JDBC 结构 ConnectorSource
- SparkConnectorSource 结构
- CatalogSource 结构
- MySQL CatalogSource 结构
- PostgreSQL 结构 CatalogSource
- OracleSQL 结构 CatalogSource
- 微软 SQL 结构 ServerCatalogSource
- CatalogKinesisSource 结构
- DirectKinesisSource 结构
- KinesisStreamingSourceOptions 结构
- CatalogKafkaSource 结构
- DirectKafkaSource 结构
- KafkaStreamingSourceOptions 结构
- RedshiftSource 结构
- AmazonRedshiftSource 结构
- AmazonRedshiftNodeData 结构
- AmazonRedshiftAdvancedOption 结构
- 选项结构
- S3 CatalogSource 结构
- S3 SourceAdditionalOptions 结构
- S3 CsvSource 结构
- DirectJDBCSource 结构
- S3 DirectSourceAdditionalOptions 结构
- S3 JsonSource 结构
- S3 ParquetSource 结构
- S3 DeltaSource 结构
- S3 CatalogDeltaSource 结构
- CatalogDeltaSource 结构
- S3 HudiSource 结构

- S3 CatalogHudiSource 结构
- CatalogHudiSource 结构
- DynamoDB 结构 CatalogSource
- RelationalCatalogSource 结构
- JDBC 结构 ConnectorTarget
- SparkConnectorTarget 结构
- BasicCatalogTarget 结构
- MySQL CatalogTarget 结构
- PostgreSQL 结构 CatalogTarget
- OracleSQL 结构 CatalogTarget
- 微软 SQL 结构 ServerCatalogTarget
- RedshiftTarget 结构
- AmazonRedshiftTarget 结构
- UpsertRedshiftTargetOptions 结构
- S3 CatalogTarget 结构
- S3 GlueParquetTarget 结构
- CatalogSchemaChangePolicy 结构
- S3 DirectTarget 结构
- S3 HudiCatalogTarget 结构
- S3 HudiDirectTarget 结构
- S3 DeltaCatalogTarget 结构
- S3 DeltaDirectTarget 结构
- DirectSchemaChangePolicy 结构
- ApplyMapping 结构
- Mapping 结构
- SelectFields 结构
- DropFields 结构
- RenameField 结构
- Spigot 结构
- Join 结构

- JoinColumn 结构
- SplitFields 结构
- SelectFromCollection 结构
- FillMissingValues 结构
- Filter 结构
- FilterExpression 结构
- FilterValue 结构
- CustomCode 结构
- SparkSQL 结构
- SqlAlias 结构
- DropNullFields 结构
- NullCheckBoxList 结构
- NullValueField 结构
- Datatype 结构
- Merge 结构
- Union 结构
- PIIDetection 结构
- Aggregate 结构
- DropDuplicates 结构
- GovernedCatalogTarget 结构
- GovernedCatalogSource 结构
- AggregateOperation 结构
- GlueSchema 结构
- GlueStudioSchemaColumn 结构
- GlueStudioColumn 结构
- DynamicTransform 结构
- TransformConfigParameter 结构
- EvaluateDataQuality 结构
- DQ 结构 ResultsPublishingOptions
- DQ 结构 StopJobOnFailureOptions

- FindMatchesMetrics 结构
- ConfusionMatrix 结构
- GlueTable 结构
- TaskRun 结构
- TransformFilterCriteria 结构
- TransformSortCriteria 结构
- TaskRunFilterCriteria 结构
- TaskRunSortCriteria 结构
- TaskRunProperties 结构
- FindMatchesTaskRunProperties 结构
- ImportLabelsTaskRunProperties 结构
- ExportLabelsTaskRunProperties 结构
- LabelingSetGenerationTaskRunProperties 结构
- SchemaColumn 结构
- TransformEncryption 结构
- MLUserDataEncryption 结构
- ColumnImportance 结构
- 操作
- CreateMLTransform 操作（Python：create_ml_transform）
- UpdateMLTransform 操作（Python：update_ml_transform）
- DeleteMLTransform 操作（Python：delete_ml_transform）
- GetMLTransform 操作（Python：get_ml_transform）
- GetMLTransforms 操作（Python：get_ml_ransforms）
- ListMLTransforms 操作（Python：list_ml_transforms）
- StartMLEvaluationTaskRun 操作（Python：start_ml_evaluation_task_run）
- StartMLLabelingSetGenerationTaskRun 操作（Python：start_ml_labeling_set_generation_task_run）
- GetMLTaskRun 操作（Python：get_ml_task_run）
- GetMLTaskRuns 操作（Python：get_ml_task_runs）
- CancelMLTaskRun 操作（Python：cancel_ml_task_run）

- InvalidInputException 结构
- InvalidStateException 结构
- InvalidTaskStatusTransitionException 结构
- JobDefinitionErrorException 结构
- JobRunInTerminalStateException 结构
- JobRunInvalidStateTransitionException 结构
- JobRunNotInTerminalStateException 结构
- LateRunnerException 结构
- NoScheduleException 结构
- OperationTimeoutException 结构
- ResourceNotReadyException 结构
- ResourceNumberLimitExceededException 结构
- SchedulerNotRunningException 结构
- SchedulerRunningException 结构
- SchedulerTransitioningException 结构
- UnrecognizedRunnerException 结构
- ValidationException 结构
- VersionMismatchException 结构

AWS Glue 中的安全性 API

安全性 API 介绍安全数据类型以及与 AWS Glue 中的安全性相关的 API。

数据类型

- DataCatalogEncryptionSettings 结构
- EncryptionAtRest 结构
- ConnectionPasswordEncryption 结构
- EncryptionConfiguration 结构
- S3Encryption 结构
- CloudWatchEncryption 结构
- JobBookmarksEncryption 结构

- SecurityConfiguration 结构
- GluePolicy 结构

DataCatalogEncryptionSettings 结构

包含用于维护数据目录安全性的配置信息。

字段

- `EncryptionAtRest` – 一个 EncryptionAtRest 对象。

  为数据目录指定静态加密配置。

- `ConnectionPasswordEncryption` – 一个 ConnectionPasswordEncryption 对象。

  启用连接密码保护后，数据目录使用客户提供的密钥作为 `CreateConnection` 或 `UpdateConnection` 的一部分来加密密码，并将密码存储在连接属性中的 `ENCRYPTED_PASSWORD` 字段中。您可以启用目录加密或仅密码加密。

EncryptionAtRest 结构

为数据目录指定静态加密配置。

字段

- `CatalogEncryptionMode` – 必填： UTF-8 字符串（有效值：`DISABLED` | `SSE-KMS="SSEKMS"` | `SSE-KMS-WITH-SERVICE-ROLE="SSEKMSWITHSERVICEROLE"`）。

  用于对数据目录数据进行加密的静态加密模式。

- `SseAwsKmsKeyId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  用于静态加密的 AWS KMS 密钥的 ID。

- `CatalogEncryptionServiceRole` – UTF-8 字符串，与 Custom string pattern #24 匹配。

  AWS Glue 为了代表调用者加密和解密数据目录对象而代入的角色。

ConnectionPasswordEncryption 结构

数据目录用于加密密码的数据结构，作为 CreateConnection 或 UpdateConnection 的一部分，并将其存储在连接属性的 ENCRYPTED_PASSWORD 字段中。您可以启用目录加密或仅密码加密。

当包含密码的 CreationConnection 请求到达时，数据目录首先使用您的 AWS KMS 密钥加密密码。然后，如果还启用了目录加密，数据目录会再次加密整个连接对象。

此加密要求您根据安全要求设置 AWS KMS 密钥权限以启用或限制对密码密钥的访问。例如，您可能只希望管理员拥有密码密钥的解密权限。

字段

- ReturnConnectionPasswordEncrypted – 必填：布尔值。

  当 ReturnConnectionPasswordEncrypted 标记设置为“true”时，在 GetConnection 和 GetConnections 响应中密码仍然保持加密状态。此加密独立于目录加密生效。
- AwsKmsKeyId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  用于加密连接密码的 AWS KMS 密钥。

  如果启用了连接密码保护，则 CreateConnection 和 UpdateConnection 的调用方至少需要指定的 AWS KMS 密钥的 kms:Encrypt 权限，才能在将密码存储在数据目录中之前加密密码。

  您可以根据安全要求设置解密权限以启用或限制对密码密钥的访问。

EncryptionConfiguration 结构

指定加密配置。

字段

- S3Encryption – S3Encryption 对象的数组。

  Amazon Simple Storage Service (Amazon S3) 数据的加密配置。
- CloudWatchEncryption – 一个 CloudWatchEncryption 对象。

  Amazon CloudWatch 的加密配置。
- JobBookmarksEncryption – 一个 JobBookmarksEncryption 对象。

作业书签的加密配置。

S3Encryption 结构

指定怎样对 Amazon Simple Storage Service（Amazon S3）数据进行加密。

字段

- `S3EncryptionMode` – UTF-8 字符串（有效值：`DISABLED` | `SSE-KMS="SSEKMS"` | `SSE-S3="SSES3"`）。

  用于 Amazon S3 数据的加密模式。
- `KmsKeyArn` – UTF-8 字符串，与 Custom string pattern #25 匹配。

  用于加密数据的 KMS 密钥的 Amazon Resource Name（ARN）。

CloudWatchEncryption 结构

指定如何加密 Amazon CloudWatch 数据。

字段

- `CloudWatchEncryptionMode` – UTF-8 字符串（有效值：`DISABLED` | `SSE-KMS="SSEKMS"`）。

  用于 CloudWatch 数据的加密模式。
- `KmsKeyArn` – UTF-8 字符串，与 Custom string pattern #25 匹配。

  用于加密数据的 KMS 密钥的 Amazon Resource Name（ARN）。

JobBookmarksEncryption 结构

指定如何加密任务书签数据。

字段

- `JobBookmarksEncryptionMode` – UTF-8 字符串（有效值：`DISABLED` | `CSE-KMS="CSEKMS"`）。

  用于任务书签数据的加密模式。

- KmsKeyArn – UTF-8 字符串，与 Custom string pattern #25 匹配。

  用于加密数据的 KMS 密钥的 Amazon Resource Name（ARN）。

SecurityConfiguration 结构

指定安全配置。

字段

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  安全配置的名称。

- CreatedTimeStamp – 时间戳。

  创建此安全配置的时间点。

- EncryptionConfiguration – 一个 EncryptionConfiguration 对象。

  与此安全配置关联的加密配置。

GluePolicy 结构

返回资源策略的结构。

字段

- PolicyInJson – UTF-8 字符串，至少 2 个字节。

  包含 JSON 格式的请求策略文档。

- PolicyHash – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  包含与此策略关联的哈希值。

- CreateTime – 时间戳。

  创建策略的日期和时间。

- UpdateTime – 时间戳。

上次更新策略的日期和时间。

操作

- GetDataCatalogEncryptionSettings 操作（Python：get_data_catalog_encryption_settings）
- PutDataCatalogEncryptionSettings 操作（Python： put_data_catalog_encryption_settings）
- PutResourcePolicy 操作（Python：put_resource_policy）
- GetResourcePolicy 操作（Python：get_resource_policy）
- DeleteResourcePolicy 操作（Python：delete_resource_policy）
- CreateSecurityConfiguration 操作（Python：create_security_configuration）
- DeleteSecurityConfiguration 操作（Python：delete_security_configuration）
- GetSecurityConfiguration 操作（Python：get_security_configuration）
- GetSecurityConfigurations 操作（Python：get_security_configurations）
- GetResourcePolicies 操作（Python：get_resource_policie）

GetDataCatalogEncryptionSettings 操作（Python：get_data_catalog_encryption_settings）

检索指定目录的安全配置。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索其安全配置的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。

响应

- `DataCatalogEncryptionSettings` – 一个 DataCatalogEncryptionSettings 对象。

  请求的安全配置。

错误

- InternalServiceException
- InvalidInputException
- OperationTimeoutException

PutDataCatalogEncryptionSettings 操作（Python：put_data_catalog_encryption_settings）

设置指定目录的安全配置。设置配置后，指定的加密将应用于之后的每个目录写入。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要设置其安全配置的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- DataCatalogEncryptionSettings – 必填：一个 DataCatalogEncryptionSettings 对象。

  要设置的安全配置。

响应

- 无响应参数。

错误

- InternalServiceException
- InvalidInputException
- OperationTimeoutException

PutResourcePolicy 操作（Python：put_resource_policy）

设置用于访问控制的数据目录资源策略。

请求

- PolicyInJson – 必需： UTF-8 字符串，至少 2 个字节。

  包含要设置的 JSON 格式的策略文档。

- ResourceArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  不使用。仅供内部使用。

- PolicyHashCondition – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  在使用 PutResourcePolicy 设置上一个策略时返回的哈希值。其用途是防止并发修改策略。如果未设置先前策略，请勿使用此参数。

- PolicyExistsCondition – UTF-8 字符串（有效值：MUST_EXIST | NOT_EXIST | NONE）。

  MUST_EXIST 的值用于更新策略。NOT_EXIST 的值用于创建新策略。如果使用 NONE 值或空值，调用不依赖于策略是否存在。

- EnableHybrid – UTF-8 字符串（有效值：TRUE | FALSE）。

  如果 'TRUE'，表示您正在使用这两种方法来授予对数据目录资源的跨账户访问权限：

  - 通过使用 PutResourePolicy 直接更新资源策略
  - 通过使用 AWS Management Console 上的授予权限命令。

  如果您已经使用管理控制台授予跨账户访问权限，则必须设置为 'TRUE'，否则调用会失败。默认设置为“FALSE”。

响应

- PolicyHash – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  刚刚设置的策略的哈希。此值必须包含在覆盖或更新该策略的后续调用中。

错误

- EntityNotFoundException
- InternalServiceException

- OperationTimeoutException
- InvalidInputException
- ConditionCheckFailureException

GetResourcePolicy 操作（Python：get_resource_policy）

检索指定的资源策略。

请求

- ResourceArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  要检索其资源策略的 AWS Glue 资源的 ARN。如果未提供，则返回数据目录资源策略。使用 GetResourcePolicies 查看所有现有资源策略。有关更多信息，请参阅指定 AWS Glue 资源 ARN。

响应

- PolicyInJson – UTF-8 字符串，至少 2 个字节。

  包含 JSON 格式的请求策略文档。
- PolicyHash – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  包含与此策略关联的哈希值。
- CreateTime – 时间戳。

  创建策略的日期和时间。
- UpdateTime – 时间戳。

  上次更新策略的日期和时间。

错误

- EntityNotFoundException
- InternalServiceException

- OperationTimeoutException
- InvalidInputException

DeleteResourcePolicy 操作（Python：delete_resource_policy）

删除指定的策略。

请求

- PolicyHashCondition – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  设置此策略时返回的哈希值。
- ResourceArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  要删除的资源策略的 AWS Glue 资源的 ARN。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ConditionCheckFailureException

CreateSecurityConfiguration 操作（Python：create_security_configuration）

创建新的安全配置。安全配置是 AWS Glue 可以使用的一组安全属性。您可以使用安全配置加密静态数据。有关使用 AWS Glue 中的安全配置的信息，请参阅加密由爬网程序、任务和开发端点写入的数据。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  新安全配置的名称。
- EncryptionConfiguration – 必填：一个 EncryptionConfiguration 对象。

  新安全配置的加密配置。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分配给新安全配置的名称。
- CreatedTimestamp – 时间戳。

  创建新安全配置的时间点。

错误

- AlreadyExistsException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException

DeleteSecurityConfiguration 操作（Python：delete_security_configuration）

删除指定的安全配置。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的安全配置的名称。

响应

- 无响应参数。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetSecurityConfiguration 操作（Python：get_security_configuration）

检索指定的安全配置。

请求

- `Name` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索的安全配置的名称。

响应

- `SecurityConfiguration` – 一个 SecurityConfiguration 对象。

  请求的安全配置。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetSecurityConfigurations 操作（Python：get_security_configurations）

检索所有安全配置的列表。

请求

- `MaxResults` – 数字（整数），不小于 1 或大于 1000。

  要返回的最大结果数量。
- `NextToken` – UTF-8 字符串。

  延续标记 (如果这是延续调用)。

响应

- `SecurityConfigurations` – SecurityConfiguration 对象的数组。

  安全配置的列表。
- `NextToken` – UTF-8 字符串。

  一个延续令牌（如果有多个安全配置要返回）。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetResourcePolicies 操作（Python：get_resource_policie）

在跨账户权限授予期间检索 AWS Resource Access Manager 在单个资源上设置的资源策略。同时检索数据目录资源策略。

如果您在数据目录设置中启用了元数据加密，并且您没有 AWS KMS 密钥的权限，则操作无法返回数据目录资源策略。

请求

- NextToken – UTF-8 字符串。

  延续令牌 (如果这是延续请求)。
- MaxResults – 数字（整数），不小于 1 或大于 1000。

  要返回的列表的最大大小。

响应

- GetResourcePoliciesResponseList – GluePolicy 对象的数组。

  单个资源策略和账户级资源策略列表。
- NextToken – UTF-8 字符串。

  延续令牌（如果返回的列表不包含上一个可用的资源策略）。

错误

- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- GlueEncryptionException

Catalog API

目录 API 描述了与使用 AWS Glue 中的目录相关的数据类型和 API。

主题

- 数据库 API
- 表 API
- 分区 API
- 连接 API
- 用户定义的函数 API

- 将 Athena 目录导入 AWS Glue

数据库 API

数据库 API 介绍数据库数据类型，还包括用于创建、删除、定位、更新和列出数据库的 API。

数据类型

- Database 结构
- DatabaseInput 结构
- PrincipalPermissions 结构
- DataLakePrincipal 结构
- DatabaseIdentifier 结构
- 联合数据库结构

Database 结构

`Database` 对象表示可能驻留在 Hive 元存储或 RDBMS 中的表的逻辑分组。

字段

- `Name` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  数据库的名称。为了确保 Hive 兼容性，它在存储时被转换为小写。
- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  对数据库的描述。
- `LocationUri` - 统一资源标识符 (uri)，不少于 1 个字节或超过 1024 个字节，与 URI address multi-line string pattern 匹配。

  数据库的位置（例如，HDFS 路径）。
- `Parameters` – 键值对的映射数组。

  每个键是一个键字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串，不超过 512000 个字节。

这些键值对定义数据库的参数和属性。

- `CreateTime` – 时间戳。

  在目录中创建元数据数据库的时间。

- `CreateTableDefaultPermissions` – PrincipalPermissions 对象的数组。

  在委托人表上创建一组默认权限。由 AWS Lake Formation 使用。正常的 AWS Glue 操作过程中不使用。

- `TargetDatabase` – 一个 DatabaseIdentifier 对象。

  描述用于资源链接的目标数据库的 `DatabaseIdentifier` 结构。

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  数据库所在的数据目录的 ID。

- `FederatedDatabase` – 一个 联合数据库 对象。

  一种引用 AWS Glue Data Catalog 外部实体的 `FederatedDatabase` 结构。

DatabaseInput 结构

用于创建或更新数据库的结构。

字段

- `Name` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  数据库的名称。为了确保 Hive 兼容性，它在存储时被转换为小写。

- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  对数据库的描述。

- `LocationUri` - 统一资源标识符 (uri)，不少于 1 个字节或超过 1024 个字节，与 URI address multi-line string pattern 匹配。

  数据库的位置（例如，HDFS 路径）。

- `Parameters` – 键值对的映射数组。

  每个键是一个键字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串，不超过 512000 个字节。

  这些键值对定义数据库的参数和属性。

  这些键值对定义数据库的参数和属性。
- `CreateTableDefaultPermissions` – PrincipalPermissions 对象的数组。

  在委托人表上创建一组默认权限。由 AWS Lake Formation 使用。正常的 AWS Glue 操作过程中不使用。
- `TargetDatabase` – 一个 DatabaseIdentifier 对象。

  描述用于资源链接的目标数据库的 `DatabaseIdentifier` 结构。
- `FederatedDatabase` – 一个 联合数据库 对象。

  一种引用 AWS Glue Data Catalog 外部实体的 `FederatedDatabase` 结构。

PrincipalPermissions 结构

向委托人授予的权限。

字段

- `Principal` – 一个 DataLakePrincipal 对象。

  被授予权限的委托人。
- `Permissions` – UTF-8 字符串数组。

  向委托人授予的权限。

DataLakePrincipal 结构

AWS Lake Formation 委托人。

字段

- `DataLakePrincipalIdentifier` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节。

AWS Lake Formation 委托人的标识符。

DatabaseIdentifier 结构

描述用于资源链接的目标数据库的结构。

字段

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  数据库所在的数据目录的 ID。

- `DatabaseName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  目录数据库的名称。

- `Region` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  目标数据库的区域。

联合数据库结构

指向 AWS Glue Data Catalog 外部实体的数据库。

字段

- `Identifier` – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节，与 Single-line string pattern 匹配。

  联合数据库的唯一标识符。

- `ConnectionName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  与外部元存储连接的名称。

操作

- CreateDatabase 操作（Python：create_database）
- UpdateDatabase 操作（Python：update_database）
- DeleteDatabase 操作（Python：delete_database）
- GetDatabase 操作（Python：get_database）
- GetDatabases 操作（Python：get_databases）

CreateDatabase 操作（Python：create_database）

在数据目录中创建新数据库。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中创建数据库的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- `DatabaseInput` – 必填：一个 DatabaseInput 对象。

  数据库的元数据。
- `Tags` – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  您分配给数据库的标签。

响应

- 无响应参数。

错误

- `InvalidInputException`
- `AlreadyExistsException`

- ResourceNumberLimitExceededException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ConcurrentModificationException
- FederatedResourceAlreadyExistsException

UpdateDatabase 操作（Python：update_database）

在数据目录中更新现有数据库定义。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  元数据数据库所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在目录中更新的数据库的名称。对于 Hive 兼容性，它会转化为小写。
- DatabaseInput – 必填：一个 DatabaseInput 对象。

  一个 DatabaseInput 对象，在目录中指定元数据数据库的新定义。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

- ConcurrentModificationException

DeleteDatabase 操作（Python：delete_database）

从数据目录中删除指定的数据库。

> **Note**
>
> 完成此操作后，您将无法再访问已删除的数据库中的这些表（以及可能属于这些表的所有表版本和分区）和用户定义函数。AWS Glue 会及时以异步方式删除这些“孤立”资源，这由服务决定。
>
> 为了确保立即删除所有相关资源，在调用 DeleteDatabase 之前，请使用 DeleteTableVersion 或 BatchDeleteTableVersion、DeletePartition 或 BatchDeletePartition、DeleteUserDefinedFunction 和 DeleteTable 或 BatchDeleteTable，删除属于该数据库的所有资源。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  数据库所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的数据库的名称。对于 Hive 兼容性，它必须是全部小写的。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException

- OperationTimeoutException
- ConcurrentModificationException

GetDatabase 操作（Python：get_database）

检索指定数据库的定义。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  数据库所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索的数据库的名称。对于 Hive 兼容性，它应该是全部小写的。

响应

- Database – 一个 数据库 对象。

  数据目录中指定数据库的定义。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- FederationSourceException

GetDatabases 操作（Python：get_databases）

检索在给定数据目录中定义的所有数据库。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中检索 Databases 的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用)。
- MaxResults – 数字（整数），不小于 1 或大于 100。

  要在一个响应中返回的数据库的最大数量。
- ResourceShareType – UTF-8 字符串（有效值：FOREIGN | ALL | FEDERATED）。

  允许您指定要列出与您的账户共享的数据库。允许的值是 FEDERATED、FOREIGN 或 ALL。
  - 如果设置为 FEDERATED，将列出与您的账户共享的联合数据库（引用外部实体）。
  - 如果设置为 FOREIGN，将列出与您的账户共享的数据库。
  - 如果设置为 ALL，将列出与您的账户共享的数据库，以及本地账户中的数据库。

响应

- DatabaseList – 必填：数据库 对象的数组。

  指定目录中的 Database 对象的列表。
- NextToken – UTF-8 字符串。

  对返回的标记列表进行分页的延续令牌 (如果列表的当前片段不是最后一个，则返回)。

错误

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

表 API

表 API 介绍与表关联的数据类型和操作。

数据类型

- Table 结构
- TableInput 结构
- FederatedTable 结构
- 列结构
- StorageDescriptor 结构
- SchemaReference 结构
- SerDeInfo 结构
- Order 结构
- SkewedInfo 结构
- TableVersion 结构
- TableError 结构
- TableVersionError 结构
- SortCriterion 结构
- TableIdentifier 结构
- KeySchemaElement 结构
- PartitionIndex 结构
- PartitionIndexDescriptor 结构
- BackfillError 结构
- IcebergInput 结构
- OpenTableFormatInput 结构
- ViewDefinition 结构
- ViewDefinitionInput 结构
- ViewRepresentation 结构
- ViewRepresentationInput 结构

Table 结构

表示用列和行组织的相关数据的集合。

字段

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  表名称。对于 Hive 兼容性，它必须是完全小写的。
- DatabaseName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表元数据所在的数据库名称。对于 Hive 兼容性，它必须是全部小写的。
- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  对表的描述。
- Owner – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的所有者。
- CreateTime – 时间戳。

  在数据目录中创建表定义的时间。
- UpdateTime – 时间戳。

  上次更新该表的时间。
- LastAccessTime – 时间戳。

  上次访问该表的时间。这通常取自 HDFS，可能不可靠。
- LastAnalyzedTime – 时间戳。

  上次计算此表的列统计信息的时间。
- Retention – 数字（整数），至多为“无”。

  此表的保留时间。
- StorageDescriptor – 一个 StorageDescriptor 对象。

一个存储描述符，包含有关此表的物理存储的信息。

- PartitionKeys – 列 对象的数组。

  表进行分区所依据的列的列表。仅支持基元类型作为分区键。

  创建 Amazon Athena 使用的表时，如果未指定任何 partitionKeys，则必须至少将 partitionKeys 的值设置为空列表。例如：

  "PartitionKeys": []

- ViewOriginalText – UTF-8 字符串，不超过 409600 个字节。

  随附以实现 Apache Hive 兼容性。在正常 AWS Glue 操作过程中不使用。如果该表是 bas VIRTUAL_VIEW e6 Athena 4 编码的特定配置。

- ViewExpandedText – UTF-8 字符串，不超过 409600 个字节。

  随附以实现 Apache Hive 兼容性。在正常 AWS Glue 操作过程中不使用。

- TableType – UTF-8 字符串，不超过 255 个字节。

  此表的类型。 AWS Glue 将使用该EXTERNAL_TABLE类型创建表。其他服务（例如） Athena可能会创建具有其他表类型的表。

  AWS Glue 相关表类型：

  EXTERNAL_TABLE

  Hive 兼容属性 - 表示非 Hive 托管表。

  GOVERNED

  由... 使用 AWS Lake Formation。 AWS Glue 数据目录可以理解GOVERNED。

- Parameters – 键值对的映射数组。

  每个键是一个键字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串，不超过 512000 个字节。

  这些键值对定义了与此表关联的属性。

- CreatedBy – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

创建表的人员或实体。

- `IsRegisteredWithLakeFormation` – 布尔值。

  表示该表是否已向注册 AWS Lake Formation。

- `TargetTable` – 一个 TableIdentifier 对象。

  描述用于资源链接的目标表的 `TableIdentifier` 结构。

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据目录的 ID。

- `VersionId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表版本 ID。

- `FederatedTable` – 一个 FederatedTable 对象。

  一种引用 AWS Glue Data Catalog外部实体的 `FederatedTable` 结构。

- `ViewDefinition` – 一个 ViewDefinition 对象。

  包含定义视图的所有信息的结构，包括视图的一个或多个方言和查询。

- `IsMultiDialectView` – 布尔值。

  指定视图是否支持一个或多个不同查询引擎的 SQL 方言，因此可以由这些引擎读取。

TableInput 结构

用于定义表的结构。

字段

- `Name` – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  表名称。为了确保 Hive 兼容性，它在存储时被转换为小写。

- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

对表的描述。

- Owner – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所有者。随附以实现 Apache Hive 兼容性。在正常 AWS Glue 操作过程中不使用。

- LastAccessTime – 时间戳。

  上次访问该表的时间。

- LastAnalyzedTime – 时间戳。

  上次计算此表的列统计信息的时间。

- Retention – 数字（整数），至多为“无”。

  此表的保留时间。

- StorageDescriptor – 一个 StorageDescriptor 对象。

  一个存储描述符，包含有关此表的物理存储的信息。

- PartitionKeys – 列 对象的数组。

  表进行分区所依据的列的列表。仅支持基元类型作为分区键。

  创建 Amazon Athena 使用的表时，如果未指定任何 partitionKeys，则必须至少将 partitionKeys 的值设置为空列表。例如：

  ```
  "PartitionKeys": []
  ```

- ViewOriginalText – UTF-8 字符串，不超过 409600 个字节。

  随附以实现 Apache Hive 兼容性。在正常 AWS Glue 操作过程中不使用。如果该表是 bas VIRTUAL_VIEW e6 Athena 4 编码的特定配置。

- ViewExpandedText – UTF-8 字符串，不超过 409600 个字节。

  随附以实现 Apache Hive 兼容性。在正常 AWS Glue 操作过程中不使用。

- TableType – UTF-8 字符串，不超过 255 个字节。

  此表的类型。 AWS Glue 将使用该EXTERNAL_TABLE类型创建表。其他服务（例如） Athena可能会创建具有其他表类型的表。

  AWS Glue 相关表类型：

EXTERNAL_TABLE

Hive 兼容属性 - 表示非 Hive 托管表。

GOVERNED

由... 使用 AWS Lake Formation。 AWS Glue 数据目录可以理解GOVERNED。

- `Parameters` – 键值对的映射数组。

  每个键是一个键字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串，不超过 512000 个字节。

  这些键值对定义了与此表关联的属性。

- `TargetTable` – 一个 TableIdentifier 对象。

  描述用于资源链接的目标表的 `TableIdentifier` 结构。

- `ViewDefinition` – 一个 ViewDefinitionInput 对象。

  包含定义视图的所有信息的结构，包括视图的一个或多个方言和查询。

FederatedTable 结构

指向 AWS Glue Data Catalog外部实体的表。

字段

- `Identifier` – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节，与 Single-line string pattern 匹配。

  联合表的唯一标识符。

- `DatabaseIdentifier` – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节，与 Single-line string pattern 匹配。

  联合数据库的唯一标识符。

- `ConnectionName` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与外部元存储连接的名称。

列结构

Table 中的列。

字段

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  Column 的名称。
- Type – UTF-8 字符串，不超过 131072 个字节，与 Single-line string pattern 匹配。

  Column 的数据类型。
- Comment – 注释字符串，不超过 255 个字节，与 Single-line string pattern 匹配。

  自由格式的文本注释。
- Parameters – 键值对的映射数组。

  每个键是一个键字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串，不超过 512000 个字节。

  这些键/值对定义了与此列关联的属性。

StorageDescriptor 结构

描述表数据的物理存储。

字段

- Columns – 列 对象的数组。

  表中的 Columns 的列表。
- Location – 位置字符串，不超过 2056 个字节，与 URI address multi-line string pattern 匹配。

  表的物理位置。默认情况下，它采用仓库位置的形式，后跟仓库中的数据库位置，然后是表名称。
- AdditionalLocations – UTF-8 字符串数组。

  指向 Delta 表所在路径的位置列表。
- InputFormat – 格式字符串，不超过 128 个字节，与 Single-line string pattern 匹配。

输入格式：SequenceFileInputFormat（二进制）或 TextInputFormat 或自定义格式。

- OutputFormat – 格式字符串，不超过 128 个字节，与 Single-line string pattern 匹配。

  输出格式：SequenceFileOutputFormat（二进制）、IgnoreKeyTextOutputFormat 或自定义格式。

- Compressed – 布尔值。

  如果对表中的数据进行压缩，则为 True，否则为 False。

- NumberOfBuckets – 数字（整数）。

  如果表包含任何维度列，则必须指定。

- SerdeInfo – 一个 SerDeInfo 对象。

  序列化/反序列化 () 信息。SerDe

- BucketColumns – UTF-8 字符串数组。

  表中的 Reducer 分组列、集群列以及桶列的列表。

- SortColumns – 顺序 对象的数组。

  指定表中的每个桶的排序顺序的列表。

- Parameters – 键值对的映射数组。

  每个键是一个键字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串，不超过 512000 个字节。

  使用键/值形式的用户提供的属性。

- SkewedInfo – 一个 SkewedInfo 对象。

  有关在列中经常出现的值（偏斜值）的信息。

- StoredAsSubDirectories – 布尔值。

  如果表数据存储在子目录中，则为 True，否则为 False。

- SchemaReference – 一个 SchemaReference 对象。

  引用存储在架构注册表中的 AWS Glue 架构的对象。

  创建表时，可以为架构传递列的空列表，而使用架构引用。

SchemaReference 结构

引用存储在架构注册表中的 AWS Glue 架构的对象。

字段

- SchemaId – 一个 SchemaId 对象。

  包含架构标识字段的结构。必须提供此值或 SchemaVersionId。

- SchemaVersionId – UTF-8 字符串，长度不少于 36 个字节或超过 36 个字节，与 Custom string pattern #17 匹配。

  分配给架构版本的唯一 ID。必须提供此值或 SchemaId。

- SchemaVersionNumber – 数字（长度），不小于 1 或大于 100000。

  架构的版本号。

SerDeInfo 结构

有关用作提取器和加载器的序列化/反序列化程序 (SerDe) 的信息。

字段

- Name – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  的名字 SerDe。

- SerializationLibrary – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  通常是实现的类 SerDe。例
  如，org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe。

- Parameters – 键值对的映射数组。

  每个键是一个键字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串，不超过 512000 个字节。

  这些键值对定义了的初始化参数。 SerDe

Order 结构

指定排序列的排序顺序。

字段

- Column – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  列的名称。
- SortOrder – 必填：数字（整数），不大于 1。

  指示是按升序（== 1）还是降序（==0）对列进行排序。

SkewedInfo 结构

指定表中的偏斜值。偏斜值是指出现频率很高的值。

字段

- SkewedColumnNames – UTF-8 字符串数组。

  包含偏斜值的列名称的列表。
- SkewedColumnValues – UTF-8 字符串数组。

  经常被认为是偏斜的值的列表。
- SkewedColumnValueLocationMaps – 键值对的映射数组。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  偏斜值到包含它们的列的映射。

TableVersion 结构

指定表的版本。

字段

- Table – 一个 表 对象。

所涉表。

- VersionId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  标识此表版本的 ID 值。VersionId 是整数的字符串表示。每个版本都会增加 1。

TableError 结构

表操作的错误记录。

字段

- TableName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的名称。对于 Hive 兼容性，它必须是完全小写的。

- ErrorDetail – 一个 ErrorDetail 对象。

  有关错误的详细信息。

TableVersionError 结构

表版本操作的错误记录。

字段

- TableName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  相关表的名称。

- VersionId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  所涉版本的 ID 值。VersionID 是整数的字符串表示。每个版本都会增加 1。

- ErrorDetail – 一个 ErrorDetail 对象。

  有关错误的详细信息。

SortCriterion 结构

指定要作为排序依据的字段和排序顺序。

字段

- `FieldName` – 值字符串，不超过 1024 个字节。

  要作为排序依据的字段的名称。
- `Sort` – UTF-8 字符串（有效值：`ASC="ASCENDING"` | `DESC="DESCENDING"`）。

  升序或降序排序。

TableIdentifier 结构

描述用于资源链接的目标表的结构。

字段

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据目录的 ID。
- `DatabaseName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  包含目标表的目录数据库的名称。
- `Name` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  目标表的名称。
- `Region` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  目标表的区域。

KeySchemaElement 结构

由名称和类型组成的分区密钥对。

字段

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  分区键的名称。
- Type – 必填：UTF-8 字符串，长度不超过 131072 个字节，与 Single-line string pattern 匹配。

  分区键的类型。

PartitionIndex 结构

分区索引的结构。

字段

- Keys – 必填：UTF-8 字符串数组，至少 1 个字符串。

  分区索引的键。
- IndexName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区索引的名称。

PartitionIndexDescriptor 结构

表中分区索引的描述符。

字段

- IndexName – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  分区索引的名称。
- Keys – 必填：KeySchemaElement对象的数组，至少有 1 个结构。

  一个或多个键的列表，例如 KeySchemaElement 结构，用于分区索引。
- IndexStatus – 必填：UTF-8 字符串（有效值：CREATING | ACTIVE | DELETING | FAILED）。

  分区索引的状态。

可能状态包括：

- CREATING：正在创建索引。当索引处于 CREATING 状态时，无法删除索引或其表。
- ACTIVE：索引创建成功。
- FAILED：索引创建失败。
- DELETING：索引将从索引列表中删除。

- `BackfillErrors` – BackfillError 对象的数组。

  为现有表注册分区索引时可能发生的错误列表。

BackfillError 结构

为现有表注册分区索引时可能发生的错误列表。

这些错误提供了有关索引注册失败的原因的详细信息，并在响应中提供了有限数量的分区，以便您可以在故障时修复分区并尝试重新注册索引。可能发生的最常见错误集分类如下：

- EncryptedPartitionError：分区已加密。
- InvalidPartitionTypeDataError：分区值与该分区列的数据类型不匹配。
- MissingPartitionValueError：分区已加密。
- UnsupportedPartitionCharacterError: 不支持分区值内的字符。例如：U+0000、U+0001 和 U+0002。
- InternalError：任何不属于其他错误代码的错误。

字段

- `Code` – UTF-8 字符串（有效值：`ENCRYPTED_PARTITION_ERROR` | `INTERNAL_ERROR` | `INVALID_PARTITION_TYPE_DATA_ERROR` | `MISSING_PARTITION_VALUE_ERROR` | `UNSUPPORTED_PARTITION_CHARACTER_ERROR`）。

  为现有表注册分区索引时发生的错误代码。

- `Partitions` – PartitionValueList 对象的数组。

  响应中有限数量的分区列表。

IcebergInput 结构

一种结构，用于定义要在目录中创建的 Apache Iceberg 元数据表。

字段

- `MetadataOperation` – 必填：UTF-8 字符串（有效值：`CREATE`）。

  必需的元数据操作。只能设置为 `CREATE`。

- `Version` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  Iceberg 表的表格版本。默认值为 2。

OpenTableFormatInput 结构

一种表示开放格式表的结构。

字段

- `IcebergInput` – 一个 IcebergInput 对象。

  指定用于定义 Apache Iceberg 元数据表的 `IcebergInput` 结构。

ViewDefinition 结构

包含表示形式详细信息的结构。

字段

- `IsProtected` – 布尔值。

  您可以将此标志设置为 true，以指示引擎在查询规划期间不要将用户提供的操作推送到视图的逻辑计划中。但是，设置此标志并不能保证引擎将遵循此要求。请参阅引擎的文档以了解所提供的保证（如果有）。

- `Definer` – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节，与 Single-line string pattern 匹配。

  SQL 中视图的定义者。

- `SubObjects` – UTF-8 字符串数组，不超过 10 个字符串。

  包含表 Amazon 资源名称（ARN）的列表。
- `Representations` – ViewRepresentation对象的数组，不少于 1 个或不超过 1000 个结构。

  包含表示形式的列表。

ViewDefinitionInput 结构

包含用于创建或更新 AWS Glue 视图的详细信息的结构。

字段

- `IsProtected` – 布尔值。

  您可以将此标志设置为 true，以指示引擎在查询规划期间不要将用户提供的操作推送到视图的逻辑计划中。但是，设置此标志并不能保证引擎将遵循此要求。请参阅引擎的文档以了解所提供的保证（如果有）。
- `Definer` – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节，与 Single-line string pattern 匹配。

  SQL 中视图的定义者。
- `Representations` – ViewRepresentationInput 对象的数组，长度不少于 1 个结构，不超过 10 个结构。

  包含视图方言和定义视图的查询的结构列表。
- `SubObjects` – UTF-8 字符串数组，不超过 10 个字符串。

  包含构成视图的基表 ARN 列表。

ViewRepresentation 结构

包含视图方言和定义视图的查询的结构。

字段

- `Dialect` – UTF-8 字符串（有效值：`REDSHIFT` | `ATHENA` | `SPARK`）。

  查询引擎的方言。

- DialectVersion – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节。

  查询引擎的方言版本。例如 3.0.0。

- ViewOriginalText – UTF-8 字符串，不超过 409600 个字节。

  客户在 CREATE VIEW DDL 期间提供的 SELECT 查询。在对视图进行查询时不使用此 SQL（现改为使用 ViewExpandedText）。ViewOriginalText 用于 SHOW CREATE VIEW 等情况（当用户想要查看创建视图的原始 DDL 命令时）。

- ViewExpandedText – UTF-8 字符串，不超过 409600 个字节。

  视图的扩展 SQL。引擎在处理视图查询时使用此 SQL。在视图创建过程中，引擎可能会执行操作，将 ViewOriginalText 转换为 ViewExpandedText。例如：

  - 完全限定的标识符：SELECT * from table1 -> SELECT * from db1.table1

- ValidationConnection – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  用于验证视图的特定表示形式的连接的名称。

- IsStale – 布尔值。

  标记为过时的方言不再有效，必须先进行更新，然后才能在其各自的查询引擎中进行查询。

ViewRepresentationInput 结构

包含表示形式详细信息的结构，用于更新或创建 Lake Formation 视图。

字段

- Dialect – UTF-8 字符串（有效值：REDSHIFT | ATHENA | SPARK）。

  用于指定特定表示形式的引擎类型的参数。

- DialectVersion – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节。

  用于指定特定表示形式的引擎版本的参数。

- ViewOriginalText – UTF-8 字符串，不超过 409600 个字节。

  用于表示描述视图的原始 SQL 查询的字符串。

- ValidationConnection – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

用于验证视图的特定表示形式的连接的名称。

- `ViewExpandedText` – UTF-8 字符串，不超过 409600 个字节。

  表示 SQL 查询的字符串，该查询描述了带有扩展资源 ARN 的视图

操作

- CreateTable 动作（Python：create_table）
- UpdateTable 动作（Python：update_table）
- DeleteTable 操作（Python：delete_table）
- BatchDeleteTable 操作（Python：batch_delete_table）
- GetTable 动作（Python：get_table）
- GetTables 动作（Python：get_tables）
- GetTableVersion 操作（Python：获取表格版本）
- GetTableVersions 操作（Python：获取表格版本）
- DeleteTableVersion 操作（Python：删除表格版本）
- BatchDeleteTableVersion 操作（Python：batch_delete_table_version）
- SearchTables 操作（Python：搜索表）
- GetPartitionIndexes 操作（Python：获取分区索引）
- CreatePartitionIndex 操作（Python：create_partition_index）
- DeletePartitionIndex 操作（Python：delete_partition_index）
- GetColumnStatisticsForTable 操作（Python：获取表格的列统计信息）
- UpdateColumnStatisticsForTable 操作（Python：更新表格的列统计信息）
- DeleteColumnStatisticsForTable 操作（Python：表格的删除列统计数据）

CreateTable 动作（Python：create_table）

在数据目录中创建新表定义。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

要在其中创建 Table 的数据目录的 ID。如果未提供任何信息，则默认使用 AWS 账户 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中创建新表的目录数据库。对于 Hive 兼容性，此名称必须是完全小写的。

- TableInput – 必填：一个 TableInput 对象。

  用于定义要在目录中创建的元数据表的 TableInput 对象。

- PartitionIndexes – PartitionIndex 对象的数组，不超过 3 个结构。

  用于表中创建的分区索引和 PartitionIndex 结构的列表。

- TransactionId – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Custom string pattern #16 匹配。

  事务的 ID。

- OpenTableFormatInput – 一个 OpenTableFormatInput 对象。

  在创建开放格式表时指定 OpenTableFormatInput 结构。

响应

- 无响应参数。

错误

- AlreadyExistsException
- InvalidInputException
- EntityNotFoundException
- ResourceNumberLimitExceededException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ConcurrentModificationException
- ResourceNotReadyException

UpdateTable 动作（Python：update_table）

更新数据目录中的元数据表。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据目录的 ID。如果未提供任何信息，则默认使用 AWS 账户 ID。
- `DatabaseName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的目录数据库的名称。对于 Hive 兼容性，此名称必须是完全小写的。
- `TableInput` – 必填：一个 TableInput 对象。

  用于定义目录中的元数据表的 `TableInput` 对象。
- `SkipArchive` – 布尔值。

  默认情况下，`UpdateTable` 始终在更新表之前创建一个存档版本。但是，如果 `skipArchive` 设置为 true，`UpdateTable` 不创建存档的版本。
- `TransactionId` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Custom string pattern #16 匹配。

  在该 ID 处更新表内容的事务 ID。
- `VersionId` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  用于更新表内容的版本 ID。
- `ViewUpdateAction` – UTF-8 字符串（有效值：`ADD` | `REPLACE` | `ADD_OR_REPLACE` | `DROP`）。

  更新视图时要执行的操作。
- `Force` – 布尔值。

  一个标志，可以设置为 true 以忽略匹配的存储描述符和子对象匹配要求。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException
- ResourceNumberLimitExceededException
- GlueEncryptionException
- ResourceNotReadyException

DeleteTable 操作（Python：delete_table）

从数据目录中创建表定义。

> **Note**
>
> 完成此操作后，您将无法再访问属于已删除的表的表版本和分区。 AWS Glue 会及时以异步方式删除这些“孤立”资源，这由服务决定。
> 为了确保立即删除所有相关资源，在调用 DeleteTable 之前，请使用 DeleteTableVersion 或 BatchDeleteTableVersion 以及 DeletePartition 或 BatchDeletePartition，删除属于该表的所有资源。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据目录的 ID。如果未提供任何信息，则默认使用 AWS 账户 ID。
- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的目录数据库的名称。对于 Hive 兼容性，此名称必须是完全小写的。
- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

要删除的表的名称。对于 Hive 兼容性，此名称必须是完全小写的。

- `TransactionId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #16 匹配。

  在该 ID 处删除表内容的事务 ID。

响应

- 无响应参数。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentModificationException`
- `ResourceNotReadyException`

BatchDeleteTable 操作（Python：batch_delete_table）

一次性删除多个表。

> **Note**
>
> 完成此操作后，您将无法再访问属于已删除的表的表版本和分区。 AWS Glue 会及时以异步方式删除这些“孤立”资源，这由服务决定。
> 为了确保立即删除所有相关资源，在调用 `BatchDeleteTable` 之前，请使用 `DeleteTableVersion` 或 `BatchDeleteTableVersion` 以及 `DeletePartition` 或 `BatchDeletePartition`，删除属于该表的所有资源。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

表所在的数据目录的 ID。如果未提供任何信息，则默认使用 AWS 账户 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的表所在的目录数据库的名称。对于 Hive 兼容性，此名称必须是完全小写的。

- TablesToDelete – 必填：UTF-8 字符串数组，不超过 100 个字符串。

  要删除的表的列表。

- TransactionId – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Custom string pattern #16 匹配。

  在该 ID 处删除表内容的事务 ID。

响应

- Errors – TableError 对象的数组。

  尝试删除指定表时遇到的错误的列表。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ResourceNotReadyException

GetTable 动作（Python：get_table）

在指定表的数据目录中检索 Table 定义。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

表所在的数据目录的 ID。如果未提供任何信息，则默认使用 AWS 账户 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的目录中的数据库的名称。对于 Hive 兼容性，此名称必须是完全小写的。

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索其定义的表的名称。对于 Hive 兼容性，此名称必须是完全小写的。

- TransactionId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #16 匹配。

  在该 ID 处读取表内容的事务 ID。

- QueryAsOfTime – 时间戳。

  截至读取表内容的时间。如果未设置，将使用最近的事务提交时间。无法与 TransactionId 一起指定。

响应

- Table – 一个 表 对象。

  用于定义指定表的 Table 对象。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ResourceNotReadyException
- FederationSourceException
- FederationSourceRetryableException

GetTables 动作（Python：get_tables）

在给定的 Database 中检索部分或所有表的定义。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据目录的 ID。如果未提供任何信息，则默认使用 AWS 账户 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  目录中要列出其表的数据库。对于 Hive 兼容性，此名称必须是完全小写的。

- Expression – UTF-8 字符串，长度不超过 2048 个字节，与 Single-line string pattern 匹配。

  正则表达式模式。如果存在，则只返回其名称与模式匹配的表。

- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用，则包括)。

- MaxResults – 数字（整数），不小于 1 或大于 100。

  要在单个响应中返回的表的最大数量。

- TransactionId – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Custom string pattern #16 匹配。

  在该 ID 处读取表内容的事务 ID。

- QueryAsOfTime – 时间戳。

  截至读取表内容的时间。如果未设置，将使用最近的事务提交时间。无法与 TransactionId 一起指定。

响应

- TableList – 表 对象的数组。

  请求的 Table 对象的列表。

- NextToken – UTF-8 字符串。

延续令牌 (如果当前列表片段不是最后一个，则呈现)。

错误

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- GlueEncryptionException
- FederationSourceException
- FederationSourceRetryableException

GetTableVersion 操作（Python：获取表格版本）

检索表的指定版本。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据目录的 ID。如果未提供任何信息，则默认使用 AWS 账户 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的目录中的数据库。对于 Hive 兼容性，此名称必须是完全小写的。

- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的名称。对于 Hive 兼容性，此名称必须是完全小写的。

- VersionId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索的表版本的 ID 值。VersionID 是整数的字符串表示。每个版本都会增加 1。

响应

- TableVersion – 一个 TableVersion 对象。

  请求的表版本。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

GetTableVersions 操作（Python：获取表格版本）

检索标识指定表的可用版本的字符串的列表。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据目录的 ID。如果未提供任何信息，则默认使用 AWS 账户 ID。
- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的目录中的数据库。对于 Hive 兼容性，此名称必须是完全小写的。
- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的名称。对于 Hive 兼容性，此名称必须是完全小写的。
- NextToken – UTF-8 字符串。

  延续标记 (如果这不是第一次调用)。
- MaxResults – 数字（整数），不小于 1 或大于 100。

  要在一个响应中返回的表版本的最大数量。

响应

- TableVersions – TableVersion 对象的数组。

  标识指定表的可用版本的字符串的列表。

- NextToken – UTF-8 字符串。

  延续令牌 (如果可用版本的列表不包括最后一个)。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

DeleteTableVersion 操作（Python：删除表格版本）

删除表的指定版本。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据目录的 ID。如果未提供任何信息，则默认使用 AWS 账户 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的目录中的数据库。对于 Hive 兼容性，此名称必须是完全小写的。

- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的名称。对于 Hive 兼容性，此名称必须是完全小写的。

- VersionId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

要删除的表版本的 ID。VersionID 是整数的字符串表示。每个版本都会增加 1。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchDeleteTableVersion 操作（Python：batch_delete_table_version）

删除表的版本的指定批次。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据目录的 ID。如果未提供任何信息，则默认使用 AWS 账户 ID。
- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的目录中的数据库。对于 Hive 兼容性，此名称必须是完全小写的。
- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的名称。对于 Hive 兼容性，此名称必须是完全小写的。
- VersionIds – 必填：UTF-8 字符串数组，不超过 100 个字符串。

  要删除版本的 ID 的列表。VersionId 是整数的字符串表示。每个版本都会增加 1。

响应

- Errors – TableVersionError 对象的数组。

  尝试删除指定表版本时遇到的错误的列表。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

SearchTables 操作（Python：搜索表）

根据表元数据以及父数据库中的属性搜索一组表。您可以根据文本或筛选条件进行搜索。

您只能基于 Lake Formation 中定义的安全策略来获取有权访问的表。您至少需要具有该表的只读访问权才能返回该表。如果您无权访问表中的所有列，则在将表列表返回给您时，不会根据这些列进行搜索。如果您有权访问这些列，但不能访问这些列中的数据，则这些列及其关联元数据将包含在搜索中。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  唯一标识符，包含 *account_id*。
- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用，则包括)。
- Filters – PropertyPredicate 对象的数组。

  键值对列表以及用于筛选搜索结果的比较器。返回与谓词匹配的所有实体。

  PropertyPredicate 结构的 Comparator 成员仅用于时间字段，并且可以省略其他字段类型。此外，当比较字符串值时，例如 Key=Name，则使用模糊匹配算法。Key 字段（例如，Name 字段值）将拆分为某些标点符号字符，例如 -、:、# 等来成为令牌。然后，每个令牌都与 PropertyPredicate 成员的 Value 完全匹配。例如，如果是 Key=Name 和 Value=link，表名为 customer-link 并且返回 xx-link-yy，但不返回 xxlinkyy。

- SearchText – 值字符串，不超过 1024 个字节。

  一个用于文本搜索的字符串。

  根据与值的精确匹配在引号筛选器中指定值。

- SortCriteria – SortCriterion 对象的数组，不超过 1 个结构。

  用于按字段名称对结果进行升序或降序排序的条件的列表。

- MaxResults – 数字（整数），不小于 1 或大于 1000。

  要在单个响应中返回的表的最大数量。

- ResourceShareType – UTF-8 字符串（有效值：FOREIGN | ALL | FEDERATED）。

  允许您指定要搜索与您的账户共享的表。允许的值是 FOREIGN 或 ALL。

  - 如果设置为 FOREIGN，将搜索与您的账户共享的表。
  - 如果设置为 ALL，将列出与您的账户共享的表，以及本账户中的表。

响应

- NextToken – UTF-8 字符串。

  延续令牌 (如果当前列表片段不是最后一个，则呈现)。

- TableList – 表 对象的数组。

  请求的 Table 对象的列表。SearchTables 响应仅返回您有权访问的表。

错误

- InternalServiceException
- InvalidInputException
- OperationTimeoutException

GetPartitionIndexes 操作（Python：获取分区索引）

检索与表关联的分区索引。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据目录的 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  指定要从中检索分区索引的数据库的名称。

- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  指定要为其检索分区索引的表的名称。

- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用，则包括)。

响应

- PartitionIndexDescriptorList – PartitionIndexDescriptor 对象的数组。

  索引描述符的列表。

- NextToken – UTF-8 字符串。

  延续令牌 (如果当前列表片段不是最后一个，则呈现)。

错误

- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- EntityNotFoundException
- ConflictException

CreatePartitionIndex 操作（Python：create_partition_index）

在现有表中创建指定的分区索引。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据目录的 ID。

- `DatabaseName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  指定要创建分区索引的数据库的名称。

- `TableName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  指定要创建分区索引的表的名称。

- `PartitionIndex` – 必填：一个 PartitionIndex 对象。

  指定 `PartitionIndex` 结构在现有表中创建分区索引。

响应

- 无响应参数。

错误

- `AlreadyExistsException`
- `InvalidInputException`
- `EntityNotFoundException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeletePartitionIndex 操作（Python：delete_partition_index）

在现有表中删除指定的分区索引。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据目录的 ID。

- `DatabaseName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  指定要从中删除分区索引的数据库的名称。

- `TableName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  指定要从中删除分区索引的表的名称。

- `IndexName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的分区索引的名称。

响应

- 无响应参数。

错误

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `EntityNotFoundException`
- `ConflictException`
- `GlueEncryptionException`

GetColumnStatisticsForTable 操作（Python：获取表格的列统计信息）

检索列的表统计数据信息。

此操作所需的 Identity and Access Management（IAM）权限是 `GetTable`。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  所涉分区所在的数据目录的 ID。如果未提供任何信息，则默认使用 AWS 账户 ID。
- `DatabaseName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区所在的目录数据库的名称。
- `TableName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区的表的名称。
- `ColumnNames` – 必填：UTF-8 字符串数组，不超过 100 个字符串。

  列名称的列表。

响应

- `ColumnStatisticsList` – ColumnStatistics 对象的数组。

  清单 ColumnStatistics.
- `Errors` – ColumnError 对象的数组。

  未能检索到其中的清单。 ColumnStatistics

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

- GlueEncryptionException

UpdateColumnStatisticsForTable 操作（Python：更新表格的列统计信息）

创建或更新列的表统计数据信息。

此操作所需的 Identity and Access Management（IAM）权限是 UpdateTable。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  所涉分区所在的数据目录的 ID。如果未提供任何信息，则默认使用 AWS 账户 ID。
- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区所在的目录数据库的名称。
- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区的表的名称。
- ColumnStatisticsList – 必填：ColumnStatistics 对象的数组，不超过 25 个结构。

  列统计数据的列表。

响应

- Errors – ColumnStatisticsError 对象的数组。

  清单 ColumnStatisticsErrors.

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

- GlueEncryptionException

DeleteColumnStatisticsForTable 操作（Python：表格的删除列统计数据）

检索列的表统计数据信息。

此操作所需的 Identity and Access Management（IAM）权限是 DeleteTable。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  所涉分区所在的数据目录的 ID。如果未提供任何信息，则默认使用 AWS 账户 ID。
- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区所在的目录数据库的名称。
- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区的表的名称。
- ColumnName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  列的名称。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

分区 API

分区 API 介绍用于处理分区的数据类型和操作。

数据类型

- Partition 结构
- PartitionInput 结构
- PartitionSpecWithSharedStorageDescriptor 结构
- PartitionListComposingSpec 结构
- PartitionSpecProxy 结构
- PartitionValueList 结构
- Segment 结构
- PartitionError 结构
- BatchUpdatePartitionFailureEntry 结构
- BatchUpdatePartitionRequestEntry 结构
- StorageDescriptor 结构
- SchemaReference 结构
- SerDeInfo 结构
- SkewedInfo 结构

Partition 结构

表示表数据的一部分。

字段

- `Values` – UTF-8 字符串数组。

  分区的值。

- `DatabaseName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中创建分区的目录数据库的名称。

- `TableName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中创建分区的数据库表的名称。

- `CreationTime` – 时间戳。

  创建分区的时间。

- `LastAccessTime` – 时间戳。

  上次访问分区的时间。

- `StorageDescriptor` – 一个 StorageDescriptor 对象。

  提供有关存储分区的物理位置的信息。

- `Parameters` – 键值对的映射数组。

  每个键是一个键字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串，不超过 512000 个字节。

  这些键值对用于定义分区参数。

- `LastAnalyzedTime` – 时间戳。

  上次为该分区计算列统计信息的时间。

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区所在的数据目录的 ID。

PartitionInput 结构

用于创建和更新分区的结构。

字段

- `Values` – UTF-8 字符串数组。

  分区的值。尽管开发工具包不需要此参数，但您必须为有效输入指定此参数。

  新分区的键值必须作为字符串对象数组传递，这些对象的顺序必须与 Amazon S3 前缀中出现的分区键的顺序相同。否则，AWS Glue 会将值添加到错误的键。

- `LastAccessTime` – 时间戳。

  上次访问分区的时间。

- `StorageDescriptor` – 一个 StorageDescriptor 对象。

  提供有关存储分区的物理位置的信息。

- `Parameters` – 键值对的映射数组。

  每个键是一个键字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串，不超过 512000 个字节。

  这些键值对用于定义分区参数。

- `LastAnalyzedTime` – 时间戳。

  上次为该分区计算列统计信息的时间。

PartitionSpecWithSharedStorageDescriptor 结构

共享物理位置的分区的分区规范。

字段

- `StorageDescriptor` – 一个 StorageDescriptor 对象。

  共享的物理存储信息。

- `Partitions` – 分区 对象的数组。

  共享该物理位置的分区的列表。

PartitionListComposingSpec 结构

列出相关的分区。

字段

- `Partitions` – 分区 对象的数组。

  编制规范中的分区的列表。

PartitionSpecProxy 结构

提供指定分区的根路径。

字段

- DatabaseName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区所在的目录数据库。

- TableName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  包含分区的表的名称。

- RootPath – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  用于寻址分区的代理的根路径。

- PartitionSpecWithSharedSD – 一个 PartitionSpecWithSharedStorageDescriptor 对象。

  共享同一物理存储位置的分区的规范。

- PartitionListComposingSpec – 一个 PartitionListComposingSpec 对象。

  指定分区的列表。

PartitionValueList 结构

包含定义分区的值的列表。

字段

- Values – 必填：UTF-8 字符串数组。

  值的列表。

Segment 结构

定义表分区的非重叠区域，从而允许并行运行多个请求。

字段

- SegmentNumber – 必填：数字（整数），至多为“无”。

  此片段的从零开始的索引编号。例如，如果片段的总数为 4，则 SegmentNumber 值的范围为 0 到 3。
- TotalSegments – 必填：数字（整数），不小于 1 或大于 10。

  片段总数。

PartitionError 结构

包含有关分区错误的信息。

字段

- PartitionValues – UTF-8 字符串数组。

  用于定义分区的值。
- ErrorDetail – 一个 ErrorDetail 对象。

  有关分区错误的详细信息。

BatchUpdatePartitionFailureEntry 结构

包含有关批量更新分区错误的信息。

字段

- PartitionValueList – UTF-8 字符串数组，不超过 100 个字符串。

  用于定义分区的值的列表。
- ErrorDetail – 一个 ErrorDetail 对象。

  有关批量更新分区错误的详细信息。

BatchUpdatePartitionRequestEntry 结构

该结构包含用于更新分区的值和结构。

字段

- PartitionValueList – 必填：UTF-8 字符串数组，不超过 100 个字符串。

  用于定义分区的值的列表。
- PartitionInput – 必填：一个 PartitionInput 对象。

  用于更新分区的结构。

StorageDescriptor 结构

描述表数据的物理存储。

字段

- Columns – 列 对象的数组。

  表中的 Columns 的列表。
- Location – 位置字符串，不超过 2056 个字节，与 URI address multi-line string pattern 匹配。

  表的物理位置。默认情况下，它采用仓库位置的形式，后跟仓库中的数据库位置，然后是表名称。
- AdditionalLocations – UTF-8 字符串数组。

  指向 Delta 表所在路径的位置列表。
- InputFormat – 格式字符串，不超过 128 个字节，与 Single-line string pattern 匹配。

  输入格式：SequenceFileInputFormat（二进制）或 TextInputFormat 或自定义格式。
- OutputFormat – 格式字符串，不超过 128 个字节，与 Single-line string pattern 匹配。

  输出格式：SequenceFileOutputFormat（二进制）、IgnoreKeyTextOutputFormat 或自定义格式。
- Compressed – 布尔值。

  如果对表中的数据进行压缩，则为 True，否则为 False。
- NumberOfBuckets – 数字（整数）。

  如果表包含任何维度列，则必须指定。
- SerdeInfo – 一个 SerDeInfo 对象。

序列化/反序列化（SerDe）信息。

- `BucketColumns` – UTF-8 字符串数组。

  表中的 Reducer 分组列、集群列以及桶列的列表。

- `SortColumns` – 顺序 对象的数组。

  指定表中的每个桶的排序顺序的列表。

- `Parameters` – 键值对的映射数组。

  每个键是一个键字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串，不超过 512000 个字节。

  使用键/值形式的用户提供的属性。

- `SkewedInfo` – 一个 SkewedInfo 对象。

  有关在列中经常出现的值（偏斜值）的信息。

- `StoredAsSubDirectories` – 布尔值。

  如果表数据存储在子目录中，则为 `True`，否则为 `False`。

- `SchemaReference` – 一个 SchemaReference 对象。

  引用存储在 AWS Glue 架构注册表中的架构的对象。

  创建表时，可以为架构传递列的空列表，而使用架构引用。

SchemaReference 结构

引用存储在 AWS Glue 架构注册表中的架构的对象。

字段

- `SchemaId` – 一个 SchemaId 对象。

  包含架构标识字段的结构。必须提供此值或 `SchemaVersionId`。

- `SchemaVersionId` – UTF-8 字符串，长度不少于 36 个字节或超过 36 个字节，与 Custom string pattern #17 匹配。

  分配给架构版本的唯一 ID。必须提供此值或 `SchemaId`。

- SchemaVersionNumber – 数字（长度），不小于 1 或大于 100000。

  架构的版本号。

SerDeInfo 结构

有关序列化/反序列化程序（SerDe）的信息，它用作抽取器和加载器。

字段

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  SerDe 的名称。
- SerializationLibrary – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  通常是实现 SerDe 的类。例如，org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe。
- Parameters – 键值对的映射数组。

  每个键是一个键字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串，不超过 512000 个字节。

  这些键值对用于定义 SerDe 的初始化参数。

SkewedInfo 结构

指定表中的偏斜值。偏斜值是指出现频率很高的值。

字段

- SkewedColumnNames – UTF-8 字符串数组。

  包含偏斜值的列名称的列表。
- SkewedColumnValues – UTF-8 字符串数组。

  经常被认为是偏斜的值的列表。
- SkewedColumnValueLocationMaps – 键值对的映射数组。

每个键是一个 UTF-8 字符串。

每个值是一个 UTF-8 字符串。

偏斜值到包含它们的列的映射。

操作

- CreatePartition 操作（Python：create_partition）
- BatchCreatePartition 操作（Python：batch_create_partition）
- UpdatePartition 操作（Python：update_partition）
- DeletePartition 操作（Python：delete_partition）
- BatchDeletePartition 操作（Python：batch_delete_partition）
- GetPartition 操作（Python：get_partition）
- GetPartitions 操作（Python：get_partitions）
- BatchGetPartition 操作（Python：batch_get_partition）
- BatchUpdatePartition 操作（Python：batch_update_partition）
- GetColumnStatisticsForPartition 操作（Python：get_column_statistics_for_partition）
- UpdateColumnStatisticsForPartition 操作（Python：update_column_statistics_for_partition）
- DeleteColumnStatisticsForPartition 操作（Python：delete_column_statistics_for_partition）

CreatePartition 操作（Python：create_partition）

创建新的分区。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中创建分区的目录的 AWS 账户 ID。
- `DatabaseName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中创建分区的元数据数据库的名称。

- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中创建分区的元数据表的名称。

- PartitionInput – 必填：一个 PartitionInput 对象。

  一个用于定义要创建的分区的 PartitionInput 结构。

响应

- 无响应参数。

错误

- InvalidInputException
- AlreadyExistsException
- ResourceNumberLimitExceededException
- InternalServiceException
- EntityNotFoundException
- OperationTimeoutException
- GlueEncryptionException

BatchCreatePartition 操作（Python：batch_create_partition）

在批量操作中创建一个或多个分区。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中创建分区的目录的 ID。目前，它应该为 AWS 账户 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中创建分区的元数据数据库的名称。

- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中创建分区的元数据表的名称。

- PartitionInputList – 必填：PartitionInput 对象的数组，不超过 100 个结构。

  用于定义要创建的分区的 PartitionInput 结构的列表。

响应

- Errors – PartitionError 对象的数组。

  在尝试创建请求的分区时遇到错误。

错误

- InvalidInputException
- AlreadyExistsException
- ResourceNumberLimitExceededException
- InternalServiceException
- EntityNotFoundException
- OperationTimeoutException
- GlueEncryptionException

UpdatePartition 操作（Python：update_partition）

更新分区。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要更新的分区所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

所涉表所在的目录数据库的名称。

- `TableName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要更新的分区所在的表的名称。

- `PartitionValueList` – 必填：UTF-8 字符串数组，不超过 100 个字符串。

  用于定义分区更新的分区键值的列表。

- `PartitionInput` – 必填：一个 PartitionInput 对象。

  要将分区更新到的新分区对象。

  此 `Values` 属性无法更改。如果要更改分区的分区键值，请删除并重新创建分区。

响应

- 无响应参数。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeletePartition 操作（Python：delete_partition）

删除指定的分区。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的分区所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  所涉表所在的目录数据库的名称。

- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  包含要删除的分区的表的名称。

- PartitionValues – 必填：UTF-8 字符串数组。

  用于定义分区的值。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchDeletePartition 操作（Python：batch_delete_partition）

在批量操作中删除一个或多个分区。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的分区所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  所涉表所在的目录数据库的名称。

- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  包含要删除的分区的表的名称。

- PartitionsToDelete – 必填：PartitionValueList 对象的数组，不超过 25 个结构。

  用于定义要删除的分区的 PartitionInput 结构的列表。

响应

- Errors – PartitionError 对象的数组。

  在尝试删除请求的分区时遇到错误。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetPartition 操作（Python：get_partition）

检索有关指定分区的信息。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  所涉分区所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区所在的目录数据库的名称。

- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

分区的表的名称。

- PartitionValues – 必填：UTF-8 字符串数组。

  用于定义分区的值。

响应

- Partition – 一个 分区 对象。

  请求的信息，采用 Partition 对象的形式。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- FederationSourceException
- FederationSourceRetryableException

GetPartitions 操作（Python：get_partitions）

检索有关表中的分区的信息。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  所涉分区所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区所在的目录数据库的名称。

- `TableName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区的表的名称。

- `Expression` – 谓词字符串，不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  筛选要返回的分区的表达式。

  该表达式使用类似于 SQL `WHERE` 筛选条件子句的 SQL 语法。SQL 语句解析器 JSQLParser 可解析表达式。

  运算符：以下是您可以在 `Expression` API 调用中使用的运算符：

  =

  检查两个操作数的值是否相等；如果是，则条件成立。

  示例：假设“变量 a”保持 10，“变量 b”保持 20。

  (a = b) 不成立。

  < >

  检查两个操作数的值是否相等；如果值不相等，则条件成立。

  示例：(a < > b) 成立。

  >

  检查左操作数的值是否大于右操作数的值；如果是，则条件成立。

  示例：(a > b) 不成立。

  <

  检查左操作数的值是否小于右操作数的值；如果是，则条件成立。

  示例：(a < b) 成立。

  >=

  检查左操作数的值是否大于或等于右操作数的值；如果是，则条件成立。

  示例：(a >= b) 不成立。

<=

检查左操作数的值是否小于或等于右操作数的值；如果是，则条件成立。

示例：(a <= b) 成立。

AND、OR、IN、BETWEEN、LIKE、NOT、IS NULL

逻辑运算符。

支持的分区键类型：以下是受支持的分区键。

- `string`
- `date`
- `timestamp`
- `int`
- `bigint`
- `long`
- `tinyint`
- `smallint`
- `decimal`

如果遇到无效类型，则会引发异常。

以下列表显示了每种类型的有效运算符。定义爬网程序时，`partitionKey` 类型将创建为 `STRING`，以与目录分区兼容。

示例 API 调用：

Example

表 `twitter_partition` 有三个分区：

```
year = 2015
        year = 2016
        year = 2017
```

Example

获取 `year` 等于 2015 的分区

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
        --expression "year*=*'2015'"
```

Example

获取 year 2016 至 2018 （不含）之间的分区

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
        --expression "year>'2016' AND year<'2018'"
```

Example

获取 year 2015 至 2018（含）之间的分区 以下 API 调用彼此等效：

```
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

通配符分区筛选条件，其中以下调用输出为分区年份=2017。LIKE 中不支持正则表达式。

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
        --expression "year LIKE '%7'"
```

- NextToken – UTF-8 字符串。

  延续令牌 (如果这不是检索这些分区的第一个调用)。

- Segment – 一个 Segment 对象。

  要在该请求中扫描的表的片段。

- MaxResults – 数字（整数），不小于 1 或大于 1000。

要在单个响应中返回的最大分区数。

- ExcludeColumnSchema – 布尔值。

  如果为真，则指定不返回分区列架构。当您只对分区值或位置等其他分区属性感兴趣时很有用。这种方法通过不返回重复数据来避免大型响应的问题。

- TransactionId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #16 匹配。

  在该 ID 处读取分区内容的事务 ID。

- QueryAsOfTime – 时间戳。

  截至读取分区内容的时间。如果未设置，将使用最近的事务提交时间。无法与 TransactionId 一起指定。

响应

- Partitions – 分区 对象的数组。

  请求的分区的列表。

- NextToken – UTF-8 字符串。

  延续令牌 (如果返回的分区列表不包括最后一个)。

错误

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- GlueEncryptionException
- InvalidStateException
- ResourceNotReadyException
- FederationSourceException
- FederationSourceRetryableException

BatchGetPartition 操作（Python：batch_get_partition）

在批处理请求中检索分区。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  所涉分区所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- `DatabaseName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区所在的目录数据库的名称。
- `TableName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区的表的名称。
- `PartitionsToGet` – 必填：PartitionValueList 对象的数组，不超过 1000 个结构。

  用于标识要检索的分区的分区值的列表。

响应

- `Partitions` – 分区 对象的数组。

  请求的分区的列表。
- `UnprocessedKeys` – PartitionValueList 对象数组，不超过 1000 个结构。

  请求中未返回分区的分区值的列表。

错误

- `InvalidInputException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `InternalServiceException`
- `GlueEncryptionException`

- InvalidStateException
- FederationSourceException
- FederationSourceRetryableException

BatchUpdatePartition 操作（Python：batch_update_partition）

在分批操作中更新一个或多个分区。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中更新分区的目录的 ID。目前，它应该为 AWS 账户 ID。
- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中更新分区的元数据数据库的名称。
- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中更新分区的元数据表的名称。
- Entries – 必填：BatchUpdatePartitionRequestEntry对象的数组，不少于 1 个或不超过 100 个结构。

  最多 100 个 BatchUpdatePartitionRequestEntry 对象进行更新的列表。

响应

- Errors – BatchUpdatePartitionFailureEntry 对象的数组。

  在尝试更新请求的分区时遇到错误。BatchUpdatePartitionFailureEntry 对象的列表。

错误

- InvalidInputException
- EntityNotFoundException

- `OperationTimeoutException`
- `InternalServiceException`
- `GlueEncryptionException`

GetColumnStatisticsForPartition 操作（Python：get_column_statistics_for_partition）

检索列的分区统计数据信息。

此操作所需的 Identity and Access Management（IAM）权限是 `GetPartition`。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  所涉分区所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- `DatabaseName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区所在的目录数据库的名称。
- `TableName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区的表的名称。
- `PartitionValues` – 必填：UTF-8 字符串数组。

  用于标识分区的分区值的列表。
- `ColumnNames` – 必填：UTF-8 字符串数组，不超过 100 个字符串。

  列名称的列表。

响应

- `ColumnStatisticsList` – ColumnStatistics 对象的数组。

  检索失败的 ColumnStatistics 的列表。
- `Errors` – ColumnError 对象的数组。

检索列统计数据时出错。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

UpdateColumnStatisticsForPartition 操作（Python：update_column_statistics_for_partition）

创建或更新分区统计数据列信息。

此操作所需的 Identity and Access Management（IAM）权限是 UpdatePartition。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  所涉分区所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区所在的目录数据库的名称。
- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区的表的名称。
- PartitionValues – 必填：UTF-8 字符串数组。

  用于标识分区的分区值的列表。
- ColumnStatisticsList – 必填：ColumnStatistics 对象的数组，不超过 25 个结构。

  列统计数据的列表。

响应

- `Errors` – ColumnStatisticsError 对象的数组。

  更新列统计数据时出错。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeleteColumnStatisticsForPartition 操作（Python：delete_column_statistics_for_partition）

删除列的分区列统计数据信息。

此操作所需的 Identity and Access Management（IAM）权限是 `DeletePartition`。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  所涉分区所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- `DatabaseName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区所在的目录数据库的名称。
- `TableName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分区的表的名称。
- `PartitionValues` – 必填：UTF-8 字符串数组。

  用于标识分区的分区值的列表。

- ColumnName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  列的名称。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

连接 API

连接 API 介绍 AWS Glue 连接数据类型，以及用于创建、删除、更新和列出连接的 API。

数据类型

- Connection 结构
- ConnectionInput 结构
- PhysicalConnectionRequirements 结构
- GetConnectionsFilter 结构

Connection 结构

定义与数据源的连接。

字段

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

连接定义的名称。

- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  连接的描述。

- ConnectionType – UTF-8 字符串（有效值：JDBC | SFTP | MONGODB | KAFKA | NETWORK | MARKETPLACE | CUSTOM | SALESFORCE）。

  连接的类型。目前不支持 SFTP。

- MatchCriteria – UTF-8 字符串数组，不超过 10 个字符串。

  可用于选择此连接的条件的列表。

- ConnectionProperties – 键值对的映射数组，不超过 100 对。

  每个键都是一个 UTF-8 字符串（有效值：HOST | PORT | USERNAME="USER_NAME" | PASSWORD | ENCRYPTED_PASSWORD | JDBC_DRIVER_JAR_URI | JDBC_DRIVER_CLASS_NAME | JDBC_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）。

  每个值是一个值字符串，不超过 1024 个字节。

  这些键值对用于定义连接的参数：

  - HOST- 主机 URI：完全限定域名 (FQDN) 或数据库主机的 IPv4 地址。

- PORT- 端口的端口号，介于 1024 和 65535 之间，数据库主机在这些端口上侦听数据库连接。
- USER_NAME- 登录数据库时使用的名称。USER_NAME 的值字符串为“USERNAME”。
- PASSWORD- 用户名使用的密码（如果使用的话）。
- ENCRYPTED_PASSWORD - 通过在数据目录加密设置中设置 ConnectionPasswordEncryption 来启用连接密码保护时，此字段会存储加密的密码。
- JDBC_DRIVER_JAR_URI - 包含要使用的 JDBC 驱动程序的 JAR 文件的 Amazon Simple Storage Service (Amazon S3) 路径。
- JDBC_DRIVER_CLASS_NAME - 要使用的 JDBC 驱动程序的类名称。
- JDBC_ENGINE - 要使用的 JDBC 引擎的名称。
- JDBC_ENGINE_VERSION - 要使用的 JDBC 引擎的版本。
- CONFIG_FILES - （留待将来使用。）
- INSTANCE_ID - 要使用的实例 ID。
- JDBC_CONNECTION_URL - 用于连接到 JDBC 数据源的 URL。
- JDBC_ENFORCE_SSL - 指定在客户端上建立 JDBC 连接时是否强制使用安全套接字层 (SSL) 及主机名匹配的布尔值字符串（true 或 false）。默认值为 false。
- CUSTOM_JDBC_CERT – 指明客户根证书的 Amazon S3 位置。AWS Glue 在连接到客户数据库时，使用此根证书来验证客户的证书。AWS Glue 仅处理 X.509 证书。提供的证书必须经过 DER 编码，并以 Base64 编码 PEM 格式提供。
- SKIP_CUSTOM_JDBC_CERT_VALIDATION – 默认情况下，这是 false。AWS Glue 将验证客户证书的签名算法和主题公有密钥算法。签名算法允许的算法仅包括 SHA256withRSA、SHA384withRSA 或 SHA512withRSA。对于主题公有密钥算法，密钥长度必须至少为 2048 位。您可以将该属性的值设置为 true 以跳过 AWS Glue 对客户证书进行的验证。
- CUSTOM_JDBC_CERT_STRING - 一个自定义的 JDBC 证书字符串，用于域匹配或可分辨名称匹配以防止中间人攻击。在 Oracle 数据库中，这将用作 SSL_SERVER_CERT_DN；在 Microsoft SQL Server 中，这将用作 hostNameInCertificate。
- CONNECTION_URL - 用于连接到一般（非 JDBC）数据源的 URL。
- SECRET_ID – 用于凭证的 Secret Manager 的密钥 ID。
- CONNECTOR_URL – MARKETPLACE 或自定义连接的连接器 URL。
- CONNECTOR_TYPE – MARKETPLACE 或自定义连接的连接器类型。
- CONNECTOR_CLASS_NAME – MARKETPLACE 或自定义连接的连接器类名称。
- KAFKA_BOOTSTRAP_SERVERS - 以逗号分隔的主机和端口对列表，它们是 Kafka 集群中 Apache Kafka 代理商的地址，Kafka 客户端将连接到该集群并自行引导。

- KAFKA_SSL_ENABLED – 是否启用或禁用 Apache Kafka 连接的 SSL。默认值为“true”。
- KAFKA_CUSTOM_CERT – 私有 CA 证书文件的 Amazon S3 URL（.pem 格式）。默认值是空字符串。
- KAFKA_SKIP_CUSTOM_CERT_VALIDATION – 是否跳过 CA 证书文件的验证。AWS Glue 验证三种算法：SHA256withRSA、SHA384withRSA 和 SHA512withRSA。默认值为“false”。
- KAFKA_CLIENT_KEYSTORE – 用于 Kafka 客户端身份验证的客户端密钥库文件的 Amazon S3 位置（可选）。
- KAFKA_CLIENT_KEYSTORE_PASSWORD – 用于访问提供的密钥库的密码（可选）。
- KAFKA_CLIENT_KEY_PASSWORD – 密钥库可以由多个密钥组成，因此这是与 Kafka 服务器端密钥一起使用的用于访问客户端密钥的密码（可选）。
- ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD – Kafka 客户端密钥库密码的加密版本（如果用户选中使用 AWS Glue 加密密码设置）。
- ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD – Kafka 客户端密钥密码的加密版本（如果用户选中使用 AWS Glue 加密密码设置）。
- KAFKA_SASL_MECHANISM – "SCRAM-SHA-512"、"GSSAPI"、"AWS_MSK_IAM" 或 "PLAIN"。这些是受支持的 SASL 机制。
- KAFKA_SASL_PLAIN_USERNAME – 用于使用“PLAIN”机制进行身份验证的明文用户名。
- KAFKA_SASL_PLAIN_PASSWORD – 用于使用“PLAIN”机制进行身份验证的明文密码。
- ENCRYPTED_KAFKA_SASL_PLAIN_PASSWORD – Kafka SASL PLAIN 密码的加密版本（如果用户选中使用 AWS Glue 加密密码设置）。
- KAFKA_SASL_SCRAM_USERNAME – 用于使用“SCRAM-SHA-512”机制进行身份验证的明文用户名。
- KAFKA_SASL_SCRAM_PASSWORD – 用于使用“SCRAM-SHA-512”机制进行身份验证的明文密码。
- ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD – Kafka SASL SCRAM 密码的加密版本（如果用户选中使用 AWS Glue 加密密码设置）。
- KAFKA_SASL_SCRAM_SECRETS_ARN - AWS Secrets Manager 中密钥的 Amazon 资源名称。
- KAFKA_SASL_GSSAPI_KEYTAB – Kerberos keytab 文件的 S3 位置。keytab 可存储一个或多个主体的长期密钥。有关更多信息，请参阅 MIT Kerberos 文档：keytab。
- KAFKA_SASL_GSSAPI_KRB5_CONF – Kerberos krb5.conf 文件的 S3 位置。krb5.conf 可存储 Kerberos 配置信息，例如 KDC 服务器的位置。有关更多信息，请参阅 MIT Kerberos 文档：krb5.conf。

  - KAFKA_SASL_GSSAPI_SERVICE – Kerberos 服务名称，如您的 Kafka 配置中的 sasl.kerberos.service.name 设置。
  - KAFKA_SASL_GSSAPI_PRINCIPAL – AWS Glue 使用的 Kerberos 主体的名称。有关更多信息，请参阅 Kafka 文档：配置 Kafka 代理。
- PhysicalConnectionRequirements – 一个 PhysicalConnectionRequirements 对象。

  成功建立此连接所需的物理连接要求，如虚拟私有云（VPC）和 SecurityGroup。
- CreationTime – 时间戳。

  创建此连接定义的时间的时间戳。
- LastUpdatedTime – 时间戳。

  上次更新此连接定义的时间的时间戳。
- LastUpdatedBy – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  上次更新此连接定义的用户、组或角色。
- Status – UTF-8 字符串（有效值：READY | IN_PROGRESS | FAILED）。

  连接的状态。可以为以下值之一：READY、IN_PROGRESS 或 FAILED。
- StatusReason – UTF-8 字符串，长度不少于 1 个字节，不超过 16384 个字节。

  连接状态原因。
- LastConnectionValidationTime – 时间戳。

  上次验证此连接的时间的时间戳。
- AuthenticationConfiguration – 一个 AuthenticationConfiguration 对象。

  连接的身份验证属性。

ConnectionInput 结构

用于指定要创建或更新的连接的结构。

字段

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

连接的名称。

- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  连接的描述。

- ConnectionType – 必填：UTF-8 字符串（有效值：JDBC | SFTP | MONGODB | KAFKA | NETWORK | MARKETPLACE | CUSTOM | SALESFORCE）。

  连接的类型。目前，支持以下类型：

  - JDBC - 通过 Java 数据库连接（JDBC）指定与数据库的连接。

    JDBC 连接使用以下 ConnectionParameters。

    - 必需：所有（HOST、PORT、JDBC_ENGINE）或 JDBC_CONNECTION_URL。
    - 必需：所有（USERNAME、PASSWORD）或 SECRET_ID。
    - 可选：JDBC_ENFORCE_SSL、CUSTOM_JDBC_CERT、CUSTOM_JDBC_CERT_STRING、SKIP_CUSTOM_
      这些参数用于通过 JDBC 配置 SSL。
  - KAFKA - 指定与 Apache Kafka 流平台的连接。

    KAFKA 连接使用以下 ConnectionParameters。

    - 必需：KAFKA_BOOTSTRAP_SERVERS。
    - 可选：KAFKA_SSL_ENABLED、KAFKA_CUSTOM_CERT、KAFKA_SKIP_CUSTOM_CERT_VALIDATION。
      这些参数用于通过 KAFKA 配置 SSL。
    - 可选：KAFKA_CLIENT_KEYSTORE、KAFKA_CLIENT_KEYSTORE_PASSWORD、KAFKA_CLIENT_KEY_P
      这些参数用于在 KAFKA 中通过 SSL 配置 TLS 客户端配置。
    - 可选：KAFKA_SASL_MECHANISM。可以指定为 SCRAM-SHA-512、GSSAPI 或 AWS_MSK_IAM。
    - 可选：KAFKA_SASL_SCRAM_USERNAME、KAFKA_SASL_SCRAM_PASSWORD、ENCRYPTED_KAFKA_SAS
      这些参数用于通过 KAFKA 配置 SASL/SCRAM-SHA-512 身份验证。

  - 可
    选：KAFKA_SASL_GSSAPI_KEYTAB、KAFKA_SASL_GSSAPI_KRB5_CONF、KAFKA_SASL_GSSAPI_
    这些参数用于通过 KAFKA 配置 SASL/GSSAPI 身份验证。
- MONGODB - 指定与 MongoDB 文档数据库的连接。

  MONGODB 连接使用以下 ConnectionParameters。

  - 必需：CONNECTION_URL。
  - 必需：所有（USERNAME、PASSWORD）或 SECRET_ID。
- SALESFORCE - 使用 OAuth 身份验证指定指向 Salesforce 的连接。
  - 需要配置 AuthenticationConfiguration 成员。
- NETWORK - 指定到 Amazon Virtual Private Cloud 环境（Amazon VPC）中的数据源的网络连接。

  NETWORK 连接不需要 ConnectionParameters。相反，提供 PhysicalConnectionRequirements。
- MARKETPLACE – 使用从 AWS Marketplace 购买的连接器中包含的配置设置来读取和写入 AWS Glue 本地不支持的数据存储。

  MARKETPLACE 连接使用以下 ConnectionParameters。

  - 必
    需：CONNECTOR_TYPE、CONNECTOR_URL、CONNECTOR_CLASS_NAME、CONNECTION_URL。
  - JDBC CONNECTOR_TYPE 连接必需：所有（USERNAME、PASSWORD）或 SECRET_ID。
- CUSTOM – 使用自定义连接器中包含的配置设置来读取和写入 AWS Glue 本地不支持的数据存储。

  不支持 SFTP。

  有关如何使用可选的 ConnectionProperties 配置 AWS Glue 中功能的更多信息，请参阅 AWS Glue 连接属性。

  有关如何使用可选的 ConnectionProperties 配置 AWS Glue Studio 中功能的更多信息，请参阅使用连接器和连接。
- MatchCriteria – UTF-8 字符串数组，不超过 10 个字符串。

  可用于选择此连接的条件的列表。
- ConnectionProperties – 必填：键值对的映射数组，不超过 100 对。

  每个键都是一个 UTF-8 字符串（有效值：HOST | PORT | USERNAME="USER_NAME" | PASSWORD | ENCRYPTED_PASSWORD | JDBC_DRIVER_JAR_URI | JDBC_DRIVER_CLASS_NAME

| JDBC_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)。

每个值是一个值字符串，不超过 1024 个字节。

这些键值对用于定义连接的参数。

- PhysicalConnectionRequirements – 一个 PhysicalConnectionRequirements 对象。

  成功建立此连接所需的物理连接要求，如虚拟私有云（VPC）和 SecurityGroup。

- AuthenticationConfiguration – 一个 AuthenticationConfigurationInput 对象。

  连接的身份验证属性。用于 Salesforce 连接。

- ValidateCredentials – 布尔值。

  用于在创建连接期间验证凭证的标志。用于 Salesforce 连接。默认设置为 true。

PhysicalConnectionRequirements 结构

GetConnection 响应中的 OAuth 客户端应用程序。

字段

- SubnetId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

连接使用的子网 ID。

- `SecurityGroupIdList` – UTF-8 字符串数组，不超过 50 个字符串。

  连接使用的安全组 ID 列表。

- `AvailabilityZone` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  连接的可用区。

GetConnectionsFilter 结构

筛选由 `GetConnections` API 操作返回的连接定义。

字段

- `MatchCriteria` – UTF-8 字符串数组，不超过 10 个字符串。

  一个条件字符串，它必须与连接定义中记录的条件相匹配，才能返回连接定义。

- `ConnectionType` – UTF-8 字符串（有效值：`JDBC` | `SFTP` | `MONGODB` | `KAFKA` | `NETWORK` | `MARKETPLACE` | `CUSTOM` | `SALESFORCE`）。

  要返回的连接的类型。目前不支持 SFTP。

操作

- CreateConnection 操作（Python：create_connection）
- DeleteConnection 操作（Python：delete_connection）
- GetConnection 操作（Python：get_connection）
- GetConnections 操作（Python：get_connections）
- UpdateConnection 操作（Python：update_connection）
- BatchDeleteConnection 操作（Python：batch_delete_connection）

CreateConnection 操作（Python：create_connection）

在数据目录中创建连接定义。

用于创建联合资源的连接需要 IAM `glue:PassConnection` 权限。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中创建连接的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。

- `ConnectionInput` – 必填：一个 ConnectionInput 对象。

  用于定义要创建的连接的 `ConnectionInput` 对象。

- `Tags` – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  您分配给连接的标签。

响应

- `CreateConnectionStatus` – UTF-8 字符串（有效值：`READY` | `IN_PROGRESS` | `FAILED`）。

  连接创建请求的状态。对于某些身份验证类型，请求可能需要一些时间，例如在 VPC 上创建包含令牌交换 OAuth 连接时。

错误

- `AlreadyExistsException`
- `InvalidInputException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `GlueEncryptionException`

DeleteConnection 操作（Python：delete_connection）

从数据目录中删除连接。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  连接所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- ConnectionName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的连接的名称。

响应

- 无响应参数。

错误

- EntityNotFoundException
- OperationTimeoutException

GetConnection 操作（Python：get_connection）

从数据目录中检索连接定义。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  连接所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索的连接定义的名称。
- HidePassword – 布尔值。

允许您在不返回密码的情况下检索连接元数据。例如，AWS Glue 控制台使用此标记来检索连接，并且不显示密码。当调用方可能无权使用 AWS KMS 密钥解密密码，但有权访问其余连接属性时，请设置此参数。

响应

- `Connection` – 一个 Connection 对象。

  请求的连接定义。

错误

- `EntityNotFoundException`
- `OperationTimeoutException`
- `InvalidInputException`
- `GlueEncryptionException`

GetConnections 操作（Python：get_connections）

从数据目录中检索连接定义的列表。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  连接所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- `Filter` – 一个 GetConnectionsFilter 对象。

  控制将返回哪些连接的筛选器。
- `HidePassword` – 布尔值。

  允许您在不返回密码的情况下检索连接元数据。例如，AWS Glue 控制台使用此标记来检索连接，并且不显示密码。当调用方可能无权使用 AWS KMS 密钥解密密码，但有权访问其余连接属性时，请设置此参数。
- `NextToken` – UTF-8 字符串。

延续标记 (如果这是延续调用)。

- MaxResults – 数字 (整数) ，不小于 1 或大于 1000。

  要在一个响应中返回的连接的最大数量。

响应

- ConnectionList – Connection 对象的数组。

  请求的连接定义的列表。

- NextToken – UTF-8 字符串。

  延续令牌 (如果返回的连接列表不包括最后一个筛选的连接)。

错误

- EntityNotFoundException
- OperationTimeoutException
- InvalidInputException
- GlueEncryptionException

UpdateConnection 操作 (Python：update_connection)

在数据目录中更新连接定义。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  连接所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要更新的连接定义的名称。

- ConnectionInput – 必填：一个 ConnectionInput 对象。

用于重新定义所涉连接的 ConnectionInput 对象。

响应

- 无响应参数。

错误

- InvalidInputException
- EntityNotFoundException
- OperationTimeoutException
- InvalidInputException
- GlueEncryptionException

BatchDeleteConnection 操作（Python：batch_delete_connection）

从数据目录中删除连接定义的列表。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  连接所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- ConnectionNameList – 必填：UTF-8 字符串数组，不超过 25 个字符串。

  要删除的连接的名称的列表。

响应

- Succeeded – UTF-8 字符串数组。

  已成功删除的连接定义的名称的列表。
- Errors – 键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

每个值都是一个 ErrorDetail 对象。

未成功删除的连接名称到错误详细信息的映射。

错误

- InternalServiceException
- OperationTimeoutException

身份验证配置

- AuthenticationConfiguration 结构
- AuthenticationConfigurationInput 结构
- OAuth2Properties 结构
- OAuth2PropertiesInput 结构
- OAuth2ClientApplication 结构
- AuthorizationCodeProperties 结构

AuthenticationConfiguration 结构

包含身份验证配置的结构。

字段

- AuthenticationType – UTF-8 字符串（有效值：BASIC | OAUTH2 | CUSTOM）。

  包含身份验证配置的结构。

- SecretArn – UTF-8 字符串，与 Custom string pattern #11 匹配。

  用于存储凭证的 Secrets Manager ARN。

- OAuth2Properties – 一个 OAuth2Properties 对象。

  OAuth2 身份验证的属性。

AuthenticationConfigurationInput 结构

在 CreateConnection 请求中包含身份验证配置的结构。

字段

- `AuthenticationType` – UTF-8 字符串（有效值：`BASIC` | `OAUTH2` | `CUSTOM`）。

  在 CreateConnection 请求中包含身份验证配置的结构。

- `SecretArn` – UTF-8 字符串，与 Custom string pattern #11 匹配。

  用于在 CreateConnection 请求中存储凭证的 Secrets Manager ARN。

- `OAuth2Properties` – 一个 OAuth2PropertiesInput 对象。

  CreateConnection 请求中 OAuth2 身份验证的属性。

OAuth2Properties 结构

包含 OAuth2 身份验证属性的结构。

字段

- `OAuth2GrantType` – UTF-8 字符串（有效值：`AUTHORIZATION_CODE` | `CLIENT_CREDENTIALS` | `JWT_BEARER`）。

  OAuth2 授权类型。例如，`AUTHORIZATION_CODE`、`JWT_BEARER` 或 `CLIENT_CREDENTIALS`。

- `OAuth2ClientApplication` – 一个 OAuth2ClientApplication 对象。

  客户端应用程序类型。例如，AWS_MANAGED 或 USER_MANAGED。

- `TokenUrl` – UTF-8 字符串，长度不超过 256 个字节，与 Custom string pattern #12 匹配。

  提供程序身份验证服务器的 URL，用于以授权代码交换访问令牌。

- `TokenUrlParametersMap` – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值都是一个 UTF-8 字符串，长度不少于 1 个字节，不超过 512 个字节。

  添加到令牌 `GET` 请求中的参数的映射。

OAuth2PropertiesInput 结构

在 CreateConnection 请求中包含 OAuth2 属性的结构。

字段

- OAuth2GrantType – UTF-8 字符串（有效值：AUTHORIZATION_CODE | CLIENT_CREDENTIALS | JWT_BEARER）。

  CreateConnection 请求中的 OAuth2 授权类型。例如，AUTHORIZATION_CODE、JWT_BEARER 或 CLIENT_CREDENTIALS。
- OAuth2ClientApplication – 一个 OAuth2ClientApplication 对象。

  CreateConnection 请求中的客户端应用程序类型。例如，AWS_MANAGED 或 USER_MANAGED。
- TokenUrl – UTF-8 字符串，长度不超过 256 个字节，与 Custom string pattern #12 匹配。

  提供程序身份验证服务器的 URL，用于以授权代码交换访问令牌。
- TokenUrlParametersMap – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值都是一个 UTF-8 字符串，长度不少于 1 个字节，不超过 512 个字节。

  添加到令牌 GET 请求中的参数的映射。
- AuthorizationCodeProperties – 一个 AuthorizationCodeProperties 对象。

  OAuth2 AUTHORIZATION_CODE 授权类型所需的属性集。

OAuth2ClientApplication 结构

用于连接的 OAuth2 客户端应用程序。

字段

- UserManagedClientApplicationClientId – UTF-8 字符串，长度不超过 2048 个字节，与 Custom string pattern #13 匹配。

  如果 ClientAppType 是 USER_MANAGED，则为客户端应用程序 clientID。
- AWSManagedClientApplicationReference – UTF-8 字符串，长度不超过 2048 个字节，与 Custom string pattern #13 匹配。

对 AWS 托管的 SaaS 端客户端应用程序的引用。

AuthorizationCodeProperties 结构

OAuth2 AUTHORIZATION_CODE 授权类型工作流所需的属性集。

字段

- AuthorizationCode – UTF-8 字符串，长度不少于 1 个字节，不超过 4096 个字节，与 Custom string pattern #13 匹配。

  将在 AUTHORIZATION_CODE 授权工作流的第三分支中使用的授权代码。这是一次性代码，一旦用于交换访问令牌后就会失效，因此可以将此值作为请求参数。

- RedirectUri – UTF-8 字符串，长度不超过 512 个字节，与 Custom string pattern #14 匹配。

  重定向 URI，在发布授权代码时授权服务器将用户重定向到此处。随后，在将授权代码交换为访问令牌时使用该 URI。

用户定义的函数 API

用户定义的函数 API 介绍用于处理函数的 AWS Glue 数据类型和操作。

数据类型

- UserDefinedFunction 结构
- UserDefinedFunctionInput 结构

UserDefinedFunction 结构

表示与 Hive 用户定义函数 (UDF) 定义等效的函数。

字段

- FunctionName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  函数的名称。

- DatabaseName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  包含函数的目录数据库的名称。

- ClassName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  包含函数代码的 Java 类。

- OwnerName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  参数的所有者。

- OwnerType – UTF-8 字符串（有效值：USER | ROLE | GROUP）。

  所有者类型。

- CreateTime – 时间戳。

  创建函数的时间。

- ResourceUris – ResourceUri 对象数组，不超过 1000 个结构。

  函数的资源 URI。

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中创建函数的数据目录的 ID。

UserDefinedFunctionInput 结构

用于创建或更新用户定义函数的结构。

字段

- FunctionName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  函数的名称。

- ClassName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

包含函数代码的 Java 类。

- `OwnerName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  参数的所有者。

- `OwnerType` – UTF-8 字符串（有效值：`USER` | `ROLE` | `GROUP`）。

  所有者类型。

- `ResourceUris` – ResourceUri 对象数组，不超过 1000 个结构。

  函数的资源 URI。

操作

- CreateUserDefinedFunction 操作（Python：create_user_defined_function）
- UpdateUserDefinedFunction 操作（Python：update_user_defined_function）
- DeleteUserDefinedFunction 操作（Python：delete_user_defined_function）
- GetUserDefinedFunction 操作（Python：get_user_defined_function）
- GetUserDefinedFunctions 操作（Python：get_user_defined_functions）

CreateUserDefinedFunction 操作（Python：create_user_defined_function）

在数据目录中创建新函数定义。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中创建函数的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。

- `DatabaseName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要在其中创建函数的目录数据库的名称。

- `FunctionInput` – 必填：一个 UserDefinedFunctionInput 对象。

一个 FunctionInput 对象，它定义要在数据目录中创建的函数。

响应

- 无响应参数。

错误

- AlreadyExistsException
- InvalidInputException
- InternalServiceException
- EntityNotFoundException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- GlueEncryptionException

UpdateUserDefinedFunction 操作（Python：update_user_defined_function）

在数据目录中更新现有函数定义。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要更新的函数所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要更新的函数所在的目录数据库的名称。
- FunctionName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  函数的名称。
- FunctionInput – 必填：一个 UserDefinedFunctionInput 对象。

一个 FunctionInput 对象，它重新定义数据目录中的函数。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

DeleteUserDefinedFunction 操作（Python：delete_user_defined_function）

从数据目录中删除现有函数定义。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的函数所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  函数所在的目录数据库的名称。
- FunctionName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的函数定义的名称。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetUserDefinedFunction 操作（Python：get_user_defined_function）

从数据目录中检索指定的函数定义。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索的函数所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  函数所在的目录数据库的名称。
- FunctionName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  函数的名称。

响应

- UserDefinedFunction – 一个 UserDefinedFunction 对象。

  请求的函数定义。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException

- OperationTimeoutException
- GlueEncryptionException

GetUserDefinedFunctions 操作（Python：get_user_defined_functions）

从数据目录中检索多个函数定义。

请求

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索的函数所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。
- DatabaseName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  函数所在的目录数据库的名称。如果未提供任何内容，则将返回来自目录中所有数据库的函数。
- Pattern – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  一个可选的函数名称模式字符串，用于筛选返回的函数定义。
- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用)。
- MaxResults – 数字（整数），不小于 1 或大于 100。

  要在一个响应中返回的函数的最大数量。

响应

- UserDefinedFunctions – UserDefinedFunction 对象的数组。

  请求的函数定义的列表。
- NextToken – UTF-8 字符串。

  延续令牌 (如果返回函数的列表不包括最后一个请求的函数)。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `GlueEncryptionException`

将 Athena 目录导入 AWS Glue

迁移 API 介绍与将 Athena 数据目录迁移到 AWS Glue 有关的 AWS Glue 数据类型和操作。

数据类型

- CatalogImportStatus 结构

CatalogImportStatus 结构

包含迁移状态信息的结构。

字段

- `ImportCompleted` – 布尔值。

  如果迁移已完成，则为 `True`，否则为 `False`。
- `ImportTime` – 时间戳。

  启动迁移的时间。
- `ImportedBy` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  启动迁移的人员的姓名。

操作

- ImportCatalogToGlue 操作（Python：import_catalog_to_glue）
- GetCatalogImportStatus 操作（Python：get_catalog_import_status）

ImportCatalogToGlue 操作（Python：import_catalog_to_glue）

将现有的 Amazon Athena 数据目录导入到 AWS Glue。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要导入的目录的 ID。目前，它应该为 AWS 账户 ID。

响应

- 无响应参数。

错误

- `InternalServiceException`
- `OperationTimeoutException`

GetCatalogImportStatus 操作（Python：get_catalog_import_status）

检索迁移操作的状态。

请求

- `CatalogId` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要迁移的目录的 ID。目前，它应该为 AWS 账户 ID。

响应

- `ImportStatus` – 一个 CatalogImportStatus 对象。

  指定目录迁移的状态。

错误

- InternalServiceException
- OperationTimeoutException

表优化器 API

表优化器 API 描述了用于启用压缩以提高读取性能的 AWS Glue API。

数据类型

- TableOptimizer 结构
- TableOptimizerConfiguration 结构
- TableOptimizerRun 结构
- RunMetrics 结构
- BatchGetTableOptimizerEntry 结构
- BatchTableOptimizer 结构
- BatchGetTableOptimizerError 结构

TableOptimizer 结构

包含有关与表相关的优化器详细信息。

字段

- type – UTF-8 字符串（有效值：compaction="COMPACTION"）。

  优化器的类型。目前唯一有效的值是 compaction。

- configuration – 一个 TableOptimizerConfiguration 对象。

  在创建或更新表优化器时指定的 TableOptimizerConfiguration 对象。

- lastRun – 一个 TableOptimizerRun 对象。

  表示表优化器的上一次运行的 TableOptimizerRun 对象。

TableOptimizerConfiguration 结构

包含有关表优化器配置的详细信息。您可以在创建或更新表优化器时传递此配置。

字段

- roleArn – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节，与 Single-line string pattern 匹配。

  由调用方传递的角色，将向服务授予代表调用方更新与优化器关联的资源的权限。

- enabled – 布尔值。

  是否启用表优化。

TableOptimizerRun 结构

包含表优化器运行的详细信息。

字段

- eventType – UTF-8 字符串（有效值：starting="STARTING" | completed="COMPLETED" | failed="FAILED" | in_progress="IN_PROGRESS"）。

  一种表示表优化器运行状态的事件类型。

- startTimestamp – 时间戳。

  表示在 Lake Formation 中启动压缩作业的纪元时间戳。

- endTimestamp – 时间戳。

  表示压缩作业结束的纪元时间戳。

- metrics – 一个 RunMetrics 对象。

  包含有关优化器运行的指标的 RunMetrics 对象。

- error – UTF-8 字符串。

  优化器运行期间出现的错误。

RunMetrics 结构

有关优化器运行的指标。

字段

- NumberOfBytesCompacted – UTF-8 字符串。

  由压缩作业运行移除的字节数。

- NumberOfFilesCompacted – UTF-8 字符串。

  由压缩作业运行移除的文件数。

- NumberOfDpus – UTF-8 字符串。

  作业使用的 DPU 小时数。

- JobDurationInHour – UTF-8 字符串。

  作业的持续时间（以小时为单位）。

BatchGetTableOptimizerEntry 结构

表示要在 BatchGetTableOptimizer 操作中检索的表优化器。

字段

- catalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的目录 ID。

- databaseName – UTF-8 字符串，至少 1 个字节。

  表所在的目录中的数据库的名称。

- tableName – UTF-8 字符串，至少 1 个字节。

  表的名称。

- type – UTF-8 字符串（有效值：compaction="COMPACTION"）。

  优化器的类型。

BatchTableOptimizer 结构

包含 BatchGetTableOptimizer 操作返回的表优化器之一的详细信息。

字段

- catalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的目录 ID。
- databaseName – UTF-8 字符串，至少 1 个字节。

  表所在的目录中的数据库的名称。
- tableName – UTF-8 字符串，至少 1 个字节。

  表的名称。
- tableOptimizer – 一个 TableOptimizer 对象。

  包含有关表优化器配置和上次运行的详细信息的 TableOptimizer 对象。

BatchGetTableOptimizerError 结构

包含 BatchGetTableOptimizer 操作所返回错误列表中的某一个错误的详细信息。

字段

- error – 一个 ErrorDetail 对象。

  包含有关错误代码和错误消息详细信息的 ErrorDetail 对象。
- catalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的目录 ID。
- databaseName – UTF-8 字符串，至少 1 个字节。

  表所在的目录中的数据库的名称。
- tableName – UTF-8 字符串，至少 1 个字节。

  表的名称。

- type – UTF-8 字符串（有效值：compaction="COMPACTION"）。

  优化器的类型。

操作

- GetTableOptimizer 动作（Python：get_table_optimizer）
- BatchGetTableOptimizer 操作（Python：batch_get_table_optimizer）
- ListTableOptimizerRuns 操作（Python：list_table_optimizer_runs）
- CreateTableOptimizer 操作（Python：create_table_optimizer）
- DeleteTableOptimizer 操作（Python：delete_table_optimizer）
- UpdateTableOptimizer 操作（Python：update_table_optimizer）

GetTableOptimizer 动作（Python：get_table_optimizer）

返回与指定表关联的所有优化器的配置。

请求

- CatalogId –必填项：目录 id 字符串，长度不少于 1 个字节，并且不超过 255 个字节，并且符合 Single-line string pattern。

  表的目录 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的目录中的数据库的名称。

- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的名称。

- Type – 必填：UTF-8 字符串（有效值：compaction="COMPACTION"）。

  优化器的类型。

响应

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的目录 ID。
- DatabaseName – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的目录中的数据库的名称。
- TableName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的名称。
- TableOptimizer – 一个 TableOptimizer 对象。

  与指定表关联的优化器。

错误

- EntityNotFoundException
- InvalidInputException
- AccessDeniedException
- InternalServiceException

BatchGetTableOptimizer 操作（Python：batch_get_table_optimizer）

返回指定表优化器的配置。

请求

- Entries – 必填：BatchGetTableOptimizerEntry 对象的数组。

  指定要检索的表优化器的 BatchGetTableOptimizerEntry 对象列表。

响应

- TableOptimizers – BatchTableOptimizer 对象的数组。

BatchTableOptimizer 对象的列表。

- Failures – BatchGetTableOptimizerError 对象的数组。

  操作中所出现错误的列表。

错误

- InternalServiceException

ListTableOptimizerRuns 操作（Python：list_table_optimizer_runs）

列出特定表的之前优化器运行历史记录。

请求

- CatalogId –必填项：目录 id 字符串，长度不少于 1 个字节，并且不超过 255 个字节，并且符合 Single-line string pattern。

  表的目录 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的目录中的数据库的名称。

- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的名称。

- Type – 必填：UTF-8 字符串（有效值：compaction="COMPACTION"）。

  优化器的类型。目前唯一有效的值是 compaction。

- MaxResults – 数字（整数）。

  每次调用时将返回的最大优化器运行数。

- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用)。

响应

- CatalogId – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的目录 ID。
- DatabaseName – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的目录中的数据库的名称。
- TableName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的名称。
- NextToken – UTF-8 字符串。

  对返回的优化器运行列表进行分页的延续令牌（如果列表的当前段不是最后一段，则返回该令牌）。
- TableOptimizerRuns – TableOptimizerRun 对象的数组。

  与表关联的优化器运行的列表。

错误

- EntityNotFoundException
- AccessDeniedException
- InvalidInputException
- InternalServiceException

CreateTableOptimizer 操作（Python：create_table_optimizer）

为特定函数创建新的表优化器。compaction 是目前唯一支持的优化器类型。

请求

- CatalogId –必填项：目录 id 字符串，长度不少于 1 个字节，并且不超过 255 个字节，并且符合 Single-line string pattern。

  表的目录 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的目录中的数据库的名称。

- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的名称。

- Type – 必填：UTF-8 字符串（有效值：compaction="COMPACTION"）。

  优化器的类型。目前唯一有效的值是 compaction。

- TableOptimizerConfiguration – 必填：一个 TableOptimizerConfiguration 对象。

  表示表优化器的配置的 TableOptimizerConfiguration 对象。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InvalidInputException
- AccessDeniedException
- AlreadyExistsException
- InternalServiceException

DeleteTableOptimizer 操作（Python：delete_table_optimizer）

删除一个表的一个优化器以及所有相关元数据。将不再对该表执行优化。

请求

- CatalogId –必填项：目录 id 字符串，长度不少于 1 个字节，并且不超过 255 个字节，并且符合 Single-line string pattern。

  表的目录 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的目录中的数据库的名称。

- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的名称。

- Type – 必填：UTF-8 字符串（有效值：compaction="COMPACTION"）。

  优化器的类型。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InvalidInputException
- AccessDeniedException
- InternalServiceException

UpdateTableOptimizer 操作（Python：update_table_optimizer）

更新现有表优化器的配置。

请求

- CatalogId –必填项：目录 id 字符串，长度不少于 1 个字节，并且不超过 255 个字节，并且符合 Single-line string pattern。

  表的目录 ID。

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的目录中的数据库的名称。

- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的名称。

- Type – 必填：UTF-8 字符串（有效值：compaction="COMPACTION"）。

  优化器的类型。目前唯一有效的值是 compaction。

- TableOptimizerConfiguration – 必填：一个 TableOptimizerConfiguration 对象。

  表示表优化器的配置的 TableOptimizerConfiguration 对象。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InvalidInputException
- AccessDeniedException
- InternalServiceException

爬网程序和分类器 API

爬网程序和分类器 API 描述了 AWS Glue 爬网程序和分类器数据类型，并包含用于创建、删除、更新和列出爬网程序或分类器的 API。

主题

- 分类器 API
- 爬网程序 API
- 列统计数据 API
- 爬网程序计划程序 API

分类器 API

分类器 API 介绍 AWS Glue 分类器数据类型，并包含用于创建、删除、更新和列出分类器的 API。

数据类型

- Classifier 结构
- GrokClassifier 结构
- XMLClassifier 结构
- JsonClassifier 结构
- CsvClassifier 结构
- CreateGrokClassifierRequest 结构
- UpdateGrokClassifierRequest 结构
- CreateXMLClassifierRequest 结构
- UpdateXMLClassifierRequest 结构
- CreateJsonClassifierRequest 结构
- UpdateJsonClassifierRequest 结构
- CreateCsvClassifierRequest 结构
- UpdateCsvClassifierRequest 结构

Classifier 结构

分类器会在爬网任务期间触发。分类器检查给定文件的格式是否可以处理。如果可以处理，分类器将以与该数据格式匹配的 `StructType` 对象的形式创建一个模式。

您可以使用 AWS Glue 提供的标准分类器，或自行编写分类器，以便更好地对数据源进行分类和指定要用于它们的合适架构。分类器可以是 `grok` 分类器、`XML` 分类器、`JSON` 分类器或自定义 `CSV` 分类器，它由 `Classifier` 对象中的字段之一指定。

字段

- `GrokClassifier` – 一个 GrokClassifier 对象。

  使用 `grok` 的分类器。
- `XMLClassifier` – 一个 XMLClassifier 对象。

XML 内容的分类器。

- `JsonClassifier` – 一个 JsonClassifier 对象。

  JSON 内容的分类器。

- `CsvClassifier` – 一个 CsvClassifier 对象。

  逗号分隔值 (CSV) 的分类器。

GrokClassifier 结构

使用 grok 模式的分类器。

字段

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分类器的名称。

- `Classification` – 必填：UTF-8 字符串。

  与分类器匹配的数据格式（例如，Twitter、JSON、Omniture 日志等）的标识符。

- `CreationTime` – 时间戳。

  注册此分类器的时间。

- `LastUpdated` – 时间戳。

  上次更新此分类器的时间。

- `Version` – 数字（长型）。

  此分类器的版本。

- `GrokPattern` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 2048 个字节，与 A Logstash Grok string pattern 匹配。

  由此分类器应用于数据存储的 grok 模式。有关更多信息，请参阅编写自定义分类器中的“内置模式”。

- `CustomPatterns` – UTF-8 字符串，不超过 16000 个字节，与 URI address multi-line string pattern 匹配。

由此分类器定义的可选自定义 grok 模式。有关更多信息，请参阅编写自定义分类器中的“自定义模式”。

XMLClassifier 结构

XML 内容的分类器。

字段

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分类器的名称。

- Classification – 必填：UTF-8 字符串。

  与分类器匹配的数据格式的标识符。

- CreationTime – 时间戳。

  注册此分类器的时间。

- LastUpdated – 时间戳。

  上次更新此分类器的时间。

- Version – 数字（长型）。

  此分类器的版本。

- RowTag – UTF-8 字符串。

  XML 标签，用于指定包含正在分析的 XML 文档中的每个记录的元素。无法识别自结束元素（以 / > 结束）。可以分析仅包含属性的空行元素，只要它以结束标签结束（例如，<row item_a="A" item_b="B"></row> 可以，但 <row item_a="A" item_b="B" /> 不可以）。

JsonClassifier 结构

JSON 内容的分类器。

字段

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分类器的名称。
- CreationTime – 时间戳。

  注册此分类器的时间。
- LastUpdated – 时间戳。

  上次更新此分类器的时间。
- Version – 数字（长型）。

  此分类器的版本。
- JsonPath – 必填：UTF-8 字符串。

  一种分类器使用的 JsonPath 字符串，该字符串定义供分类器分类的 JSON 数据。AWS Glue 支持小部分适用于 JsonPath 的运算符，如编写 JsonPath 自定义分类器中所述。

CsvClassifier 结构

自定义 CSV 内容的分类器。

字段

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分类器的名称。
- CreationTime – 时间戳。

  注册此分类器的时间。
- LastUpdated – 时间戳。

  上次更新此分类器的时间。
- Version – 数字（长型）。

  此分类器的版本。

- `Delimiter` – UTF-8 字符串，长度不少于 1 个字节或超过 1 个字节，与 Custom string pattern #10 匹配。

  一个自定义符号，表示分隔行中每个列条目的内容。

- `QuoteSymbol` – UTF-8 字符串，长度不少于 1 个字节或超过 1 个字节，与 Custom string pattern #10 匹配。

  一个自定义符号，表示将内容组合为单个列值的内容。它必须与列分隔符不同。

- `ContainsHeader` – UTF-8 字符串（有效值：`UNKNOWN` | `PRESENT` | `ABSENT`）。

  指示 CSV 文件是否包含标头。

- `Header` – UTF-8 字符串数组。

  表示列名称的字符串列表。

- `DisableValueTrimming` – 布尔值。

  指定在标识列值类型之前不去除值。默认值为 `true`。

- `AllowSingleColumn` – 布尔值。

  允许处理仅包含一列的文件。

- `CustomDatatypeConfigured` – 布尔值。

  允许配置自定义数据类型。

- `CustomDatatypes` – UTF-8 字符串数组。

  自定义数据类型列表包括“BINARY”、“BOOLEAN”、“DATE”、“DECIMAL”、“DOUBLE”、“FLOAT”、“INT”、“LONG”、“SHORT”、

- `Serde` – UTF-8 字符串（有效值：`OpenCSVSerDe` | `LazySimpleSerDe` | `None`）。

  设置用于在分类器中处理 CSV 的 SerDe，该分类器将应用于 Data Catalog。有效值包括 `OpenCSVSerDe`、`LazySimpleSerDe` 和 `None`。当您想让爬网程序执行检测时，可以指定 `None` 值。

CreateGrokClassifierRequest 结构

为要创建的 `CreateClassifier` 指定 `grok` 分类器。

字段

- `Classification` – 必填：UTF-8 字符串。

  与分类器匹配的数据格式 (例如，Twitter、JSON、Omniture 日志、Amazon CloudWatch Logs 等) 的标识符。
- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  新分类器的名称。
- `GrokPattern` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 2048 个字节，与 A Logstash Grok string pattern 匹配。

  此分类器使用的 grok 模式。
- `CustomPatterns` – UTF-8 字符串，不超过 16000 个字节，与 URI address multi-line string pattern 匹配。

  此分类器使用的可选自定义 grok 模式。

UpdateGrokClassifierRequest 结构

指定要在传递到 `UpdateClassifier` 时更新的 grok 分类器。

字段

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  `GrokClassifier` 的名称。
- `Classification` – UTF-8 字符串。

  与分类器匹配的数据格式 (例如，Twitter、JSON、Omniture 日志、Amazon CloudWatch Logs 等) 的标识符。
- `GrokPattern` – UTF-8 字符串，不少于 1 个字节或超过 2048 个字节，与 A Logstash Grok string pattern 匹配。

  此分类器使用的 grok 模式。
- `CustomPatterns` – UTF-8 字符串，不超过 16000 个字节，与 URI address multi-line string pattern 匹配。

此分类器使用的可选自定义 grok 模式。

CreateXMLClassifierRequest 结构

为要创建的 CreateClassifier 指定 XML 分类器。

字段

- Classification – 必填：UTF-8 字符串。

  与分类器匹配的数据格式的标识符。

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分类器的名称。

- RowTag – UTF-8 字符串。

  XML 标签，用于指定包含正在分析的 XML 文档中的每个记录的元素。无法识别自结束元素（以 / > 结束）。可以分析仅包含属性的空行元素，只要它以结束标签结束（例如，<row item_a="A" item_b="B"></row> 可以，但 <row item_a="A" item_b="B" /> 不可以）。

UpdateXMLClassifierRequest 结构

指定要更新的 XML 分类器。

字段

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分类器的名称。

- Classification – UTF-8 字符串。

  与分类器匹配的数据格式的标识符。

- RowTag – UTF-8 字符串。

  XML 标签，用于指定包含正在分析的 XML 文档中的每个记录的元素。请注意，此标签无法识别自结束元素 (由 /> 结束)。可以分析仅包含属性的空行元素，只要它以结束标签结束（例如，<row

item_a="A" item_b="B"></row> 可以，但 <row item_a="A" item_b="B" /> 不可以）。

CreateJsonClassifierRequest 结构

为要创建的 CreateClassifier 指定 JSON 分类器。

字段

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分类器的名称。

- JsonPath – 必填：UTF-8 字符串。

  一种分类器使用的 JsonPath 字符串，该字符串定义供分类器分类的 JSON 数据。AWS Glue 支持小部分适用于 JsonPath 的运算符，如编写 JsonPath 自定义分类器中所述。

UpdateJsonClassifierRequest 结构

指定要更新的 JSON 分类器。

字段

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分类器的名称。

- JsonPath – UTF-8 字符串。

  一种分类器使用的 JsonPath 字符串，该字符串定义供分类器分类的 JSON 数据。AWS Glue 支持小部分适用于 JsonPath 的运算符，如编写 JsonPath 自定义分类器中所述。

CreateCsvClassifierRequest 结构

为要创建的 CreateClassifier 指定自定义 CSV 分类器。

字段

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分类器的名称。

- Delimiter – UTF-8 字符串，长度不少于 1 个字节或超过 1 个字节，与 Custom string pattern #10 匹配。

  一个自定义符号，表示分隔行中每个列条目的内容。

- QuoteSymbol – UTF-8 字符串，长度不少于 1 个字节或超过 1 个字节，与 Custom string pattern #10 匹配。

  一个自定义符号，表示将内容组合为单个列值的内容。必须与列分隔符不同。

- ContainsHeader – UTF-8 字符串（有效值：UNKNOWN | PRESENT | ABSENT）。

  指示 CSV 文件是否包含标头。

- Header – UTF-8 字符串数组。

  表示列名称的字符串列表。

- DisableValueTrimming – 布尔值。

  指定在标识列值类型之前不去除值。默认值为 true。

- AllowSingleColumn – 布尔值。

  允许处理仅包含一列的文件。

- CustomDatatypeConfigured – 布尔值。

  允许配置自定义数据类型。

- CustomDatatypes – UTF-8 字符串数组。

  创建受支持的自定义数据类型列表。

- Serde – UTF-8 字符串（有效值：OpenCSVSerDe | LazySimpleSerDe | None）。

  设置用于在分类器中处理 CSV 的 SerDe，该分类器将应用于 Data Catalog。有效值包括 OpenCSVSerDe、LazySimpleSerDe 和 None。当您想让爬网程序执行检测时，可以指定 None 值。

UpdateCsvClassifierRequest 结构

指定要更新的自定义 CSV 分类器。

字段

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分类器的名称。

- Delimiter – UTF-8 字符串，长度不少于 1 个字节或超过 1 个字节，与 Custom string pattern #10 匹配。

  一个自定义符号，表示分隔行中每个列条目的内容。

- QuoteSymbol – UTF-8 字符串，长度不少于 1 个字节或超过 1 个字节，与 Custom string pattern #10 匹配。

  一个自定义符号，表示将内容组合为单个列值的内容。它必须与列分隔符不同。

- ContainsHeader – UTF-8 字符串（有效值：UNKNOWN | PRESENT | ABSENT）。

  指示 CSV 文件是否包含标头。

- Header – UTF-8 字符串数组。

  表示列名称的字符串列表。

- DisableValueTrimming – 布尔值。

  指定在标识列值类型之前不去除值。默认值为 true。

- AllowSingleColumn – 布尔值。

  允许处理仅包含一列的文件。

- CustomDatatypeConfigured – 布尔值。

  指定配置自定义数据类型。

- CustomDatatypes – UTF-8 字符串数组。

  指定受支持的自定义数据类型列表。

- Serde – UTF-8 字符串（有效值：OpenCSVSerDe | LazySimpleSerDe | None）。

设置用于在分类器中处理 CSV 的 SerDe，该分类器将应用于 Data Catalog。有效值包括 `OpenCSVSerDe`、`LazySimpleSerDe` 和 `None`。当您想让爬网程序执行检测时，可以指定 `None` 值。

操作

- CreateClassifier 操作（Python：create_classifier）
- DeleteClassifier 操作（Python：delete_classifier）
- GetClassifier 操作（Python：get_classifier）
- GetClassifiers 操作（Python：get_classifiers）
- UpdateClassifier 操作（Python：update_classifier）

CreateClassifier 操作（Python：create_classifier）

在用户的账户中创建分类器。这可以是 `GrokClassifier`、`XMLClassifier`、`JsonClassifier` 或 `CsvClassifier`，具体取决于请求的哪个字段存在。

请求

- `GrokClassifier` – 一个 CreateGrokClassifierRequest 对象。

  一个指定要创建的分类器的 `GrokClassifier` 对象。
- `XMLClassifier` – 一个 CreateXMLClassifierRequest 对象。

  一个指定要创建的分类器的 `XMLClassifier` 对象。
- `JsonClassifier` – 一个 CreateJsonClassifierRequest 对象。

  一个指定要创建的分类器的 `JsonClassifier` 对象。
- `CsvClassifier` – 一个 CreateCsvClassifierRequest 对象。

  一个指定要创建的分类器的 `CsvClassifier` 对象。

响应

- 无响应参数。

错误

- `AlreadyExistsException`
- `InvalidInputException`
- `OperationTimeoutException`

DeleteClassifier 操作（Python：delete_classifier）

从数据目录中删除分类器。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的分类器名称。

响应

- 无响应参数。

错误

- `EntityNotFoundException`
- `OperationTimeoutException`

GetClassifier 操作（Python：get_classifier）

按名称检索分类器。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索的分类器的名称。

响应

- `Classifier` – 一个 分类器 对象。

  请求的分类器。

错误

- `EntityNotFoundException`
- `OperationTimeoutException`

GetClassifiers 操作（Python：get_classifiers）

列出数据目录中的所有分类器对象。

请求

- `MaxResults` – 数字（整数），不小于 1 或大于 1000。

  要返回的列表的大小 (可选)。

- `NextToken` – UTF-8 字符串。

  一个可选延续令牌。

响应

- `Classifiers` – 分类器 对象的数组。

  请求的分类器对象的列表。

- `NextToken` – UTF-8 字符串。

  一个延续令牌。

错误

- `OperationTimeoutException`

UpdateClassifier 操作（Python：update_classifier）

修改现有分类器（GrokClassifier、XMLClassifier、JsonClassifier 或 CsvClassifier，具体取决于存在的字段）。

请求

- GrokClassifier – 一个 UpdateGrokClassifierRequest 对象。

  一个包含已更新字段的 GrokClassifier 对象。
- XMLClassifier – 一个 UpdateXMLClassifierRequest 对象。

  一个包含已更新字段的 XMLClassifier 对象。
- JsonClassifier – 一个 UpdateJsonClassifierRequest 对象。

  一个包含已更新字段的 JsonClassifier 对象。
- CsvClassifier – 一个 UpdateCsvClassifierRequest 对象。

  一个包含已更新字段的 CsvClassifier 对象。

响应

- 无响应参数。

错误

- InvalidInputException
- VersionMismatchException
- EntityNotFoundException
- OperationTimeoutException

爬网程序 API

Crawler API 描述了 AWS Glue 抓取工具的数据类型，以及用于创建、删除、更新和列出抓取工具的 API。

数据类型

- Crawler 结构
- Schedule 结构
- CrawlerTargets 结构
- S3Target 结构
- S3 DeltaCatalogTarget 结构
- S3 DeltaDirectTarget 结构
- JdbcTarget 结构
- MongoDBTarget 结构
- DynamoDBTarget 结构
- DeltaTarget 结构
- IcebergTarget 结构
- HudiTarget 结构
- CatalogTarget 结构
- CrawlerMetrics 结构
- CrawlerHistory 结构
- CrawlsFilter 结构
- SchemaChangePolicy 结构
- LastCrawlInfo 结构
- RecrawlPolicy 结构
- LineageConfiguration 结构
- LakeFormationConfiguration 结构

Crawler 结构

指定一个爬网程序，该程序将检查数据源并使用分类器来尝试确定其架构。如果成功，该爬网程序将记录与 AWS Glue Data Catalog中的数据源相关的元数据。

字段

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

爬网程序的名称。

- `Role` – UTF-8 字符串。

  用于访问客户资源的 IAM 角色的 Amazon 资源名称（ARN），如 Amazon Simple Storage Service (Amazon S3) 数据。

- `Targets` – 一个 CrawlerTargets 对象。

  要爬网的目标的集合。

- `DatabaseName` – UTF-8 字符串。

  存储爬网程序输出的数据库的名称。

- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  爬网程序的描述。

- `Classifiers` – UTF-8 字符串数组。

  指定与爬网程序关联的自定义分类器的 UTF-8 字符串列表。

- `RecrawlPolicy` – 一个 RecrawlPolicy 对象。

  指定是否再次网络爬取整个数据集，还是仅网络爬取自上次爬网程序运行以来添加的文件夹的策略。

- `SchemaChangePolicy` – 一个 SchemaChangePolicy 对象。

  指定爬网程序的更新和删除行为的策略。

- `LineageConfiguration` – 一个 LineageConfiguration 对象。

  指定是否为爬网程序启用数据系统的配置。

- `State` – UTF-8 字符串（有效值：`READY` | `RUNNING` | `STOPPING`）。

  指示是爬网程序正在运行还是运行正在等待处理。

- `TablePrefix` – UTF-8 字符串，长度不超过 128 个字节。

  添加到创建的表的名称的前缀。

- `Schedule` – 一个 计划 对象。

  对于计划的爬网程序，是爬网程序运行时的计划。

- `CrawlElapsedTime` – 数字（长型）。

如果爬网程序正在运行，则包含自上次爬网开始已用的总时间。

- `CreationTime` – 时间戳。

  创建爬网程序的时间。

- `LastUpdated` – 时间戳。

  上次更新爬网程序的时间。

- `LastCrawl` – 一个 LastCrawlInfo 对象。

  上次爬网的状态，以及出错时可能显示的错误信息。

- `Version` – 数字（长型）。

  爬网程序的版本。

- `Configuration` – UTF-8 字符串。

  爬网程序配置信息。此受版本控制的 JSON 字符串允许用户指定爬网程序的行为的各个方面。有关更多信息，请参阅设置爬网程序配置选项。

- `CrawlerSecurityConfiguration` – UTF-8 字符串，长度不超过 128 个字节。

  该爬网程序将使用的 `SecurityConfiguration` 结构的名称。

- `LakeFormationConfiguration` – 一个 LakeFormationConfiguration 对象。

  指定爬网程序是否应使用爬网程序的 AWS Lake Formation 证书而不是 IAM 角色证书。

Schedule 结构

一个使用 `cron` 语句计划事件的计划对象。

字段

- `ScheduleExpression` – UTF-8 字符串。

  用于指定计划的 `cron` 表达式 (请参阅用于作业和爬网程序的基于时间的计划）。例如，要每天 12:15 UTC 运行某些任务，您应该指定：`cron(15 12 * * ? *)`。

- `State` – UTF-8 字符串（有效值：`SCHEDULED` | `NOT_SCHEDULED` | `TRANSITIONING`）。

  计划的状态。

CrawlerTargets 结构

指定要爬网的数据存储。

字段

- `S3Targets` – S3Target 对象的数组。

  指定 Amazon Simple Storage Service（Amazon S3）目标。
- `JdbcTargets` – JdbcTarget 对象的数组。

  指定 JDBC 目标。
- `MongoDBTargets` – MongoDBTarget 对象的数组。

  指定 Amazon DocumentDB 或 MongoDB 目标。
- `DynamoDBTargets` – DynamoDBTarget 对象的数组。

  指定 Amazon DynamoDB 目标。
- `CatalogTargets` – CatalogTarget 对象的数组。

  指定 AWS Glue Data Catalog 目标。
- `DeltaTargets` – DeltaTarget 对象的数组。

  指定 Delta 数据存储目标。
- `IcebergTargets` – IcebergTarget 对象的数组。

  指定 Apache Iceberg 数据存储目标。
- `HudiTargets` – HudiTarget 对象的数组。

  指定 Apache Hudi 数据存储目标。

S3Target 结构

指定 Amazon Simple Storage Service（Amazon S3）中的数据存储。

字段

- `Path` – UTF-8 字符串。

  Amazon S3 目标的路径。

- `Exclusions` – UTF-8 字符串数组。

  要从爬网中排除的 glob 模式的列表。有关更多信息，请参阅使用爬网程序为表编制目录。

- `ConnectionName` – UTF-8 字符串。

  允许作业或爬网程序在 Amazon Virtual Private Cloud（Amazon VPC）内访问 Amazon S3 中的数据的连接名称。

- `SampleSize` – 数字（整数）。

  设置网络爬取数据集中的示例文件时，每个叶文件夹中要网络爬取的文件数。如果未设置，则会网络爬取所有文件。有效值是介于 1 到 249 之间的整数。

- `EventQueueArn` – UTF-8 字符串。

  有效的 Amazon SQS ARN。例如，`arn:aws:sqs:region:account:sqs`。

- `DlqEventQueueArn` – UTF-8 字符串。

  有效的 Amazon 死信 SQS ARN。例如，`arn:aws:sqs:region:account:deadLetterQueue`。

S3 DeltaCatalogTarget 结构

指定写入数据目录中的 Delta Lake AWS Glue 数据源的目标。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。

- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。

- `PartitionKeys` – UTF-8 字符串数组。

  使用一系列键指定本机分区。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的数据库中的表的名称。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

要向其写入的数据库的名称。

- AdditionalOptions – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定用于连接器的其他连接选项。

- SchemaChangePolicy – 一个 CatalogSchemaChangePolicy 对象。

  一项指定爬网程序的更新行为的策略。

S3 DeltaDirectTarget 结构

指定写入中三角洲湖数据源的目标 Amazon S3。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。

- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。

- PartitionKeys – UTF-8 字符串数组。

  使用一系列键指定本机分区。

- Path – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的 Delta Lake 数据来源的 Amazon S3 路径。

- Compression – 必填：UTF-8 字符串（有效值：uncompressed="UNCOMPRESSED" | snappy="SNAPPY"）。

  指定数据压缩方式。通常，如果数据有标准文件扩展名，则不需要指定。可能的值为 "gzip" 和 "bzip"。

- Format – 必填：UTF-8 字符串（有效值：json="JSON" | csv="CSV" | avro="AVRO" | orc="ORC" | parquet="PARQUET" | hudi="HUDI" | delta="DELTA"）。

指定目标的数据输出格式。

- `AdditionalOptions` – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定用于连接器的其他连接选项。

- `SchemaChangePolicy` – 一个 DirectSchemaChangePolicy 对象。

  一项指定爬网程序的更新行为的策略。

JdbcTarget 结构

指定要爬网的 JDBC 数据存储。

字段

- `ConnectionName` – UTF-8 字符串。

  要用于连接到 JDBC 目标的连接的名称。

- `Path` – UTF-8 字符串。

  JDBC 目标的路径。

- `Exclusions` – UTF-8 字符串数组。

  要从爬网中排除的 glob 模式的列表。有关更多信息，请参阅使用爬网程序为表编制目录。

- `EnableAdditionalMetadata` – UTF-8 字符串数组。

  将值指定为 `RAWTYPES` 或 `COMMENTS`，以在表响应中启用其他元数据。`RAWTYPES` 提供原生级别的数据类型。`COMMENTS` 提供与数据库中的列或表相关的注释。

  如果您不需要其他元数据，请将该字段留空。

MongoDBTarget 结构

指定要网络爬取的 Amazon DocumentDB 或 MongoDB 数据存储。

字段

- ConnectionName – UTF-8 字符串。

  要用于连接到 Amazon DocumentDB 或 MongoDB 目标的连接名称。
- Path – UTF-8 字符串。

  Amazon DocumentDB 或 MongoDB 目标（数据库/集合）的路径。
- ScanAll – 布尔值。

  指示是扫描所有记录，还是对表中的行进行采样。当表不是高吞吐量表时，扫描所有记录会花费很长时间。

  值为 true 表示扫描所有记录，值为 false 表示对记录进行采样。如果未指定任何值，则该值默认为 true。

DynamoDBTarget 结构

指定 Amazon DynamoDB 表以爬网。

字段

- Path – UTF-8 字符串。

  要爬网的 DynamoDB 表的名称。
- scanAll – 布尔值。

  指示是扫描所有记录，还是对表中的行进行采样。当表不是高吞吐量表时，扫描所有记录会花费很长时间。

  值为 true 表示扫描所有记录，值为 false 表示对记录进行采样。如果未指定任何值，则该值默认为 true。
- scanRate – 数字（double）。

  C AWS Glue rawler 要使用的已配置读取容量单位的百分比。读取容量单位是一个由 DynamoDB 定义的术语，它是一个数值，用作每秒可对表执行的读取次数的速率限制器。

  有效值为空或一个介于 0.1 和 1.5 之间的值。当用户未提供值时，使用空值，该值默认为已配置的读取容量单位的 1/2（对于预配置的表）或配置的最大读取容量单位的 1/4（对于使用按需模式的表）。

DeltaTarget 结构

指定 Delta 数据存储，爬取一个或多个 Delta 表。

字段

- `DeltaTables` – UTF-8 字符串数组。

  指向 Delta 表的 Amazon S3 路径列表。
- `ConnectionName` – UTF-8 字符串。

  要用于连接到 Delta 表目标的连接名称。
- `WriteManifest` – 布尔值。

  指定是否将清单文件写入 Delta 表路径。
- `CreateNativeDeltaTable` – 布尔值。

  指定爬网程序是否将创建原生表，以允许与支持直接查询 Delta 事务日志的查询引擎集成。

IcebergTarget 结构

指定 Amazon S3中存储 Iceberg 表的 Apache Iceberg 数据来源。

字段

- `Paths` – UTF-8 字符串数组。

  包含 Iceberg 元数据文件夹的一个或多个 Amazon S3 路径为`s3://bucket/prefix`.
- `ConnectionName` – UTF-8 字符串。

  要用于连接到 Iceberg 目标的连接的名称。
- `Exclusions` – UTF-8 字符串数组。

  要从爬网中排除的 glob 模式的列表。有关更多信息，请参阅使用爬网程序为表编制目录。
- `MaximumTraversalDepth` – 数字（整数）。

  爬虫可以遍历的最大 Amazon S3 路径深度，以发现路径中的 Iceberg 元数据文件夹。 Amazon S3 用于限制爬网程序运行时间。

HudiTarget 结构

指定 Apache Hudi 数据来源。

字段

- `Paths` – UTF-8 字符串数组。

  Hudi 的 Amazon S3 位置字符串数组，每个位置字符串都指示 Hudi 表的元数据文件所在的根文件夹。Hudi 文件夹可能位于根文件夹的子文件夹中。

  爬网程序将扫描路径下的所有文件夹，寻找 Hudi 文件夹。

- `ConnectionName` – UTF-8 字符串。

  要用于连接到 Hudi 目标的连接的名称。如果您的 Hudi 文件存储在需要 VPC 授权的存储桶中，则可以在此处设置其连接属性。

- `Exclusions` – UTF-8 字符串数组。

  要从爬网中排除的 glob 模式的列表。有关更多信息，请参阅使用爬网程序为表编制目录。

- `MaximumTraversalDepth` – 数字（整数）。

  爬虫可以遍历的最大 Amazon S3 路径深度，以发现路径中的 Hudi 元数据文件夹。 Amazon S3 用于限制爬网程序运行时间。

CatalogTarget 结构

指定 AWS Glue Data Catalog 目标。

字段

- `DatabaseName` – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  要同步的数据库的名称。

- `Tables` – 必填：UTF-8 字符串数组，至少 1 个字符串。

  要同步的表的列表。

- `ConnectionName` – UTF-8 字符串。

当使用与 Catalog 连接类型配对的 NETWORK 连接类型时，作为网络爬取目标的 Amazon S3 支持的数据目录表的连接名称。

- EventQueueArn – UTF-8 字符串。

  有效的 Amazon SQS ARN。例如，arn:aws:sqs:region:account:sqs。

- DlqEventQueueArn – UTF-8 字符串。

  有效的 Amazon 死信 SQS ARN。例如，arn:aws:sqs:region:account:deadLetterQueue。

CrawlerMetrics 结构

指定爬网程序的指标。

字段

- CrawlerName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  爬网程序的名称。

- TimeLeftSeconds – 数字（双数），至多为“无”。

  完成正在运行的爬网的估计剩余时间。

- StillEstimating – 布尔值。

  如果爬网程序仍在估算完成此运行所需的时长，则为 True。

- LastRuntimeSeconds – 数字（双数），至多为“无”。

  爬网程序的最近一次运行的持续时间 (秒)。

- MedianRuntimeSeconds – 数字（双数），至多为“无”。

  此爬网程序的运行的持续时间中值 (秒)。

- TablesCreated – 数字（整数），至多为“无”。

  此爬网程序创建的表的数量。

- TablesUpdated – 数字（整数），至多为“无”。

  此爬网程序更新的表的数量。

- TablesDeleted – 数字（整数），至多为“无”。

  此爬网程序删除的表的数量。

CrawlerHistory 结构

包含有关运行爬网程序的信息。

字段

- CrawlId – UTF-8 字符串。

  每个爬取的 UUID 标识符。

- State – UTF-8 字符串（有效值：RUNNING | COMPLETED | FAILED | STOPPED）。

  爬网的状态。

- StartTime – 时间戳。

  爬网操作的开始日期和时间。

- EndTime – 时间戳。

  爬网操作的结束日期和时间。

- Summary – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  JSON 中特定爬取的运行摘要。包含已添加、更新或删除的目录表和分区。

- ErrorMessage – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  如果发生错误，则为与爬网关联的错误消息。

- LogGroup – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节，与 Log group string pattern 匹配。

  与爬网关联的日志组。

- LogStream – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节，与 Log-stream string pattern 匹配。

  与爬网关联的日志流。

- `MessagePrefix` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  有关此抓取的 CloudWatch 消息的前缀。

- `DPUHour` – 数字（双数），至多为“无”。

  用于爬取的数据处理单元（DPU）的数量（以小时为单位）。

CrawlsFilter 结构

可用于筛选指定爬网程序的爬网程序运行的字段、比较运算符和值的列表。

字段

- `FieldName` – UTF-8 字符串（有效值：`CRAWL_ID` | `STATE` | `START_TIME` | `END_TIME` | `DPU_HOUR`）。

  用于筛选指定爬网程序的爬网程序运行的键。每个字段名称的有效值为：

  - `CRAWL_ID`：表示爬取的 UUID 标识符的字符串。
  - `STATE`：表示爬取状态的字符串。
  - `START_TIME` 和 `END_TIME`：纪元时间戳（以毫秒为单位）。
  - `DPU_HOUR`：用于爬网的数据处理单元（DPU）的数量（以小时为单位）。

- `FilterOperator` – UTF-8 字符串（有效值：`GT` | `GE` | `LT` | `LE` | `EQ` | `NE`）。

  对值进行操作的已定义比较运算符。可用的运算符有：

  - `GT`：大于。
  - `GE`：大于或等于。
  - `LT`：小于。
  - `LE`：小于或等于。
  - `EQ`：等于。
  - `NE`：不等于。

- `FieldValue` – UTF-8 字符串。

  为在爬取字段上进行比较而提供的值。

SchemaChangePolicy 结构

一项指定爬网程序的更新和删除行为的策略。

字段

- UpdateBehavior – UTF-8 字符串（有效值：LOG | UPDATE_IN_DATABASE）。

  爬网程序发现已更改的架构时的更新行为。
- DeleteBehavior – UTF-8 字符串（有效值：LOG | DELETE_FROM_DATABASE | DEPRECATE_IN_DATABASE）。

  爬网程序发现已删除的对象时的删除行为。

LastCrawlInfo 结构

有关最近一次爬网的状态和错误信息。

字段

- Status – UTF-8 字符串（有效值：SUCCEEDED | CANCELLED | FAILED）。

  上次爬网的状态。
- ErrorMessage – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  如果出现错误，则为有关上次爬网的错误信息。
- LogGroup – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节，与 Log group string pattern 匹配。

  上次爬网的日志组。
- LogStream – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节，与 Log-stream string pattern 匹配。

  上次爬网的日志流。
- MessagePrefix – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  有关此爬网的消息的前缀。

- `StartTime` – 时间戳。

  爬网开始的时间。

RecrawlPolicy 结构

在第一次网络爬取完成后对 Amazon S3 数据源进行网络爬取时，请指定是再次网络爬取整个数据集还是仅网络爬取自上次爬网程序运行以来添加的文件夹。有关更多信息，请参阅开发人员指南中的 AWS Glue中的增量爬网。

字段

- `RecrawlBehavior` – UTF-8 字符串（有效值：`CRAWL_EVERYTHING` | `CRAWL_NEW_FOLDERS_ONLY` | `CRAWL_EVENT_MODE`）。

  指定是否再次网络爬取整个数据集还是仅网络爬取自上次爬网程序运行以来添加的文件夹。

  值 `CRAWL_EVERYTHING` 指定再次网络爬取整个数据集。

  值 `CRAWL_NEW_FOLDERS_ONLY` 指定仅网络爬取自上次爬网程序运行以来添加的文件夹。

  `CRAWL_EVENT_MODE` 的值指定只网络爬取由 Amazon S3 事件标识的更改。

LineageConfiguration 结构

指定爬网程序的数据系统配置设置。

字段

- `CrawlerLineageSettings` – UTF-8 字符串（有效值：`ENABLE` | `DISABLE`）。

  指定是否已为爬网程序启用数据关联。有效值为：

  - ENABLE：启用爬网程序的数据关联
  - DISABLE：禁用爬网程序的数据关联

LakeFormationConfiguration 结构

指定爬网程序的 AWS Lake Formation 配置设置。

字段

- `UseLakeFormationCredentials` – 布尔值。

  指定是否使用爬网程序 AWS Lake Formation 凭证而不是 IAM 角色证书。

- `AccountId` – UTF-8 字符串，长度不超过 12 个字节。

  对于跨账户爬取是必需的。如果是与目标数据相同的账户爬取，可以将其保留为空。

操作

- CreateCrawler 动作（Python：create_crawler）
- DeleteCrawler 动作（Python：delete_crawler）
- GetCrawler 动作（Python：get_crawler）
- GetCrawlers 动作（Python：get_crawlers）
- GetCrawlerMetrics 操作（Python：get_crawler_metrics）
- UpdateCrawler 动作（Python：update_crawler）
- StartCrawler 动作（Python：start_crawler）
- StopCrawler 动作（Python：stop_crawler）
- BatchGetCrawlers 动作（Python：batch_get_crawlers）
- ListCrawlers 动作（Python：list_crawlers）
- ListCrawls 动作（Python：list_crawls）

CreateCrawler 动作（Python：create_crawler）

使用指定的目标、角色、配置和可选计划创建新的爬网程序。必须指定至少一个爬网目标（在 `s3Targets` 字段、`jdbcTargets` 字段或 `DynamoDBTargets` 字段中）。

请求

- `Name` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  新爬网程序的名称。

- `Role` – 必填：UTF-8 字符串。

新爬网程序用来访问客户资源的 IAM 角色或 IAM 角色的 Amazon Resource Name (ARN)。

- `DatabaseName` – UTF-8 字符串。

  写入结果 AWS Glue 的数据库，例如：`arn:aws:daylight:us-east-1::database/sometable/*`。

- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  新爬网程序的描述。

- `Targets` – 必填：一个 CrawlerTargets 对象。

  要爬网的目标的集合列表。

- `Schedule` – UTF-8 字符串。

  用于指定计划的 `cron` 表达式 (请参阅用于作业和爬网程序的基于时间的计划）。例如，要每天 12:15 UTC 运行某些任务，您应该指定：`cron(15 12 * * ? *)`。

- `Classifiers` – UTF-8 字符串数组。

  用户已注册的自定义分类器的列表。默认情况下，所有内置分类器均包含在爬网中，但这些自定义分类器始终会覆盖给定分类的默认分类器。

- `TablePrefix` – UTF-8 字符串，长度不超过 128 个字节。

  用于创建的目录表的表前缀。

- `SchemaChangePolicy` – 一个 SchemaChangePolicy 对象。

  适用于爬网程序的更新和删除行为的策略。

- `RecrawlPolicy` – 一个 RecrawlPolicy 对象。

  指定是否再次网络爬取整个数据集，还是仅网络爬取自上次爬网程序运行以来添加的文件夹的策略。

- `LineageConfiguration` – 一个 LineageConfiguration 对象。

  指定爬网程序的数据系统配置设置。

- `LakeFormationConfiguration` – 一个 LakeFormationConfiguration 对象。

  指定爬网程序的 AWS Lake Formation 配置设置。

- `Configuration` – UTF-8 字符串。

爬网程序配置信息。此受版本控制的 JSON 字符串允许用户指定爬网程序的行为的各个方面。有关更多信息，请参阅设置爬网程序配置选项。

- `CrawlerSecurityConfiguration` – UTF-8 字符串，长度不超过 128 个字节。

  该爬网程序将使用的 `SecurityConfiguration` 结构的名称。

- `Tags` – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  要用于此爬网程序请求的标签。您可以使用标签来限制对爬网程序的访问。有关中标签的更多信息 AWS Glue，请参阅开发者指南AWS Glue中的AWS 标签。

响应

- 无响应参数。

错误

- `InvalidInputException`
- `AlreadyExistsException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`

DeleteCrawler 动作（Python：delete_crawler）

从中移除指定的爬虫 AWS Glue Data Catalog，除非爬网程序状态为。`RUNNING`

请求

- `Name` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的爬网程序的名称。

响应

- 无响应参数。

错误

- `EntityNotFoundException`
- `CrawlerRunningException`
- `SchedulerTransitioningException`
- `OperationTimeoutException`

GetCrawler 动作（Python：get_crawler）

检索指定爬网程序的元数据。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索其元数据的爬网程序的名称。

响应

- `Crawler` – 一个 爬网程序 对象。

  指定爬网程序的元数据。

错误

- `EntityNotFoundException`
- `OperationTimeoutException`

GetCrawlers 动作（Python：get_crawlers）

检索在客户账户中定义的所有爬网程序的元数据。

请求

- MaxResults – 数字（整数），不小于 1 或大于 1000。

  每次调用时要返回的爬网程序的数量。
- NextToken – UTF-8 字符串。

  延续令牌 (如果这是延续请求)。

响应

- Crawlers – 爬网程序 对象的数组。

  爬网程序元数据的列表。
- NextToken – UTF-8 字符串。

  延续令牌 (如果返回的列表未到达此客户账户中定义的令牌的结尾)。

错误

- OperationTimeoutException

GetCrawlerMetrics 操作（Python：get_crawler_metrics）

检索有关指定爬网程序的指标。

请求

- CrawlerNameList – UTF-8 字符串数组，不超过 100 个字符串。

  有关要检索其指标的爬网程序的名称的列表。
- MaxResults – 数字（整数），不小于 1 或大于 1000。

  要返回的列表的最大大小。
- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用)。

响应

- CrawlerMetricsList – CrawlerMetrics 对象的数组。

  指定爬网程序的指标的列表。

- NextToken – UTF-8 字符串。

  延续令牌 (如果返回的列表不包含上一个可用的指标)。

错误

- OperationTimeoutException

UpdateCrawler 动作（Python：update_crawler）

更新爬网程序。如果爬网程序正在运行，您必须在更新它之前使用 StopCrawler 停止它。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  新爬网程序的名称。

- Role – UTF-8 字符串。

  新爬网程序用于访问客户资源的 IAM 角色或 IAM 角色的 Amazon Resource Name (ARN)。

- DatabaseName – UTF-8 字符串。

  存储结果 AWS Glue 的数据库，例如：arn:aws:daylight:us-east-1::database/sometable/*。

- Description – UTF-8 字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  新爬网程序的描述。

- Targets – 一个 CrawlerTargets 对象。

  要爬网的目标的列表。

- Schedule – UTF-8 字符串。

用于指定计划的 cron 表达式 (请参阅用于作业和爬网程序的基于时间的计划）。例如，要每天 12:15 UTC 运行某些任务，您应该指定：cron(15 12 * * ? *)。

- Classifiers – UTF-8 字符串数组。

  用户已注册的自定义分类器的列表。默认情况下，所有内置分类器均包含在爬网中，但这些自定义分类器始终会覆盖给定分类的默认分类器。

- TablePrefix – UTF-8 字符串，长度不超过 128 个字节。

  用于创建的目录表的表前缀。

- SchemaChangePolicy – 一个 SchemaChangePolicy 对象。

  适用于爬网程序的更新和删除行为的策略。

- RecrawlPolicy – 一个 RecrawlPolicy 对象。

  指定是否再次网络爬取整个数据集，还是仅网络爬取自上次爬网程序运行以来添加的文件夹的策略。

- LineageConfiguration – 一个 LineageConfiguration 对象。

  指定爬网程序的数据系统配置设置。

- LakeFormationConfiguration – 一个 LakeFormationConfiguration 对象。

  指定爬网程序的 AWS Lake Formation 配置设置。

- Configuration – UTF-8 字符串。

  爬网程序配置信息。此受版本控制的 JSON 字符串允许用户指定爬网程序的行为的各个方面。有关更多信息，请参阅设置爬网程序配置选项。

- CrawlerSecurityConfiguration – UTF-8 字符串，长度不超过 128 个字节。

  该爬网程序将使用的 SecurityConfiguration 结构的名称。

响应

- 无响应参数。

错误

- InvalidInputException
- VersionMismatchException

- EntityNotFoundException
- CrawlerRunningException
- OperationTimeoutException

StartCrawler 动作（Python：start_crawler）

使用指定爬网程序启动爬网，无论计划了什么。如果爬虫已经在运行，则返回 a。CrawlerRunningException

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要启动的爬网程序的名称。

响应

- 无响应参数。

错误

- EntityNotFoundException
- CrawlerRunningException
- OperationTimeoutException

StopCrawler 动作（Python：stop_crawler）

如果指定爬网程序正在运行，请停止爬网。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要停止的爬网程序的名称。

响应

- 无响应参数。

错误

- EntityNotFoundException
- CrawlerNotRunningException
- CrawlerStoppingException
- OperationTimeoutException

BatchGetCrawlers 动作（Python：batch_get_crawlers）

返回给定爬网程序名称列表的资源元数据的列表。调用 ListCrawlers 操作后，您可以调用此操作来访问您有权访问的数据。此操作支持所有 IAM 权限，包括使用标签的权限条件。

请求

- CrawlerNames – 必填：UTF-8 字符串数组，不超过 100 个字符串。

  爬网程序名称列表，这些名称可能是通过 ListCrawlers 操作返回的名称。

响应

- Crawlers – 爬网程序 对象的数组。

  爬网程序定义的列表。
- CrawlersNotFound – UTF-8 字符串数组，不超过 100 个字符串。

  未找到的 Crawler 名称的列表。

错误

- InvalidInputException
- OperationTimeoutException

ListCrawlers 动作（Python：list_crawlers）

检索此 AWS 账户中所有 Crawler 资源的名称，或者检索带有指定标签的资源的名称。此操作可让您查看您账户中可用的资源及其名称。

此操作采用可选的 Tags 字段，您可以将其用作响应的筛选器，以便将标记的资源作为一个组进行检索。如果您选择使用标签筛选，则仅检索带标签的资源。

请求

- MaxResults – 数字（整数），不小于 1 或大于 1000。

  要返回的列表的最大大小。

- NextToken – UTF-8 字符串。

  延续令牌 (如果这是延续请求)。

- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  指定仅返回这些已标记的资源。

响应

- CrawlerNames – UTF-8 字符串数组，不超过 100 个字符串。

  账户中所有爬网程序的名称或带指定标签的爬网程序。

- NextToken – UTF-8 字符串。

  延续令牌 (如果返回的列表不包含上一个可用的指标)。

错误

- OperationTimeoutException

ListCrawls 动作（Python：list_crawls）

返回指定爬网程序的所有爬取结果。仅返回自爬网程序历史记录功能启动之日起发生的爬网操作，最多只能保留 12 个月的爬取结果。较早的爬取结果将不会被返回。

您可以使用此 API 执行以下操作：

- 检索指定爬网程序的所有爬网。
- 在有限的计数内检索指定爬网程序的所有爬取结果。
- 检索指定爬网程序在特定时间范围内的所有爬取结果。
- 检索具有特定状态、爬网 ID 或 DPU 小时值的指定爬网程序的所有爬取结果。

请求

- `CrawlerName` – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  要检索其运行的爬网程序的名称。

- `MaxResults` – 数字（整数），不小于 1 或大于 1000。

  要返回的最大结果数量。默认值为 20，最大值为 100。

- `Filters` – CrawlsFilter 对象的数组。

  按照您在 `CrawlsFilter` 对象列表中指定的标准筛选爬取结果。

- `NextToken` – UTF-8 字符串。

  延续标记 (如果这是延续调用)。

响应

- `Crawls` – CrawlerHistory 对象的数组。

  代表符合您的标准的爬取运行的 `CrawlerHistory` 对象列表。

- `NextToken` – UTF-8 字符串。

  对返回的标记列表进行分页的延续令牌 (如果列表的当前片段不是最后一个，则返回)。

错误

- EntityNotFoundException
- OperationTimeoutException
- InvalidInputException

列统计数据 API

列统计数据 API 介绍了用于返回表中各列统计数据的 AWS Glue API。

数据类型

- ColumnStatisticsTaskRun 结构
- ColumnStatisticsTaskRunningException 结构
- ColumnStatisticsTaskNotRunningException 结构
- ColumnStatisticsTaskStoppingException 结构

ColumnStatisticsTaskRun 结构

显示列统计数据运行详细信息的对象。

字段

- CustomerId – UTF-8 字符串，长度不超过 12 个字节。

  AWS 账户 ID。
- ColumnStatisticsTaskRunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  特定列统计数据任务运行的标识符。
- DatabaseName – UTF-8 字符串。

  表所在的数据库。
- TableName – UTF-8 字符串。

  生成列统计数据的表的名称。
- ColumnNameList – UTF-8 字符串数组。

列名称的列表。如果不提供此参数，则默认情况下将使用表的所有列名。

- `CatalogID` – 目录 id 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。

- `Role` – UTF-8 字符串。

  服务为了生成统计数据而代入的 IAM 角色。

- `SampleSize` – 数值（双精度），不超过 100。

  用于生成统计数据的行百分比。如果不提供此参数，则将用整个表来生成统计数据。

- `SecurityConfiguration` – UTF-8 字符串，长度不超过 128 个字节。

  用于为列统计数据任务运行的 CloudWatch 日志加密的安全配置的名称。

- `NumberOfWorkers` - 数字（整数），至少为 1。

  生成列统计数据的 Worker 线程数。此作业已预先配置为可自动扩展至不超过 25 个实例。

- `WorkerType` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  用于生成统计数据的 Worker 线程类型。默认为 `g.1x`。

- `Status` – UTF-8 字符串（有效值：`STARTING` | `RUNNING` | `SUCCEEDED` | `FAILED` | `STOPPED`）。

  任务运行的状态。

- `CreationTime` – 时间戳。

  此任务的创建时间。

- `LastUpdated` – 时间戳。

  上次修改此任务的时间点。

- `StartTime` – 时间戳。

  任务的开始时间。

- `EndTime` – 时间戳。

  任务的结束时间。

- ErrorMessage – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  作业的错误消息。

- DPUSeconds – 数字（双数），至多为“无”。

  所有自动扩展的 Worker 线程的计算 DPU 使用量（以秒为单位）。

ColumnStatisticsTaskRunningException 结构

在运行列统计数据生成作业时尝试启动其他作业引发的异常。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

ColumnStatisticsTaskNotRunningException 结构

在没有任务运行时尝试停止任务运行引发的异常。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

ColumnStatisticsTaskStoppingException 结构

在尝试停止任务运行引发的异常。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

操作

- StartColumnStatisticsTaskRun 操作（Python：start_column_statistics_task_run）
- GetColumnStatisticsTaskRun 操作（Python：get_column_statistics_task_run）
- GetColumnStatisticsTaskRuns 操作（Python：get_column_statistics_task_runs）
- ListColumnStatisticsTaskRuns 操作（Python：list_column_statistics_task_runs）
- StopColumnStatisticsTaskRun 操作（Python：stop_column_statistics_task_run）

StartColumnStatisticsTaskRun 操作（Python：start_column_statistics_task_run）

为指定的表和列启动列统计数据任务运行。

请求

- `DatabaseName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据库的名称。
- `TableName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要生成统计数据的表的名称。
- `ColumnNameList` – UTF-8 字符串数组。

  生成统计数据的列名列表。如果不提供此参数，则默认情况下将使用表的所有列名。
- `Role` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  服务为了生成统计数据而代入的 IAM 角色。
- `SampleSize` – 数值（双精度），不超过 100。

  用于生成统计数据的行百分比。如果不提供此参数，则将用整个表来生成统计数据。
- `CatalogID` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的数据目录的 ID。如果没有提供，则默认情况下使用 AWS 账户 ID。

- SecurityConfiguration – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  用于为列统计数据任务运行的 CloudWatch 日志加密的安全配置的名称。

响应

- ColumnStatisticsTaskRunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  列统计数据任务运行的标识符。

错误

- AccessDeniedException
- EntityNotFoundException
- ColumnStatisticsTaskRunningException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- InvalidInputException

GetColumnStatisticsTaskRun 操作（Python：get_column_statistics_task_run）

在已知任务运行 ID 的情况下，获取任务运行的相关元数据/信息。

请求

- ColumnStatisticsTaskRunId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  特定列统计数据任务运行的标识符。

响应

- ColumnStatisticsTaskRun – 一个 ColumnStatisticsTaskRun 对象。

  表示列统计数据运行详细信息的 ColumnStatisticsTaskRun 对象。

错误

- EntityNotFoundException
- OperationTimeoutException
- InvalidInputException

GetColumnStatisticsTaskRuns 操作（Python：get_column_statistics_task_runs）

检索与指定表关联的所有运行的信息。

请求

- DatabaseName – 必填：UTF-8 字符串。

  表所在的数据库的名称。
- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的名称。
- MaxResults – 数字（整数），不小于 1 或大于 1000。

  响应的最大大小。
- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用)。

响应

- ColumnStatisticsTaskRuns – ColumnStatisticsTaskRun 对象的数组。

  列统计数据任务运行列表。
- NextToken – UTF-8 字符串。

  延续令牌（如果尚未返回所有任务运行）。

错误

- OperationTimeoutException

ListColumnStatisticsTaskRuns 操作（Python：list_column_statistics_task_runs）

列出特定账户的所有任务运行。

请求

- MaxResults – 数字（整数），不小于 1 或大于 1000。

  响应的最大大小。
- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用)。

响应

- ColumnStatisticsTaskRunIds – UTF-8 字符串数组，不超过 100 个字符串。

  列统计数据任务运行 ID 列表。
- NextToken – UTF-8 字符串。

  延续令牌（如果尚未返回所有任务运行 ID）。

错误

- OperationTimeoutException

StopColumnStatisticsTaskRun 操作（Python：stop_column_statistics_task_run）

停止指定表的任务运行。

请求

- DatabaseName – 必填：UTF-8 字符串。

  表所在的数据库的名称。
- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表的名称。

响应

- 无响应参数。

错误

- `EntityNotFoundException`
- `ColumnStatisticsTaskNotRunningException`
- `ColumnStatisticsTaskStoppingException`
- `OperationTimeoutException`

爬网程序计划程序 API

爬网程序调度器 API 介绍 AWS Glue 爬网程序数据类型，以及用于创建、删除、更新和列出爬网程序的 API。

数据类型

- Schedule 结构

Schedule 结构

一个使用 `cron` 语句计划事件的计划对象。

字段

- `ScheduleExpression` – UTF-8 字符串。

  用于指定计划的 `cron` 表达式 (请参阅用于作业和爬网程序的基于时间的计划）。例如，要每天 12:15 UTC 运行某些任务，您应该指定：`cron(15 12 * * ? *)`。
- `State` – UTF-8 字符串（有效值：`SCHEDULED` | `NOT_SCHEDULED` | `TRANSITIONING`）。

  计划的状态。

操作

- UpdateCrawlerSchedule 操作（Python：update_crawler_schedule）

- StartCrawlerSchedule 操作（Python：start_crawler_schedule）
- StopCrawlerSchedule 操作（Python：stop_crawler_schedule）

UpdateCrawlerSchedule 操作（Python：update_crawler_schedule）

使用 cron 表达式更新爬网程序的计划。

请求

- CrawlerName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要更新其计划的爬网程序的名称。

- Schedule – UTF-8 字符串。

  用于指定计划的更新的 cron 表达式（请参阅用于作业和爬网程序的基于时间的计划）。例如，要每天 12:15 UTC 运行某些任务，您应该指定：cron(15 12 * * ? *)。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InvalidInputException
- VersionMismatchException
- SchedulerTransitioningException
- OperationTimeoutException

StartCrawlerSchedule 操作（Python：start_crawler_schedule）

将指定爬网程序的计划状态更改为 SCHEDULED，除非爬网程序已在运行或者计划状态已为 SCHEDULED。

请求

- `CrawlerName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要计划的爬网程序的名称。

响应

- 无响应参数。

错误

- `EntityNotFoundException`
- `SchedulerRunningException`
- `SchedulerTransitioningException`
- `NoScheduleException`
- `OperationTimeoutException`

StopCrawlerSchedule 操作（Python：stop_crawler_schedule）

将指定爬网程序的计划状态设置为 `NOT_SCHEDULED`，但不停止爬网程序 (如果已在运行)。

请求

- `CrawlerName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要设置其计划状态的爬网程序的名称。

响应

- 无响应参数。

错误

- `EntityNotFoundException`

- SchedulerNotRunningException
- SchedulerTransitioningException
- OperationTimeoutException

自动生成 ETL 脚本 API

ETL 脚本生成 API 介绍用于在 AWS Glue 中生成 ETL 脚本的数据类型和 API。

数据类型

- CodeGenNode 结构
- CodeGenNodeArg 结构
- CodeGenEdge 结构
- Location 结构
- CatalogEntry 结构
- MappingEntry 结构

CodeGenNode 结构

表示有向无环图 (DAG) 中的节点

字段

- Id – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Identifier string pattern 匹配。

  节点图中唯一的节点标识符。
- NodeType – 必填：UTF-8 字符串。

  该节点的类型。
- Args – 必填：CodeGenNodeArg 对象的数组，不超过 50 个结构。

  节点的属性，采用名称-值对形式。
- LineNumber – 数字（整数）。

  节点的行号。

CodeGenNodeArg 结构

节点的参数或属性。

字段

- Name – 必填：UTF-8 字符串。

  参数或属性的名称。

- Value – 必填：UTF-8 字符串。

  参数或属性的值。

- Param – 布尔值。

  如果值用作参数，则为 True。

CodeGenEdge 结构

表示有向无环图 (DAG) 中的方向边缘。

字段

- Source – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Identifier string pattern 匹配。

  边缘开始的节点的 ID。

- Target – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Identifier string pattern 匹配。

  边缘结束的节点的 ID。

- TargetParameter – UTF-8 字符串。

  边缘的目标。

Location 结构

资源的位置。

字段

- Jdbc – CodeGenNodeArg 对象的数组，不超过 50 个结构。

  JDBC 位置。
- S3 – CodeGenNodeArg 对象的数组，不超过 50 个结构。

  Amazon Simple Storage Service (Amazon S3) 位置
- DynamoDB – CodeGenNodeArg 对象的数组，不超过 50 个结构。

  Amazon DynamoDB 表位置。

CatalogEntry 结构

在 AWS Glue Data Catalog 中指定表定义。

字段

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  表元数据所在的数据库。
- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  相关表的名称。

MappingEntry 结构

定义映射。

字段

- SourceTable – UTF-8 字符串。

  源表的名称。
- SourcePath – UTF-8 字符串。

  源路径。

- SourceType – UTF-8 字符串。

  源类型

- TargetTable – UTF-8 字符串。

  目标表。

- TargetPath – UTF-8 字符串。

  目标路径。

- TargetType – UTF-8 字符串。

  目标类型。

操作

- CreateScript 操作（Python：create_script）
- GetDataflowGraph 操作（Python：get_dataflow_graph）
- GetMapping 操作（Python：get_mapping）
- GetPlan 操作（Python：get_plan）

CreateScript 操作（Python：create_script）

将有向无环图 (DAG) 转换为代码。

请求

- DagNodes – CodeGenNode 对象的数组。

  DAG 中的节点的列表。

- DagEdges – CodeGenEdge 对象的数组。

  DAG 中的边缘的列表。

- Language – UTF-8 字符串（有效值：PYTHON | SCALA）。

  从 DAG 生成的代码的编程语言。

响应

- `PythonScript` – UTF-8 字符串。

  从 DAG 生成的 Python 脚本。
- `ScalaCode` – UTF-8 字符串。

  从 DAG 生成的 Scala 代码。

错误

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetDataflowGraph 操作（Python：get_dataflow_graph）

将 Python 脚本转换为有向无环图 (DAG)。

请求

- `PythonScript` – UTF-8 字符串。

  要转换的 Python 脚本。

响应

- `DagNodes` – CodeGenNode 对象的数组。

  生成的 DAG 中的节点的列表。
- `DagEdges` – CodeGenEdge 对象的数组。

  生成的 DAG 中的边缘的列表。

错误

- `InvalidInputException`
- `InternalServiceException`

- OperationTimeoutException

GetMapping 操作（Python：get_mapping）

创建映射。

请求

- Source – 必填：一个 CatalogEntry 对象。

  指定源表。
- Sinks – CatalogEntry 对象的数组。

  目标表的列表。
- Location – 一个 位置 对象。

  映射的参数。

响应

- Mapping – 必填：MappingEntry 对象的数组。

  指定目标的映射的列表。

错误

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- EntityNotFoundException

GetPlan 操作（Python：get_plan）

获取代码以执行指定的映射。

请求

- Mapping – 必填：MappingEntry 对象的数组。

从源表到目标表的映射的列表。

- `Source` – 必填：一个 CatalogEntry 对象。

  源表。

- `Sinks` – CatalogEntry 对象的数组。

  目标表。

- `Location` – 一个 位置 对象。

  映射的参数。

- `Language` – UTF-8 字符串（有效值：`PYTHON` | `SCALA`）。

  用于执行映射的代码的编程语言。

- `AdditionalPlanOptionsMap` – 键值对的映射数组。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  用于保存其他可选键值参数的映射。

  目前支持以下键值对：

  - `inferSchema` – 指定是否为 AWS Glue 任务生成的默认脚本将 `inferSchema` 设置为 true 或 false。例如，要将 `inferSchema` 设置为 true，则传递以下键值对：

    ```
    --additional-plan-options-map '{"inferSchema":"true"}'
    ```

响应

- `PythonScript` – UTF-8 字符串。

  用于执行映射的 Python 脚本。

- `ScalaCode` – UTF-8 字符串。

  用于执行映射的 Scala 代码。

错误

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

可视化作业 API

Visual Job API 允许您使用表示作业可视化配置的 JSON 对象中的 AWS Glue API 来创建数据集成 AWS Glue 作业。

为创建或更新作业 API 提供了列表，用于在 AWS Glue Studio 中为已创建的作业注册 DAG 并生成关联代码。`CodeGenConfigurationNodes`

数据类型

- CodeGenConfigurationNode 结构
- JDBC 结构 ConnectorOptions
- StreamingDataPreviewOptions 结构
- AthenaConnectorSource 结构
- JDBC 结构 ConnectorSource
- SparkConnectorSource 结构
- CatalogSource 结构
- MySQL CatalogSource 结构
- PostgreSQL 结构 CatalogSource
- OracleSQL 结构 CatalogSource
- 微软 SQL 结构 ServerCatalogSource
- CatalogKinesisSource 结构
- DirectKinesisSource 结构
- KinesisStreamingSourceOptions 结构
- CatalogKafkaSource 结构
- DirectKafkaSource 结构

- KafkaStreamingSourceOptions 结构
- RedshiftSource 结构
- AmazonRedshiftSource 结构
- AmazonRedshiftNodeData 结构
- AmazonRedshiftAdvancedOption 结构
- 选项结构
- S3 CatalogSource 结构
- S3 SourceAdditionalOptions 结构
- S3 CsvSource 结构
- DirectJDBCSource 结构
- S3 DirectSourceAdditionalOptions 结构
- S3 JsonSource 结构
- S3 ParquetSource 结构
- S3 DeltaSource 结构
- S3 CatalogDeltaSource 结构
- CatalogDeltaSource 结构
- S3 HudiSource 结构
- S3 CatalogHudiSource 结构
- CatalogHudiSource 结构
- DynamoDB 结构 CatalogSource
- RelationalCatalogSource 结构
- JDBC 结构 ConnectorTarget
- SparkConnectorTarget 结构
- BasicCatalogTarget 结构
- MySQL CatalogTarget 结构
- PostgreSQL 结构 CatalogTarget
- OracleSQL 结构 CatalogTarget
- 微软 SQL 结构 ServerCatalogTarget

- RedshiftTarget 结构
- AmazonRedshiftTarget 结构
- UpsertRedshiftTargetOptions 结构
- S3 CatalogTarget 结构
- S3 GlueParquetTarget 结构
- CatalogSchemaChangePolicy 结构
- S3 DirectTarget 结构
- S3 HudiCatalogTarget 结构
- S3 HudiDirectTarget 结构
- S3 DeltaCatalogTarget 结构
- S3 DeltaDirectTarget 结构
- DirectSchemaChangePolicy 结构
- ApplyMapping 结构
- Mapping 结构
- SelectFields 结构
- DropFields 结构
- RenameField 结构
- Spigot 结构
- Join 结构
- JoinColumn 结构
- SplitFields 结构
- SelectFromCollection 结构
- FillMissingValues 结构
- Filter 结构
- FilterExpression 结构
- FilterValue 结构
- CustomCode 结构
- SparkSQL 结构

- SqlAlias 结构
- DropNullFields 结构
- NullCheckBoxList 结构
- NullValueField 结构
- Datatype 结构
- Merge 结构
- Union 结构
- PIIDetection 结构
- Aggregate 结构
- DropDuplicates 结构
- GovernedCatalogTarget 结构
- GovernedCatalogSource 结构
- AggregateOperation 结构
- GlueSchema 结构
- GlueStudioSchemaColumn 结构
- GlueStudioColumn 结构
- DynamicTransform 结构
- TransformConfigParameter 结构
- EvaluateDataQuality 结构
- DQ 结构 ResultsPublishingOptions
- DQ 结构 StopJobOnFailureOptions
- EvaluateDataQualityMultiFrame 结构
- 脚本结构
- RecipeReference 结构
- SnowflakeNodeData 结构
- SnowflakeSource 结构
- SnowflakeTarget 结构
- ConnectorDataSource 结构

- ConnectorDataTarget 结构

CodeGenConfigurationNode 结构

CodeGenConfigurationNode 枚举全部有效的节点类型。可以填充其中一个成员变量，并且只能填充一个。

字段

- AthenaConnectorSource – 一个 AthenaConnectorSource 对象。

  指定一个指向 Amazon Athena 数据源的连接器。
- JDBCConnectorSource – 一个 JDBC ConnectorSource 对象。

  指定一个指向 JDBC 数据源的连接器。
- SparkConnectorSource – 一个 SparkConnectorSource 对象。

  指定一个指向 Apache Spark 数据源的连接器。
- CatalogSource – 一个 CatalogSource 对象。

  在数据目录中指定 AWS Glue 数据存储。
- RedshiftSource – 一个 RedshiftSource 对象。

  指定一个 Amazon Redshift 数据存储。
- S3CatalogSource – 一个 S3 CatalogSource 对象。

  在数据目录中指定 Amazon S3 AWS Glue 数据存储。
- S3CsvSource – 一个 S3 CsvSource 对象。

  指定一个存储在 Amazon S3 中的命令分隔值 (CSV) 数据存储。
- S3JsonSource – 一个 S3 JsonSource 对象。

  指定一个存储在 Amazon S3 中的 JSON 数据存储。
- S3ParquetSource – 一个 S3 ParquetSource 对象。

  指定一个存储在 Amazon S3 中的 Apache Parquet 数据存储。
- RelationalCatalogSource – 一个 RelationalCatalogSource 对象。

在数据目录中指定关系目录 AWS Glue 数据存储。

- `DynamoDBCatalogSource` – 一个 DynamoDB CatalogSource 对象。

  在数据目录中指定 DynamoDBC 目录 AWS Glue 数据存储。

- `JDBCConnectorTarget` – 一个 JDBC ConnectorTarget 对象。

  指定一个在 Apache Paric 列式存储中写入 Amazon S3 的数据目标。

- `SparkConnectorTarget` – 一个 SparkConnectorTarget 对象。

  指定一个使用 Apache Spark 连接器的目标。

- `CatalogTarget` – 一个 BasicCatalogTarget 对象。

  指定使用 AWS Glue 数据目录表的目标。

- `RedshiftTarget` – 一个 RedshiftTarget 对象。

  指定一个使用 Amazon Redshift 的目标。

- `S3CatalogTarget` – 一个 S3 CatalogTarget 对象。

  指定使用数据目录写入 Amazon S3 AWS Glue 的数据目标。

- `S3GlueParquetTarget` – 一个 S3 GlueParquetTarget 对象。

  指定一个在 Apache Paric 列式存储中写入 Amazon S3 的数据目标。

- `S3DirectTarget` – 一个 S3 DirectTarget 对象。

  指定一个写入 Amazon S3 的数据目标。

- `ApplyMapping` – 一个 ApplyMapping 对象。

  指定一个将数据源中的数据属性键映射到数据目标中的数据属性键的转换。您可以重命名键、修改键的数据类型以及选择要从数据集中删除的键。

- `SelectFields` – 一个 SelectFields 对象。

  指定一个选择要保留的数据属性键的转换。

- `DropFields` – 一个 DropFields 对象。

  指定一个选择要删除的数据属性键的转换。

- `RenameField` – 一个 RenameField 对象。

指定一个重命名单个数据属性键的转换。

- `Spigot` – 一个 Spigot 对象。

  指定一个将数据样本写入 Amazon S3 存储桶的转换。

- `Join` – 一个 Join 对象。

  指定一个转换，它将使用指定数据属性键上的比较短语将两个数据集联接到一个数据集。您可以使用内部、外部、左、右、左半和左反联接。

- `SplitFields` – 一个 SplitFields 对象。

  指定一个将数据属性键拆分为两个 `DynamicFrames` 的转换。输出是 `DynamicFrames` 的集合：一个包含选定的数据属性键，另一个包含剩余的数据属性键。

- `SelectFromCollection` – 一个 SelectFromCollection 对象。

  指定一个从 `DynamicFrames` 的集合中选择一个 `DynamicFrame` 的转换。输出是选定的 `DynamicFrame`

- `FillMissingValues` – 一个 FillMissingValues 对象。

  指定一个转换，它将查找数据集中缺少值的记录，并添加包含通过推算确定的值的新字段。输入数据集用于训练机器学习模型，该模型确定缺失值应该是什么。

- `Filter` – 一个 筛选条件 对象。

  指定一个转换，它将基于筛选条件将一个数据集拆分为两个。

- `CustomCode` – 一个 CustomCode 对象。

  指定一个转换，它将使用您提供的自定义代码执行数据转换。输出是一个集合 DynamicFrames。

- `SparkSQL` – 一个 SparkSQL 对象。

  指定一个转换，您可以在其中使用 Spark SQL 语法输入 SQL 查询以转换数据。输出为单个 `DynamicFrame`。

- `DirectKinesisSource` – 一个 DirectKinesisSource 对象。

  指定一个直接 Amazon Kinesis 数据源。

- `DirectKafkaSource` – 一个 DirectKafkaSource 对象。

  指定一个 Apache Kafka 数据存储。

- `CatalogKinesisSource` – 一个 CatalogKinesisSource 对象。

在数据目录中指定 Kinesis AWS Glue 数据源。

- `CatalogKafkaSource` – 一个 CatalogKafkaSource 对象。

  指定数据目录中的一个 Apache Kafka 数据存储。

- `DropNullFields` – 一个 DropNullFields 对象。

  指定一个转换，如果列中的所有值均为“null”，则该转换将从数据集中删除这些列。默认情况下，AWS Glue Studio 会识别空对象，但是某些值（例如空字符串、“null” 字符串、-1 个整数或其他占位符（例如零）不会自动识别为空值。

- `Merge` – 一个 Merge 对象。

  指定一个转换，它将基于指定的主键将 `DynamicFrame` 与暂存 `DynamicFrame` 合并以标识记录。不会对重复记录（具有相同主键的记录）去除重复。

- `Union` – 一个 Union 对象。

  指定一个转换，它将两个或更多数据集中的行合并到单个结果中。

- `PIIDetection` – 一个 PIIDetection 对象。

  指定用于识别、删除或掩盖 PII 数据的转换。

- `Aggregate` – 一个 聚合 对象。

  指定一个转换，用于按选定字段对行进行分组并通过指定函数计算聚合值。

- `DropDuplicates` – 一个 DropDuplicates 对象。

  指定一个用于从数据集中删除重复数据行的转换。

- `GovernedCatalogTarget` – 一个 GovernedCatalogTarget 对象。

  指定一个用于写入监管目录的数据目标。

- `GovernedCatalogSource` – 一个 GovernedCatalogSource 对象。

  指定监管数据目录中的一个数据源。

- `MicrosoftSQLServerCatalogSource` – 一个 微软 SQL ServerCatalogSource 对象。

  在 AWS Glue 数据目录中指定一个 Microsoft SQL Server 数据源。

- `MySQLCatalogSource` – 一个 MySQL CatalogSource 对象。

  在数据目录中指定一个 MySQL AWS Glue 数据源。

- `OracleSQLCatalogSource` – 一个 OracleSQL CatalogSource 对象。

  在数据目录中指定 Oracle AWS Glue 数据源。

- `PostgreSQLCatalogSource` – 一个 PostgreSQL CatalogSource 对象。

  在数据目录中指定 PostgresSQL 数据源。 AWS Glue

- `MicrosoftSQLServerCatalogTarget` – 一个 微软 SQL ServerCatalogTarget 对象。

  指定一个使用 Microsoft SQL 的目标。

- `MySQLCatalogTarget` – 一个 MySQL CatalogTarget 对象。

  指定一个使用 MySQL 的目标。

- `OracleSQLCatalogTarget` – 一个 OracleSQL CatalogTarget 对象。

  指定一个使用 Oracle SQL 的目标。

- `PostgreSQLCatalogTarget` – 一个 PostgreSQL CatalogTarget 对象。

  指定一个使用 Postgres SQL 的目标。

- `DynamicTransform` – 一个 DynamicTransform 对象。

  指定由用户创建的自定义视觉转换。

- `EvaluateDataQuality` – 一个 EvaluateDataQuality 对象。

  指定您的数据质量评估标准。

- `S3CatalogHudiSource` – 一个 S3 CatalogHudiSource 对象。

  指定在数据目录中注册的 Hudi AWS Glue 数据源。数据源必须存储在 Amazon S3。

- `CatalogHudiSource` – 一个 CatalogHudiSource 对象。

  指定在数据目录中注册的 Hudi AWS Glue 数据源。

- `S3HudiSource` – 一个 S3 HudiSource 对象。

  指定存储在中的 Amazon S3 Hudi 数据源。

- `S3HudiCatalogTarget` – 一个 S3 HudiCatalogTarget 对象。

  指定写入数据目录中的 Hudi 数据源的目标。 AWS Glue

- `S3HudiDirectTarget` – 一个 S3 HudiDirectTarget 对象。

指定写入中 Hudi 数据源的目标。 Amazon S3

- `S3CatalogDeltaSource` – 一个 S3 CatalogDeltaSource 对象。

  指定在数据目录中注册的 Delta Lake AWS Glue 数据源。数据源必须存储在 Amazon S3。

- `CatalogDeltaSource` – 一个 CatalogDeltaSource 对象。

  指定在数据目录中注册的 Delta Lake AWS Glue 数据源。

- `S3DeltaSource` – 一个 S3 DeltaSource 对象。

  指定存储在中的三角洲湖数据源 Amazon S3。

- `S3DeltaCatalogTarget` – 一个 S3 DeltaCatalogTarget 对象。

  指定写入数据目录中的 Delta Lake AWS Glue 数据源的目标。

- `S3DeltaDirectTarget` – 一个 S3 DeltaDirectTarget 对象。

  指定写入中三角洲湖数据源的目标 Amazon S3。

- `AmazonRedshiftSource` – 一个 AmazonRedshiftSource 对象。

  指定在 Amazon Redshift 中写入数据来源的目标。

- `AmazonRedshiftTarget` – 一个 AmazonRedshiftTarget 对象。

  指定在 Amazon Redshift 中写入数据目标的目标。

- `EvaluateDataQualityMultiFrame` – 一个 EvaluateDataQualityMultiFrame 对象。

  指定您的数据质量评估标准。允许多个输入数据并返回动态帧的集合。

- `Recipe` – 一个 配方 对象。

  指定 AWS Glue DataBrew 配方节点。

- `SnowflakeSource` – 一个 SnowflakeSource 对象。

  指定 Snowflake 数据来源。

- `SnowflakeTarget` – 一个 SnowflakeTarget 对象。

  指定写入 Snowflake 数据来源的目标。

- `ConnectorDataSource` – 一个 ConnectorDataSource 对象。

  指定使用标准连接选项生成的源。

- ConnectorDataTarget – 一个 ConnectorDataTarget 对象。

  指定使用标准连接选项生成的目标。

JDBC 结构 ConnectorOptions

用于连接器的其他连接选项。

字段

- FilterPredicate – UTF-8 字符串，与 Custom string pattern #40 匹配。

  用于筛选源中的数据的额外条件子句。例如：

  ```
  BillingCity='Mountain View'
  ```

  使用查询（而不是表名称）时，您应验证查询是否适用于指定的 filterPredicate。
- PartitionColumn – UTF-8 字符串，与 Custom string pattern #40 匹配。

  用于分区的整数列的名称。此选项仅在包含 lowerBound、upperBound 和 numPartitions 时有效。此选项的工作方式与 Spark SQL JDBC 阅读器中的工作方式相同。
- LowerBound – 数字（长型），至多为“无”。

  用于确定分区步长的最小 partitionColumn 值。
- UpperBound – 数字（长型），至多为“无”。

  用于确定分区步长的最大 partitionColumn 值。
- NumPartitions – 数字（长型），至多为“无”。

  分区的数量。此值以及 lowerBound（包含）和 upperBound（排除）为用于拆分 partitionColumn 而生成的 WHERE 子句表达式构成分区步长。
- JobBookmarkKeys – UTF-8 字符串数组。

  要作为排序依据的任务书签键的名称。
- JobBookmarkKeysSortOrder – UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定升序或降序排序顺序。
- DataTypeMapping – 键值对的映射数组。

每个键都是一个 UTF-8 字符串（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）。

每个值都是一个 UTF-8字符串（有效值：DATE | STRING | TIMESTAMP | INT | FLOAT | LONG | BIGDECIMAL | BYTE | SHORT | DOUBLE）。

用于构建从 JDBC 数据类型到 AWS Glue 数据类型的映射的自定义数据类型映射。例如，该选项通过调用驱动程序的ResultSet.getString()方法FLOAT将 JDBC String 类型的数据字段"dataTypeMapping":{"FLOAT":"STRING"}映射到 Java 类型，并使用它来生成记录。 AWS Glue ResultSet 对象由每个驱动程序实现，因此行为特定于您使用的驱动程序。请参阅 JDBC 驱动程序的文档，了解驱动程序执行转换的方式。

StreamingDataPreviewOptions 结构

指定与用于查看数据样本的数据预览相关的选项。

字段

- PollingTime – 数字（长度），至少为 10。

  轮询时间（以毫秒为单位）。
- RecordPollingLimit – 数字（长度），至少为 1。

  已轮询的记录的数量限制。

AthenaConnectorSource 结构

指定一个指向 Amazon Athena 数据源的连接器。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据源的名称。

- ConnectionName – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  与连接器关联的连接的名称。
- ConnectorName – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  帮助访问 AWS Glue Studio 中数据存储的连接器的名称。
- ConnectionType – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定指向 Amazon Athena 数据存储的连接的连接类型，如 marketplace.athena 或 custom.athena。
- ConnectionTable – UTF-8 字符串，与 Custom string pattern #41 匹配。

  数据源中的表的名称。
- SchemaName – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的 CloudWatch 日志组的名称。例如，/aws-glue/jobs/output。
- OutputSchemas – GlueSchema 对象的数组。

  指定自定义 Athena 源的数据架构。

JDBC 结构 ConnectorSource

指定一个指向 JDBC 数据源的连接器。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据源的名称。
- ConnectionName – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  与连接器关联的连接的名称。
- ConnectorName – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  帮助访问 AWS Glue Studio 中数据存储的连接器的名称。
- ConnectionType – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定指向 JDBC 数据存储的连接的连接类型，如 marketplace.jdbc 或 custom.jdbc。
- AdditionalOptions – 一个 JDBC ConnectorOptions 对象。

用于连接器的其他连接选项。

- ConnectionTable – UTF-8 字符串，与 Custom string pattern #41 匹配。

  数据源中的表的名称。

- Query – UTF-8 字符串，与 Custom string pattern #42 匹配。

  从中获取数据的表或 SQL 查询。您可以指定 ConnectionTable 或 query，但不能同时指定两者。

- OutputSchemas – GlueSchema 对象的数组。

  指定自定义 JDBC 源的数据架构。

SparkConnectorSource 结构

指定一个指向 Apache Spark 数据源的连接器。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据源的名称。

- ConnectionName – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  与连接器关联的连接的名称。

- ConnectorName – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  帮助访问 AWS Glue Studio 中数据存储的连接器的名称。

- ConnectionType – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定指向 Apache Spark 数据存储的连接的连接类型，如 marketplace.spark 或 custom.spark。

- AdditionalOptions – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  用于连接器的其他连接选项。

- OutputSchemas – GlueSchema 对象的数组。

指定自定义 Spark 源的数据架构。

CatalogSource 结构

在数据目录中指定 AWS Glue 数据存储。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据存储的名称。
- Database – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库的名称。
- Table – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库中的表的名称。

MySQL CatalogSource 结构

在数据目录中指定一个 MySQL AWS Glue 数据源。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据源的名称。
- Database – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库的名称。
- Table – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库中的表的名称。

PostgreSQL 结构 CatalogSource

在数据目录中指定 PostgresSQL 数据源。 AWS Glue

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据源的名称。
- Database – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库的名称。
- Table – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库中的表的名称。

OracleSQL 结构 CatalogSource

在数据目录中指定 Oracle AWS Glue 数据源。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据源的名称。
- Database – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库的名称。
- Table – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库中的表的名称。

微软 SQL 结构 ServerCatalogSource

在 AWS Glue 数据目录中指定一个 Microsoft SQL Server 数据源。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据源的名称。
- Database – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库的名称。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库中的表的名称。

CatalogKinesisSource 结构

在数据目录中指定 Kinesis AWS Glue 数据源。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据源的名称。
- `WindowSize` – 数字（整数），至多为“无”。

  处理每个微批处理所花费的时间量。
- `DetectSchema` – 布尔值。

  是否从传入的数据中自动确定架构。
- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库中的表的名称。
- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库的名称。
- `StreamingOptions` – 一个 KinesisStreamingSourceOptions 对象。

  用于 Kinesis 串流数据源的其他选项。
- `DataPreviewOptions` – 一个 StreamingDataPreviewOptions 对象。

  用于数据预览的其他选项。

DirectKinesisSource 结构

指定一个直接 Amazon Kinesis 数据源。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

数据源的名称。

- `WindowSize` – 数字（整数），至多为“无”。

  处理每个微批处理所花费的时间量。

- `DetectSchema` – 布尔值。

  是否从传入的数据中自动确定架构。

- `StreamingOptions` – 一个 KinesisStreamingSourceOptions 对象。

  用于 Kinesis 串流数据源的其他选项。

- `DataPreviewOptions` – 一个 StreamingDataPreviewOptions 对象。

  用于数据预览的其他选项。

KinesisStreamingSourceOptions 结构

用于 Amazon Kinesis 串流数据源的其他选项。

字段

- `EndpointUrl` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  Kinesis 端点的 URL。

- `StreamName` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  Kinesis 数据流的名称。

- `Classification` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  一个可选分类。

- `Delimiter` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定分隔符。

- `StartingPosition` – UTF-8 字符串（有效值：`latest="LATEST"` | `trim_horizon="TRIM_HORIZON"` | `earliest="EARLIEST"` | `timestamp="TIMESTAMP"`）。

  要从中读取数据的 Kinesis 数据流中的起始位置。可能的值是 `"latest"`、`"trim_horizon"`、`"earliest"` 或以模式 `yyyy-mm-ddTHH:MM:SSZ`

采用 UTC 格式的时间戳字符串（其中 Z 表示带有 +/-的 UTC 时区偏移量。例如：“2023-04-04T08:00:00-04:00”）。默认值为 "latest"。

注意：仅版本 4.0 或更高 AWS Glue 版本支持使用 UTC 格式的时间戳字符串值来表示“StartingPosition”。

- MaxFetchTimeInMs – 数字（长型），至多为“无”。

  作业执行程序从 Kinesis 数据流中读取当前批处理记录所花费的最长时间，以毫秒为单位指定。在这段时间内可以进行多次 GetRecords API 调用。默认值为 1000。

- MaxFetchRecordsPerShard – 数字（长型），至多为“无”。

  每个微批次将从 Kinesis 数据流中的每个分片获取的最大记录数。注意：如果流式传输作业已经从 Kinesis 读取了额外的记录（在同一个 get-records 调用中），则客户端可以超过此限制。如果 MaxFetchRecordsPerShard 需要严格，则必须是 MaxRecordPerRead 的整数倍。默认值为 100000。

- MaxRecordPerRead – 数字（长型），至多为“无”。

  每项 getRecords 操作中要从 Kinesis 数据流获取的最大记录数。默认值为 10000。

- AddIdleTimeBetweenReads – 布尔值。

  在两项连续 getRecords 操作之间添加时间延迟。默认值为 "False"。此选项仅适用于 Glue 版本 2.0 及更高版本。

- IdleTimeBetweenReadsInMs – 数字（长型），至多为“无”。

  两项连续 getRecords 操作之间的最短时间延迟，以毫秒为单位指定。默认值为 1000。此选项仅适用于 Glue 版本 2.0 及更高版本。

- DescribeShardInterval – 数字（长型），至多为“无”。

  脚本需要考虑重新分片的两次 ListShards API 调用之间的最短时间间隔。默认值为 1s。

- NumRetries – 数字（整数），至多为“无”。

  Kinesis Data Streams API 请求的最大重试次数。默认值为 3。

- RetryIntervalMs – 数字（长型），至多为“无”。

  重试 Kinesis Data Streams API 调用之前的冷却时间（以毫秒为单位指定）。默认值为 1000。

- MaxRetryIntervalMs – 数字（长型），至多为“无”。

Kinesis Data Streams API 调用的两次重试之间的最长冷却时间（以毫秒为单位指定）。默认值为 10000。

- AvoidEmptyBatches – 布尔值。

  在批处理开始之前检查 Kinesis 数据流中是否有未读数据，避免创建空白微批处理任务。默认值为 "False"。

- StreamArn – UTF-8 字符串，与 Custom string pattern #40 匹配。

  Kinesis 数据流的 Amazon Resource Name (ARN)。

- RoleArn – UTF-8 字符串，与 Custom string pattern #40 匹配。

  要使用 AWS Security Token Service (AWS STS) 代入的角色的 Amazon Resource Name (ARN)。此角色必须拥有针对 Kinesis 数据流执行描述或读取记录操作的权限。在访问其他账户中的数据流时，必须使用此参数。与 "awsSTSSessionName" 结合使用。

- RoleSessionName – UTF-8 字符串，与 Custom string pattern #40 匹配。

  使用 AWS STS 代入角色的会话的标识符。在访问其他账户中的数据流时，必须使用此参数。与 "awsSTSRoleARN" 结合使用。

- AddRecordTimestamp – UTF-8 字符串，与 Custom string pattern #40 匹配。

  当选项设置为 'true' 时，数据输出将包含一个名为 "__src_timestamp" 的附加列，表示数据流收到相应记录的时间。默认值为‘false’。4.0 或更高 AWS Glue 版本支持此选项。

- EmitConsumerLagMetrics – UTF-8 字符串，与 Custom string pattern #40 匹配。

  当此选项设置为 “true” 时，对于每个批次，它将发出从直播收到的最旧记录到其到达时间之间的持续时间内的 AWS Glue 指标。 CloudWatch该指标的名字是 “glue.driver.streaming”。maxConsumerLagInMs”。默认值为‘false’。4.0 或更高 AWS Glue 版本支持此选项。

- StartingTimestamp – UTF-8 字符串。

  Kinesis 数据流中开始读取数据的记录的时间戳。可能的值是以模式 yyyy-mm-ddTHH:MM:SSZ 采用 UTC 格式的时间戳字符串（其中 表示带有 +/-的 UTC 时区偏移量。例如：“2023-04-04T08:00:00+08:00”）。

CatalogKafkaSource 结构

指定数据目录中的一个 Apache Kafka 数据存储。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据存储的名称。
- WindowSize – 数字（整数），至多为“无”。

  处理每个微批处理所花费的时间量。
- DetectSchema – 布尔值。

  是否从传入的数据中自动确定架构。
- Table – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库中的表的名称。
- Database – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库的名称。
- StreamingOptions – 一个 KafkaStreamingSourceOptions 对象。

  指定串流选项。
- DataPreviewOptions – 一个 StreamingDataPreviewOptions 对象。

  指定与用于查看数据样本的数据预览相关的选项。

DirectKafkaSource 结构

指定一个 Apache Kafka 数据存储。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据存储的名称。
- StreamingOptions – 一个 KafkaStreamingSourceOptions 对象。

  指定串流选项。
- WindowSize – 数字（整数），至多为“无”。

  处理每个微批处理所花费的时间量。

- DetectSchema – 布尔值。

  是否从传入的数据中自动确定架构。

- DataPreviewOptions – 一个 StreamingDataPreviewOptions 对象。

  指定与用于查看数据样本的数据预览相关的选项。

KafkaStreamingSourceOptions 结构

用于串流的其他选项。

字段

- BootstrapServers – UTF-8 字符串，与 Custom string pattern #40 匹配。

  引导服务器 URL 的列表，例如，作为 b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094。此选项必须在 API 调用中指定，或在数据目录的表元数据中定义。

- SecurityProtocol – UTF-8 字符串，与 Custom string pattern #40 匹配。

  用于与代理通信的协议。可能的值为 "SSL" 或 "PLAINTEXT"。

- ConnectionName – UTF-8 字符串，与 Custom string pattern #40 匹配。

  连接的名称。

- TopicName – UTF-8 字符串，与 Custom string pattern #40 匹配。

  Apache Kafka 中指定的主题名称。您必须指定 "topicName"、"assign" 或 "subscribePattern" 中的至少一个。

- Assign – UTF-8 字符串，与 Custom string pattern #40 匹配。

  要使用的特定 TopicPartitions。您必须指定 "topicName"、"assign" 或 "subscribePattern" 中的至少一个。

- SubscribePattern – UTF-8 字符串，与 Custom string pattern #40 匹配。

  标识要订阅的主题列表的 Java 正则表达式字符串。您必须指定 "topicName"、"assign" 或 "subscribePattern" 中的至少一个。

- Classification – UTF-8 字符串，与 Custom string pattern #40 匹配。

  一个可选分类。

- `Delimiter` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定分隔符。

- `StartingOffsets` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  Kafka 主题中读取数据的起始位置。可能的值为 `"earliest"` 或 `"latest"`。默认值为 `"latest"`。

- `EndingOffsets` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  批处理查询结束时的终点。可能值为 `"latest"`，或者为每个 `TopicPartition` 指定结束偏移的 JSON 字符串。

- `PollTimeoutMs` – 数字（长型），至多为“无”。

  Spark 任务执行程序中，从 Kafka 轮询数据的超时时间（以毫秒为单位）。默认值为 `512`。

- `NumRetries` – 数字（整数），至多为“无”。

  获取 Kafka 偏移失败前的重试次数。默认值为 `3`。

- `RetryIntervalMs` – 数字（长型），至多为“无”。

  重试获取 Kafka 偏移前的等待时间（以毫秒为单位）。默认值为 `10`。

- `MaxOffsetsPerTrigger` – 数字（长型），至多为“无”。

  每个触发间隔处理的最大偏移数的速率限制。指定的总偏移数跨不同卷的 `topicPartitions` 按比例分割。默认值为 null，这意味着使用者读取所有偏移，直到已知的最新偏移。

- `MinPartitions` – 数字（整数），至多为“无”。

  从 Kafka 读取数据的所需最小分区数。默认值为 null，这意味着 Spark 分区数等于 Kafka 分区数。

- `IncludeHeaders` – 布尔值。

  是否包含 Kafka 标头。当选项设置为“true”时，数据输出将包含一个名为“glue_streaming_kafka_headers”的附加列，类型为 `Array[Struct(key: String, value: String)]`。默认值为“false”。此选项仅在 3.0 或更高 AWS Glue 版本中可用。

- `AddRecordTimestamp` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  当选项设置为 'true' 时，数据输出将包含一个名为 "__src_timestamp" 的附加列，表示主题收到相应记录的时间。默认值为‘false’。4.0 或更高 AWS Glue 版本支持此选项。

- `EmitConsumerLagMetrics` – UTF-8 字符串，与 Custom string pattern #40 匹配。

当此选项设置为 “true” 时，对于每个批次，它将发出从主题收到的最旧记录到该记录到达的时间之间的持续时间内的 AWS Glue 指标。 CloudWatch该指标的名字是 “glue.driver.streaming”。maxConsumerLagInMs”。默认值为‘false’。4.0 或更高 AWS Glue 版本支持此选项。

- `StartingTimestamp` – UTF-8 字符串。

  Kafka 主题中开始读取数据的记录时间戳。可能的值是以模式 `yyyy-mm-ddTHH:MM:SSZ` 采用 UTC 格式的时间戳字符串（其中 表示带有 +/-的 UTC 时区偏移量。例如：“2023-04-04T08:00:00+08:00”）。

  只能设置一个 `StartingTimestamp` 或 `StartingOffsets`。

RedshiftSource 结构

指定一个 Amazon Redshift 数据存储。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  Amazon Redshift 数据存储的名称。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库表。

- `RedshiftTmpDir` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  从数据库中复制时，可以用于暂存临时数据的 Amazon S3 路径。

- `TmpDirIAMRole` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  拥有权限的 IAM 角色。

AmazonRedshiftSource 结构

指定 Amazon Redshift 来源。

字段

- Name – UTF-8 字符串，与 Custom string pattern #43 匹配。

  Amazon Redshift 来源的名称。
- Data – 一个 AmazonRedshiftNodeData 对象。

  指定 Amazon Redshift 源节点的数据。

AmazonRedshiftNodeData 结构

指定一个 Amazon Redshift 节点。

字段

- AccessType – UTF-8 字符串，与 Custom string pattern #39 匹配。

  Redshift 连接的访问类型。可以是直接连接或目录连接。
- SourceType – UTF-8 字符串，与 Custom string pattern #39 匹配。

  用于指定特定表是源查询还是自定义查询的源类型。
- Connection – 一个 选项 对象。

  与 Redshift 集群的 AWS Glue 连接。
- Schema – 一个 选项 对象。

  使用直接连接时的 Redshift 架构名称。
- Table – 一个 选项 对象。

  使用直接连接时的 Redshift 表名称。
- CatalogDatabase – 一个 选项 对象。

  使用 AWS Glue 数据目录时数据目录数据库的名称。
- CatalogTable – 一个 选项 对象。

  使用 AWS Glue 数据目录时的数据目录表名。
- CatalogRedshiftSchema – UTF-8 字符串。

  使用数据目录时的 Redshift 架构名称。

- `CatalogRedshiftTable` – UTF-8 字符串。

  要从中进行读取的数据库表。

- `TempDir` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  从数据库中复制时，可以用于暂存临时数据的 Amazon S3 路径。

- `IamRole` – 一个 选项 对象。

  可选。连接到 S3 时使用的角色名称。留空时，IAM 角色将默认为作业中的角色。

- `AdvancedOptions` – AmazonRedshiftAdvancedOption 对象的数组。

  连接到 Redshift 集群时为可选值。

- `SampleQuery` – UTF-8 字符串。

  当 Redshift 源为 “查询” 时，用于从 Redshift 源中获取数据 SourceType的 SQL。

- `PreAction` – UTF-8 字符串。

  使用 upsert 运行 MERGE 或 APPEND 之前使用的 SQL。

- `PostAction` – UTF-8 字符串。

  使用 upsert 运行 MERGE 或 APPEND 之前使用的 SQL。

- `Action` – UTF-8 字符串。

  指定写入 Redshift 集群的操作方式。

- `TablePrefix` – UTF-8 字符串，与 Custom string pattern #39 匹配。

  指定表的前缀。

- `Upsert` – 布尔值。

  执行 APPEND 时，在 Redshift 上使用的操作会失效。

- `MergeAction` – UTF-8 字符串，与 Custom string pattern #39 匹配。

  该操作用于确定如何处理 Redshift 接收器中的 MERGE。

- `MergeWhenMatched` – UTF-8 字符串，与 Custom string pattern #39 匹配。

  当现有记录与新记录匹配时，该操作用于确定如何处理 Redshift 接收器中的 MERGE。

- `MergeWhenNotMatched` – UTF-8 字符串，与 Custom string pattern #39 匹配。

当现有记录与新记录不匹配时，该操作用于确定如何处理 Redshift 接收器中的 MERGE。

- `MergeClause` – UTF-8 字符串。

  自定义合并中用于处理匹配记录的 SQL。

- `CrawlerConnection` – UTF-8 字符串。

  指定与所用目录表关联的连接的名称。

- `TableSchema` – 选项 对象的数组。

  给定节点的架构输出数组。

- `StagingTable` – UTF-8 字符串。

  使用 upsert 执行 MERGE 或 APPEND 时使用的临时暂存表的名称。

- `SelectedColumns` – 选项 对象的数组。

  使用 upsert 执行 MERGE 或 APPEND 时用于确定匹配记录的列名列表。

AmazonRedshiftAdvancedOption 结构

连接到 Redshift 集群时指定可选值。

字段

- `Key` – UTF-8 字符串。

  其他连接选项的键。

- `Value` – UTF-8 字符串。

  其他连接选项的值。

选项结构

指定选项值。

字段

- `Value` – UTF-8 字符串，与 Custom string pattern #40 匹配。

指定选项的值。

- `Label` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定选项的标签。

- `Description` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定选项的描述。

S3 CatalogSource 结构

在数据目录中指定 Amazon S3 AWS Glue 数据存储。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据存储的名称。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库表。

- `PartitionPredicate` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  满足此谓词的分区将被删除。这些分区中保留期内的文件不会被删除。设置为 "" – 默认情况下为空。

- `AdditionalOptions` – 一个 S3 SourceAdditionalOptions 对象。

  指定其他连接选项。

S3 SourceAdditionalOptions 结构

为 Amazon S3 数据存储指定其他连接选项。

字段

- `BoundedSize` – 数字（长型）。

设置要处理的数据集的目标大小的上限（以字节为单位）。

- BoundedFiles – 数字（长型）。

  设置要处理的文件的目标数量的上限。

S3 CsvSource 结构

指定一个存储在 Amazon S3 中的命令分隔值 (CSV) 数据存储。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据存储的名称。

- Paths – 必填：UTF-8 字符串数组。

  要从中进行读取的 Amazon S3 路径的列表。

- CompressionType – UTF-8 字符串（有效值：gzip="GZIP" | bzip2="BZIP2"）。

  指定数据压缩方式。通常，如果数据有标准文件扩展名，则不需要指定。可能的值为 "gzip" 和 "bzip"。

- Exclusions – UTF-8 字符串数组。

  包含要排除的 Unix 样式 glob 模式的 JSON 列表的字符串。例如，"[\"**.pdf\"]" 排除所有 PDF 文件。

- GroupSize – UTF-8 字符串，与 Custom string pattern #40 匹配。

  目标组大小（以字节为单位）。默认值根据输入数据大小和群集大小进行计算。当少于 50,000 个输入文件时，"groupFiles" 必须设置为 "inPartition"，此选项才能生效。

- GroupFiles – UTF-8 字符串，与 Custom string pattern #40 匹配。

  当输入包含超过 50,000 个文件时，预设情况下将启用文件分组。当少于 50,000 个文件时，要启用分组，请将此参数设置为“inPartition”。当超过 50,000 个文件时，若要禁用分组，请将此参数设置为 "none"。

- Recurse – 布尔值。

  如果设置为 true（真），则以递归方式读取指定路径下的所有子目录中的文件。

- MaxBand – 数字（整数），至多为“无”。

  此选项控制 s3 列表可能保持一致的持续时间（以毫秒为单位）。为了 JobBookmarks考虑 Amazon S3 的最终一致性，修改时间戳在最近 MaxBand 毫秒以内的文件会被特别跟踪。大多数用户不需要设置此选项。默认值为 900000 毫秒或 15 分钟。

- MaxFilesInBand – 数字（整数），至多为“无”。

  此选项指定在最后 maxBand 秒内可保存的最大文件数量。如果超过此值，额外的文件将会跳过，且只能在下一次作业运行中处理。

- AdditionalOptions – 一个 S3 DirectSourceAdditionalOptions 对象。

  指定其他连接选项。

- Separator – 必填：UTF-8 字符串（有效值：comma="COMMA" | ctrla="CTRLA" | pipe="PIPE" | semicolon="SEMICOLON" | tab="TAB"）。

  指定分隔符。默认值为逗号：“,”，但也可以指定任何其他字符。

- Escaper – UTF-8 字符串，与 Custom string pattern #41 匹配。

  指定要用于转义的字符。此选项仅在读取 CSV 文件时使用。默认值为 none。如果启用，则按原样使用紧跟其后的字符，一小组已知的转义符（\n、\r、\t 和 \0）除外。

- QuoteChar – 必填：UTF-8 字符串（有效值：quote="QUOTE" | quillemet="QUILLEMET" | single_quote="SINGLE_QUOTE" | disabled="DISABLED"）。

  指定要用于引用的字符。默认值为双引号：'"'。将这设置为 -1 可完全关闭引用。

- Multiline – 布尔值。

  指定单个记录能否跨越多行的布尔值。当字段包含带引号的换行符时，会出现此选项。如果有任何记录跨越多行，则您必须将此选项设置为 True（真）。默认值为 False，它允许在分析过程中更积极地拆分文件。

- WithHeader – 布尔值。

  指定是否将第一行视为标题的布尔值。默认值为 False。

- WriteHeader – 布尔值。

  指定是否将标题写入输出的布尔值。默认值为 True。

- SkipFirst – 布尔值。

  指定是否跳过第一个数据行的布尔值。默认值为 False。

- `OptimizePerformance` – 布尔值。

  指定是否使用高级 SIMD CSV 读取器以及基于 Apache Arrow 的列式内存格式的布尔值。仅在 3.0 AWS Glue 版本中可用。

- `OutputSchemas` – GlueSchema 对象的数组。

  指定 S3 CSV 源的数据架构。

DirectJDBCSource 结构

指定直接 JDBC 数据源连接。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  JDBC 数据源连接的名称。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  JDBC 数据源连接的数据库。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  JDBC 数据源连接的表。

- `ConnectionName` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  JDBC 数据源的连接名称。

- `ConnectionType` – 必填：UTF-8 字符串（有效值：`sqlserver` | `mysql` | `oracle` | `postgresql` | `redshift` ）。

  JDBC 源的连接类型。

- `RedshiftTmpDir` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  JDBC Redshift 数据源的临时目录。

S3 DirectSourceAdditionalOptions 结构

为 Amazon S3 数据存储指定其他连接选项。

字段

- BoundedSize – 数字（长型）。

  设置要处理的数据集的目标大小的上限（以字节为单位）。
- BoundedFiles – 数字（长型）。

  设置要处理的文件的目标数量的上限。
- EnableSamplePath – 布尔值。

  设置选项以启用示例路径。
- SamplePath – UTF-8 字符串，与 Custom string pattern #40 匹配。

  如果启用，请指定示例路径。

S3 JsonSource 结构

指定一个存储在 Amazon S3 中的 JSON 数据存储。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据存储的名称。
- Paths – 必填：UTF-8 字符串数组。

  要从中进行读取的 Amazon S3 路径的列表。
- CompressionType – UTF-8 字符串（有效值：gzip="GZIP" | bzip2="BZIP2"）。

  指定数据压缩方式。通常，如果数据有标准文件扩展名，则不需要指定。可能的值为 "gzip" 和 "bzip"。
- Exclusions – UTF-8 字符串数组。

  包含要排除的 Unix 样式 glob 模式的 JSON 列表的字符串。例如，"[\"**.pdf\"]" 排除所有 PDF 文件。
- GroupSize – UTF-8 字符串，与 Custom string pattern #40 匹配。

  目标组大小（以字节为单位）。默认值根据输入数据大小和群集大小进行计算。当少于 50,000 个输入文件时，"groupFiles" 必须设置为 "inPartition"，此选项才能生效。

- `GroupFiles` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  当输入包含超过 50,000 个文件时，预设情况下将启用文件分组。当少于 50,000 个文件时，要启用分组，请将此参数设置为“inPartition”。当超过 50,000 个文件时，若要禁用分组，请将此参数设置为 `"none"`。

- `Recurse` – 布尔值。

  如果设置为 true（真），则以递归方式读取指定路径下的所有子目录中的文件。

- `MaxBand` – 数字（整数），至多为“无”。

  此选项控制 s3 列表可能保持一致的持续时间（以毫秒为单位）。为了 JobBookmarks考虑 Amazon S3 的最终一致性，修改时间戳在最近 MaxBand 毫秒以内的文件会被特别跟踪。大多数用户不需要设置此选项。默认值为 900000 毫秒或 15 分钟。

- `MaxFilesInBand` – 数字（整数），至多为“无”。

  此选项指定在最后 maxBand 秒内可保存的最大文件数量。如果超过此值，额外的文件将会跳过，且只能在下一次作业运行中处理。

- `AdditionalOptions` – 一个 S3 DirectSourceAdditionalOptions 对象。

  指定其他连接选项。

- `JsonPath` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  定义 JSON 数据的 JsonPath 字符串。

- `Multiline` – 布尔值。

  指定单个记录能否跨越多行的布尔值。当字段包含带引号的换行符时，会出现此选项。如果有任何记录跨越多行，则您必须将此选项设置为 True（真）。默认值为 `False`，它允许在分析过程中更积极地拆分文件。

- `OutputSchemas` – GlueSchema 对象的数组。

  指定 S3 JSON 源的数据架构。

S3 ParquetSource 结构

指定一个存储在 Amazon S3 中的 Apache Parquet 数据存储。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据存储的名称。

- Paths – 必填：UTF-8 字符串数组。

  要从中进行读取的 Amazon S3 路径的列表。

- CompressionType – UTF-8 字符串（有效值：snappy="SNAPPY" | lzo="LZO" | gzip="GZIP" | uncompressed="UNCOMPRESSED" | none="NONE"）。

  指定数据压缩方式。通常，如果数据有标准文件扩展名，则不需要指定。可能的值为 "gzip" 和 "bzip"。

- Exclusions – UTF-8 字符串数组。

  包含要排除的 Unix 样式 glob 模式的 JSON 列表的字符串。例如，"[\"**.pdf\"]" 排除所有 PDF 文件。

- GroupSize – UTF-8 字符串，与 Custom string pattern #40 匹配。

  目标组大小（以字节为单位）。默认值根据输入数据大小和群集大小进行计算。当少于 50,000 个输入文件时，"groupFiles" 必须设置为 "inPartition"，此选项才能生效。

- GroupFiles – UTF-8 字符串，与 Custom string pattern #40 匹配。

  当输入包含超过 50,000 个文件时，预设情况下将启用文件分组。当少于 50,000 个文件时，要启用分组，请将此参数设置为“inPartition”。当超过 50,000 个文件时，若要禁用分组，请将此参数设置为 "none"。

- Recurse – 布尔值。

  如果设置为 true（真），则以递归方式读取指定路径下的所有子目录中的文件。

- MaxBand – 数字（整数），至多为“无”。

  此选项控制 s3 列表可能保持一致的持续时间（以毫秒为单位）。为了 JobBookmarks考虑 Amazon S3 的最终一致性，修改时间戳在最近 MaxBand 毫秒以内的文件会被特别跟踪。大多数用户不需要设置此选项。默认值为 900000 毫秒或 15 分钟。

- MaxFilesInBand – 数字（整数），至多为“无”。

  此选项指定在最后 maxBand 秒内可保存的最大文件数量。如果超过此值，额外的文件将会跳过，且只能在下一次作业运行中处理。

- `AdditionalOptions` – 一个 S3 DirectSourceAdditionalOptions 对象。

  指定其他连接选项。
- `OutputSchemas` – GlueSchema 对象的数组。

  指定 S3 Parquet 源的数据架构。

S3 DeltaSource 结构

指定存储在中的三角洲湖数据源 Amazon S3。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  Delta Lake 源的名称。
- `Paths` – 必填：UTF-8 字符串数组。

  要从中进行读取的 Amazon S3 路径的列表。
- `AdditionalDeltaOptions` – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定其他连接选项。
- `AdditionalOptions` – 一个 S3 DirectSourceAdditionalOptions 对象。

  为连接器指定其他选项。
- `OutputSchemas` – GlueSchema 对象的数组。

  指定 Delta Lake 源的数据架构。

S3 CatalogDeltaSource 结构

指定在数据目录中注册的 Delta Lake AWS Glue 数据源。数据源必须存储在 Amazon S3。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

Delta Lake 数据源的名称。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库的名称。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库中的表的名称。

- `AdditionalDeltaOptions` – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定其他连接选项。

- `OutputSchemas` – GlueSchema 对象的数组。

  指定 Delta Lake 源的数据架构。

CatalogDeltaSource 结构

指定在数据目录中注册的 Delta Lake AWS Glue 数据源。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  Delta Lake 数据源的名称。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库的名称。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库中的表的名称。

- `AdditionalDeltaOptions` – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

指定其他连接选项。

- `OutputSchemas` – GlueSchema 对象的数组。

  指定 Delta Lake 源的数据架构。

S3 HudiSource 结构

指定存储在中的 Amazon S3 Hudi 数据源。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  Hudi 源的名称。

- `Paths` – 必填：UTF-8 字符串数组。

  要从中进行读取的 Amazon S3 路径的列表。

- `AdditionalHudiOptions` – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定其他连接选项。

- `AdditionalOptions` – 一个 S3 DirectSourceAdditionalOptions 对象。

  为连接器指定其他选项。

- `OutputSchemas` – GlueSchema 对象的数组。

  指定 Hudi 源的数据架构。

S3 CatalogHudiSource 结构

指定在数据目录中注册的 Hudi AWS Glue 数据源。Hudi 数据源必须存储在。 Amazon S3

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

数据来源的名称。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库的名称。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库中的表的名称。

- `AdditionalHudiOptions` – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定其他连接选项。

- `OutputSchemas` – GlueSchema 对象的数组。

  指定 Hudi 源的数据架构。

CatalogHudiSource 结构

指定在数据目录中注册的 Hudi AWS Glue 数据源。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据来源的名称。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库的名称。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库中的表的名称。

- `AdditionalHudiOptions` – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

指定其他连接选项。

- `OutputSchemas` – GlueSchema 对象的数组。

  指定 Hudi 源的数据架构。

DynamoDB 结构 CatalogSource

在数据目录中指定 DynamoDB 数据源。 AWS Glue

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据源的名称。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库的名称。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库中的表的名称。

RelationalCatalogSource 结构

指定 AWS Glue 数据目录中的一个关系数据库数据源。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据源的名称。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库的名称。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库中的表的名称。

JDBC 结构 ConnectorTarget

指定一个在 Apache Paric 列式存储中写入 Amazon S3 的数据目标。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。

- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。

- `ConnectionName` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  与连接器关联的连接的名称。

- `ConnectionTable` – 必填： UTF-8 字符串，与 Custom string pattern #41 匹配。

  数据目标中表的名称。

- `ConnectorName` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  将使用的连接器的名称。

- `ConnectionType` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定指向 JDBC 数据目标的连接的连接类型，如 marketplace.jdbc 或 custom.jdbc。

- `AdditionalOptions` – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  用于连接器的其他连接选项。

- `OutputSchemas` – GlueSchema 对象的数组。

  指定 JDBC 目标的数据架构。

SparkConnectorTarget 结构

指定一个使用 Apache Spark 连接器的目标。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。
- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。
- ConnectionName – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  Apache Spark 连接器的连接名称。
- ConnectorName – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  Apache Spark 连接器的名称。
- ConnectionType – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定指向 Apache Spark 数据存储的连接的连接类型，如 marketplace.spark 或 custom.spark。
- AdditionalOptions – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  用于连接器的其他连接选项。
- OutputSchemas – GlueSchema 对象的数组。

  指定自定义 Spark 目标的数据架构。

BasicCatalogTarget 结构

指定使用 AWS Glue 数据目录表的目标。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。
- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  包含您要用作目标的表的数据库。此数据库必须已存在于数据目录中。
- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  定义输出数据架构的表。此表必须已存在于数据目录中。

MySQL CatalogTarget 结构

指定一个使用 MySQL 的目标。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。
- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。
- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要向其写入的数据库的名称。
- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的数据库中的表的名称。

PostgreSQL 结构 CatalogTarget

指定一个使用 Postgres SQL 的目标。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。
- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。
- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

要向其写入的数据库的名称。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的数据库中的表的名称。

OracleSQL 结构 CatalogTarget

指定一个使用 Oracle SQL 的目标。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。

- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要向其写入的数据库的名称。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的数据库中的表的名称。

微软 SQL 结构 ServerCatalogTarget

指定一个使用 Microsoft SQL 的目标。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。

- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

要向其写入的数据库的名称。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的数据库中的表的名称。

RedshiftTarget 结构

指定一个使用 Amazon Redshift 的目标。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。

- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要向其写入的数据库的名称。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的数据库中的表的名称。

- `RedshiftTmpDir` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  从数据库中复制时，可以用于暂存临时数据的 Amazon S3 路径。

- `TmpDirIAMRole` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  拥有权限的 IAM 角色。

- `UpsertRedshiftOptions` – 一个 UpsertRedshiftTargetOptions 对象。

  写入 Redshift 目标时用于配置 upsert 操作的一组选项。

AmazonRedshiftTarget 结构

指定一个 Amazon Redshift 目标。

字段

- Name – UTF-8 字符串，与 Custom string pattern #43 匹配。

  Amazon Redshift 目标的名称。
- Data – 一个 AmazonRedshiftNodeData 对象。

  指定 Amazon Redshift 目标节点的数据。
- Inputs – UTF-8 字符串数组，不少于 1 个字符串，不超过 1 个字符串。

  作为数据目标输入的节点。

UpsertRedshiftTargetOptions 结构

写入 Redshift 目标时用于配置 upsert 操作的选项。

字段

- TableLocation – UTF-8 字符串，与 Custom string pattern #40 匹配。

  Redshift 表的物理位置。
- ConnectionName – UTF-8 字符串，与 Custom string pattern #40 匹配。

  用于写入 Redshift 的连接的名称。
- UpsertKeys – UTF-8 字符串数组。

  用于确定是执行更新还是插入的键。

S3 CatalogTarget 结构

指定使用数据目录写入 Amazon S3 AWS Glue 的数据目标。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。
- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。

- PartitionKeys – UTF-8 字符串数组。

  使用一系列键指定本机分区。

- Table – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的数据库中的表的名称。

- Database – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要向其写入的数据库的名称。

- SchemaChangePolicy – 一个 CatalogSchemaChangePolicy 对象。

  一项指定爬网程序的更新行为的策略。

S3 GlueParquetTarget 结构

指定一个在 Apache Paric 列式存储中写入 Amazon S3 的数据目标。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。

- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。

- PartitionKeys – UTF-8 字符串数组。

  使用一系列键指定本机分区。

- Path – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的单个 Amazon S3 路径。

- Compression – UTF-8 字符串（有效值：snappy="SNAPPY" | lzo="LZO" | gzip="GZIP" | uncompressed="UNCOMPRESSED" | none="NONE"）。

  指定数据压缩方式。通常，如果数据有标准文件扩展名，则不需要指定。可能的值为 "gzip" 和 "bzip"。

- SchemaChangePolicy – 一个 DirectSchemaChangePolicy 对象。

  一项指定爬网程序的更新行为的策略。

CatalogSchemaChangePolicy 结构

一项指定爬网程序的更新行为的策略。

字段

- `EnableUpdateCatalog` – 布尔值。

  当爬网程序发现已更改的架构时，是否使用指定的更新行为。
- `UpdateBehavior` – UTF-8 字符串（有效值：`UPDATE_IN_DATABASE` | `LOG`）。

  爬网程序发现已更改的架构时的更新行为。

S3 DirectTarget 结构

指定一个写入 Amazon S3 的数据目标。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。
- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。
- `PartitionKeys` – UTF-8 字符串数组。

  使用一系列键指定本机分区。
- `Path` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的单个 Amazon S3 路径。
- `Compression` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定数据压缩方式。通常，如果数据有标准文件扩展名，则不需要指定。可能的值为 `"gzip"` 和 `"bzip"`。
- `Format` – 必填：UTF-8 字符串（有效值：`json="JSON"` | `csv="CSV"` | `avro="AVRO"` | `orc="ORC"` | `parquet="PARQUET"` | `hudi="HUDI"` | `delta="DELTA"`）。

  指定目标的数据输出格式。

- SchemaChangePolicy – 一个 DirectSchemaChangePolicy 对象。

  一项指定爬网程序的更新行为的策略。

S3 HudiCatalogTarget 结构

指定写入数据目录中的 Hudi 数据源的目标。 AWS Glue

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。
- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。
- PartitionKeys – UTF-8 字符串数组。

  使用一系列键指定本机分区。
- Table – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的数据库中的表的名称。
- Database – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要向其写入的数据库的名称。
- AdditionalOptions – 必填：键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定用于连接器的其他连接选项。
- SchemaChangePolicy – 一个 CatalogSchemaChangePolicy 对象。

  一项指定爬网程序的更新行为的策略。

S3 HudiDirectTarget 结构

指定写入中 Hudi 数据源的目标。 Amazon S3

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。

- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。

- Path – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的 Hudi 数据来源的Amazon S3 路径。

- Compression – 必填：UTF-8 字符串（有效值：gzip="GZIP" | lzo="LZO" | uncompressed="UNCOMPRESSED" | snappy="SNAPPY" ）。

  指定数据压缩方式。通常，如果数据有标准文件扩展名，则不需要指定。可能的值为 "gzip" 和 "bzip"。

- PartitionKeys – UTF-8 字符串数组。

  使用一系列键指定本机分区。

- Format – 必填：UTF-8 字符串（有效值：json="JSON" | csv="CSV" | avro="AVRO" | orc="ORC" | parquet="PARQUET" | hudi="HUDI" | delta="DELTA" ）。

  指定目标的数据输出格式。

- AdditionalOptions – 必填：键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定用于连接器的其他连接选项。

- SchemaChangePolicy – 一个 DirectSchemaChangePolicy 对象。

  一项指定爬网程序的更新行为的策略。

S3 DeltaCatalogTarget 结构

指定写入数据目录中的 Delta Lake AWS Glue 数据源的目标。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。
- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。
- PartitionKeys – UTF-8 字符串数组。

  使用一系列键指定本机分区。
- Table – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的数据库中的表的名称。
- Database – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要向其写入的数据库的名称。
- AdditionalOptions – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定用于连接器的其他连接选项。
- SchemaChangePolicy – 一个 CatalogSchemaChangePolicy 对象。

  一项指定爬网程序的更新行为的策略。

S3 DeltaDirectTarget 结构

指定写入中三角洲湖数据源的目标 Amazon S3。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。
- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。

- PartitionKeys – UTF-8 字符串数组。

  使用一系列键指定本机分区。

- Path – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的 Delta Lake 数据来源的 Amazon S3 路径。

- Compression – 必填：UTF-8 字符串（有效值：uncompressed="UNCOMPRESSED" | snappy="SNAPPY"）。

  指定数据压缩方式。通常，如果数据有标准文件扩展名，则不需要指定。可能的值为 "gzip" 和 "bzip"。

- Format – 必填：UTF-8 字符串（有效值：json="JSON" | csv="CSV" | avro="AVRO" | orc="ORC" | parquet="PARQUET" | hudi="HUDI" | delta="DELTA"）。

  指定目标的数据输出格式。

- AdditionalOptions – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定用于连接器的其他连接选项。

- SchemaChangePolicy – 一个 DirectSchemaChangePolicy 对象。

  一项指定爬网程序的更新行为的策略。

DirectSchemaChangePolicy 结构

一项指定爬网程序的更新行为的策略。

字段

- EnableUpdateCatalog – 布尔值。

  当爬网程序发现已更改的架构时，是否使用指定的更新行为。

- UpdateBehavior – UTF-8 字符串（有效值：UPDATE_IN_DATABASE | LOG）。

  爬网程序发现已更改的架构时的更新行为。

- Table – UTF-8 字符串，与 Custom string pattern #40 匹配。

指定应用架构更改策略的数据库中的表。

- Database – UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定应用架构更改策略的数据库。

ApplyMapping 结构

指定一个将数据源中的数据属性键映射到数据目标中的数据属性键的转换。您可以重命名键、修改键的数据类型以及选择要从数据集中删除的键。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。

- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  通过其节点名称标识的数据输入。

- Mapping – 必填：Mapping 对象的数组。

  指定数据源中的数据属性键与数据目标中的数据属性键的映射。

Mapping 结构

指定数据属性键的映射。

字段

- ToKey – UTF-8 字符串，与 Custom string pattern #40 匹配。

  应用映射之后，列的名称应该是什么。可与 FromPath 相同。

- FromPath – UTF-8 字符串数组。

  要修改的表或列。

- FromType – UTF-8 字符串，与 Custom string pattern #40 匹配。

  要修改的数据的类型。

- ToType – UTF-8 字符串，与 Custom string pattern #40 匹配。

要修改的数据的数据类型。

- Dropped – 布尔值。

  如果为 true（真），则删除列。

- Children – Mapping 对象的数组。

  仅适用于嵌套数据结构。如果要更改父结构，但也要更改其子结构之一，则可填写此数据结构。它也是 Mapping，但其 FromPath 将是父结构的 FromPath 再加上来自此结构的 FromPath。

  对于子部分，假设您拥有结构：

  ```
  { "FromPath": "OuterStructure", "ToKey": "OuterStructure", "ToType":
  "Struct", "Dropped": false, "Chidlren": [{ "FromPath": "inner", "ToKey":
  "inner", "ToType": "Double", "Dropped": false, }] }
  ```

  您可以指定一个类似如下的 Mapping：

  ```
  { "FromPath": "OuterStructure", "ToKey": "OuterStructure", "ToType":
  "Struct", "Dropped": false, "Chidlren": [{ "FromPath": "inner", "ToKey":
  "inner", "ToType": "Double", "Dropped": false, }] }
  ```

SelectFields 结构

指定一个选择要保留的数据属性键的转换。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。

- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  通过其节点名称标识的数据输入。

- Paths – 必填：UTF-8 字符串数组。

  指向数据结构中变量的 JSON 路径。

DropFields 结构

指定一个选择要删除的数据属性键的转换。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。
- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  通过其节点名称标识的数据输入。
- Paths – 必填：UTF-8 字符串数组。

  指向数据结构中变量的 JSON 路径。

RenameField 结构

指定一个重命名单个数据属性键的转换。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。
- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  通过其节点名称标识的数据输入。
- SourcePath – 必填：UTF-8 字符串数组。

  指向源数据的数据结构中变量的 JSON 路径。
- TargetPath – 必填：UTF-8 字符串数组。

  指向目标数据的数据结构中变量的 JSON 路径。

Spigot 结构

指定一个将数据样本写入 Amazon S3 存储桶的转换。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。

- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  通过其节点名称标识的数据输入。

- Path – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  Amazon S3 中的一个路径，在该路径中，转换会将数据集中的记录子集写入 Amazon S3 存储桶中的 JSON 文件。

- Topk – 数字（整型），不超过 100。

  指定从数据集开头开始写入的一些记录。

- Prob – 数字（双精度），不超过 1。

  选择任何给定记录的概率（最大值为 1 的十进制值）。值 1 表示从数据集中读取的每一行都应包括在示例输出中。

Join 结构

指定一个转换，它将使用指定数据属性键上的比较短语将两个数据集联接到一个数据集。您可以使用内部、外部、左、右、左半和左反联接。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。

- Inputs – 必填：UTF-8 字符串数组，不少于 2 个或不超过 2 个字符串。

  通过其节点名称标识的数据输入。

- JoinType – 必填：UTF-8 字符串（有效值：`equijoin="EQUIJOIN"` | `left="LEFT"` | `right="RIGHT"` | `outer="OUTER"` | `leftsemi="LEFT_SEMI"` | `leftanti="LEFT_ANTI"`）。

  指定要针对数据集执行的联接的类型。

- `Columns` – 必填：JoinColumn 对象的数组，不少于 2 个或不超过 2 个结构。

  要联接的两列的列表。

JoinColumn 结构

指定一个要联接的列。

字段

- `From` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要联接的列。
- `Keys` – 必填：UTF-8 字符串数组。

  要联接的列的键。

SplitFields 结构

指定一个将数据属性键拆分为两个 DynamicFrames 的转换。输出是 DynamicFrames 的集合：一个包含选定的数据属性键，另一个包含剩余的数据属性键。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。
- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  通过其节点名称标识的数据输入。
- `Paths` – 必填：UTF-8 字符串数组。

  指向数据结构中变量的 JSON 路径。

SelectFromCollection 结构

指定一个从 DynamicFrames 的集合中选择一个 DynamicFrame 的转换。输出是选定的 DynamicFrame

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。
- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  通过其节点名称标识的数据输入。
- Index – 必填：数字（整数），至多为“无”。

  DynamicFrame 要选择的索引。

FillMissingValues 结构

指定一个转换，它将查找数据集中缺少值的记录，并添加包含通过推算确定的值的新字段。输入数据集用于训练机器学习模型，该模型确定缺失值应该是什么。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。
- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  通过其节点名称标识的数据输入。
- ImputedPath – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  指向推算的数据集的数据结构中变量的 JSON 路径。
- FilledPath – UTF-8 字符串，与 Custom string pattern #40 匹配。

  指向被填充的数据集的数据结构中变量的 JSON 路径。

Filter 结构

指定一个转换，它将基于筛选条件将一个数据集拆分为两个。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

转换节点的名称。

- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  通过其节点名称标识的数据输入。

- `LogicalOperator` – 必填：UTF-8 字符串（有效值：`AND` | `OR`）。

  用于通过将键值与指定值进行比较来筛选行的运算符。

- `Filters` – 必填：FilterExpression 对象的数组。

  指定一个筛选条件表达式。

FilterExpression 结构

指定一个筛选条件表达式。

字段

- `Operation` – 必填：UTF-8 字符串（有效值：`EQ` | `LT` | `GT` | `LTE` | `GTE` | `REGEX` | `ISNULL`）。

  要在该表达式中执行的操作的类型。

- `Negated` – 布尔值。

  是否要否定该表达式。

- `Values` – 必填：FilterValue 对象的数组。

  筛选条件值的列表。

FilterValue 结构

表示 `FilterExpression` 的值的列表中的单个条目。

字段

- `Type` – 必填：UTF-8 字符串（有效值：`COLUMNEXTRACTED` | `CONSTANT`）。

  筛选条件值的类型。

- `Value` – 必填：UTF-8 字符串数组。

要关联的值。

CustomCode 结构

指定一个转换，它将使用您提供的自定义代码执行数据转换。输出是一个集合 DynamicFrames。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。
- Inputs – 必填：UTF-8 字符串数组，至少 1 个字符串。

  通过其节点名称标识的数据输入。
- Code – 必填： UTF-8 字符串，与 Custom string pattern #35 匹配。

  用于执行数据转换的自定义代码。
- ClassName – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  为自定义代码节点类定义的名称。
- OutputSchemas – GlueSchema 对象的数组。

  指定自定义代码转换的数据架构。

SparkSQL 结构

指定一个转换，您可以在其中使用 Spark SQL 语法输入 SQL 查询以转换数据。输出为单个 DynamicFrame。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。
- Inputs – 必填：UTF-8 字符串数组，至少 1 个字符串。

  通过其节点名称标识的数据输入。您可以将表名称与要在 SQL 查询中使用的每个输入节点关联起来。您选择的名称必须满足 Spark SQL 命名限制。

- `SqlQuery` – 必填： UTF-8 字符串，与 Custom string pattern #42 匹配。

  必须使用 Spark SQL 语法并返回单个数据集的 SQL 查询。

- `SqlAliases` – 必填：SqlAlias 对象的数组。

  别名列表。别名允许您指定在 SQL 中为给定输入使用什么名称。例如，您有一个名为 “” MyDataSource 的数据源。如果你指定`From`为 MyDataSource、as SqlName，`Alias`那么在你的 SQL 中你可以这样做：

  ```
  select * from SqlName
  ```

  然后从中获取数据 MyDataSource。

- `OutputSchemas` – GlueSchema 对象的数组。

  指定 SparkSQL 转换的数据架构。

SqlAlias 结构

表示 `SqlAliases` 的值的列表中的单个条目。

字段

- `From` – 必填： UTF-8 字符串，与 Custom string pattern #39 匹配。

  表，或表中的列。

- `Alias` – 必填： UTF-8 字符串，与 Custom string pattern #41 匹配。

  为表或表中的列提供的临时名称。

DropNullFields 结构

指定一个转换，如果列中的所有值均为“null”，则该转换将从数据集中删除这些列。默认情况下， AWS Glue Studio 会识别空对象，但是某些值（例如空字符串、“null” 字符串、-1 个整数或其他占位符（例如零）不会自动识别为空值。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。

- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  通过其节点名称标识的数据输入。
- `NullCheckBoxList` – 一个 NullCheckBoxList 对象。

  一种结构，它表示是否将某些值识别为要删除的 null 值。
- `NullTextList` – NullValueField 对象的数组，不超过 50 个结构。

  一种结构，它指定了一系列 NullValueField 结构，这些结构表示自定义空值，例如零或其他用作数据集独有的空占位符的值。

  仅当 null 占位符的值和数据类型与数据匹配时，`DropNullFields` 转换才会删除自定义 null 值。

NullCheckBoxList 结构

表示某些值是否被识别为要删除的 null 值。

字段

- `IsEmpty` – 布尔值。

  指定将一个空字符串视为 null 值。
- `IsNullString` – 布尔值。

  指定将一个拼写“null”一词的值视为 null 值。
- `IsNegOne` – 布尔值。

  指定将一个为 -1 的整数值视为 null 值。

NullValueField 结构

表示自定义 null 值，如零值或其他值，用作数据集唯一的 null 占位符。

字段

- `Value` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  null 占位符的值。
- `Datatype` – 必填：一个 DataType 对象。

值的数据类型。

Datatype 结构

表示值的数据类型的结构。

字段

- Id – 必填： UTF-8 字符串，与 Custom string pattern #39 匹配。

  值的数据类型。
- Label – 必填： UTF-8 字符串，与 Custom string pattern #39 匹配。

  分配给 datatype（数据类型）的标签。

Merge 结构

指定一个转换，它将基于指定的主键将 DynamicFrame 与暂存 DynamicFrame 合并以标识记录。不会对重复记录（具有相同主键的记录）去除重复。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。
- Inputs – 必填：UTF-8 字符串数组，不少于 2 个或不超过 2 个字符串。

  通过其节点名称标识的数据输入。
- Source – 必填： UTF-8 字符串，与 Custom string pattern #39 匹配。

  将与暂存 DynamicFrame 合并的源 DynamicFrame。
- PrimaryKeys – 必填：UTF-8 字符串数组。

  要匹配源和暂存动态帧中的记录的主键字段列表。

Union 结构

指定一个转换，它将两个或更多数据集中的行合并到单个结果中。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。

- Inputs – 必填：UTF-8 字符串数组，不少于 2 个或不超过 2 个字符串。

  节点 ID 输入到转换。

- UnionType – 必填：UTF-8 字符串（有效值：ALL | DISTINCT）。

  指示 Union 转换的类型。

  指定ALL将数据源中的所有行联接到生成的行 DynamicFrame。生成的并集不会删除重复行。

  指定DISTINCT删除结果中的重复行 DynamicFrame。

PIIDetection 结构

指定用于识别、删除或掩盖 PII 数据的转换。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。

- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  节点 ID 输入到转换。

- PiiType – 必填：UTF-8 字符串（有效值：RowAudit | RowMasking | ColumnAudit | ColumnMasking）。

  指示 PIIDetection 转换的类型。

- EntityTypesToDetect – 必填：UTF-8 字符串数组。

  指示 PIIDetection 转换将标识为 PII 数据的实体类型。

  PII 类型的实体包括：
  PERSON_NAME、DATE、USA_SNN、EMAIL、USA_ITIN、USA_PASSPORT_NUMBER、PHONE_NUM

- OutputColumnName – UTF-8 字符串，与 Custom string pattern #40 匹配。

指示包含该行中检测到的任何实体类型的输出列名称。

- `SampleFraction` – 数字（双精度），不超过 1。

  指示要在扫描 PII 实体时采样的数据的部分。

- `ThresholdFraction` – 数字（双精度），不超过 1。

  指示要将列标识为 PII 数据所必须满足的数据部分。

- `MaskValue` – UTF-8 字符串，长度不超过 256 个字节，与 Custom string pattern #37 匹配。

  指示将替代检测到的实体的值。

Aggregate 结构

指定一个转换，用于按选定字段对行进行分组并通过指定函数计算聚合值。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  转换节点的名称。

- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  指定要用作聚合转换输入的字段和行。

- `Groups` – 必填：UTF-8 字符串数组。

  指定要对其进行分组的字段。

- `Aggs`：必填：AggregateOperation 对象的数组，不少于 1 个或不超过 30 个结构。

  指定要对指定字段执行的聚合函数。

DropDuplicates 结构

指定一个用于从数据集中删除重复数据行的转换。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

转换节点的名称。

- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  通过其节点名称标识的数据输入。

- `Columns` – UTF-8 字符串数组。

  重复时需要合并或删除的列的名称。

GovernedCatalogTarget 结构

指定使用数据目录写入 Amazon S3 AWS Glue 的数据目标。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据目标的名称。

- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为数据目标输入的节点。

- `PartitionKeys` – UTF-8 字符串数组。

  使用一系列键指定本机分区。

- `Table` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要写入的数据库中的表的名称。

- `Database` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要向其写入的数据库的名称。

- `SchemaChangePolicy` – 一个 CatalogSchemaChangePolicy 对象。

  用于指定监管目录的更新行为的策略。

GovernedCatalogSource 结构

在受管控的数据目录中指定 AWS Glue 数据存储。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据存储的名称。

- Database – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库。

- Table – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  要从中进行读取的数据库表。

- PartitionPredicate – UTF-8 字符串，与 Custom string pattern #40 匹配。

  满足此谓词的分区将被删除。这些分区中保留期内的文件不会被删除。设置为 "" – 默认情况下为空。

- AdditionalOptions – 一个 S3 SourceAdditionalOptions 对象。

  指定其他连接选项。

AggregateOperation 结构

指定在聚合转换中执行聚合所需的一组参数。

字段

- Column – 必填：UTF-8 字符串数组。

  指定数据集上将应用聚合函数的列。

- AggFunc：必填：UTF-8 字符串（有效值：avg | countDistinct | count | first | last | kurtosis | max | min | skewness | stddev_samp | stddev_pop | sum | sumDistinct | var_samp | var_pop ）。

  指定要应用的聚合函数。

  可能的聚合函数包括：avg countDinstinct、count、first、last、kurtosis、max、min、skewness、stddev_samp、stddev_pop、sum、

GlueSchema 结构

在无法确定架构时，指定一个用户定义的架构 AWS Glue。

字段

- `Columns` – GlueStudioSchemaColumn 对象的数组。

  指定构成 AWS Glue 架构的列定义。

GlueStudioSchemaColumn 结构

在 AWS Glue 架构定义中指定单个列。

字段

- `Name` – 必填：UTF-8 字符串，长度不超过 1024 个字节，与 Single-line string pattern 匹配。

  AWS Glue Studio 架构中该列的名称。
- `Type` – UTF-8 字符串，不超过 131072 个字节，与 Single-line string pattern 匹配。

  AWS Glue Studio 架构中此列的配置单元类型。

GlueStudioColumn 结构

在 AWS Glue Studio 中指定单列。

字段

- `Key` – 必填： UTF-8 字符串，与 Custom string pattern #41 匹配。

  AWS Glue Studio 中专栏的关键。
- `FullPath` – 必填：UTF-8 字符串数组。

  AWS Glue Studio 中该专栏的完整网址。
- `Type` – 必填： UTF-8 字符串（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") 。

AWS Glue Studio 中专栏的类型。

- Children – 结构数组。

  AWS Glue Studio 中父列的子项。

DynamicTransform 结构

指定执行动态转换所需的一组参数。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定动态转换的名称。

- TransformName – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定动态变换在 AWS Glue Studio 可视化编辑器中显示的名称。

- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  指定所需的动态转换输入。

- Parameters – TransformConfigParameter 对象的数组。

  指定动态转换的参数。

- FunctionName – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定动态转换的函数名称。

- `Path` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定动态转换源和配置文件的路径。
- `Version` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  此字段未使用，将在未来版本中弃用。
- `OutputSchemas` – GlueSchema 对象的数组。

  指定动态转换的数据架构。

TransformConfigParameter 结构

指定动态转换的配置文件的参数。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定动态转换的配置文件的参数名称。
- `Type` – 必填：UTF-8 字符串（有效值：`str="STR"` | `int="INT"` | `float="FLOAT"` | `complex="COMPLEX"` | `bool="BOOL"` | `list="LIST"` | `null="NULL"`）。

  指定动态转换的配置文件中的参数类型。
- `ValidationRule` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定动态转换的配置文件的验证规则。
- `ValidationMessage` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定动态转换的配置文件的验证消息。
- `Value` – UTF-8 字符串数组。

  指定动态转换的配置文件的参数值。
- `ListType` – UTF-8 字符串（有效值：`str="STR"` | `int="INT"` | `float="FLOAT"` | `complex="COMPLEX"` | `bool="BOOL"` | `list="LIST"` | `null="NULL"`）。

  指定动态转换的配置文件的参数列表类型。
- `IsOptional` – 布尔值。

  指定该参数在动态转换的配置文件中是否可选。

EvaluateDataQuality 结构

指定您的数据质量评估标准。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据质量评估的名称。
- Inputs – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  您的数据质量评估的输入。
- Ruleset – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 65536 个字节，与 Custom string pattern #38 匹配。

  数据质量评估的规则集。
- Output – UTF-8 字符串（有效值：PrimaryInput | EvaluationResults）。

  您的数据质量评估的输出。
- PublishingOptions – 一个 DQ ResultsPublishingOptions 对象。

  用于配置结果发布方式的选项。
- StopJobOnFailureOptions – 一个 DQ StopJobOnFailureOptions 对象。

  用于配置在数据质量评估失败时如何停止作业的选项。

DQ 结构 ResultsPublishingOptions

用于配置数据质量评估结果发布方式的选项。

字段

- EvaluationContext – UTF-8 字符串，与 Custom string pattern #39 匹配。

  评估的背景。
- ResultsS3Prefix – UTF-8 字符串，与 Custom string pattern #40 匹配。

  Amazon S3 前缀位于结果之前。
- CloudWatchMetricsEnabled – 布尔值。

为您的数据质量结果启用指标。

- `ResultsPublishingEnabled` – 布尔值。

  为您的数据质量结果启用发布。

DQ 结构 StopJobOnFailureOptions

用于配置在数据质量评估失败时如何停止作业的选项。

字段

- `StopJobOnFailureTiming` – UTF-8 字符串（有效值：`Immediate` | `AfterDataLoad`）。

  如果您的数据质量评估失败，何时停止作业。选项有 “即时” 或 AfterDataLoad。

EvaluateDataQualityMultiFrame 结构

指定您的数据质量评估标准。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  数据质量评估的名称。

- `Inputs` – 必填：UTF-8 字符串数组，至少 1 个字符串。

  您的数据质量评估的输入。此列表中的第一个输入是主数据来源。

- `AdditionalDataSources` – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #43 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  除主数据来源之外的所有数据来源的别名。

- `Ruleset` – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 65536 个字节，与 Custom string pattern #38 匹配。

  数据质量评估的规则集。

- `PublishingOptions` – 一个 DQ ResultsPublishingOptions 对象。

用于配置结果发布方式的选项。

- `AdditionalOptions` – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串（有效值：`performanceTuning.caching="CacheOption" | observations.scope="ObservationsOption"`）。

  每个值是一个 UTF-8 字符串。

  用于配置转换运行时行为的选项。

- `StopJobOnFailureOptions` – 一个 DQ StopJobOnFailureOptions 对象。

  用于配置在数据质量评估失败时如何停止作业的选项。

脚本结构

在 AWS Glue 作业中使用 AWS Glue DataBrew 配方的 AWS Glue Studio 节点。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  AWS Glue 工作室节点的名称。

- `Inputs` – 必填：UTF-8 字符串数组，不少于 1 个或不超过 1 个字符串。

  作为脚本节点的输入的节点，由 id 标识。

- `RecipeReference` – 必填：一个 RecipeReference 对象。

  对节点使用的 DataBrew 配方的引用。

RecipeReference 结构

对 AWS Glue DataBrew 食谱的引用。

字段

- `RecipeArn` – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  食谱的 ARN。 DataBrew

- `RecipeVersion` – 必填： UTF-8 字符串，长度不少于 1 个字节，不超过 16 个字节。

DataBrew 食谱 RecipeVersion 中的那个。

SnowflakeNodeData 结构

指定 Studio 中雪花节点的 AWS Glue 配置。

字段

- `SourceType` – UTF-8 字符串，与 Custom string pattern #39 匹配。

  指定检索数据的指定方式。有效值："table"、 "query"。
- `Connection` – 一个 选项 对象。

  指定与 Snowflake 端点 AWS Glue 的数据目录连接。
- `Schema` – UTF-8 字符串。

  为您的节点指定要使用的 Snowflake 数据库架构。
- `Table` – UTF-8 字符串。

  为您的节点指定要使用的 Snowflake 表。
- `Database` – UTF-8 字符串。

  为您的节点指定要使用的 Snowflake 数据库。
- `TempDir` – UTF-8 字符串，与 Custom string pattern #40 匹配。

  当前未使用。
- `IamRole` – 一个 选项 对象。

  当前未使用。
- `AdditionalOptions` – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  每个值都是一个 UTF-8 字符串，与 Custom string pattern #40 匹配。

  指定传递给 Snowflake 连接器的其他选项。如果在此节点中以其他地方指定了选项，则优先使用该选项。
- `SampleQuery` – UTF-8 字符串。

用于检索 query 源类型数据的 SQL 字符串。

- PreAction – UTF-8 字符串。

  在 Snowflake 连接器执行其标准操作之前运行的 SQL 字符串。

- PostAction – UTF-8 字符串。

  在 Snowflake 连接器执行其标准操作之后运行的 SQL 字符串。

- Action – UTF-8 字符串。

  指定在写入包含先前存在数据的表时要执行的操作。有效值：append、merge、truncate、drop。

- Upsert – 布尔值。

  在“操作”为 append 时使用。指定行已存在时的解析行为。如果为 true，则先前存在的行将被更新。如果为 false，则将插入这些行。

- MergeAction – UTF-8 字符串，与 Custom string pattern #39 匹配。

  指定合并操作。有效值：simple、custom。如果是简单，则合并行为由 MergeWhenMatched 和 MergeWhenNotMatched 定义。如果是自定义，则 MergeClause 由定义。

- MergeWhenMatched – UTF-8 字符串，与 Custom string pattern #39 匹配。

  指定在合并时如何解析与先前存在的数据相匹配的记录。有效值： update、delete。

- MergeWhenNotMatched – UTF-8 字符串，与 Custom string pattern #39 匹配。

  指定在合并时如何处理与先前存在的数据不匹配的记录。有效值：insert、none。

- MergeClause – UTF-8 字符串。

  指定自定义合并行为的 SQL 语句。

- StagingTable – UTF-8 字符串。

  执行 merge 或更新插入 append 操作时使用的暂存表的名称。数据被写入此表，然后通过生成的后期操作将其移动到 table。

- SelectedColumns – 选项 对象的数组。

  指定在检测到合并和更新插入的匹配项时用于标识记录的组合列。带有 value、label 和 description 键的结构列表。每个结构都描述了一列。

- AutoPushdown – 布尔值。

指定是否启用自动查询下推。如果启用了下推，那么当在 Spark 上运行查询时，如果可以将部分查询“下推”到 Snowflake 服务器，则会将其下推。这提高了某些查询的性能。

- `TableSchema` – 选项 对象的数组。

  手动定义节点的目标架构。带有 `value`、`label` 和 `description` 键的结构列表。每个结构都定义了一列。

SnowflakeSource 结构

指定 Snowflake 数据来源。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  Snowflake 数据来源的名称。

- `Data` – 必填：一个 SnowflakeNodeData 对象。

  Snowflake 数据来源的配置。

- `OutputSchemas` – GlueSchema 对象的数组。

  为输出数据指定用户定义的架构。

SnowflakeTarget 结构

指定 Snowflake 目标。

字段

- `Name` – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  Snowflake 目标的名称。

- `Data` – 必填：一个 SnowflakeNodeData 对象。

  指定 Snowflake 目标节点的数据。

- `Inputs` – UTF-8 字符串数组，不少于 1 个字符串，不超过 1 个字符串。

  作为数据目标输入的节点。

ConnectorDataSource 结构

指定使用标准连接选项生成的源。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

  此源节点的名称。
- ConnectionType – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  提供给底层 AWS Glue 库的。connectionType此节点类型支持以下连接类型：
  - opensearch
  - azuresql
  - azurecosmos
  - bigquery
  - saphana
  - teradata
  - vertica
- Data – 必填：键值对的映射数组。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  指定节点连接选项的映射。您可以在 AWS Glue 文档的 “连接参数” 部分中找到相应连接类型的标准连接选项。
- OutputSchemas – GlueSchema 对象的数组。

  指定该源的数据 Schema。

ConnectorDataTarget 结构

指定使用标准连接选项生成的目标。

字段

- Name – 必填： UTF-8 字符串，与 Custom string pattern #43 匹配。

此目标节点的名称。

- ConnectionType – 必填： UTF-8 字符串，与 Custom string pattern #40 匹配。

  提供给底层 AWS Glue 库的。connectionType此节点类型支持以下连接类型：

  - opensearch
  - azuresql
  - azurecosmos
  - bigquery
  - saphana
  - teradata
  - vertica
- Data – 必填：键值对的映射数组。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  指定节点连接选项的映射。您可以在 AWS Glue 文档的 “连接参数” 部分中找到相应连接类型的标准连接选项。
- Inputs – UTF-8 字符串数组，不少于 1 个字符串，不超过 1 个字符串。

  作为数据目标输入的节点。

作业 API

任务 API 介绍任务数据类型，并包含用于在 AWS Glue 中处理任务、任务运行和触发器的 API。

主题

- 任务
- 任务运行
- 触发

任务

作业 API 描述了与在中创建、更新、删除或查看作业相关的数据类型和 API AWS Glue。

数据类型

- Job 结构
- ExecutionProperty 结构
- NotificationProperty 结构
- JobCommand 结构
- ConnectionsList 结构
- JobUpdate 结构
- SourceControlDetails 结构

Job 结构

指定作业定义。

字段

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  您分配给该作业定义的名称。
- JobMode – UTF-8 字符串（有效值：SCRIPT="" | VISUAL="" | NOTEBOOK=""）。

  描述任务是如何创建的模式。有效值为：
  - SCRIPT-该作业是使用 AWS Glue Studio 脚本编辑器创建的。
  - VISUAL-该作业是使用 AWS Glue Studio 可视化编辑器创建的。
  - NOTEBOOK – 该任务使用交互式会话笔记本创建。

  当 JobMode 字段缺失或为空时，SCRIPT 将指定为默认值。
- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  作业的描述。
- LogUri – UTF-8 字符串。

  该字段保留，以供将来使用。
- Role – UTF-8 字符串。

与此作业关联的 IAM 角色的名称或 Amazon 资源名称（ARN）。

- `CreatedOn` – 时间戳。

  创建此作业定义的时间和日期。

- `LastModifiedOn` – 时间戳。

  修改此作业定义时的最后一个时间点。

- `ExecutionProperty` – 一个 ExecutionProperty 对象。

  一个 `ExecutionProperty`，指定该作业允许的最大并发运行数。

- `Command` – 一个 JobCommand 对象。

  运行此任务的 `JobCommand`。

- `DefaultArguments` – 键值对的映射数组。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  此作业每个运行的默认参数，指定为名称-值对。

  您可以在此处指定自己的作业执行脚本使用的参数，以及自己消耗的 AWS Glue 参数。

  可能会记录任务参数。不要将明文密钥作为参数传递。如果您打算将机密保留在 Job 中，请从 C AWS Glue onnection AWS Secrets Manager 或其他机密管理机制中检索机密。

  有关如何指定和使用您自己的任务参数的信息，请参阅开发人员指南中的在 Python 中调用 AWS Glue API 主题。

  有关在配置 Spark 作业时可以为该字段提供的参数的信息，请参阅《开发人员指南》中的 Special Parameters Used by AWS Glue 主题。

  有关在配置 Spark 作业时可以为该字段提供的参数的信息，请参阅《开发人员指南》中的 Using job parameters in Ray jobs。

- `NonOverridableArguments` – 键值对的映射数组。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

在作业运行中提供作业参数时，此作业的参数不会被覆盖，而是指定为名称-值对。

- `Connections` – 一个 ConnectionsList 对象。

  用于此作业的连接。

- `MaxRetries` – 数字（整数）。

  JobRun 失败后重试此任务的最大次数。

- `AllocatedCapacity` – 数字（整数）。

  此字段已弃用。请改用 `MaxCapacity`。

  分配给此作业运行 AWS Glue 的数据处理单元 (DPU) 的数量。您可以分配至少 2 个 DPU；默认值为 10。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅 AWS Glue 价格页面。

- `Timeout` - 数字（整数），至少为 1。

  作业超时（以分钟为单位）。这是任务运行在终止并进入 `TIMEOUT` 状态前可以使用资源的最长时间。批处理作业的默认值为 2880 分钟（48 小时）。

  流式传输任务的超时值必须小于 7 天或 10080 分钟。如果将该值留空，则如果您尚未设置维护窗口，则该任务将在 7 天后重新启动。如果您设置了维护时段，则该任务将在 7 天后的维护时段内重新启动。

- `MaxCapacity` – 数字（double）。

  对于 Glue 版本 1.0 或更早版本的作业，使用标准工作器类型，即该作业运行时可以分配 AWS Glue 的数据处理单元 (DPU) 的数量。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅 AWS Glue 价格页面。

  对于 Glue 版本 2.0 或更高版本的作业，则不能指定 `Maximum capacity`。而应指定 `Worker type` 和 `Number of workers`。

  如果使用 `WorkerType` 和 `NumberOfWorkers`，请勿设置 `MaxCapacity`。

  可为 `MaxCapacity` 分配的值取决于您运行的是 Python shell 作业、Apache Spark ETL 作业，还是 Apache Spark 流 ETL 作业：

  - 当您指定 Python shell 作业 (`JobCommand.Name`="pythonshell")，您可以分配 0.0625 或 1 DPU。默认值为 0.0625 DPU。

  - 当您指定 Apache Spark ETL 作业 (`JobCommand.Name="glueetl"`) 或 Apache Spark 流 ETL 作业 (`JobCommand.Name="gluestreaming"`) 时，您可以分配 2 到 100 个 DPU。默认为 10 个 DPU。此任务类型不能具有小数 DPU 分配。

- `WorkerType` – UTF-8 字符串（有效值：`Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""`）。

  任务运行时分配的预定义工作线程的类型。接受 G.1X、G.2X、G.4X、G.8X 或 G.025X for Spark 作业的值。接受 Ray 作业的值 Z.2X。

  - 对于 `G.1X` 工作线程类型，每个工作线程映射到 1 个 DPU（4 个 vCPU，16GB 内存），84GB 磁盘（34GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议将这种工作线程类型用于数据转换、联接和查询等工作负载，以提供一种可扩展且经济实惠的方式来运行大多数作业。
  - 对于 `G.2X` 工作线程类型，每个工作线程映射到 2 个 DPU（8 个 vCPU，32GB 内存），128GB 磁盘（77GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议将这种工作线程类型用于数据转换、联接和查询等工作负载，以提供一种可扩展且经济实惠的方式来运行大多数作业。
  - 对于 `G.4X` 工作线程类型，每个工作线程映射到 4 个 DPU（16 个 vCPU，64GB 内存），256GB 磁盘（235GB 可用空间），并且每个工作线程提供 1 个执行程序。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作人员类型仅适用于以下 AWS 区域的 3.0 或更高 AWS Glue 版本的 Spark ETL 职位：美国东部（俄亥俄州）、美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）、加拿大（中部）、欧洲（法兰克福）、欧洲（爱尔兰）和欧洲（斯德哥尔摩）。
  - 对于 `G.8X` 工作线程类型，每个工作线程映射到 8 个 DPU（32 个 vCPU，128GB 内存），512GB 磁盘（487GB 可用空间），并且每个工作线程提供 1 个执行程序。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作器类型仅适用于 3.0 或更高 AWS Glue 版本的 Spark ETL 作业，其 AWS 区域与该`G.4X`工作人员类型支持的区域相同。
  - 对于 `G.025X` 工作线程类型，每个工作线程映射到 0.25 个 DPU（2 个 vCPU，4GB 内存），84GB 磁盘（34GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议为低容量串流任务使用此 Worker 类型。此工作器类型仅适用于 3.0 AWS Glue 版本的流媒体作业。
  - 对于 `Z.2X` 工作线程类型，每个工作线程映射到 2 个 M-DPU（8 个 vCPU，64GB 内存），128GB 磁盘（120GB 可用空间），基于自动缩放器，最多提供 8 个 Ray 工作线程。

- `NumberOfWorkers` – 数字（整数）。

  任务运行时分配的定义 `workerType` 的工作线程数。

- `SecurityConfiguration` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

该作业将使用的 SecurityConfiguration 结构的名称。

- NotificationProperty – 一个 NotificationProperty 对象。

  指定作业通知的配置属性。

- Running – 布尔值。

  该字段保留，以供将来使用。

- GlueVersion – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Custom string pattern #20 匹配。

  在 Spark 作业中，GlueVersion确定作业中 AWS Glue 可用的 Apache Spark 和 Python 版本。Python 版本指示了 Spark 类型的任务支持的版本。

  Ray 作业应将 GlueVersion 设置为 4.0 或更高。但是，Ray 作业中可用的 Ray、Python 和其他库的版本由 Job 命令的 Runtime 参数决定。

  有关可用 AWS Glue 版本以及相应的 Spark 和 Python 版本的更多信息，请参阅开发者指南中的 Glue 版本。

  在未指定 Glue 版本的情况下创建的任务默认为 Glue 0.9。

- CodeGenConfigurationNodes – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #39 匹配。

  每个值都是一个 CodeGenConfigurationNode 对象。

  定向非循环图表的表示形式，Glue Studio 可视化组件和 Glue Studio 代码生成都是基于该图表的。

- ExecutionClass – UTF-8 字符串，长度不超过 16 个字节（有效值：FLEX="" | STANDARD=""）。

  指示任务是使用标准执行类还是灵活执行类运行的。标准执行类非常适合需要快速任务启动和专用资源的时间敏感型工作负载。

  灵活执行类适用于启动和完成时间可能不同的时间不敏感型作业。

  只有 AWS Glue 版本为 3.0 及以上且命令类型的作业glueetl才允许设置ExecutionClass为FLEX。灵活的执行类可用于 Spark 任务。

- SourceControlDetails – 一个 SourceControlDetails 对象。

作业源代码控制配置的详细信息，允许将作业构件同步到远程存储库或从远程存储库同步。

- `MaintenanceWindow` – UTF-8 字符串，与 Custom string pattern #30 匹配。

  此字段为流式处理作业的维护窗口指定一周中的某一天和一小时。 AWS Glue 定期执行维护活动。在这些维护时段内， AWS Glue 需要重新启动您的直播作业。

  AWS Glue 将在指定维护时段后的 3 小时内重新启动作业。例如，如果您将维护时段设置为 GMT 时间星期一上午 10:00，则您的任务将在 GMT 时间上午 10:00 至下午 1:00 之间重新启动。

- `ProfileName` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与任务关联的 AWS Glue 使用情况配置文件的名称。

ExecutionProperty 结构

作业的执行属性。

字段

- `MaxConcurrentRuns` – 数字（整数）。

  作业允许的最大并发运行数。默认 为 1。达到此阈值时，将返回一个错误。可以指定的最大值由服务限制控制。

NotificationProperty 结构

指定通知的配置属性。

字段

- `NotifyDelayAfter` - 数字（整数），至少为 1。

  在作业运行开始后，发送作业运行延迟通知之前等待的分钟数。

JobCommand 结构

指定任务运行时运行的代码。

字段

- Name – UTF-8 字符串。

  作业命令的名称。对于 Apache Spark ETL 作业，这必须是 glueetl。对于 Python shell 作业，它必须为 pythonshell。对于 Apache Spark 流 ETL 作业，这必须是 gluestreaming。对于 Ray 作业，必须是 glueray。

- ScriptLocation – UTF-8 字符串，长度不超过 400000 个字节。

  指定运行任务的脚本的 Amazon Simple Storage Service（Amazon S3）路径。

- PythonVersion – UTF-8 字符串，与 Custom string pattern #21 匹配。

  运行 Python shell 任务所用的 Python 版本。允许的值为 2 或 3。

- Runtime - UTF-8 字符串，长度不超过 64 个字节，与 Custom string pattern #29 匹配。

  在 Ray 作业中，运行时用于指定环境中可用的 Ray、Python 和其他库的版本。此字段不用于其他作业类型。有关支持的运行时环境值，请参阅《AWS Glue 开发人员指南》中支持的 Ray 运行时环境。

ConnectionsList 结构

指定作业所使用的连接。

字段

- Connections – UTF-8 字符串数组。

  作业所使用的连接的列表。

JobUpdate 结构

指定用于更新现有作业定义的信息。此信息会完全覆盖先前的作业定义。

字段

- JobMode – UTF-8 字符串（有效值：SCRIPT="" | VISUAL="" | NOTEBOOK=""）。

  描述任务是如何创建的模式。有效值为：

  - SCRIPT-该作业是使用 AWS Glue Studio 脚本编辑器创建的。

- VISUAL-该作业是使用 AWS Glue Studio 可视化编辑器创建的。
- NOTEBOOK – 该任务使用交互式会话笔记本创建。

当 JobMode 字段缺失或为空时，SCRIPT 将指定为默认值。

- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  要定义的作业的描述。

- LogUri – UTF-8 字符串。

  该字段保留，以供将来使用。

- Role – UTF-8 字符串。

  与此作业关联的 IAM 角色的名称或 Amazon Resource Name (ARN)（必填）。

- ExecutionProperty – 一个 ExecutionProperty 对象。

  一个 ExecutionProperty，指定该作业允许的最大并发运行数。

- Command – 一个 JobCommand 对象。

  运行此任务的 JobCommand（必填）。

- DefaultArguments – 键值对的映射数组。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  此作业每个运行的默认参数，指定为名称-值对。

  您可以在此处指定自己的作业执行脚本使用的参数，以及自己消耗的 AWS Glue 参数。

  可能会记录任务参数。不要将明文密钥作为参数传递。如果您打算将机密保留在 Job 中，请从 C AWS Glue onnection AWS Secrets Manager 或其他机密管理机制中检索机密。

  有关如何指定和使用您自己的任务参数的信息，请参阅开发人员指南中的在 Python 中调用 AWS Glue API 主题。

  有关在配置 Spark 作业时可以为该字段提供的参数的信息，请参阅《开发人员指南》中的 Special Parameters Used by AWS Glue 主题。

有关在配置 Spark 作业时可以为该字段提供的参数的信息，请参阅《开发人员指南》中的 Using job parameters in Ray jobs。

- NonOverridableArguments – 键值对的映射数组。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  在作业运行中提供作业参数时，此作业的参数不会被覆盖，而是指定为名称-值对。

- Connections – 一个 ConnectionsList 对象。

  用于此作业的连接。

- MaxRetries – 数字（整数）。

  在该作业失败时重试的最大次数。

- AllocatedCapacity – 数字（整数）。

  此字段已弃用。请改用 MaxCapacity。

  要分配给此作业 AWS Glue 的数据处理单元 (DPU) 的数量。您可以分配至少 2 个 DPU；默认值为 10。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅 AWS Glue 价格页面。

- Timeout - 数字（整数），至少为 1。

  作业超时（以分钟为单位）。这是任务运行在终止并进入 TIMEOUT 状态前可以使用资源的最长时间。批处理作业的默认值为 2880 分钟（48 小时）。

  流式传输任务的超时值必须小于 7 天或 10080 分钟。如果将该值留空，则如果您尚未设置维护窗口，则该任务将在 7 天后重新启动。如果您设置了维护时段，则该任务将在 7 天后的维护时段内重新启动。

- MaxCapacity – 数字（double）。

  对于 Glue 版本 1.0 或更早版本的作业，使用标准工作器类型，即该作业运行时可以分配 AWS Glue 的数据处理单元 (DPU) 的数量。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅 AWS Glue 价格页面。

  对于 Glue 版本 2.0+ 作业，不能指定 Maximum capacity。而应指定 Worker type 和 Number of workers。

如果使用 WorkerType 和 NumberOfWorkers，请勿设置 MaxCapacity。

可为 MaxCapacity 分配的值取决于您运行的是 Python shell 作业、Apache Spark ETL 作业，还是 Apache Spark 流 ETL 作业：

- 当您指定 Python shell 作业 (JobCommand.Name="pythonshell")，您可以分配 0.0625 或 1 DPU。默认值为 0.0625 DPU。
- 当您指定 Apache Spark ETL 作业 (JobCommand.Name="glueetl") 或 Apache Spark 流 ETL 作业 (JobCommand.Name="gluestreaming") 时，您可以分配 2 到 100 个 DPU。默认为 10 个 DPU。此任务类型不能具有小数 DPU 分配。

- WorkerType – UTF-8 字符串（有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""）。

  任务运行时分配的预定义工作线程的类型。接受 G.1X、G.2X、G.4X、G.8X 或 G.025X for Spark 作业的值。接受 Ray 作业的值 Z.2X。

  - 对于 G.1X 工作线程类型，每个工作线程映射到 1 个 DPU（4 个 vCPU，16GB 内存），84GB 磁盘（34GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议将这种工作线程类型用于数据转换、联接和查询等工作负载，以提供一种可扩展且经济实惠的方式来运行大多数作业。
  - 对于 G.2X 工作线程类型，每个工作线程映射到 2 个 DPU（8 个 vCPU，32GB 内存），128GB 磁盘（77GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议将这种工作线程类型用于数据转换、联接和查询等工作负载，以提供一种可扩展且经济实惠的方式来运行大多数作业。
  - 对于 G.4X 工作线程类型，每个工作线程映射到 4 个 DPU（16 个 vCPU，64GB 内存），256GB 磁盘（235GB 可用空间），并且每个工作线程提供 1 个执行程序。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作人员类型仅适用于以下 AWS 区域的 3.0 或更高 AWS Glue 版本的 Spark ETL 职位：美国东部（俄亥俄州）、美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）、加拿大（中部）、欧洲（法兰克福）、欧洲（爱尔兰）和欧洲（斯德哥尔摩）。
  - 对于 G.8X 工作线程类型，每个工作线程映射到 8 个 DPU（32 个 vCPU，128GB 内存），512GB 磁盘（487GB 可用空间），并且每个工作线程提供 1 个执行程序。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作器类型仅适用于 3.0 或更高 AWS Glue 版本的 Spark ETL 作业，其 AWS 区域与该G.4X工作人员类型支持的区域相同。
  - 对于 G.025X 工作线程类型，每个工作线程映射到 0.25 个 DPU（2 个 vCPU，4GB 内存），84GB 磁盘（34GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议为低容量串流任务使用此 Worker 类型。此工作器类型仅适用于 3.0 AWS Glue 版本的流媒体作业。

  - 对于 `Z.2X` 工作线程类型，每个工作线程映射到 2 个 M-DPU（8 个 vCPU，64GB 内存），128GB 磁盘（120GB 可用空间），基于自动缩放器，最多提供 8 个 Ray 工作线程。
- `NumberOfWorkers` – 数字（整数）。

  任务运行时分配的定义 `workerType` 的工作线程数。
- `SecurityConfiguration` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  该作业将使用的 `SecurityConfiguration` 结构的名称。
- `NotificationProperty` – 一个 NotificationProperty 对象。

  指定作业通知的配置属性。
- `GlueVersion` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #20 匹配。

  在 Spark 作业中，`GlueVersion`确定作业中 AWS Glue 可用的 Apache Spark 和 Python 版本。Python 版本指示了 Spark 类型的任务支持的版本。

  Ray 作业应将 `GlueVersion` 设置为 `4.0` 或更高。但是，Ray 作业中可用的 Ray、Python 和其他库的版本由 Job 命令的 `Runtime` 参数决定。

  有关可用 AWS Glue 版本以及相应的 Spark 和 Python 版本的更多信息，请参阅开发者指南中的 Glue 版本。

  在未指定 Glue 版本的情况下创建的任务默认为 Glue 0.9。
- `CodeGenConfigurationNodes` – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #39 匹配。

  每个值都是一个 CodeGenConfigurationNode 对象。

  定向非循环图表的表示形式，Glue Studio 可视化组件和 Glue Studio 代码生成都是基于该图表的。
- `ExecutionClass` – UTF-8 字符串，长度不超过 16 个字节（有效值：`FLEX=""` | `STANDARD=""`）。

  指示任务是使用标准执行类还是灵活执行类运行的。标准执行类非常适合需要快速任务启动和专用资源的时间敏感型工作负载。

  灵活执行类适用于启动和完成时间可能不同的时间不敏感型作业。

只有 AWS Glue 版本为 3.0 及以上且命令类型的作业`glueetl`才允许设置`ExecutionClass`为`FLEX`。灵活的执行类可用于 Spark 任务。

- `SourceControlDetails` – 一个 SourceControlDetails 对象。

  作业源代码控制配置的详细信息，允许将作业构件同步到远程存储库或从远程存储库同步。

- `MaintenanceWindow` – UTF-8 字符串，与 Custom string pattern #30 匹配。

  此字段为流式处理作业的维护窗口指定一周中的某一天和一小时。 AWS Glue 定期执行维护活动。在这些维护时段内， AWS Glue 需要重新启动您的直播作业。

  AWS Glue 将在指定维护时段后的 3 小时内重新启动作业。例如，如果您将维护时段设置为 GMT 时间星期一上午 10:00，则您的任务将在 GMT 时间上午 10:00 至下午 1:00 之间重新启动。

- `ProfileName` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与任务关联的 AWS Glue 使用情况配置文件的名称。

SourceControlDetails 结构

作业源代码控制配置的详细信息，允许将作业构件同步到远程存储库或从远程存储库同步。

字段

- `Provider` – UTF-8 字符串。

  远程存储库的提供者。

- `Repository` – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节。

  包含作业构件的远程存储库的名称。

- `Owner` – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节。

  包含作业构件的远程存储库的所有者。

- `Branch` – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节。

  远程存储库中的可选分支。

- `Folder` – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节。

  远程存储库中的可选文件夹。

- `LastCommitId` – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节。

  远程存储库中提交的最后一次提交 ID。
- `LastSyncTimestamp` – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节。

  上次执行作业同步的日期和时间。
- `AuthStrategy` – UTF-8 字符串。

  身份验证的类型，可以是存储在 S AWS ecrets Manager 中的身份验证令牌，也可以是个人访问令牌。
- `AuthToken` – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节。

  授权令牌的值。

操作

- CreateJob 动作（Python：create_job）
- UpdateJob 动作（Python：update_job）
- GetJob 动作（Python：get_job）
- GetJobs 动作（Python：get_jobs）
- DeleteJob 操作（Python：delete_job）
- ListJobs 动作（Python：列出任务）
- BatchGetJobs 动作（Python：batch_get_jobs）

CreateJob 动作（Python：create_job）

创建新的作业定义。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  您分配给该作业定义的名称。它在您的 账户中必须是唯一的。
- `JobMode` – UTF-8 字符串（有效值：`SCRIPT=""` | `VISUAL=""` | `NOTEBOOK=""`）。

  描述任务是如何创建的模式。有效值为：

- SCRIPT-该作业是使用 AWS Glue Studio 脚本编辑器创建的。
- VISUAL-该作业是使用 AWS Glue Studio 可视化编辑器创建的。
- NOTEBOOK – 该任务使用交互式会话笔记本创建。

当 JobMode 字段缺失或为空时，SCRIPT 将指定为默认值。

- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  要定义的作业的描述。
- LogUri – UTF-8 字符串。

  该字段保留，以供将来使用。
- Role – 必填：UTF-8 字符串。

  与此作业关联的 IAM 角色的名称或 Amazon 资源名称（ARN）。
- ExecutionProperty – 一个 ExecutionProperty 对象。

  一个 ExecutionProperty，指定该作业允许的最大并发运行数。
- Command – 必填：一个 JobCommand 对象。

  运行此任务的 JobCommand。
- DefaultArguments – 键值对的映射数组。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  此作业每个运行的默认参数，指定为名称-值对。

  您可以在此处指定自己的作业执行脚本使用的参数，以及自己消耗的 AWS Glue 参数。

  可能会记录任务参数。不要将明文密钥作为参数传递。如果您打算将机密保留在 Job 中，请从 C AWS Glue onnection AWS Secrets Manager 或其他机密管理机制中检索机密。

  有关如何指定和使用您自己的任务参数的信息，请参阅开发人员指南中的在 Python 中调用 AWS Glue API 主题。

  有关在配置 Spark 作业时可以为该字段提供的参数的信息，请参阅《开发人员指南》中的 Special Parameters Used by AWS Glue 主题。

有关在配置 Spark 作业时可以为该字段提供的参数的信息，请参阅《开发人员指南》中的 Using job parameters in Ray jobs。

- NonOverridableArguments – 键值对的映射数组。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  在作业运行中提供作业参数时，此作业的参数不会被覆盖，而是指定为名称-值对。

- Connections – 一个 ConnectionsList 对象。

  用于此作业的连接。

- MaxRetries – 数字（整数）。

  在该作业失败时重试的最大次数。

- AllocatedCapacity – 数字（整数）。

  此参数已被弃用。请改用 MaxCapacity。

  要分配给此 Job AWS Glue 的数据处理单元 (DPU) 的数量。您可以分配至少 2 个 DPU；默认值为 10。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅 AWS Glue 价格页面。

- Timeout - 数字（整数），至少为 1。

  作业超时（以分钟为单位）。这是任务运行在终止并进入 TIMEOUT 状态前可以使用资源的最长时间。批处理作业的默认值为 2880 分钟（48 小时）。

  流式传输任务的超时值必须小于 7 天或 10080 分钟。如果将该值留空，则如果您尚未设置维护窗口，则该任务将在 7 天后重新启动。如果您设置了维护时段，则该任务将在 7 天后的维护时段内重新启动。

- MaxCapacity – 数字（double）。

  对于 Glue 版本 1.0 或更早版本的作业，使用标准工作器类型，即该作业运行时可以分配 AWS Glue 的数据处理单元 (DPU) 的数量。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅 AWS Glue 价格页面。

  对于 Glue 版本 2.0+ 作业，不能指定 Maximum capacity。而应指定 Worker type 和 Number of workers。

如果使用 WorkerType 和 NumberOfWorkers，请勿设置 MaxCapacity。

可为 MaxCapacity 分配的值取决于您运行的是 Python shell 作业、Apache Spark ETL 作业，还是 Apache Spark 流 ETL 作业：

- 当您指定 Python shell 作业 (JobCommand.Name="pythonshell")，您可以分配 0.0625 或 1 DPU。默认值为 0.0625 DPU。
- 当您指定 Apache Spark ETL 作业 (JobCommand.Name="glueetl") 或 Apache Spark 流 ETL 作业 (JobCommand.Name="gluestreaming") 时，您可以分配 2 到 100 个 DPU。默认为 10 个 DPU。此任务类型不能具有小数 DPU 分配。

- SecurityConfiguration – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  该作业将使用的 SecurityConfiguration 结构的名称。

- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  要用于此作业的标签。您可以使用标签来限制对作业的访问。有关中标签的更多信息 AWS Glue，请参阅开发者指南AWS Glue中的AWS 标签。

- NotificationProperty – 一个 NotificationProperty 对象。

  指定作业通知的配置属性。

- GlueVersion – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #20 匹配。

  在 Spark 作业中，GlueVersion确定作业中 AWS Glue 可用的 Apache Spark 和 Python 版本。Python 版本指示了 Spark 类型的任务支持的版本。

  Ray 作业应将 GlueVersion 设置为 4.0 或更高。但是，Ray 作业中可用的 Ray、Python 和其他库的版本由 Job 命令的 Runtime 参数决定。

  有关可用 AWS Glue 版本以及相应的 Spark 和 Python 版本的更多信息，请参阅开发者指南中的 Glue 版本。

  在未指定 Glue 版本的情况下创建的任务默认为 Glue 0.9。

- NumberOfWorkers – 数字（整数）。

任务运行时分配的定义 workerType 的工作线程数。

- WorkerType – UTF-8 字符串（有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""）。

  任务运行时分配的预定义工作线程的类型。接受 G.1X、G.2X、G.4X、G.8X 或 G.025X for Spark 作业的值。接受 Ray 作业的值 Z.2X。

  - 对于 G.1X 工作线程类型，每个工作线程映射到 1 个 DPU（4 个 vCPU，16GB 内存），84GB 磁盘（34GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议将这种工作线程类型用于数据转换、联接和查询等工作负载，以提供一种可扩展且经济实惠的方式来运行大多数作业。
  - 对于 G.2X 工作线程类型，每个工作线程映射到 2 个 DPU（8 个 vCPU，32GB 内存），128GB 磁盘（77GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议将这种工作线程类型用于数据转换、联接和查询等工作负载，以提供一种可扩展且经济实惠的方式来运行大多数作业。
  - 对于 G.4X 工作线程类型，每个工作线程映射到 4 个 DPU（16 个 vCPU，64GB 内存），256GB 磁盘（235GB 可用空间），并且每个工作线程提供 1 个执行程序。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作人员类型仅适用于以下 AWS 区域的 3.0 或更高 AWS Glue 版本的 Spark ETL 职位：美国东部（俄亥俄州）、美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）、加拿大（中部）、欧洲（法兰克福）、欧洲（爱尔兰）和欧洲（斯德哥尔摩）。
  - 对于 G.8X 工作线程类型，每个工作线程映射到 8 个 DPU（32 个 vCPU，128GB 内存），512GB 磁盘（487GB 可用空间），并且每个工作线程提供 1 个执行程序。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作器类型仅适用于 3.0 或更高 AWS Glue 版本的 Spark ETL 作业，其 AWS 区域与该G.4X工作人员类型支持的区域相同。
  - 对于 G.025X 工作线程类型，每个工作线程映射到 0.25 个 DPU（2 个 vCPU，4GB 内存），84GB 磁盘（34GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议为低容量串流任务使用此 Worker 类型。此工作器类型仅适用于 3.0 AWS Glue 版本的流媒体作业。
  - 对于 Z.2X 工作线程类型，每个工作线程映射到 2 个 M-DPU（8 个 vCPU，64GB 内存），128GB 磁盘（120GB 可用空间），基于自动缩放器，最多提供 8 个 Ray 工作线程。
- CodeGenConfigurationNodes – 键值对的映射数组。

  每个键都是一个 UTF-8 字符串，与 Custom string pattern #39 匹配。

  每个值都是一个 CodeGenConfigurationNode 对象。

  定向非循环图表的表示形式，Glue Studio 可视化组件和 Glue Studio 代码生成都是基于该图表的。

- `ExecutionClass` – UTF-8 字符串，长度不超过 16 个字节（有效值：`FLEX=""` | `STANDARD=""`）。

  指示任务是使用标准执行类还是灵活执行类运行的。标准执行类非常适合需要快速任务启动和专用资源的时间敏感型工作负载。

  灵活执行类适用于启动和完成时间可能不同的时间不敏感型作业。

  只有 AWS Glue 版本为 3.0 及以上且命令类型的作业`glueetl`才允许设置`ExecutionClass`为`FLEX`。灵活的执行类可用于 Spark 任务。
- `SourceControlDetails` – 一个 SourceControlDetails 对象。

  作业源代码控制配置的详细信息，允许将作业构件同步到远程存储库或从远程存储库同步。
- `MaintenanceWindow` – UTF-8 字符串，与 Custom string pattern #30 匹配。

  此字段为流式处理作业的维护窗口指定一周中的某一天和一小时。 AWS Glue 定期执行维护活动。在这些维护时段内， AWS Glue 需要重新启动您的直播作业。

  AWS Glue 将在指定维护时段后的 3 小时内重新启动作业。例如，如果您将维护时段设置为 GMT 时间星期一上午 10:00，则您的任务将在 GMT 时间上午 10:00 至下午 1:00 之间重新启动。
- `ProfileName` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与任务关联的 AWS Glue 使用情况配置文件的名称。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  为此作业定义提供的唯一名称。

错误

- `InvalidInputException`
- `IdempotentParameterMismatchException`
- `AlreadyExistsException`
- `InternalServiceException`

- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentModificationException

UpdateJob 动作（Python：update_job）

更新现有的作业定义。此信息会完全覆盖先前的作业定义。

请求

- JobName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要更新的作业定义的名称。
- JobUpdate – 必填：一个 JobUpdate 对象。

  指定用于更新作业定义的值。未指定的配置已移除或重置为默认值。
- ProfileName – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与任务关联的 AWS Glue 使用情况配置文件的名称。

响应

- JobName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  返回更新的作业定义的名称。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException

GetJob 动作（Python：get_job）

检索现有的作业定义。

请求

- JobName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索的作业定义的名称。

响应

- Job – 一个 作业 对象。

  请求的作业定义。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetJobs 动作（Python：get_jobs）

检索所有当前作业定义。

请求

- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用)。
- MaxResults – 数字（整数），不小于 1 或大于 1000。

  响应的最大大小。

响应

- Jobs – 作业 对象的数组。

  作业定义的列表。
- NextToken – UTF-8 字符串。

  延续令牌 (如果尚未返回所有作业定义)。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

DeleteJob 操作（Python：delete_job）

删除指定的作业定义。如果找不到该作业定义，则不会引发异常。

请求

- JobName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的作业定义的名称。

响应

- JobName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  已删除的作业定义的名称。

错误

- InvalidInputException
- InternalServiceException

- OperationTimeoutException

ListJobs 动作（Python：列出任务）

检索此 AWS 账户中所有任务资源的名称，或者检索带有指定标签的资源的名称。此操作可让您查看您账户中可用的资源及其名称。

此操作采用可选的 Tags 字段，您可以将其用作响应的筛选器，以便将标记的资源作为一个组进行检索。如果您选择使用标签筛选，则仅检索带标签的资源。

请求

- NextToken – UTF-8 字符串。

  延续令牌 (如果这是延续请求)。
- MaxResults – 数字（整数），不小于 1 或大于 1000。

  要返回的列表的最大大小。
- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  指定仅返回这些已标记的资源。

响应

- JobNames – UTF-8 字符串数组。

  账户中所有作业的名称或带指定标签的作业。
- NextToken – UTF-8 字符串。

  延续令牌 (如果返回的列表不包含上一个可用的指标)。

错误

- InvalidInputException
- EntityNotFoundException

- InternalServiceException
- OperationTimeoutException

BatchGetJobs 动作（Python：batch_get_jobs）

返回给定作业名称列表的资源元数据的列表。调用 ListJobs 操作后，您可以调用此操作来访问您有权访问的数据。此操作支持所有 IAM 权限，包括使用标签的权限条件。

请求

- JobNames – 必填：UTF-8 字符串数组。

  作业名称列表，这些名称可能是通过 ListJobs 操作返回的名称。

响应

- Jobs – 作业 对象的数组。

  作业定义的列表。
- JobsNotFound – UTF-8 字符串数组。

  未找到作业名称的列表。

错误

- InternalServiceException
- OperationTimeoutException
- InvalidInputException

任务运行

Jobs Runs API 描述了中与启动、停止或查看作业运行以及重置作业书签相关的数据类型和 API。AWS Glue您可以在 90 天内访问工作流和任务运行的任务运行历史记录。

数据类型

- JobRun 结构
- Predecessor 结构

- JobBookmarkEntry 结构
- BatchStopJobRunSuccessfulSubmission 结构
- BatchStopJobRunError 结构
- NotificationProperty 结构

JobRun 结构

包含有关任务运行的信息。

字段

- Id – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  此任务运行的 ID。
- Attempt – 数字（整数）。

  尝试运行此任务的次数。
- PreviousRunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  此任务以前运行时的 ID。例如，StartJobRun 操作中指定的 JobRunId。
- TriggerName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  启动此任务运行的触发器的名称。
- JobName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  此运行中使用的任务定义的名称。
- JobMode – UTF-8 字符串（有效值：SCRIPT="" | VISUAL="" | NOTEBOOK=""）。

  描述任务是如何创建的模式。有效值为：
  - SCRIPT-该作业是使用 AWS Glue Studio 脚本编辑器创建的。
  - VISUAL-该作业是使用 AWS Glue Studio 可视化编辑器创建的。
  - NOTEBOOK – 该任务使用交互式会话笔记本创建。

  当 JobMode 字段缺失或为空时，SCRIPT 将指定为默认值。

- `StartedOn` – 时间戳。

  此任务运行的启动日期和时间。

- `LastModifiedOn` – 时间戳。

  此作业运行的上次修改时间。

- `CompletedOn` – 时间戳。

  此任务运行的完成日期和时间。

- `JobRunState` - UTF-8 字符串（有效值：`STARTING` | `RUNNING` | `STOPPING` | `STOPPED` | `SUCCEEDED` | `FAILED` | `TIMEOUT` | `ERROR` | `WAITING` | `EXPIRED`）。

  任务运行的当前状态。有关异常终止的任务的状态的更多信息，请参阅 AWS Glue 任务运行状态。

- `Arguments` – 键值对的映射数组。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  与此运行关联的任务参数。对于此任务运行，它们会替换任务定义本身中的默认参数集。

  您可以在此处指定自己的作业执行脚本使用的参数，以及自己消耗的 AWS Glue 参数。

  可能会记录任务参数。不要将明文密钥作为参数传递。如果您打算将机密保留在 Job 中，请从 C AWS Glue onnection AWS Secrets Manager 或其他机密管理机制中检索机密。

  有关如何指定和使用您自己的任务参数的信息，请参阅开发人员指南中的在 Python 中调用 AWS Glue API 主题。

  有关在配置 Spark 作业时可以为该字段提供的参数的信息，请参阅《开发人员指南》中的 Special Parameters Used by AWS Glue 主题。

  有关在配置 Spark 作业时可以为该字段提供的参数的信息，请参阅《开发人员指南》中的 Using job parameters in Ray jobs。

- `ErrorMessage` – UTF-8 字符串。

  与此任务运行关联的错误消息。

- `PredecessorRuns` – Predecessor 对象的数组。

  此任务运行的前身列表。

- AllocatedCapacity – 数字（整数）。

  此字段已弃用。请改用 MaxCapacity。

  分配给它的 AWS Glue 数据处理单元 (DPU) 的 JobRun数量。可以分配 2 到 100 个 DPU；默认值为 10。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅 AWS Glue 价格页面。

- ExecutionTime – 数字（整数）。

  作业运行使用资源的时间长度 (以秒为单位)。

- Timeout - 数字（整数），至少为 1。

  JobRun 超时（分钟）这是任务运行在终止并进入 TIMEOUT 状态前可以使用资源的最长时间。此值会覆盖父任务中设置的超时值。

  流式传输任务的超时值必须小于 7 天或 10080 分钟。如果将该值留空，则如果您尚未设置维护窗口，则该任务将在 7 天后重新启动。如果您设置了维护时段，则该任务将在 7 天后的维护时段内重新启动。

- MaxCapacity – 数字（double）。

  对于 Glue 版本 1.0 或更早版本的作业，使用标准工作器类型，即该作业运行时可以分配 AWS Glue 的数据处理单元 (DPU) 的数量。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅 AWS Glue 价格页面。

  对于 Glue 版本 2.0+ 作业，不能指定 Maximum capacity。而应指定 Worker type 和 Number of workers。

  如果使用 WorkerType 和 NumberOfWorkers，请勿设置 MaxCapacity。

  可为 MaxCapacity 分配的值取决于您运行的是 Python shell 作业、Apache Spark ETL 作业，还是 Apache Spark 流 ETL 作业：

  - 当您指定 Python shell 作业 (JobCommand.Name="pythonshell")，您可以分配 0.0625 或 1 DPU。默认值为 0.0625 DPU。
  - 当您指定 Apache Spark ETL 作业 (JobCommand.Name="glueetl") 或 Apache Spark 流 ETL 作业 (JobCommand.Name="gluestreaming") 时，您可以分配 2 到 100 个 DPU。默认为 10 个 DPU。此任务类型不能具有小数 DPU 分配。

- WorkerType – UTF-8 字符串（有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""）。

任务运行时分配的预定义工作线程的类型。接受 G.1X、G.2X、G.4X、G.8X 或 G.025X for Spark 作业的值。接受 Ray 作业的值 Z.2X。

- 对于 G.1X 工作线程类型，每个工作线程映射到 1 个 DPU（4 个 vCPU，16GB 内存），84GB 磁盘（34GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议将这种工作线程类型用于数据转换、联接和查询等工作负载，以提供一种可扩展且经济实惠的方式来运行大多数作业。
- 对于 G.2X 工作线程类型，每个工作线程映射到 2 个 DPU（8 个 vCPU，32GB 内存），128GB 磁盘（77GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议将这种工作线程类型用于数据转换、联接和查询等工作负载，以提供一种可扩展且经济实惠的方式来运行大多数作业。
- 对于 G.4X 工作线程类型，每个工作线程映射到 4 个 DPU（16 个 vCPU，64GB 内存），256GB 磁盘（235GB 可用空间），并且每个工作线程提供 1 个执行程序。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作人员类型仅适用于以下 AWS 区域的 3.0 或更高 AWS Glue 版本的 Spark ETL 职位：美国东部（俄亥俄州）、美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）、加拿大（中部）、欧洲（法兰克福）、欧洲（爱尔兰）和欧洲（斯德哥尔摩）。
- 对于 G.8X 工作线程类型，每个工作线程映射到 8 个 DPU（32 个 vCPU，128GB 内存），512GB 磁盘（487GB 可用空间），并且每个工作线程提供 1 个执行程序。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作器类型仅适用于 3.0 或更高 AWS Glue 版本的 Spark ETL 作业，其 AWS 区域与该G.4X工作人员类型支持的区域相同。
- 对于 G.025X 工作线程类型，每个工作线程映射到 0.25 个 DPU（2 个 vCPU，4GB 内存），84GB 磁盘（34GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议为低容量串流任务使用此 Worker 类型。此工作器类型仅适用于 3.0 AWS Glue 版本的流媒体作业。
- 对于 Z.2X 工作线程类型，每个工作线程映射到 2 个 M-DPU（8 个 vCPU，64GB 内存），128GB 磁盘（120GB 可用空间），基于自动缩放器，最多提供 8 个 Ray 工作线程。

- NumberOfWorkers – 数字（整数）。

  任务运行时分配的定义 workerType 的工作线程数。

- SecurityConfiguration – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  运行该任务将使用的 SecurityConfiguration 结构的名称。

- LogGroupName – UTF-8 字符串。

  用于安全日志的日志组的名称，可 CloudWatch 使用 AWS KMS在 Amazon 中进行服务器端加密。此名称可以是 /aws-glue/jobs/，在这种情况下，默认加密为 NONE。如果您添加角色

名称和 SecurityConfiguration 名称（换句话说，/aws-glue/jobs-yourRoleName-yourSecurityConfigurationName/），则该安全配置将用于加密该日志组。

- NotificationProperty – 一个 NotificationProperty 对象。

  指定任务运行通知的配置属性。

- GlueVersion – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #20 匹配。

  在 Spark 作业中，GlueVersion确定作业中 AWS Glue 可用的 Apache Spark 和 Python 版本。Python 版本指示了 Spark 类型的任务支持的版本。

  Ray 作业应将 GlueVersion 设置为 4.0 或更高。但是，Ray 作业中可用的 Ray、Python 和其他库的版本由 Job 命令的 Runtime 参数决定。

  有关可用 AWS Glue 版本以及相应的 Spark 和 Python 版本的更多信息，请参阅开发者指南中的 Glue 版本。

  在未指定 Glue 版本的情况下创建的任务默认为 Glue 0.9。

- DPUSeconds – 数字（double）。

  此字段可针对使用 FLEX 执行类运行的任务设置，也可以在启用自动扩缩功能时设置，等于任务运行生命周期内每个执行器运行的总时间（以秒为单位）乘以 DPU 因子（G.1X 个工作线程时为 1，G.2X 个工作线程时为 2，G.025X 个工作线程时为 0.25）。此值可能不同于弹性伸缩任务中的 executionEngineRuntime * MaxCapacity，因为在给定时间运行的执行程序数量可能少于 MaxCapacity。因此，DPUSeconds 的值可能小于 executionEngineRuntime * MaxCapacity。

- ExecutionClass – UTF-8 字符串，长度不超过 16 个字节（有效值：FLEX="" | STANDARD=""）。

  指示任务是使用标准执行类还是灵活执行类运行的。标准执行类非常适合需要快速任务启动和专用资源的时间敏感型工作负载。

  灵活执行类适用于启动和完成时间可能不同的时间不敏感型作业。

  只有 AWS Glue 版本为 3.0 及以上且命令类型的作业glueetl才允许设置ExecutionClass为FLEX。灵活的执行类可用于 Spark 任务。

- MaintenanceWindow – UTF-8 字符串，与 Custom string pattern #30 匹配。

此字段为流式处理作业的维护窗口指定一周中的某一天和一小时。 AWS Glue 定期执行维护活动。在这些维护时段内， AWS Glue 需要重新启动您的直播作业。

AWS Glue 将在指定维护时段后的 3 小时内重新启动作业。例如，如果您将维护时段设置为 GMT 时间星期一上午 10:00，则您的任务将在 GMT 时间上午 10:00 至下午 1:00 之间重新启动。

- `ProfileName` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与作业运行关联的 AWS Glue 使用情况配置文件的名称。

Predecessor 结构

在触发此任务运行的条件触发器的谓词中使用的任务运行。

字段

- `JobName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  任务运行前身使用的任务定义的名称。
- `RunId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  任务运行前身的任务运行 ID。

JobBookmarkEntry 结构

定义任务可以恢复处理的点。

字段

- `JobName` – UTF-8 字符串。

  相关任务的名称。
- `Version` – 数字（整数）。

  任务的版本。
- `Run` – 数字（整数）。

运行 ID 号。

- Attempt – 数字（整数）。

  尝试 ID 号。

- PreviousRunId – UTF-8 字符串。

  与上次任务运行关联的唯一运行标识符。

- RunId – UTF-8 字符串。

  运行 ID 号。

- JobBookmark – UTF-8 字符串。

  书签本身。

BatchStopJobRunSuccessfulSubmission 结构

记录成功的对停止指定 JobRun 的请求。

字段

- JobName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  已停止的任务运行中使用的任务定义的名称。

- JobRunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  已停止的任务运行的 JobRunId。

BatchStopJobRunError 结构

记录在尝试停止指定的任务运行时出现的错误。

字段

- JobName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  相关任务运行中使用的任务业定义的名称。

- JobRunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  相关任务运行的 JobRunId。
- ErrorDetail – 一个 ErrorDetail 对象。

  指定有关遇到的错误的详细信息。

NotificationProperty 结构

指定通知的配置属性。

字段

- NotifyDelayAfter - 数字（整数），至少为 1。

  在作业运行开始后，发送作业运行延迟通知之前等待的分钟数。

操作

- StartJobRun 操作（Python：start_job_run）
- BatchStopJobRun 操作（Python：batch_stop_job_run）
- GetJobRun 动作（Python：get_job_run）
- GetJobRuns 动作（Python：get_job_runs）
- GetJobBookmark 动作（Python：get_job_bookmark）
- GetJobBookmarks 操作（Python：获取工作书签）
- ResetJobBookmark 动作（Python：重置作业书签）

StartJobRun 操作（Python：start_job_run）

使用任务定义启动任务运行。

请求

- JobName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要使用的任务定义的名称。

- `JobRunId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要重试的以前的 `JobRun` 的 ID。

- `Arguments` – 键值对的映射数组。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  与此运行关联的任务参数。对于此任务运行，它们会替换任务定义本身中的默认参数集。

  您可以在此处指定自己的作业执行脚本使用的参数，以及自己消耗的 AWS Glue 参数。

  可能会记录任务参数。不要将明文密钥作为参数传递。如果您打算将机密保留在 Job 中，请从 C AWS Glue onnection AWS Secrets Manager 或其他机密管理机制中检索机密。

  有关如何指定和使用您自己的任务参数的信息，请参阅开发人员指南中的在 Python 中调用 AWS Glue API 主题。

  有关在配置 Spark 作业时可以为该字段提供的参数的信息，请参阅《开发人员指南》中的 Special Parameters Used by AWS Glue 主题。

  有关在配置 Spark 作业时可以为该字段提供的参数的信息，请参阅《开发人员指南》中的 Using job parameters in Ray jobs。

- `AllocatedCapacity` – 数字（整数）。

  此字段已弃用。请改用 `MaxCapacity`。

  要分配给 AWS Glue 它的数据处理单元 (DPU) 的 JobRun数量。您可以分配至少 2 个 DPU；默认值为 10。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅 AWS Glue 价格页面。

- `Timeout` - 数字（整数），至少为 1。

  `JobRun` 超时（分钟）这是任务运行在终止并进入 `TIMEOUT` 状态前可以使用资源的最长时间。此值会覆盖父任务中设置的超时值。

  流式传输任务的超时值必须小于 7 天或 10080 分钟。如果将该值留空，则如果您尚未设置维护窗口，则该任务将在 7 天后重新启动。如果您设置了维护时段，则该任务将在 7 天后的维护时段内重新启动。

- `MaxCapacity` – 数字（double）。

  对于 Glue 版本 1.0 或更早版本的作业，使用标准工作器类型，即该作业运行时可以分配 AWS Glue 的数据处理单元 (DPU) 的数量。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅 AWS Glue 价格页面。

  对于 Glue 版本 2.0+ 作业，不能指定 `Maximum capacity`。而应指定 `Worker type` 和 `Number of workers`。

  如果使用 `WorkerType` 和 `NumberOfWorkers`，请勿设置 `MaxCapacity`。

  可为 `MaxCapacity` 分配的值取决于您运行的是 Python shell 作业、Apache Spark ETL 作业，还是 Apache Spark 流 ETL 作业：

  - 当您指定 Python shell 作业 (`JobCommand.Name="pythonshell"`)，您可以分配 0.0625 或 1 DPU。默认值为 0.0625 DPU。
  - 当您指定 Apache Spark ETL 作业 (`JobCommand.Name="glueetl"`) 或 Apache Spark 流 ETL 作业 (`JobCommand.Name="gluestreaming"`) 时，您可以分配 2 到 100 个 DPU。默认为 10 个 DPU。此任务类型不能具有小数 DPU 分配。
- `SecurityConfiguration` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  运行该任务将使用的 `SecurityConfiguration` 结构的名称。
- `NotificationProperty` – 一个 NotificationProperty 对象。

  指定任务运行通知的配置属性。
- `WorkerType` – UTF-8 字符串（有效值：`Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""`）。

  任务运行时分配的预定义工作线程的类型。接受 G.1X、G.2X、G.4X、G.8X 或 G.025X for Spark 作业的值。接受 Ray 作业的值 Z.2X。

  - 对于 `G.1X` 工作线程类型，每个工作线程映射到 1 个 DPU（4 个 vCPU，16GB 内存），84GB 磁盘（34GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议将这种工作线程类型用于数据转换、联接和查询等工作负载，以提供一种可扩展且经济实惠的方式来运行大多数作业。
  - 对于 `G.2X` 工作线程类型，每个工作线程映射到 2 个 DPU（8 个 vCPU，32GB 内存），128GB 磁盘（77GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议将这种工作线程类型用于数据转换、联接和查询等工作负载，以提供一种可扩展且经济实惠的方式来运行大多数作业。

- 对于 G.4X 工作线程类型，每个工作线程映射到 4 个 DPU（16 个 vCPU，64GB 内存），256GB 磁盘（235GB 可用空间），并且每个工作线程提供 1 个执行程序。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作人员类型仅适用于以下 AWS 区域的 3.0 或更高 AWS Glue 版本的 Spark ETL 职位：美国东部（俄亥俄州）、美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）、加拿大（中部）、欧洲（法兰克福）、欧洲（爱尔兰）和欧洲（斯德哥尔摩）。
- 对于 G.8X 工作线程类型，每个工作线程映射到 8 个 DPU（32 个 vCPU，128GB 内存），512GB 磁盘（487GB 可用空间），并且每个工作线程提供 1 个执行程序。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作器类型仅适用于 3.0 或更高 AWS Glue 版本的 Spark ETL 作业，其 AWS 区域与该G.4X工作人员类型支持的区域相同。
- 对于 G.025X 工作线程类型，每个工作线程映射到 0.25 个 DPU（2 个 vCPU，4GB 内存），84GB 磁盘（34GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议为低容量串流任务使用此 Worker 类型。此工作器类型仅适用于 3.0 AWS Glue 版本的流媒体作业。
- 对于 Z.2X 工作线程类型，每个工作线程映射到 2 个 M-DPU（8 个 vCPU，64GB 内存），128GB 磁盘（120GB 可用空间），基于自动缩放器，最多提供 8 个 Ray 工作线程。

- NumberOfWorkers – 数字（整数）。

  任务运行时分配的定义 workerType 的工作线程数。

- ExecutionClass – UTF-8 字符串，长度不超过 16 个字节（有效值：FLEX="" | STANDARD=""）。

  指示任务是使用标准执行类还是灵活执行类运行的。标准执行类非常适合需要快速任务启动和专用资源的时间敏感型工作负载。

  灵活执行类适用于启动和完成时间可能不同的时间不敏感型作业。

  只有 AWS Glue 版本为 3.0 及以上且命令类型的作业glueetl才允许设置ExecutionClass为FLEX。灵活的执行类可用于 Spark 任务。

- ProfileName – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与作业运行关联的 AWS Glue 使用情况配置文件的名称。

响应

- JobRunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分配给此任务运行的 ID。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentRunsExceededException

BatchStopJobRun 操作（Python：batch_stop_job_run）

停止指定的任务定义的一个或多次任务运行。

请求

- JobName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要停止其任务运行的任务定义的名称。
- JobRunIds – 必填：UTF-8 字符串数组，不少于 1 个或不超过 25 个字符串。

  应为该任务定义停止的 JobRunIds 列表。

响应

- SuccessfulSubmissions – BatchStopJobRunSuccessfulSubmission 对象的数组。

  已成功提交以 JobRuns 停止的列表。
- Errors – BatchStopJobRunError 对象的数组。

  尝试停止 JobRuns 时遇到的错误列表，包括遇到每个错误的 JobRunId 和有关错误的详细信息。

错误

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetJobRun 动作（Python：get_job_run）

检索给定任务运行的元数据。您可以在 90 天内访问工作流和任务运行的任务运行历史记录。

请求

- JobName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  正在运行的任务定义的名称。
- RunId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  任务运行的 ID。
- PredecessorsIncluded – 布尔值。

  如果应返回运行前身的列表，则为 True。

响应

- JobRun – 一个 JobRun 对象。

  请求的任务运行元数据。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetJobRuns 动作（Python：get_job_runs）

检索给定任务定义的所有运行的元数据。

请求

- JobName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索其所有任务运行的作业定义的名称。
- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用)。
- MaxResults – 数字（整数），不小于 1 或大于 200。

  响应的最大大小。

响应

- JobRuns – JobRun 对象的数组。

  任务运行元数据对象的列表。
- NextToken – UTF-8 字符串。

  延续令牌（如果尚未返回所有请求的任务运行）。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetJobBookmark 动作（Python：get_job_bookmark）

返回有关任务书签条目的信息。

更多有关启用和使用作业书签的信息，请参阅：

- 使用作业书签跟踪已处理的数据
- 使用的 Job 参数 AWS Glue
- 作业结构

请求

- JobName – 必填：UTF-8 字符串。

  相关任务的名称。
- Version – 数字（整数）。

  任务的版本。
- RunId – UTF-8 字符串。

  与此任务运行关联的唯一运行标识符。

响应

- JobBookmarkEntry – 一个 JobBookmarkEntry 对象。

  用于定义任务可以恢复处理的点的结构。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ValidationException

GetJobBookmarks 操作（Python：获取工作书签）

返回有关任务书签条目的信息。该列表按版本号递减的顺序排列。

更多有关启用和使用作业书签的信息，请参阅：

- 使用作业书签跟踪已处理的数据

- 使用的 Job 参数 AWS Glue
- 作业结构

请求

- JobName – 必填：UTF-8 字符串。

  相关任务的名称。
- MaxResults – 数字（整数）。

  响应的最大大小。
- NextToken – 数字（整数）。

  延续标记 (如果这是延续调用)。

响应

- JobBookmarkEntries – JobBookmarkEntry 对象的数组。

  用于定义任务可以恢复处理的点的任务书签条目列表。
- NextToken – 数字（整数）。

  延续令牌，如果返回所有条目，则其值为 1；如果未返回所有请求的任务运行，则其值 > 1。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

ResetJobBookmark 动作（Python：重置作业书签）

重置书签条目。

更多有关启用和使用作业书签的信息，请参阅：

- 使用作业书签跟踪已处理的数据
- 使用的 Job 参数 AWS Glue
- 作业结构

请求

- JobName – 必填：UTF-8 字符串。

  相关任务的名称。
- RunId – UTF-8 字符串。

  与此任务运行关联的唯一运行标识符。

响应

- JobBookmarkEntry – 一个 JobBookmarkEntry 对象。

  重置的书签条目。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

触发

触发器 API 介绍与在 AWS Glue 中创建、更新或删除以及开启和停止任务触发器相关的数据类型和 API。

数据类型

- Trigger 结构
- TriggerUpdate 结构
- Predicate 结构

- Condition 结构
- Action 结构
- EventBatchingCondition 结构

Trigger 结构

有关特定触发器的信息。

字段

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  触发器的名称。
- WorkflowName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  与触发器关联的工作流程的名称。
- Id – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  留待将来使用。
- Type – UTF-8 字符串（有效值：SCHEDULED |CONDITIONAL | ON_DEMAND | EVENT）。

  触发器的类型。
- State – UTF-8 字符串（有效值：CREATING | CREATED | ACTIVATING | ACTIVATED | DEACTIVATING | DEACTIVATED | DELETING | UPDATING）。

  触发器的当前状态。
- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  此触发器的描述。
- Schedule – UTF-8 字符串。

  用于指定计划的 cron 表达式 (请参阅用于作业和爬网程序的基于时间的计划）。例如，要每天 12:15 UTC 运行某些任务，您应该指定：cron(15 12 * * ? *)。
- Actions – 操作 对象的数组。

通过此触发器发起的操作。

- `Predicate` – 一个 谓词 对象。

  此触发器的谓词，用于定义触发器将在何时触发。

- `EventBatchingCondition` – 一个 EventBatchingCondition 对象。

  在 EventBridge 事件触发器触发之前必须满足的批处理条件（接收的事件数量或批处理时间段已过期）。

TriggerUpdate 结构

提供用于更新触发器的信息的结构。此对象通过完全覆盖以前的触发器定义来更新以前的触发器定义。

字段

- `Name` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  留待将来使用。

- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  此触发器的描述。

- `Schedule` – UTF-8 字符串。

  用于指定计划的 `cron` 表达式 (请参阅用于作业和爬网程序的基于时间的计划）。例如，要每天 12:15 UTC 运行某些任务，您应该指定：`cron(15 12 * * ? *)`。

- `Actions` – 操作 对象的数组。

  通过此触发器发起的操作。

- `Predicate` – 一个 谓词 对象。

  此触发器的谓词，用于定义触发器将在何时触发。

- `EventBatchingCondition` – 一个 EventBatchingCondition 对象。

  在 EventBridge 事件触发器触发之前必须满足的批处理条件（接收的事件数量或批处理时间段已过期）。

Predicate 结构

定义触发器的谓词，确定触发器何时触发。

字段

- `Logical` – UTF-8 字符串（有效值：`AND` | `ANY`）。

  如果只列出了一个条件，则为可选字段。如果列出了多个条件，则此字段为必需字段。
- `Conditions` – 状况 对象的数组。

  确定触发器将何时触发的条件列表。

Condition 结构

定义触发器将触发的条件。

字段

- `LogicalOperator` – UTF-8 字符串（有效值：`EQUALS`）。

  一个逻辑运算符。
- `JobName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  此条件应用于其 `JobRuns` 并且此触发器在其上等待的作业的名称。
- `State` - UTF-8 字符串（有效值：`STARTING` | `RUNNING` | `STOPPING` | `STOPPED` | `SUCCEEDED` | `FAILED` | `TIMEOUT` | `ERROR` | `WAITING` | `EXPIRED`）。

  条件状态。目前，触发器可侦听的仅有作业为 `SUCCEEDED`、`STOPPED`、`FAILED` 和 `TIMEOUT`。触发器可侦听的仅有爬网程序状态为 `SUCCEEDED`、`FAILED` 和 `CANCELLED`。
- `CrawlerName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  此条件应用于的爬网程序的名称。
- `CrawlState` – UTF-8 字符串（有效值：`RUNNING` | `CANCELLING` | `CANCELLED` | `SUCCEEDED` | `FAILED` | `ERROR`）。

  此条件应用于的爬网程序的状态。

Action 结构

定义触发器发起的操作。

字段

- JobName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要运行的任务的名称。
- Arguments – 键值对的映射数组。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  此触发器触发时使用的作业参数。对于此任务运行，它们会替换任务定义本身中的默认参数集。

  可以在此处指定您自己的任务执行脚本使用的参数以及 AWS Glue 本身使用的参数。

  有关如何指定和使用您自己的任务参数的信息，请参阅开发人员指南中的在 Python 中调用 AWS Glue API 主题。

  有关 AWS Glue 用于设置任务的键值对的信息，请参阅开发人员指南中的由 AWS Glue 使用的特殊参数主题。
- Timeout - 数字（整数），至少为 1。

  JobRun 超时（分钟）这是任务运行在终止并进入 TIMEOUT 状态前可以使用资源的最长时间。默认值为 2880 分钟（48 小时）。此值会覆盖父任务中设置的超时值。
- SecurityConfiguration – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  该操作将使用的 SecurityConfiguration 结构的名称。
- NotificationProperty – 一个 NotificationProperty 对象。

  指定任务运行通知的配置属性。
- CrawlerName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要用于此操作的爬网程序的名称。

EventBatchingCondition 结构

在 EventBridge 事件触发器触发之前必须满足的批处理条件（接收的事件数量或批处理时间段已过期）。

字段

- `BatchSize` – 必填：数字（整数），不小于 1 或大于 100。

  在 EventBridge 事件触发器触发之前，必须从 Amazon EventBridge 接收的事件数。
- `BatchWindow` – 数字（整数），不小于 1 或大于 900。

  EventBridge 事件触发器触发的时段（以秒为单位）。时段在收到第一个事件时开始计算。

操作

- CreateTrigger 操作（Python：create_trigger）
- StartTrigger 操作（Python：start_trigger）
- GetTrigger 操作（Python：get_trigger）
- GetTriggers 操作（Python：get_triggers）
- UpdateTrigger 操作（Python：update_trigger）
- StopTrigger 操作（Python：stop_trigger）
- DeleteTrigger 操作（Python：delete_trigger）
- ListTriggers 操作（Python：list_triggers）
- BatchGetTriggers 操作（Python：batch_get_triggers）

CreateTrigger 操作（Python：create_trigger）

创建新的触发器。

请求

- `Name` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  触发器的名称。

- WorkflowName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  与触发器关联的工作流程的名称。

- Type – 必填：UTF-8 字符串（有效值：SCHEDULED | CONDITIONAL | ON_DEMAND | EVENT）。

  新触发器的类型。

- Schedule – UTF-8 字符串。

  用于指定计划的 cron 表达式 (请参阅用于作业和爬网程序的基于时间的计划）。例如，要每天 12:15 UTC 运行某些任务，您应该指定：cron(15 12 * * ? *)。

  当触发类型为 SCHEDULED 时，此字段为必需字段。

- Predicate – 一个 谓词 对象。

  指定新的触发器应何时触发的谓词。

  当触发类型为 CONDITIONAL 时，此字段为必需字段。

- Actions – 必填：操作 对象的数组。

  触发器触发时所发起的操作。

- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  新触发器的描述。

- StartOnCreation – 布尔值。

  设置 true 为，可在创建时开启 SCHEDULED 和 CONDITIONAL 触发器。ON_DEMAND 触发器不支持 True。

- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  要用于此触发器的标签。您可以使用标签来限制对触发器的访问。有关 AWS Glue 中的标签的更多信息，请参阅开发人员指南中的 AWS Glue 中的 AWS 标签。

- EventBatchingCondition – 一个 EventBatchingCondition 对象。

在 EventBridge 事件触发器触发之前必须满足的批处理条件（接收的事件数量或批处理时间段已过期）。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  触发器的名称。

错误

- `AlreadyExistsException`
- `EntityNotFoundException`
- `InvalidInputException`
- `IdempotentParameterMismatchException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `ConcurrentModificationException`

StartTrigger 操作（Python：start_trigger）

启动现有触发器。请参阅触发作业了解有关如何启动不同类型的触发器的信息。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要启动的触发器名称。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

已启动的触发器名称。

错误

- InvalidInputException
- InternalServiceException
- EntityNotFoundException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentRunsExceededException

GetTrigger 操作（Python：get_trigger）

检索触发器的定义。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索的触发器名称。

响应

- Trigger – 一个 触发器 对象。

  请求的触发器定义。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetTriggers 操作（Python：get_triggers）

获取与一个作业关联的所有触发器。

请求

- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用)。

- DependentJobName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要为其检索触发器的作业的名称。返回可以启动此作业的触发器，如果没有这样的触发器，则返回所有触发器。

- MaxResults – 数字（整数），不小于 1 或大于 200。

  响应的最大大小。

响应

- Triggers – 触发器 对象的数组。

  指定作业的触发器列表。

- NextToken – UTF-8 字符串。

  延续令牌 (如果尚未返回所有请求的触发器)。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

UpdateTrigger 操作（Python：update_trigger）

更新触发器定义。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要更新的触发器的名称。

- TriggerUpdate – 必填：一个 TriggerUpdate 对象。

  用来更新触发器的新值。

响应

- Trigger – 一个 触发器 对象。

  生成的触发器定义。

错误

- InvalidInputException
- InternalServiceException
- EntityNotFoundException
- OperationTimeoutException
- ConcurrentModificationException

StopTrigger 操作（Python：stop_trigger）

停止指定的触发器。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要停止的触发器名称。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  已停止的触发器名称。

错误

- `InvalidInputException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

DeleteTrigger 操作（Python：delete_trigger）

删除指定的触发器。未找到该触发器不会引发异常。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的触发器的名称。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  已删除的触发器的名称。

错误

- `InvalidInputException`
- `InternalServiceException`

- OperationTimeoutException
- ConcurrentModificationException

ListTriggers 操作（Python：list_triggers）

检索此 AWS 账户中所有触发器资源的名称或带指定标签的资源。此操作可让您查看您账户中可用的资源及其名称。

此操作采用可选的 Tags 字段，您可以将其用作响应的筛选器，以便将标记的资源作为一个组进行检索。如果您选择使用标签筛选，则仅检索带标签的资源。

请求

- NextToken – UTF-8 字符串。

  延续令牌 (如果这是延续请求)。

- DependentJobName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要为其检索触发器的作业的名称。返回可启动此作业的触发器。如果没有这样的触发器，则返回所有触发器。

- MaxResults – 数字（整数），不小于 1 或大于 200。

  要返回的列表的最大大小。

- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  指定仅返回这些已标记的资源。

响应

- TriggerNames – UTF-8 字符串数组。

  账户中所有触发器的名称或带指定标签的触发器。

- NextToken – UTF-8 字符串。

延续令牌 (如果返回的列表不包含上一个可用的指标)。

错误

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchGetTriggers 操作（Python：batch_get_triggers）

返回给定触发器名称列表的资源元数据的列表。调用 ListTriggers 操作后，您可以调用此操作来访问您有权访问的数据。此操作支持所有 IAM 权限，包括使用标签的权限条件。

请求

- TriggerNames – 必填：UTF-8 字符串数组。

  触发器名称列表，这些名称可能是通过 ListTriggers 操作返回的名称。

响应

- Triggers – 触发器 对象的数组。

  触发器定义的列表。
- TriggersNotFound – UTF-8 字符串数组。

  未找到触发器名称的列表。

错误

- InternalServiceException
- OperationTimeoutException
- InvalidInputException

交互式会话 API

交互式会话 API 描述了与使用 AWS Glue 交互式会话来构建和测试用于数据集成的提取、转换和加载 (ETL) 脚本相关的 AWS Glue API。

数据类型

- Session 结构
- SessionCommand 结构
- Statement 结构
- StatementOutput 结构
- StatementOutputData 结构
- ConnectionsList 结构

Session 结构

远程 Spark 运行时环境的运行时段。

字段

- `Id` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  会话 ID。
- `CreatedOn` – 时间戳。

  创建会话的日期和时间。
- `Status` – UTF-8 字符串（有效值：`PROVISIONING` | `READY` | `FAILED` | `TIMEOUT` | `STOPPING` | `STOPPED`）。

  会话状态。
- `ErrorMessage` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  会话期间显示的错误消息。
- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

会话描述。

- `Role` – UTF-8 字符串，不少于 20 个字节，不超过 2048 个字节，与 Custom string pattern #26 匹配。

  与此会话关联的 IAM 角色的名称或 Amazon Resource Name (ARN)。

- `Command` – 一个 SessionCommand 对象。

  命令对象。请参阅。 SessionCommand

- `DefaultArguments` – 键值对的映射数组，不超过 75 对。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  每个值都是一个 UTF-8 字符串，长度不超过 4096 个字节，与 URI address multi-line string pattern 匹配。

  键值对的映射数组。最多 75 对。

- `Connections` – 一个 ConnectionsList 对象。

  用于会话的连接数。

- `Progress` – 数字（double）。

  会话的代码执行进度。

- `MaxCapacity` – 数字（double）。

  作业运行时可以分配 AWS Glue 的数据处理单元 (DPU) 的数量。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。

- `SecurityConfiguration` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  要用于会话的 SecurityConfiguration 结构的名称。

- `GlueVersion` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Custom string pattern #20 匹配。

  该 AWS Glue 版本决定了 AWS Glue 支持的 Apache Spark 和 Python 版本。 GlueVersion 必须大于 2.0。

- `DataAccessId` – UTF-8 字符串，长度不少于 1 个字节，不超过 36 个字节。

会话的数据访问 ID。

- PartitionId – UTF-8 字符串，长度不少于 1 个字节，不超过 36 个字节。

  会话的分区 ID。

- NumberOfWorkers – 数字（整数）。

  用于会话的已定义 WorkerType 的工件数量。

- WorkerType – UTF-8 字符串（有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""）。

  会话运行时分配的预定义工作线程的类型。接受 G.1X、G.2X、G.4X、G.8X for Spark 会话的值。接受 Z.2X for Ray 的值。

- CompletedOn – 时间戳。

  此会话的完成日期和时间。

- ExecutionTime – 数字（double）。

  会话运行的总时间。

- DPUSeconds – 数字（double）。

  会话消耗的 DPU（公式： ExecutionTime * MaxCapacity）。

- IdleTimeout – 数字（整数）。

  会话超时前的空闲分钟数。

- ProfileName – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与会话关联的 AWS Glue 使用情况配置文件的名称。

SessionCommand 结构

运行任务的 SessionCommand。

字段

- Name – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

指定名称 SessionCommand。可以是“gluetel”或“gluestreaming”。

- PythonVersion – UTF-8 字符串，与 Custom string pattern #21 匹配。

  指定 Python 版本。Python 版本指示了 Spark 类型的任务支持的版本。

Statement 结构

在会话中实现特定操作的语句或请求。

字段

- Id – 数字（整数）。

  语句的 ID。
- Code – UTF-8 字符串。

  语句的执行代码。
- State – UTF-8 字符串（有效值：WAITING | RUNNING | AVAILABLE | CANCELLING | CANCELLED | ERROR）。

  执行请求时的状态。
- Output – 一个 StatementOutput 对象。

  JSON 中的输出。
- Progress – 数字（double）。

  代码执行进度。
- StartedOn – 数字（长型）。

  任务定义开始的 Unix 时间和日期。
- CompletedOn – 数字（长型）。

  任务定义完成的 Unix 时间和日期。

StatementOutput 结构

JSON 格式的代码执行输出。

字段

- `Data` – 一个 StatementOutputData 对象。

  代码执行输出。

- `ExecutionCount` – 数字（整数）。

  输出的执行计数。

- `Status` – UTF-8 字符串（有效值：`WAITING` | `RUNNING` | `AVAILABLE` | `CANCELLING` | `CANCELLED` | `ERROR`）。

  代码执行输出的状态。

- `ErrorName` – UTF-8 字符串。

  输出中的错误名称。

- `ErrorValue` – UTF-8 字符串。

  输出的错误值。

- `Traceback` – UTF-8 字符串数组。

  输出的回溯。

StatementOutputData 结构

JSON 格式的代码执行输出。

字段

- `TextPlain` – UTF-8 字符串。

  文本格式的代码执行输出。

ConnectionsList 结构

指定作业所使用的连接。

字段

- `Connections` – UTF-8 字符串数组。

作业所使用的连接的列表。

操作

- CreateSession 操作（Python：create_session）
- StopSession 动作（Python：停止会话）
- DeleteSession 操作（Python：删除会话）
- GetSession 动作（Python：get_session）
- ListSessions 动作（Python：列表会话）
- RunStatement 动作（Python：run_statement）
- CancelStatement 操作（Python：取消语句）
- GetStatement 动作（Python：get_statement）
- ListStatements 动作（Python：列表语句）

CreateSession 操作（Python：create_session）

创建新的会话。

请求

请求创建新的会话。

- `Id` – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  会话请求 ID。
- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  会话描述。
- `Role` – 必需：UTF-8 字符串，长度不少于 20 个字节，不超过 2048 个字节，与 Custom string pattern #26 匹配。

  IAM 角色 ARN
- `Command` – 必填：一个 SessionCommand 对象。

运行任务的 SessionCommand。

- Timeout - 数字（整数），至少为 1。

  会话超时前的分钟数。Spark ETL 作业的默认值为 48 小时（2880 分钟），是该作业类型的最大会话存在期。有关其他作业类型，请查阅文档。

- IdleTimeout - 数字（整数），至少为 1。

  会话超时前的空闲分钟数。Spark ETL 作业的默认值为“超时”。有关其他作业类型，请查阅文档。

- DefaultArguments – 键值对的映射数组，不超过 75 对。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  每个值都是一个 UTF-8 字符串，长度不超过 4096 个字节，与 URI address multi-line string pattern 匹配。

  键值对的映射数组。最多 75 对。

- Connections – 一个 ConnectionsList 对象。

  用于会话的连接数。

- MaxCapacity – 数字（double）。

  作业运行时可以分配 AWS Glue 的数据处理单元 (DPU) 的数量。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。

- NumberOfWorkers – 数字（整数）。

  用于会话的已定义 WorkerType 的工件数量。

- WorkerType – UTF-8 字符串（有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""）。

  任务运行时分配的预定义工作线程的类型。接受 G.1X、G.2X、G.4X 或 G.8X for Spark 作业的值。接受 Ray 笔记本的值 Z.2X。

  - 对于 G.1X 工作线程类型，每个工作线程映射到 1 个 DPU（4 个 vCPU，16GB 内存），84GB 磁盘（34GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议将这种工作线程类型用于数据转换、联接和查询等工作负载，以提供一种可扩展且经济实惠的方式来运行大多数作业。

- 对于 G.2X 工作线程类型，每个工作线程映射到 2 个 DPU（8 个 vCPU，32GB 内存），128GB 磁盘（77GB 可用空间），并且每个工作线程提供 1 个执行程序。我们建议将这种工作线程类型用于数据转换、联接和查询等工作负载，以提供一种可扩展且经济实惠的方式来运行大多数作业。
- 对于 G.4X 工作线程类型，每个工作线程映射到 4 个 DPU（16 个 vCPU，64GB 内存），256GB 磁盘（235GB 可用空间），并且每个工作线程提供 1 个执行程序。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作人员类型仅适用于以下 AWS 区域的 3.0 或更高 AWS Glue 版本的 Spark ETL 职位：美国东部（俄亥俄州）、美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）、加拿大（中部）、欧洲（法兰克福）、欧洲（爱尔兰）和欧洲（斯德哥尔摩）。
- 对于 G.8X 工作线程类型，每个工作线程映射到 8 个 DPU（32 个 vCPU，128GB 内存），512GB 磁盘（487GB 可用空间），并且每个工作线程提供 1 个执行程序。对于工作负载包含要求最高的转换、聚合、联接和查询的作业，我们建议使用这种工作线程类型。此工作器类型仅适用于 3.0 或更高 AWS Glue 版本的 Spark ETL 作业，其 AWS 区域与该G.4X工作人员类型支持的区域相同。
- 对于 Z.2X 工作线程类型，每个工作线程映射到 2 个 M-DPU（8 个 vCPU，64 GB 内存），128GB 磁盘（120GB 可用空间），基于自动缩放器，最多提供 8 个 Ray 工作线程。

- SecurityConfiguration – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  要用于会话的 SecurityConfiguration 结构的名称

- GlueVersion – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Custom string pattern #20 匹配。

  该 AWS Glue 版本决定了 AWS Glue 支持的 Apache Spark 和 Python 版本。 GlueVersion 必须大于 2.0。

- DataAccessId – UTF-8 字符串，长度不少于 1 个字节，不超过 36 个字节。

  会话的数据访问 ID。

- PartitionId – UTF-8 字符串，长度不少于 1 个字节，不超过 36 个字节。

  会话的分区 ID。

- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

属于会话的键值对（标签）的映射。

- RequestOrigin – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  请求的源。

- ProfileName – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与会话关联的 AWS Glue 使用情况配置文件的名称。

响应

- Session – 一个 会话 对象。

  返回响应中的会话对象。

错误

- AccessDeniedException
- IdempotentParameterMismatchException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ValidationException
- AlreadyExistsException
- ResourceNumberLimitExceededException

StopSession 动作（Python：停止会话）

停止会话。

请求

- Id – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

要停止的会话 ID。

- `RequestOrigin` – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  请求的源。

响应

- `Id` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  返回已停止会话的 ID。

错误

- `AccessDeniedException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `IllegalSessionStateException`
- `ConcurrentModificationException`

DeleteSession 操作（Python：删除会话）

删除会话。

请求

- `Id` – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的会话 ID。

- `RequestOrigin` – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  删除会话请求的源名称。

响应

- Id – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  返回要删除会话 ID。

错误

- AccessDeniedException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- IllegalSessionStateException
- ConcurrentModificationException

GetSession 动作（Python：get_session）

检索会话。

请求

- Id – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  会话 ID。
- RequestOrigin – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  请求的源。

响应

- Session – 一个 会话 对象。

  响应中会返回会话对象。

错误

- AccessDeniedException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException

ListSessions 动作（Python：列表会话）

检索会话列表。

请求

- NextToken – UTF-8 字符串，长度不超过 400000 个字节。

  下一组结果的令牌，没有更多结果时为 null。
- MaxResults – 数字（整数），不小于 1 或大于 1000。

  最大结果数量。
- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  属于会话的标签。
- RequestOrigin – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  请求的源。

响应

- Ids – UTF-8 字符串数组。

  返回会话 ID。
- Sessions – 会话 对象的数组。

返回会话对象。

- NextToken – UTF-8 字符串，长度不超过 400000 个字节。

  下一组结果的令牌，没有更多结果时为 null。

错误

- AccessDeniedException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

RunStatement 动作（Python：run_statement）

执行语句。

请求

- SessionId – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  要运行的语句的会话 ID。

- Code – 必需： UTF-8 字符串，长度不超过 68000 个字节。

  要运行的语句代码。

- RequestOrigin – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  请求的源。

响应

- Id – 数字（整数）。

  返回所运行的语句的 ID。

错误

- EntityNotFoundException
- AccessDeniedException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ValidationException
- ResourceNumberLimitExceededException
- IllegalSessionStateException

CancelStatement 操作（Python：取消语句）

取消语句。

请求

- SessionId – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  要取消的语句的会话 ID。
- Id – 必填：数字（整数）。

  要取消的语句的 ID。
- RequestOrigin – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  用于取消语句的请求的源。

响应

- 无响应参数。

错误

- AccessDeniedException

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- IllegalSessionStateException

GetStatement 动作（Python：get_statement）

检索语句。

请求

- SessionId – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  语句的会话 ID。
- Id – 必填：数字（整数）。

  语句的 ID。
- RequestOrigin – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  请求的源。

响应

- Statement – 一个 语句 对象。

  返回语句。

错误

- AccessDeniedException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

- InvalidInputException
- IllegalSessionStateException

ListStatements 动作（Python：列表语句）

列出会话的语句。

请求

- SessionId – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  语句的会话 ID。
- RequestOrigin – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  用于列出语句的请求的源。
- NextToken – UTF-8 字符串，长度不超过 400000 个字节。

  延续标记 (如果这是延续调用)。

响应

- Statements – 语句 对象的数组。

  返回语句列表。
- NextToken – UTF-8 字符串，长度不超过 400000 个字节。

  延续令牌（如果尚未返回所有语句）。

错误

- AccessDeniedException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException

- IllegalSessionStateException

开发终端节点 API

开发端点 API 介绍与使用自定义 DevEndpoint 进行测试相关的 AWS Glue API。

数据类型

- DevEndpoint 结构
- DevEndpointCustomLibraries 结构

DevEndpoint 结构

开发人员可以远程调试提取、转换和加载 (ETL) 脚本的开发终端节点。

字段

- EndpointName – UTF-8 字符串。

  DevEndpoint 的名称。
- RoleArn – UTF-8 字符串，与 AWS IAM ARN string pattern 匹配。

  该 DevEndpoint 中使用的 IAM 角色的 Amazon 资源名称（ARN）。
- SecurityGroupIds – UTF-8 字符串数组。

  此 DevEndpoint 中使用的安全组标识符的列表。
- SubnetId – UTF-8 字符串。

  此 DevEndpoint 的子网 ID。
- YarnEndpointAddress – UTF-8 字符串。

  此 DevEndpoint 使用的 YARN 终端节点地址。
- PrivateAddress – UTF-8 字符串。

  私有 IP 地址用于访问 VPC 中的 DevEndpoint（如果在 VPC 中创建了 DevEndpoint）。只有在 VPC 创建 DevEndpoint 时才会显示 PrivateAddress 字段。
- ZeppelinRemoteSparkInterpreterPort – 数字（整数）。

远程 Apache Spark 解释器的 Apache Zeppelin 端口。

- `PublicAddress` – UTF-8 字符串。

  此 `DevEndpoint` 使用的公有 IP 地址。仅当您创建非 virtual private cloud (VPC) `DevEndpoint` 时提供 `PublicAddress` 字段。

- `Status` – UTF-8 字符串。

  该 `DevEndpoint` 的当前状态。

- `WorkerType` – UTF-8 字符串（有效值：`Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""`）。

  分配给开发端点的预定义工作线程的类型。接受的值为 Standard、G.1X 或 G.2X。

  - 对于 `Standard` 工作线程类型，每个工作线程提供 4 个 vCPU、16 GB 内存和 50GB 磁盘，并且每个工作线程提供 2 个执行器。
  - 对于 `G.1X` 工作线程类型，每个工作线程映射到 1 个 DPU（4 个 vCPU，16 GB 内存，64 GB 磁盘），并且每个工作线程提供 1 个执行器。我们建议内存密集型作业使用该工作线程类型。
  - 对于 `G.2X` 工作线程类型，每个工作线程映射到 2 个 DPU（8 个 vCPU，32 GB 内存，128 GB 磁盘），并且每个工作线程提供 1 个执行器。我们建议内存密集型作业使用该工作线程类型。

  已知问题：当使用 `G.2X` `WorkerType` 配置创建开发端点时，开发端点的 Spark 驱动程序将在 4 个 vCPU、16 GB 内存和 64 GB 磁盘上运行。

- `GlueVersion` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #20 匹配。

  Glue 版本决定了 AWS Glue 支持的 Apache Spark 和 Python 版本。Python 版本表示支持在开发端点上运行 ETL 脚本的版本。

  有关可用的 AWS Glue 版本以及相应的 Spark 和 Python 版本的更多信息，请参阅开发人员指南中的 Glue 版本。

  在未指定 Glue 版本的情况下创建的开发端点默认为 Glue 0.9。

  可以使用 `CreateDevEndpoint` 或 `UpdateDevEndpoint` API 中的 `Arguments` 参数指定支持开发端点的 Python 版本。如果未提供参数，则版本默认为 Python 2。

- `NumberOfWorkers` – 数字（整数）。

  分配给开发端点的已定义 `workerType` 的工作线程数。

您可以定义的最大工作线程数是 299 (G.1X)，以及 149 (G.2X)。

- NumberOfNodes – 数字（整数）。

  分配给此 DevEndpoint 的 AWS Glue 数据处理单元（DPU）的数量。

- AvailabilityZone – UTF-8 字符串。

  此 DevEndpoint 所在的 AWS 可用区。

- VpcId – UTF-8 字符串。

  此 DevEndpoint 使用的 Virtual Private Cloud (VPC) 的 ID。

- ExtraPythonLibsS3Path – UTF-8 字符串。

  Amazon S3 存储桶中应加载到您的 DevEndpoint 中的一个或多个 Python 库的路径。多个值必须是以逗号分隔的完整路径。

  > **Note**
  >
  > 您只能与 DevEndpoint 一同使用纯 Python 库。目前不支持依赖于 C 扩展的库，如 pandas Python 数据分析库。

- ExtraJarsS3Path – UTF-8 字符串。

  S3 存储桶中应加载到您的 DevEndpoint 中的一个或多个 Java .jar 文件的路径。

  > **Note**
  >
  > 您只能与 DevEndpoint 一同使用纯 Java/Scala 库。

- FailureReason – UTF-8 字符串。

  此 DevEndpoint 中的当前失败的原因。

- LastUpdateStatus – UTF-8 字符串。

  上次更新的状态。

- CreatedTimestamp – 时间戳。

  创建此 DevEndpoint 的时间点。

- LastModifiedTimestamp – 时间戳。

上次修改此 DevEndpoint 的时间点。

- PublicKey – UTF-8 字符串。

  此 DevEndpoint 用于身份验证的公有密钥。提供此属性是为了向后兼容，因为要使用的推荐属性是公有密钥。

- PublicKeys – UTF-8 字符串数组，不超过 5 个字符串。

  DevEndpoints 用于身份验证的公有密钥列表。优先于单个公有密钥使用此属性，因为公有密钥允许您为每个客户端使用不同的私有密钥。

  > **Note**
  >
  > 如果以前使用公有密钥创建了端点，则必须删除该密钥才能设置公有密钥列表。使用 addPublicKeys 属性中的公有密钥内容和 deletePublicKeys 属性中的新密钥列表调用 UpdateDevEndpoint API 操作。

- SecurityConfiguration – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  该 DevEndpoint 将使用的 SecurityConfiguration 结构的名称。

- Arguments – 键值对的映射数组，不超过 100 对。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  用于配置 DevEndpoint 的参数的映射。

  有效参数为：

  - "--enable-glue-datacatalog": ""

  可以使用 CreateDevEndpoint 或 UpdateDevEndpoint API 中的 Arguments 参数指定支持开发端点的 Python 版本。如果未提供参数，则版本默认为 Python 2。

DevEndpointCustomLibraries 结构

要加载到开发终端节点中的自定义库。

字段

- `ExtraPythonLibsS3Path` – UTF-8 字符串。

  Amazon Simple Storage Service (Amazon S3) 存储桶中应加载到您的 `DevEndpoint` 中的一个或多个 Python 库的路径。多个值必须是以逗号分隔的完整路径。

  > **Note**
  >
  > 您只能与 `DevEndpoint` 一同使用纯 Python 库。目前不支持依赖于 C 扩展的库，如 pandas Python 数据分析库。

- `ExtraJarsS3Path` – UTF-8 字符串。

  S3 存储桶中应加载到您的 `DevEndpoint` 中的一个或多个 Java `.jar` 文件的路径。

  > **Note**
  >
  > 您只能与 `DevEndpoint` 一同使用纯 Java/Scala 库。

操作

- CreateDevEndpoint 操作（Python：create_dev_endpoint）
- UpdateDevEndpoint 操作（Python：update_dev_endpoint）
- DeleteDevEndpoint 操作（Python：delete_dev_endpoint）
- GetDevEndpoint 操作（Python：get_dev_endpoint）
- GetDevEndpoints 操作（Python：get_dev_endpoints）
- BatchGetDevEndpoints 操作（Python：batch_get_dev_endpoints）
- ListDevEndpoints 操作（Python：list_dev_endpoints）

CreateDevEndpoint 操作（Python：create_dev_endpoint）

创建新的开发终端节点。

请求

- `EndpointName` – 必填：UTF-8 字符串。

要为新 DevEndpoint 分配的名称。

- RoleArn – 必填： UTF-8 字符串，与 AWS IAM ARN string pattern 匹配。

  DevEndpoint 的 IAM 角色。

- SecurityGroupIds – UTF-8 字符串数组。

  新的 DevEndpoint 将使用的安全组的安全组 ID。

- SubnetId – UTF-8 字符串。

  新的 DevEndpoint 将使用的子网 ID。

- PublicKey – UTF-8 字符串。

  此 DevEndpoint 用于身份验证的公有密钥。提供此属性是为了向后兼容，因为要使用的推荐属性是公有密钥。

- PublicKeys – UTF-8 字符串数组，不超过 5 个字符串。

  开发终端节点用于身份验证的公有密钥列表。优先于单个公有密钥使用此属性，因为公有密钥允许您为每个客户端使用不同的私有密钥。

  > **Note**
  >
  > 如果以前使用公有密钥创建了端点，则必须删除该密钥才能设置公有密钥列表。使用 addPublicKeys 属性中的公有密钥内容和 deletePublicKeys 属性中的新密钥列表调用 UpdateDevEndpoint API。

- NumberOfNodes – 数字（整数）。

  要分配给该 DevEndpoint 的 AWS Glue 数据处理单元（DPU）的数量。

- WorkerType – UTF-8 字符串（有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""）。

  分配给开发端点的预定义工作线程的类型。接受的值为 Standard、G.1X 或 G.2X。

  - 对于 Standard 工作线程类型，每个工作线程提供 4 个 vCPU、16 GB 内存和 50GB 磁盘，并且每个工作线程提供 2 个执行器。
  - 对于 G.1X 工作线程类型，每个工作线程映射到 1 个 DPU（4 个 vCPU，16 GB 内存，64 GB 磁盘），并且每个工作线程提供 1 个执行器。我们建议内存密集型作业使用该工作线程类型。

- 对于 G.2X 工作线程类型，每个工作线程映射到 2 个 DPU（8 个 vCPU，32 GB 内存，128 GB 磁盘），并且每个工作线程提供 1 个执行器。我们建议内存密集型作业使用该工作线程类型。

  已知问题：当使用 G.2X WorkerType 配置创建开发端点时，开发端点的 Spark 驱动程序将在 4 个 vCPU、16 GB 内存和 64 GB 磁盘上运行。

- GlueVersion – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #20 匹配。

  Glue 版本决定了 AWS Glue 支持的 Apache Spark 和 Python 版本。Python 版本表示支持在开发端点上运行 ETL 脚本的版本。

  有关可用的 AWS Glue 版本以及相应的 Spark 和 Python 版本的更多信息，请参阅开发人员指南中的 Glue 版本。

  在未指定 Glue 版本的情况下创建的开发端点默认为 Glue 0.9。

  可以使用 CreateDevEndpoint 或 UpdateDevEndpoint API 中的 Arguments 参数指定支持开发端点的 Python 版本。如果未提供参数，则版本默认为 Python 2。

- NumberOfWorkers – 数字（整数）。

  分配给开发端点的已定义 workerType 的工作线程数。

  您可以定义的最大工作线程数是 299 (G.1X)，以及 149 (G.2X)。

- ExtraPythonLibsS3Path – UTF-8 字符串。

  Amazon S3 存储桶中应加载到您的 DevEndpoint 中的一个或多个 Python 库的路径。多个值必须是以逗号分隔的完整路径。

  > **Note**
  >
  > 您只能与 DevEndpoint 一同使用纯 Python 库。尚不支持依赖于 C 扩展的库，如 pandas Python 数据分析库。

- ExtraJarsS3Path – UTF-8 字符串。

  S3 存储桶中应加载到您的 DevEndpoint 中的一个或多个 Java .jar 文件的路径。

- SecurityConfiguration – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

该 DevEndpoint 将使用的 SecurityConfiguration 结构的名称。

- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  要用于此 DevEndpoint 的标签。您可以使用标签来限制对 DevEndpoint 的访问。有关 AWS Glue 中的标签的更多信息，请参阅开发人员指南中的 AWS Glue 中的 AWS 标签。
- Arguments – 键值对的映射数组，不超过 100 对。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  用于配置 DevEndpoint 的参数的映射。

响应

- EndpointName – UTF-8 字符串。

  为新的 DevEndpoint 指定的名称。
- Status – UTF-8 字符串。

  新的 DevEndpoint 的当前状态。
- SecurityGroupIds – UTF-8 字符串数组。

  分配给新 DevEndpoint 的安全组。
- SubnetId – UTF-8 字符串。

  分配给新的 DevEndpoint 的子网 ID。
- RoleArn – UTF-8 字符串，与 AWS IAM ARN string pattern 匹配。

  分配给新的 DevEndpoint 的角色的 Amazon Resource Name（ARN）。
- YarnEndpointAddress – UTF-8 字符串。

  此 DevEndpoint 使用的 YARN 终端节点的地址。
- ZeppelinRemoteSparkInterpreterPort – 数字（整数）。

远程 Apache Spark 解释器的 Apache Zeppelin 端口。

- `NumberOfNodes` – 数字（整数）。

  分配给此 DevEndpoint 的 AWS Glue 数据处理单元（DPU）的数量。

- `WorkerType` – UTF-8 字符串（有效值：`Standard=""` | `G.1X=""` | `G.2X=""` | `G.025X=""` | `G.4X=""` | `G.8X=""` | `Z.2X=""`）。

  分配给开发端点的预定义工作线程的类型。可能的值为 Standard、G.1X 或 G.2X。

- `GlueVersion` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #20 匹配。

  Glue 版本决定了 AWS Glue 支持的 Apache Spark 和 Python 版本。Python 版本表示支持在开发端点上运行 ETL 脚本的版本。

  有关可用的 AWS Glue 版本以及相应的 Spark 和 Python 版本的更多信息，请参阅开发人员指南中的 Glue 版本。

- `NumberOfWorkers` – 数字（整数）。

  分配给开发端点的已定义 `workerType` 的工作线程数。

- `AvailabilityZone` – UTF-8 字符串。

  此 `DevEndpoint` 所在的 AWS 可用区。

- `VpcId` – UTF-8 字符串。

  此 `DevEndpoint` 使用的 Virtual Private Cloud (VPC) 的 ID。

- `ExtraPythonLibsS3Path` – UTF-8 字符串。

  S3 存储桶中将加载到您的 `DevEndpoint` 中的一个或多个 Python 库的路径。

- `ExtraJarsS3Path` – UTF-8 字符串。

  S3 存储桶中应加载到您的 `DevEndpoint` 中的一个或多个 Java `.jar` 文件的路径。

- `FailureReason` – UTF-8 字符串。

  此 `DevEndpoint` 中的当前失败的原因。

- `SecurityConfiguration` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

该 DevEndpoint 使用的 SecurityConfiguration 结构的名称。

- CreatedTimestamp – 时间戳。

  创建此 DevEndpoint 的时间点。

- Arguments – 键值对的映射数组，不超过 100 对。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  用于配置该 DevEndpoint 的参数的映射。

  有效参数为：

  - "--enable-glue-datacatalog": ""

  可以使用 CreateDevEndpoint 或 UpdateDevEndpoint API 中的 Arguments 参数指定支持开发端点的 Python 版本。如果未提供参数，则版本默认为 Python 2。

错误

- AccessDeniedException
- AlreadyExistsException
- IdempotentParameterMismatchException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ValidationException
- ResourceNumberLimitExceededException

UpdateDevEndpoint 操作（Python：update_dev_endpoint）

更新指定的开发终端节点。

请求

- EndpointName – 必填：UTF-8 字符串。

要更新的 DevEndpoint 名称。

- PublicKey – UTF-8 字符串。

  DevEndpoint 要使用的公有密钥。

- AddPublicKeys – UTF-8 字符串数组，不超过 5 个字符串。

  该 DevEndpoint 将使用的公有密钥的列表。

- DeletePublicKeys – UTF-8 字符串数组，不超过 5 个字符串。

  要从 DevEndpoint 中删除的公有密钥的列表。

- CustomLibraries – 一个 DevEndpointCustomLibraries 对象。

  要加载到 DevEndpoint 中的自定义 Python 或 Java 库。

- UpdateEtlLibraries – 布尔值。

  如果需要更新要加载到开发终端节点中的自定义库列表，则为 True，否则为 False。

- DeleteArguments – UTF-8 字符串数组。

  要从用于配置 DevEndpoint 的参数映射中删除的参数键的列表。

- AddArguments – 键值对的映射数组，不超过 100 对。

  每个键是一个 UTF-8 字符串。

  每个值是一个 UTF-8 字符串。

  用于添加用来配置 DevEndpoint 的参数映射的参数的映射。

  有效参数为：

  - "--enable-glue-datacatalog": ""

  可以使用 CreateDevEndpoint 或 UpdateDevEndpoint API 中的 Arguments 参数指定支持开发端点的 Python 版本。如果未提供参数，则版本默认为 Python 2。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ValidationException

DeleteDevEndpoint 操作（Python：delete_dev_endpoint）

删除指定的开发终端节点。

请求

- EndpointName – 必填：UTF-8 字符串。

  DevEndpoint 的名称。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException

GetDevEndpoint 操作（Python：get_dev_endpoint）

检索有关指定开发终端节点的信息。

> **Note**
>
> 当您在 Virtual Private Cloud (VPC) 中创建开发终端节点时，AWS Glue 仅返回一个私有 IP 地址，并且不会填充公有 IP 地址字段。在创建非 VPC 开发终端节点时，AWS Glue 仅返回一个公有 IP 地址。

请求

- `EndpointName` – 必填：UTF-8 字符串。

  要为其检索信息的 `DevEndpoint` 名称。

响应

- `DevEndpoint` – 一个 DevEndpoint 对象。

  `DevEndpoint` 定义。

错误

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

GetDevEndpoints 操作（Python：get_dev_endpoints）

检索此 AWS 账户中的所有开发终端节点。

> **Note**
>
> 当您在 Virtual Private Cloud（VPC）中创建开发终端节点时，AWS Glue 仅返回一个私有 IP 地址，并且不会填充公有 IP 地址字段。在创建非 VPC 开发终端节点时，AWS Glue 仅返回一个公有 IP 地址。

请求

- MaxResults – 数字（整数），不小于 1 或大于 1000。

  要返回的信息的最大大小。
- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用)。

响应

- DevEndpoints – DevEndpoint 对象的数组。

  DevEndpoint 定义的列表。
- NextToken – UTF-8 字符串。

  延续令牌 (如果尚未返回所有 DevEndpoint 定义)。

错误

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException

BatchGetDevEndpoints 操作（Python：batch_get_dev_endpoints）

返回给定开发终端节点名称列表的资源元数据的列表。调用 ListDevEndpoints 操作后，您可以调用此操作来访问您有权访问的数据。此操作支持所有 IAM 权限，包括使用标签的权限条件。

请求

- customerAccountId – UTF-8 字符串。

  AWS 账户 ID。
- DevEndpointNames – 必填：UTF-8 字符串数组，不少于 1 个或不超过 25 个字符串。

  DevEndpoint 名称列表，这些名称可能是通过 ListDevEndpoint 操作返回的名称。

响应

- DevEndpoints – DevEndpoint 对象的数组。

  DevEndpoint 定义的列表。
- DevEndpointsNotFound – UTF-8 字符串数组，不少于 1 个或不超过 25 个字符串。

  未找到 DevEndpoints 列表。

错误

- AccessDeniedException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException

ListDevEndpoints 操作（Python：list_dev_endpoints）

检索此 AWS 账户中所有 DevEndpoint 资源的名称或带指定标签的资源。此操作可让您查看您账户中可用的资源及其名称。

此操作采用可选的 Tags 字段，您可以将其用作响应的筛选器，以便将标记的资源作为一个组进行检索。如果您选择使用标签筛选，则仅检索带标签的资源。

请求

- NextToken – UTF-8 字符串。

  延续令牌 (如果这是延续请求)。
- MaxResults – 数字（整数），不小于 1 或大于 1000。

  要返回的列表的最大大小。
- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  指定仅返回这些已标记的资源。

响应

- DevEndpointNames – UTF-8 字符串数组。

  账户中所有 DevEndpoint 的名称或带指定标签的 DevEndpoint。
- NextToken – UTF-8 字符串。

  延续令牌 (如果返回的列表不包含上一个可用的指标)。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

架构注册表

架构注册表 API 描述了与使用中的 AWS Glue架构相关的数据类型和 API。

数据类型

- RegistryId 结构
- RegistryListItem 结构
- MetadataInfo 结构
- OtherMetadataValueListItem 结构
- SchemaListItem 结构
- SchemaVersionListItem 结构
- MetadataKeyValuePair 结构
- SchemaVersionErrorItem 结构
- ErrorDetails 结构
- SchemaVersionNumber 结构
- SchemaId 结构

RegistryId 结构

包装结构，包含注册表名称和 Amazon Resource Name（ARN）。

字段

- `RegistryName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  注册表的名称。仅用于查找。必须提供 `RegistryArn` 或 `RegistryName` 中的一个。
- `RegistryArn` – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  要更新的注册表的 ARN。必须提供 `RegistryArn` 或 `RegistryName` 中的一个。

RegistryListItem 结构

包含注册表详细信息的结构。

字段

- `RegistryName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  注册表的名称。
- `RegistryArn` – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  注册表的 Amazon Resource Name（ARN）。
- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  注册表的描述。
- `Status` – UTF-8 字符串（有效值：`AVAILABLE | DELETING`）。

  注册表的状态。
- `CreatedTime` – UTF-8 字符串。

  注册表创建的数据。

- `UpdatedTime` – UTF-8 字符串。

  注册表更新的日期。

MetadataInfo 结构

包含用于架构版本的元数据信息的结构。

字段

- `MetadataValue` – UTF-8 字符串，长度不少于 1 个字节或超过 256 个字节，与 Custom string pattern #33 匹配。

  元数据键的相应值。
- `CreatedTime` – UTF-8 字符串。

  创建条目的时间。
- `OtherMetadataValueList` – OtherMetadataValueListItem 对象的数组。

  属于同一元数据键的其他元数据。

OtherMetadataValueListItem 结构

包含属于同一元数据键用于架构版本的其他元数据的结构。

字段

- `MetadataValue` – UTF-8 字符串，长度不少于 1 个字节或超过 256 个字节，与 Custom string pattern #33 匹配。

  元数据键的其他元数据的相应值属于同一元数据键。
- `CreatedTime` – UTF-8 字符串。

  创建条目的时间。

SchemaListItem 结构

包含架构的最小详细信息的对象。

字段

- RegistryName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  架构所在的注册表的名称。

- SchemaName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  架构的名称。

- SchemaArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  架构的 Amazon Resource Name（ARN）。

- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  架构描述。

- SchemaStatus – UTF-8 字符串（有效值：AVAILABLE | PENDING | DELETING）。

  架构的状态。

- CreatedTime – UTF-8 字符串。

  架构的创建日期和时间。

- UpdatedTime – UTF-8 字符串。

  架构的更新日期和时间。

SchemaVersionListItem 结构

包含有关架构版本的详细信息的对象。

字段

- SchemaArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  架构的 Amazon Resource Name（ARN）。

- SchemaVersionId – UTF-8 字符串，长度不少于 36 个字节或超过 36 个字节，与 Custom string pattern #17 匹配。

  架构版本的唯一标识符。
- VersionNumber – 数字（长度），不小于 1 或大于 100000。

  架构的版本号。
- Status – UTF-8 字符串（有效值：AVAILABLE |PENDING | FAILURE | DELETING）。

  架构版本的状态。
- CreatedTime – UTF-8 字符串。

  此架构版本的创建日期和时间。

MetadataKeyValuePair 结构

包含元数据键值对的结构。

字段

- MetadataKey – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #33 匹配。

  元数据键
- MetadataValue – UTF-8 字符串，长度不少于 1 个字节或超过 256 个字节，与 Custom string pattern #33 匹配。

  元数据键的相应值。

SchemaVersionErrorItem 结构

包含架构版本操作错误详细信息的对象。

字段

- VersionNumber – 数字（长度），不小于 1 或大于 100000。

  架构的版本号。
- ErrorDetails – 一个 ErrorDetails 对象。

架构版本的错误的详细信息。

ErrorDetails 结构

包含错误详细信息的对象。

字段

- `ErrorCode` – UTF-8 字符串。

  错误的错误代码。
- `ErrorMessage` – UTF-8 字符串。

  错误的错误消息。

SchemaVersionNumber 结构

包含架构版本信息的结构。

字段

- `LatestVersion` – 布尔值。

  可用于架构的最新版本。
- `VersionNumber` – 数字（长度），不小于 1 或大于 100000。

  架构的版本号。

SchemaId 结构

架构注册表中架构的唯一 ID。 AWS Glue

字段

- `SchemaArn` – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  架构的 Amazon Resource Name（ARN）。必须提供 `SchemaArn` 或 `SchemaName` 中的一个。

- SchemaName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  架构的名称。必须提供 SchemaArn 或 SchemaName 中的一个。

- RegistryName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  包含该架构的架构注册表的名称。

操作

- CreateRegistry 操作（Python：创建注册表）
- CreateSchema 操作（Python：创建架构）
- GetSchema 操作（Python：获取架构）
- ListSchemaVersions 操作（Python：列表架构版本）
- GetSchemaVersion 操作（Python：get_schema_version）
- GetSchemaVersionsDiff 动作（Python：get_schema_versions_diff）
- ListRegistries 操作（Python：列表_注册表）
- ListSchemas 操作（Python：列表架构）
- RegisterSchemaVersion 操作（Python：register_schema_version）
- UpdateSchema 操作（Python：更新架构）
- CheckSchemaVersionValidity 操作（Python：检查架构版本有效性）
- UpdateRegistry 操作（Python：update_registry）
- GetSchemaByDefinition 动作（Python：按定义获取架构）
- GetRegistry 操作（Python：获取注册表）
- PutSchemaVersionMetadata 操作（Python：put_schema_version_metadata）
- QuerySchemaVersionMetadata 操作（Python：查询架构版本元数据）
- RemoveSchemaVersionMetadata 操作（Python：移除架构版本元数据）
- DeleteRegistry 操作（Python：删除注册表）
- DeleteSchema 操作（Python：删除架构）
- DeleteSchemaVersions 操作（Python：删除架构版本）

CreateRegistry 操作（Python：创建注册表）

创建可用于保存架构集合的新注册表。

请求

- `RegistryName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  要创建的注册表的名称，其最大长度不得超过 255 个字符，且只能包含字母、数字、连字符、下划线、美元符号或哈希标记。无空格。

- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  注册表的描述。如果未提供描述，则不会有任何默认值。

- `Tags` – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  AWS 包含键值对的标签，可以通过控制台、命令行或 API 进行搜索。

响应

- `RegistryArn` – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  服务注册表的 Amazon Resource Name（ARN）。

- `RegistryName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  注册表的名称。

- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  注册表的描述。

- `Tags` – 键值对的映射数组，不超过 50 对。

每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

每个值是一个 UTF-8 字符串，不超过 256 个字节。

注册表的标签。

错误

- `InvalidInputException`
- `AccessDeniedException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`
- `ConcurrentModificationException`
- `InternalServiceException`

CreateSchema 操作（Python：创建架构）

创建新的架构集并注册架构定义。如果架构集已经存在，但没有实际注册版本，则返回错误。

创建架构集时，版本检查点将设置为第一个版本。兼容模式“DISABLED”限制在第一个架构版本之后添加任何其他架构版本。对于所有其他兼容模式，在使用 `RegisterSchemaVersion` API 时，兼容性设置的验证将仅从第二个版本开始应用。

当不使用 `RegistryId` 调用此 API 时，这将为注册表数据库表中的“default-registry”创建一个条目（如果它尚未存在）。

请求

- `RegistryId` – 一个 RegistryId 对象。

  这是包含注册表标识字段的包装形状。如果未提供此选项，则会使用默认的注册表。相同的 ARN 格式为：`arn:aws:glue:us-east-2:<customer id>:registry/default-registry:random-5-letter-id`。
- `SchemaName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

要创建的架构的名称，其最大长度不得超过 255 个字符，且只能包含字母、数字、连字符、下划线、美元符号或哈希标记。无空格。

- DataFormat – 必填： UTF-8 字符串（有效值：AVRO | JSON | PROTOBUF）。

  架构定义的数据格式。目前支持 AVRO、JSON 和 PROTOBUF。

- Compatibility – UTF-8 字符串（有效值：NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL）。

  架构的兼容模式。可能的值包括：

  - NONE：不适用兼容模式。您可以在开发场景或者在不知道要应用于架构的兼容性模式时使用此选项。无需进行兼容性检查，接受添加的任何新版本。
  - DISABLED：此兼容性选项可防止对特定架构进行版本控制。您可以使用此选项来防止将来对架构进行版本控制。
  - BACKWARD：建议使用此兼容性选项，因为它允许数据接收器读取架构的最新版本和上一个版本。这意味着，例如，新架构版本不能删除数据字段或更改这些字段的类型，因此使用先前版本的读取器无法读取这些字段。
  - BACKWARD_ALL：此兼容性选项允许数据接收器读取架构的最新版本和所有先前版本。当您需要删除字段或添加可选字段以及检查所有先前架构版本的兼容性时，您可以使用此选项。
  - FORWARD：此兼容性选项允许数据接收器读取当前和下一个架构版本，但不一定是更高版本。当您需要添加字段或删除可选字段，但仅检查上一个架构版本的兼容性时，您可以使用此选项。
  - FORWARD_ALL：此兼容性选项允许数据接收器读取任何新的注册架构的创建器写入。当您需要添加字段或删除可选字段以及检查所有先前架构版本的兼容性时，您可以使用此选项。
  - FULL：此兼容性选项允许数据接收器读取使用先前版本或下一版本的架构，但不一定是早期版本或更高版本的创建器写入的数据。当您需要添加或删除可选字段，但仅检查上一个架构版本的兼容性时，您可以使用此选项。
  - FULL_ALL：此兼容性选项允许数据接收器读取使用所有架构先前版本的创建器写入的数据。当您需要添加或删除可选字段以及检查所有先前架构版本的兼容性时，您可以使用此选项。

- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  架构的描述（可选）。如果未提供描述，则不会有自动任何默认值。

- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

每个值是一个 UTF-8 字符串，不超过 256 个字节。

AWS 包含键值对的标签，可以通过控制台、命令行或 API 进行搜索。如果指定，则遵循 AWS tags-on-create 模式。

- SchemaDefinition – UTF-8 字符串，长度不少于 1 个字节或超过 170000 个字节，与 Custom string pattern #32 匹配。

  使用 DataFormat 设置 SchemaName 的架构定义。

响应

- RegistryName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  注册表的名称。

- RegistryArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  注册表的 Amazon Resource Name（ARN）。

- SchemaName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  架构的名称。

- SchemaArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  架构的 Amazon Resource Name（ARN）。

- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  对架构（如果在创建时指定）的描述。

- DataFormat – UTF-8 字符串（有效值：AVRO | JSON | PROTOBUF）。

  架构定义的数据格式。目前支持 AVRO、JSON 和 PROTOBUF。

- Compatibility – UTF-8 字符串（有效值：NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL）。

架构的兼容模式。

- SchemaCheckpoint – 数字（长度），不小于 1 或大于 100000。

  检查点的版本号（上次更改兼容模式时）。

- LatestSchemaVersion – 数字（长度），不小于 1 或大于 100000。

  与返回的架构定义相关联的架构的最新版本。

- NextSchemaVersion – 数字（长度），不小于 1 或大于 100000。

  与返回的架构定义相关联的架构的下一个版本。

- SchemaStatus – UTF-8 字符串（有效值：AVAILABLE | PENDING | DELETING）。

  架构的状态。

- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  架构的标签。

- SchemaVersionId – UTF-8 字符串，长度不少于 36 个字节或超过 36 个字节，与 Custom string pattern #17 匹配。

  第一个架构版本的唯一标识符。

- SchemaVersionStatus – UTF-8 字符串（有效值：AVAILABLE |PENDING | FAILURE | DELETING）。

  所创建的第一个架构版本的状态。

错误

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- AlreadyExistsException
- ResourceNumberLimitExceededException

- ConcurrentModificationException
- InternalServiceException

GetSchema 操作（Python：获取架构）

详细描述指定的架构。

请求

- SchemaId – 必填：一个 SchemaId 对象。

  包含架构标识字段的包装结构。结构包含：

  - SchemaId$SchemaArn：架构的亚马逊资源名称 (ARN)。必须提供 SchemaArn，或者 SchemaName 和 RegistryName。
  - SchemaId$SchemaName：架构的名称。必须提供 SchemaArn，或者 SchemaName 和 RegistryName。

响应

- RegistryName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  注册表的名称。
- RegistryArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  注册表的 Amazon Resource Name（ARN）。
- SchemaName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  架构的名称。
- SchemaArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  架构的 Amazon Resource Name（ARN）。
- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

对架构（如果在创建时指定）的描述。

- DataFormat – UTF-8 字符串（有效值：AVRO | JSON | PROTOBUF）。

  架构定义的数据格式。目前支持 AVRO、JSON 和 PROTOBUF。

- Compatibility – UTF-8 字符串（有效值：NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL）。

  架构的兼容模式。

- SchemaCheckpoint – 数字（长度），不小于 1 或大于 100000。

  检查点的版本号（上次更改兼容模式时）。

- LatestSchemaVersion – 数字（长度），不小于 1 或大于 100000。

  与返回的架构定义相关联的架构的最新版本。

- NextSchemaVersion – 数字（长度），不小于 1 或大于 100000。

  与返回的架构定义相关联的架构的下一个版本。

- SchemaStatus – UTF-8 字符串（有效值：AVAILABLE | PENDING | DELETING）。

  架构的状态。

- CreatedTime – UTF-8 字符串。

  架构的创建日期和时间。

- UpdatedTime – UTF-8 字符串。

  架构的更新日期和时间。

错误

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

ListSchemaVersions 操作（Python：列表架构版本）

返回已创建的架构版本列表，其中包含最少信息。结果中将不会包含处于已删除状态的架构版本。如果没有可用的架构版本，则返回空白结果。

请求

- SchemaId – 必填：一个 SchemaId 对象。

  包含架构标识字段的包装结构。结构包含：

  - SchemaId$SchemaArn：架构的亚马逊资源名称 (ARN)。必须提供 SchemaArn，或者 SchemaName 和 RegistryName。
  - SchemaId$SchemaName：架构的名称。必须提供 SchemaArn，或者 SchemaName 和 RegistryName。
- MaxResults – 数字（整数），不小于 1 或大于 100。

  每页所需的最大结果数量。如果未提供该值，则每页默认最大数量为 25。
- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用)。

响应

- Schemas – SchemaVersionListItem 对象的数组。

  SchemaVersionList 对象的数组，包含每个架构版本的详细信息。
- NextToken – UTF-8 字符串。

  对返回的标记列表进行分页的延续令牌 (如果列表的当前片段不是最后一个，则返回)。

错误

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

GetSchemaVersion 操作（Python：get_schema_version）

通过在创建或注册架构版本时分配的唯一 ID 获取指定架构。结果中将不会包含处于已删除状态的架构版本。

请求

- SchemaId – 一个 SchemaId 对象。

  包含架构标识字段的包装结构。结构包含：

  - SchemaId$SchemaArn：架构的亚马逊资源名称 (ARN)。必须提供 SchemaArn，或者 SchemaName 和 RegistryName。
  - SchemaId$SchemaName：架构的名称。必须提供 SchemaArn，或者 SchemaName 和 RegistryName。
- SchemaVersionId – UTF-8 字符串，长度不少于 36 个字节或超过 36 个字节，与 Custom string pattern #17 匹配。

  架构版本的 SchemaVersionId。此字段是按架构 ID 进行读取的必填字段。必须提供此字段或 SchemaId 包装。
- SchemaVersionNumber – 一个 SchemaVersionNumber 对象。

  架构的版本号。

响应

- SchemaVersionId – UTF-8 字符串，长度不少于 36 个字节或超过 36 个字节，与 Custom string pattern #17 匹配。

  架构版本的 SchemaVersionId。
- SchemaDefinition – UTF-8 字符串，长度不少于 1 个字节或超过 170000 个字节，与 Custom string pattern #32 匹配。

  架构 ID 的架构定义。
- DataFormat – UTF-8 字符串（有效值：AVRO | JSON | PROTOBUF）。

  架构定义的数据格式。目前支持 AVRO、JSON 和 PROTOBUF。
- SchemaArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

架构的 Amazon Resource Name（ARN）。

- VersionNumber – 数字（长度），不小于 1 或大于 100000。

  架构的版本号。

- Status – UTF-8 字符串（有效值：AVAILABLE |PENDING | FAILURE | DELETING）。

  架构版本的状态。

- CreatedTime – UTF-8 字符串。

  此架构版本的创建日期和时间。

错误

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

GetSchemaVersionsDiff 动作（Python：get_schema_versions_diff）

获取架构注册表中两个存储架构版本之间指定差异类型的架构版本差异。

此 API 允许您比较同一架构下的两个架构定义之间的两个架构版本。

请求

- SchemaId – 必填：一个 SchemaId 对象。

  包含架构标识字段的包装结构。结构包含：

  - SchemaId$SchemaArn：架构的亚马逊资源名称 (ARN)。必须提供 SchemaArn 或 SchemaName 中的一个。
  - SchemaId$SchemaName：架构的名称。必须提供 SchemaArn 或 SchemaName 中的一个。

- FirstSchemaVersionNumber – 必填：一个 SchemaVersionNumber 对象。

  要比较的两个架构版本中的第一个版本。

- SecondSchemaVersionNumber – 必填：一个 SchemaVersionNumber 对象。

要比较的两个架构版本中的第二个版本。

- SchemaDiffType – 必填：UTF-8 字符串（有效值：SYNTAX_DIFF）。

  指的是 SYNTAX_DIFF，这是当前支持的差异类型。

响应

- Diff – UTF-8 字符串，长度不少于 1 个字节或超过 340000 个字节，与 Custom string pattern #32 匹配。

  JsonPatch 格式化为字符串的模式之间的区别。

错误

- InvalidInputException
- EntityNotFoundException
- AccessDeniedException
- InternalServiceException

ListRegistries 操作（Python：列表_注册表）

返回您已创建的注册表列表，其中包含最少的注册表信息。结果中将不会包含处于 Deleting 状态的注册表。如果没有可用的注册表，将返回空白结果。

请求

- MaxResults – 数字（整数），不小于 1 或大于 100。

  每页所需的最大结果数量。如果未提供该值，则每页默认最大数量为 25。

- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用)。

响应

- Registries – RegistryListItem 对象的数组。

RegistryDetailedListItem 对象的数组，其中包含每个注册表的最少详细信息。

- NextToken – UTF-8 字符串。

  对返回的标记列表进行分页的延续令牌 (如果列表的当前片段不是最后一个，则返回)。

错误

- InvalidInputException
- AccessDeniedException
- InternalServiceException

ListSchemas 操作（Python：列表架构）

返回具有最少详细信息的架构列表。结果中将不会包含处于 Deleting (正在删除) 状态的架构。如果没有可用的架构，则返回空白结果。

未提供 RegistryId 时，跨注册表的所有架构都将成为 API 响应的一部分。

请求

- RegistryId – 一个 RegistryId 对象。

  包装结构，包含注册表名称和 Amazon Resource Name（ARN）。

- MaxResults – 数字（整数），不小于 1 或大于 100。

  每页所需的最大结果数量。如果未提供该值，则每页默认最大数量为 25。

- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用)。

响应

- Schemas – SchemaListItem 对象的数组。

  SchemaListItem 对象数组，包含每个架构的详细信息。

- NextToken – UTF-8 字符串。

  对返回的标记列表进行分页的延续令牌 (如果列表的当前片段不是最后一个，则返回)。

错误

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

RegisterSchemaVersion 操作（Python：register_schema_version）

将新版本添加到现有架构。如果新版本的架构不符合架构集的兼容性要求，则返回错误。如果架构集尚未存在于架构注册表中，此 API 将不会创建新的架构集，并返回 404 错误。

如果是要在架构注册表中注册的第一个架构定义，则此 API 将存储架构版本并立即返回。否则，由于兼容模式，此调用可能会比其他操作运行时间长。您可以调用带有 SchemaVersionId 的 GetSchemaVersion API 来检查兼容性模式。

如果同一架构定义已作为版本存储在架构注册表中，则现有架构的架构 ID 将返回给调用方。

请求

- SchemaId – 必填：一个 SchemaId 对象。

  包含架构标识字段的包装结构。结构包含：

  - SchemaId$SchemaArn：架构的亚马逊资源名称 (ARN)。必须提供 SchemaArn，或者 SchemaName 和 RegistryName。
  - SchemaId$SchemaName：架构的名称。必须提供 SchemaArn，或者 SchemaName 和 RegistryName。
- SchemaDefinition – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 170000 个字节，与 Custom string pattern #32 匹配。

  使用 DataFormat 设置 SchemaName 的架构定义。

响应

- SchemaVersionId – UTF-8 字符串，长度不少于 36 个字节或超过 36 个字节，与 Custom string pattern #17 匹配。

  表示此架构版本的唯一 ID。

- VersionNumber – 数字（长度），不小于 1 或大于 100000。

  此架构的版本（仅适用于同步流，以防这是第一个版本）。
- Status – UTF-8 字符串（有效值：AVAILABLE |PENDING | FAILURE | DELETING）。

  架构版本的状态。

错误

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- ResourceNumberLimitExceededException
- ConcurrentModificationException
- InternalServiceException

UpdateSchema 操作（Python：更新架构）

更新架构集的描述、兼容性设置或版本检查点。

为了更新兼容性设置，调用不会验证具有新兼容性设置的整个架构版本集的兼容性。如果提供用于 Compatibility 的值，还需要 VersionNumber（检查点）。API 将验证检查点版本号的一致性。

如果提供 VersionNumber（检查点）的值，则 Compatibility 是可选的，这可用于设置/重置架构的检查点。

仅当架构处于 AVAILABLE (可用) 状态时，才会进行此更新。

请求

- SchemaId – 必填：一个 SchemaId 对象。

  包含架构标识字段的包装结构。结构包含：
  - SchemaId$SchemaArn：架构的亚马逊资源名称 (ARN)。必须提供 SchemaArn 或 SchemaName 中的一个。
  - SchemaId$SchemaName：架构的名称。必须提供 SchemaArn 或 SchemaName 中的一个。
- SchemaVersionNumber – 一个 SchemaVersionNumber 对象。

通过检查点检验所需的版本号。必须提供 VersionNumber 或 Compatibility 中的一个。

- Compatibility – UTF-8 字符串（有效值：NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL ）。

  架构的新兼容性设置。

- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  架构的新描述。

响应

- SchemaArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  架构的 Amazon Resource Name（ARN）。

- SchemaName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  架构的名称。

- RegistryName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  包含该架构的注册表的名称。

错误

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- ConcurrentModificationException
- InternalServiceException

CheckSchemaVersionValidity 操作（Python：检查架构版本有效性）

验证提供的架构。这个调用没有副作用，它只是使用所提供的架构（使用 DataFormat 作为格式）进行验证。由于它不采用架构集名称，因此不会执行兼容性检查。

请求

- DataFormat – 必填： UTF-8 字符串（有效值：AVRO | JSON | PROTOBUF）。

  架构定义的数据格式。目前支持 AVRO、JSON 和 PROTOBUF。
- SchemaDefinition – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 170000 个字节，与 Custom string pattern #32 匹配。

  必须验证的架构的定义。

响应

- Valid – 布尔值。

  如果模式有效，则返回“true”，否则返回“false”。
- Error – UTF-8 字符串，长度不少于 1 个字节或超过 5000 个字节。

  验证失败错误消息。

错误

- InvalidInputException
- AccessDeniedException
- InternalServiceException

UpdateRegistry 操作（Python：update_registry）

更新用于保存架构集的现有注册表。更新的属性与注册表相关，并且不会修改注册表中的任何架构。

请求

- RegistryId – 必填：一个 RegistryId 对象。

  这是包装结构，包含注册表名称和 Amazon Resource Name（ARN）。

- `Description` – 必填：描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  注册表的描述。如果未提供描述，则不会更新此字段。

响应

- `RegistryName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  更新的注册表的名称。
- `RegistryArn` – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  更新的注册表的 Amazon Resource name（ARN）。

错误

- `InvalidInputException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `ConcurrentModificationException`
- `InternalServiceException`

GetSchemaByDefinition 动作（Python：按定义获取架构）

通过 `SchemaDefinition` 来检索架构。架构定义将发送到架构注册表、进行标准化和哈希处理。如果哈希在 `SchemaName` 或 ARN（或默认注册表，如果未提供）的范围内匹配，则返回该架构的元数据。否则，将返回 404 或 NotFound 错误。结果中将不会包含处于 `Deleted` 状态的架构版本。

请求

- `SchemaId` – 必填：一个 SchemaId 对象。

  包含架构标识字段的包装结构。结构包含：

  - SchemaId$SchemaArn：架构的亚马逊资源名称 (ARN)。必须提供 `SchemaArn` 或 `SchemaName` 中的一个。

  - SchemaId$SchemaName：架构的名称。必须提供 SchemaArn 或 SchemaName 中的一个。
- SchemaDefinition – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 170000 个字节，与 Custom string pattern #32 匹配。

  需要架构详细信息的架构的定义。

响应

- SchemaVersionId – UTF-8 字符串，长度不少于 36 个字节或超过 36 个字节，与 Custom string pattern #17 匹配。

  架构版本的架构 ID。
- SchemaArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  架构的 Amazon Resource Name（ARN）。
- DataFormat – UTF-8 字符串（有效值：AVRO | JSON | PROTOBUF）。

  架构定义的数据格式。目前支持 AVRO、JSON 和 PROTOBUF。
- Status – UTF-8 字符串（有效值：AVAILABLE |PENDING | FAILURE | DELETING）。

  架构版本的状态。
- CreatedTime – UTF-8 字符串。

  架构的创建日期和时间。

错误

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

GetRegistry 操作（Python：获取注册表）

详细描述指定的注册表。

请求

- RegistryId – 必填：一个 RegistryId 对象。

  这是包装结构，包含注册表名称和 Amazon Resource Name（ARN）。

响应

- RegistryName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  注册表的名称。

- RegistryArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  注册表的 Amazon Resource Name（ARN）。

- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  注册表的描述。

- Status – UTF-8 字符串（有效值：AVAILABLE | DELETING）。

  注册表的状态。

- CreatedTime – UTF-8 字符串。

  注册表的创建日期和时间。

- UpdatedTime – UTF-8 字符串。

  注册表的更新日期和时间。

错误

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

PutSchemaVersionMetadata 操作（Python：put_schema_version_metadata）

放置指定架构版本 ID 的元数据键值对。每个架构版本最多允许 10 个键值对。它们可以通过一个或多个调用添加。

请求

- `SchemaId` – 一个 SchemaId 对象。

  架构的唯一 ID。

- `SchemaVersionNumber` – 一个 SchemaVersionNumber 对象。

  架构的版本号。

- `SchemaVersionId` – UTF-8 字符串，长度不少于 36 个字节或超过 36 个字节，与 Custom string pattern #17 匹配。

  架构版本的唯一版本 ID。

- `MetadataKeyValue` – 必填：一个 MetadataKeyValuePair 对象。

  元数据键的相应值。

响应

- `SchemaArn` – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  架构的 Amazon Resource Name（ARN）。

- `SchemaName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  架构的名称。

- `RegistryName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  注册表的名称。

- `LatestVersion` – 布尔值。

架构的最新版本。

- `VersionNumber` – 数字（长度），不小于 1 或大于 100000。

  架构的版本号。

- `SchemaVersionId` – UTF-8 字符串，长度不少于 36 个字节或超过 36 个字节，与 Custom string pattern #17 匹配。

  架构版本的唯一版本 ID。

- `MetadataKey` – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #33 匹配。

  元数据键。

- `MetadataValue` – UTF-8 字符串，长度不少于 1 个字节或超过 256 个字节，与 Custom string pattern #33 匹配。

  元数据键的值。

错误

- `InvalidInputException`
- `AccessDeniedException`
- `AlreadyExistsException`
- `EntityNotFoundException`
- `ResourceNumberLimitExceededException`

QuerySchemaVersionMetadata 操作（Python：查询架构版本元数据）

架构版本元数据信息的查询。

请求

- `SchemaId` – 一个 SchemaId 对象。

  包装结构，包含架构名称和 Amazon Resource Name（ARN）。

- `SchemaVersionNumber` – 一个 SchemaVersionNumber 对象。

架构的版本号。

- SchemaVersionId – UTF-8 字符串，长度不少于 36 个字节或超过 36 个字节，与 Custom string pattern #17 匹配。

  架构版本的唯一版本 ID。

- MetadataList – MetadataKeyValuePair 对象的数组。

  搜索元数据的键值对，如果未提供键值对，则将获取所有元数据信息。

- MaxResults – 数字（整数），不小于 1 或大于 50。

  每页所需的最大结果数量。如果未提供该值，则每页默认最大数量为 25。

- NextToken – UTF-8 字符串。

  延续标记 (如果这是延续调用)。

响应

- MetadataInfoMap – 键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 128 个字节，与 Custom string pattern #33 匹配。

  每个值都是一个 MetadataInfo 对象。

  元数据键和关联值的映射。

- SchemaVersionId – UTF-8 字符串，长度不少于 36 个字节或超过 36 个字节，与 Custom string pattern #17 匹配。

  架构版本的唯一版本 ID。

- NextToken – UTF-8 字符串。

  对返回的标记列表进行分页的延续令牌 (如果列表的当前片段不是最后一个，则返回)。

错误

- InvalidInputException
- AccessDeniedException

- EntityNotFoundException

RemoveSchemaVersionMetadata 操作（Python：移除架构版本元数据）

从指定架构版本 ID 的架构版本元数据中删除键值对。

请求

- SchemaId – 一个 SchemaId 对象。

  包装结构，包含架构名称和 Amazon Resource Name（ARN）。
- SchemaVersionNumber – 一个 SchemaVersionNumber 对象。

  架构的版本号。
- SchemaVersionId – UTF-8 字符串，长度不少于 36 个字节或超过 36 个字节，与 Custom string pattern #17 匹配。

  架构版本的唯一版本 ID。
- MetadataKeyValue – 必填：一个 MetadataKeyValuePair 对象。

  元数据键的值。

响应

- SchemaArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  架构的 Amazon Resource Name（ARN）。
- SchemaName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  架构的名称。
- RegistryName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  注册表的名称。
- LatestVersion – 布尔值。

架构的最新版本。

- `VersionNumber` – 数字（长度），不小于 1 或大于 100000。

  架构的版本号。

- `SchemaVersionId` – UTF-8 字符串，长度不少于 36 个字节或超过 36 个字节，与 Custom string pattern #17 匹配。

  架构版本的版本 ID。

- `MetadataKey` – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #33 匹配。

  元数据键。

- `MetadataValue` – UTF-8 字符串，长度不少于 1 个字节或超过 256 个字节，与 Custom string pattern #33 匹配。

  元数据键的值。

错误

- `InvalidInputException`
- `AccessDeniedException`
- `EntityNotFoundException`

DeleteRegistry 操作（Python：删除注册表）

删除包含架构及其所有版本的整个注册表。要获取删除操作的状态，您可以在异步调用后调用 `GetRegistry` API。删除注册表将停用注册表的所有联机操作，如 `UpdateRegistry`、`CreateSchema`、`UpdateSchema` 和 `RegisterSchemaVersion` API。

请求

- `RegistryId` – 必填：一个 RegistryId 对象。

  这是包装结构，包含注册表名称和 Amazon Resource Name（ARN）。

响应

- RegistryName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  正在删除的注册表的名称。

- RegistryArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  服务注册表的 Amazon Resource Name（ARN）。

- Status – UTF-8 字符串（有效值：AVAILABLE | DELETING）。

  注册表的状态。成功的操作将返回 Deleting 状态。

错误

- InvalidInputException
- EntityNotFoundException
- AccessDeniedException
- ConcurrentModificationException

DeleteSchema 操作（Python：删除架构）

删除整个架构集，包括架构集及其所有版本。要获取删除操作的状态，您可以在异步调用后调用 GetSchema API。删除注册表将停用架构的所有联机操作，例如 GetSchemaByDefinition 和 RegisterSchemaVersion API。

请求

- SchemaId – 必填：一个 SchemaId 对象。

  这是包装结构，包含架构名称和 Amazon Resource Name（ARN）。

响应

- SchemaArn – UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

要删除的架构的 Amazon Resource Name（ARN）。

- SchemaName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #18 匹配。

  正在删除的架构的名称。

- Status – UTF-8 字符串（有效值：AVAILABLE | PENDING | DELETING）。

  架构的状态。

错误

- InvalidInputException
- EntityNotFoundException
- AccessDeniedException
- ConcurrentModificationException

DeleteSchemaVersions 操作（Python：删除架构版本）

从指定架构中删除版本。可以提供版本号或范围。如果兼容模式禁止删除必要的版本（如 BACKWARDS_FULL），则返回错误。此调用后调用的 GetSchemaVersions API 将列出已删除版本的状态。

当版本号范围包含检查指向版本时，API 将返回 409 冲突，并且不会继续删除操作。您必须首先使用 DeleteSchemaCheckpoint API，然后再使用此 API。

您无法使用 DeleteSchemaVersions API 删除架构集中的第一个架构版本。第一个架构版本只能通过 DeleteSchema API 删除。此操作还将删除附加在架构版本下的 SchemaVersionMetadata。硬性删除将在数据库上强制执行。

如果兼容模式禁止删除必要的版本（如 BACKWARDS_FULL），则返回错误。

请求

- SchemaId – 必填：一个 SchemaId 对象。

  这是包装结构，包含架构名称和 Amazon Resource Name（ARN）。

- Versions – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 100000 个字节，与 Custom string pattern #34 匹配。

可以提供一个版本范围，其格式可能为：

- 单个版本号，5
- 一个范围，5-8：删除版本 5、6、7、8

响应

- `SchemaVersionErrors` – SchemaVersionErrorItem 对象的数组。

  `SchemaVersionErrorItem` 对象列表，每个对象都包含错误和架构版本。

错误

- `InvalidInputException`
- `EntityNotFoundException`
- `AccessDeniedException`
- `ConcurrentModificationException`

工作流程

工作流 API 介绍与在 AWS Glue 中创建、更新或查看工作流相关的数据类型和 API。您可以在 90 天内访问工作流和任务运行的任务运行历史记录。

数据类型

- JobNodeDetails 结构
- CrawlerNodeDetails 结构
- TriggerNodeDetails 结构
- Crawl 结构
- Node 结构
- Edge 结构
- Workflow 结构
- WorkflowGraph 结构
- WorkflowRun 结构

- WorkflowRunStatistics 结构
- StartingEventBatchCondition 结构
- Blueprint 结构
- BlueprintDetails 结构
- LastActiveDefinition 结构
- BlueprintRun 结构

JobNodeDetails 结构

工作流程中展示的作业节点的详细信息。

字段

- `JobRuns` – JobRun 对象的数组。

  作业节点表示的作业运行的信息。

CrawlerNodeDetails 结构

工作流程中存在的爬网程序节点的详细信息。

字段

- `Crawls` – 爬网 对象的数组。

  爬网节点表示的爬网的列表。

TriggerNodeDetails 结构

工作流程中存在的触发器节点的详细信息。

字段

- `Trigger` – 一个 触发器 对象。

  触发器节点表示的触发器的信息。

Crawl 结构

工作流程中的爬网的详细信息。

字段

- State – UTF-8 字符串（有效值：RUNNING | CANCELLING | CANCELLED | SUCCEEDED | FAILED | ERROR）。

  爬网程序的状态。

- StartedOn – 时间戳。

  爬网操作的开始日期和时间。

- CompletedOn – 时间戳。

  爬网操作的完成日期和时间。

- ErrorMessage – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  与爬网关联的错误消息。

- LogGroup – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节，与 Log group string pattern 匹配。

  与爬网关联的日志组。

- LogStream – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节，与 Log-stream string pattern 匹配。

  与爬网关联的日志流。

Node 结构

节点在工作流图表中表示 AWS Glue 组件（触发器、爬网程序或任务）。

字段

- Type – UTF-8 字符串（有效值：CRAWLER | JOB | TRIGGER）。

  节点表示的 AWS Glue 组件的类型。

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  节点表示的 AWS Glue 组件的名称。

- UniqueId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分配给工作流程中的节点的唯一 ID。

- TriggerDetails – 一个 TriggerNodeDetails 对象。

  触发器的详细信息（当节点表示触发器时）。

- JobDetails – 一个 JobNodeDetails 对象。

  作业的详细信息（当节点表示作业时）。

- CrawlerDetails – 一个 CrawlerNodeDetails 对象。

  爬网程序的详细信息（当节点表示爬网程序时）。

Edge 结构

边缘表示两个 AWS Glue 组件之间的定向连接，这两个组件是边缘所属的工作流的一部分。

字段

- SourceId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  边缘在其中启动的工作流程中节点的唯一 ID。

- DestinationId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  边缘在其中终止的工作流程中节点的唯一 ID。

Workflow 结构

工作流是为完成复杂 ETL 任务而运行的多个相互依赖的 AWS Glue 任务和爬网程序的集合。工作流管理其所有任务和爬网程序的执行和监控。

字段

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  工作流的名称。
- Description – UTF-8 字符串。

  工作流程的描述。
- DefaultRunProperties – 键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串。

  要用作每个工作流程执行的一部分的属性的集合。运行属性可供工作流中的每个任务使用。任务可以修改工作流中后续任务的属性。
- CreatedOn – 时间戳。

  工作流程的创建日期和时间。
- LastModifiedOn – 时间戳。

  工作流程的最后修改日期和时间。
- LastRun – 一个 WorkflowRun 对象。

  有关上次工作流程执行的信息。
- Graph – 一个 WorkflowGraph 对象。

  将属于工作流的所有 AWS Glue 组件表示为节点，并将它们之间的定向连接表示为边缘的图表。
- CreationStatus – UTF-8 字符串（有效值：CREATING | CREATED | CREATION_FAILED）。

  工作流程的创建状态。
- MaxConcurrentRuns – 数字（整数）。

  您可以使用此参数防止系统对数据进行多次不必要的更新，来控制成本，或者在某些情况下，防止系统超过任何组件任务的最大并发运行次数。如果您将此参数留空，则系统对并发工作流运行的次数没有限制。
- BlueprintDetails – 一个 BlueprintDetails 对象。

此结构指示创建此特定工作流的蓝图的详细信息。

WorkflowGraph 结构

工作流图表表示完整的工作流，其中包含工作流中存在的所有 AWS Glue 组件以及它们之间的所有定向连接。

字段

- `Nodes` – 节点 对象的数组。

  属于工作流程的表示为节点的 AWS Glue 组件的列表。
- `Edges` – Edge 对象的数组。

  属于工作流程的节点之间的所有定向连接的列表。

WorkflowRun 结构

工作流程运行是提供所有运行时信息的工作流程的执行。

字段

- `Name` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  已运行的工作流的名称。
- `WorkflowRunId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  此工作流程运行的 ID。
- `PreviousRunId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  先前工作流运行的 ID。
- `WorkflowRunProperties` – 键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

每个值是一个 UTF-8 字符串。

运行期间设置的工作流程运行属性。

- StartedOn – 时间戳。

  工作流程运行的开始日期和时间。

- CompletedOn – 时间戳。

  工作流程运行的完成日期和时间。

- Status – UTF-8 字符串（有效值：RUNNING | COMPLETED | STOPPING | STOPPED | ERROR）。

  工作流程运行的状态。

- ErrorMessage – UTF-8 字符串。

  此错误消息描述了在启动工作流运行时可能发生的任何错误。目前唯一的错误消息是“超过工作流的并发运行：foo。”

- Statistics – 一个 WorkflowRunStatistics 对象。

  运行的统计数据。

- Graph – 一个 WorkflowGraph 对象。

  将属于工作流的所有 AWS Glue 组件表示为节点，并将它们之间的定向连接表示为边缘的图表。

- StartingEventBatchCondition – 一个 StartingEventBatchCondition 对象。

  启动工作流运行的批处理条件。

WorkflowRunStatistics 结构

工作流程运行统计数据提供了有关工作流程运行的统计数据。

字段

- TotalActions – 数字（整数）。

  工作流程运行中的操作的总数。

- TimeoutActions – 数字（整数）。

  超时操作的总数。

- FailedActions – 数字（整数）。

  失败操作的总数。

- StoppedActions – 数字（整数）。

  已停止操作的总数。

- SucceededActions – 数字（整数）。

  成功操作的总数。

- RunningActions – 数字（整数）。

  处于正在运行状态的操作的总数。

- ErroredActions – 数字（整数）。

  指示工作流运行中处于 ERROR 状态的任务计数。

- WaitingActions – 数字（整数）。

  指示工作流运行中处于 WAITING 状态的任务运行计数。

StartingEventBatchCondition 结构

启动工作流运行的批处理条件。批处理大小中的事件数到达（在这种情况下，BatchSize 成员为非零），或批处理时间已过期（在这种情况下，BatchWindow 成员为非零）。

字段

- BatchSize – 数字（整数）。

  批处理中的事件数。

- BatchWindow – 数字（整数）。

  批处理时间的持续时间，以秒为单位。

Blueprint 结构

蓝图的详细信息。

字段

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  蓝图的名称。

- Description – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节。

  蓝图的描述。

- CreatedOn – 时间戳。

  蓝图的注册日期和时间。

- LastModifiedOn – 时间戳。

  蓝图的上次修改日期和时间。

- ParameterSpec – UTF-8 字符串，长度不少于 1 个字节或超过 131072 个字节。

  指示蓝图参数规范列表的 JSON 字符串。

- BlueprintLocation – UTF-8 字符串。

  指定 Amazon S3 中发布蓝图的路径。

- BlueprintServiceLocation – UTF-8 字符串。

  当您调用 CreateBlueprint/UpdateBlueprint 将蓝图注册到 AWS Glue 时，指定 Amazon S3 中复制蓝图的路径。

- Status – UTF-8 字符串（有效值：CREATING | ACTIVE | UPDATING | FAILED）。

  蓝图注册的状态。

  - 正在创建 – 正在进行蓝图注册。
  - 激活 – 蓝图已成功注册。
  - 正在更新 – 正在更新蓝图注册。
  - 失败 – 蓝图注册失败。

- ErrorMessage – UTF-8 字符串。

  错误消息。

- LastActiveDefinition – 一个 LastActiveDefinition 对象。

如果蓝图有多个版本，且最新版本出现一些错误，则此属性指示该服务可用的最后一个成功的蓝图定义。

BlueprintDetails 结构

蓝图的详细信息。

字段

- `BlueprintName` – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  蓝图的名称。
- `RunId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  此蓝图的运行 ID。

LastActiveDefinition 结构

如果蓝图有多个版本，且最新版本出现一些错误，则此属性指示该服务可用的最后一个成功的蓝图定义。

字段

- `Description` – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节。

  蓝图的描述。
- `LastModifiedOn` – 时间戳。

  蓝图的上次修改日期和时间。
- `ParameterSpec` – UTF-8 字符串，长度不少于 1 个字节或超过 131072 个字节。

  指定蓝图参数的 JSON 字符串。
- `BlueprintLocation` – UTF-8 字符串。

  指定 Amazon S3 中由 AWS Glue 开发人员发布的蓝图的路径。
- `BlueprintServiceLocation` – UTF-8 字符串。

当您创建或更新蓝图时，指定 Amazon S3 中复制蓝图的路径。

BlueprintRun 结构

蓝图运行的详细信息。

字段

- `BlueprintName` – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  蓝图的名称。

- `RunId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  此蓝图运行的运行 ID。

- `WorkflowName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  因蓝图运行成功而创建的工作流的名称。如果蓝图运行出现错误，则不会创建工作流。

- `State` – UTF-8 字符串（有效值：`RUNNING` | `SUCCEEDED` | `FAILED` | `ROLLING_BACK`）。

  蓝图运行的状态。可能的值有：

  - 正在运行 – 正在进行蓝图运行。
  - 成功 – 蓝图运行已成功完成。
  - 失败 – 蓝图运行失败，已完成回滚。
  - 回滚 – 蓝图运行失败，正在回滚。

- `StartedOn` – 时间戳。

  蓝图运行的开始日期和时间。

- `CompletedOn` – 时间戳。

  蓝图运行的完成日期和时间。

- `ErrorMessage` – UTF-8 字符串。

  表示运行蓝图时出现的任何错误。

- `RollbackErrorMessage` – UTF-8 字符串。

  如果在创建工作流实体时出现任何错误，我们会尝试回滚至创建实体的该点，然后将实体删除。此属性指示尝试删除创建的实体时出现的错误。

- `Parameters` – UTF-8 字符串，长度不少于 1 个字节或超过 131072 个字节。

  以字符串形式显示的蓝图参数。您必须为 `Blueprint$ParameterSpec` 中定义的参数规范所需的每个键提供一个值。

- `RoleArn` – UTF-8 字符串，不少于 1 个字节或超过 1024 个字节，与 Custom string pattern #26 匹配。

  角色 ARN。此角色将由 AWS Glue 服务代入，并将用于创建工作流和工作流的其他实体。

操作

- CreateWorkflow 操作（Python：create_workflow）
- UpdateWorkflow 操作（Python：update_workflow）
- DeleteWorkflow 操作（Python：delete_workflow）
- GetWorkflow 操作（Python：get_workflow）
- ListWorkflows 操作（Python：list_workflows）
- BatchGetWorkflows 操作（Python：batch_get_workflows）
- GetWorkflowRun 操作（Python：get_workflow_run）
- GetWorkflowRuns 操作（Python：get_workflow_runs）
- GetWorkflowRunProperties 操作（Python：get_workflow_run_properties）
- PutWorkflowRunProperties 操作（Python：put_workflow_run_properties）
- CreateBlueprint 操作（Python：create_blueprint）
- UpdateBlueprint 操作（Python：update_blueprint）
- DeleteBlueprint 操作（Python：delete_blueprint）
- ListBlueprints 操作（Python：list_blueprint）
- BatchGetBlueprints 操作（Python：batch_get_blueprints）
- StartBlueprintRun 操作（Python：start_blueprint_run）
- GetBlueprintRun 操作（Python：get_blueprint_run）

- GetBlueprintRuns 操作（Python：get_blueprint_runs）
- StartWorkflowRun 操作（Python：start_workflow_run）
- StopWorkflowRun 操作（Python：stop_workflow_run）
- ResumeWorkflowRun 操作（Python：resume_workflow_run）

CreateWorkflow 操作（Python：create_workflow）

创建新的工作流程。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要向工作流程分配的名称。它应在您的账户中是唯一的。
- Description – UTF-8 字符串。

  工作流程的描述。
- DefaultRunProperties – 键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串。

  要用作每个工作流程执行的一部分的属性的集合。
- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  要用于此工作流程的标签。
- MaxConcurrentRuns – 数字（整数）。

  您可以使用此参数防止系统对数据进行多次不必要的更新，来控制成本，或者在某些情况下，防止系统超过任何组件任务的最大并发运行次数。如果您将此参数留空，则系统对并发工作流运行的次数没有限制。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  作为请求的一部分提供的工作流程的名称。

错误

- `AlreadyExistsException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `ConcurrentModificationException`

UpdateWorkflow 操作（Python：update_workflow）

更新现有工作流程。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要更新的工作流程的名称。
- `Description` – UTF-8 字符串。

  工作流程的描述。
- `DefaultRunProperties` – 键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串。

  要用作每个工作流程执行的一部分的属性的集合。
- `MaxConcurrentRuns` – 数字（整数）。

您可以使用此参数防止系统对数据进行多次不必要的更新，来控制成本，或者在某些情况下，防止系统超过任何组件任务的最大并发运行次数。如果您将此参数留空，则系统对并发工作流运行的次数没有限制。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  输入中指定的工作流程的名称。

错误

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

DeleteWorkflow 操作（Python：delete_workflow）

删除工作流程。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的工作流程的名称。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  输入中指定的工作流程的名称。

错误

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException

GetWorkflow 操作（Python：get_workflow）

检索工作流程的资源元数据。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索的工作流程的名称。
- IncludeGraph – 布尔值。

  指定在返回工作流程资源元数据时是否包含图表。

响应

- Workflow – 一个 工作流 对象。

  工作流程的资源元数据。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

ListWorkflows 操作（Python：list_workflows）

列出账户中创建的工作流程的名称。

请求

- NextToken – UTF-8 字符串。

  延续令牌 (如果这是延续请求)。
- MaxResults – 数字（整数），不小于 1 或大于 25。

  要返回的列表的最大大小。

响应

- Workflows – UTF-8 字符串数组，不少于 1 个或不超过 25 个字符串。

  账户中工作流程的名称的列表。
- NextToken – UTF-8 字符串。

  延续令牌（如果尚未返回所有工作流程名称）。

错误

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchGetWorkflows 操作（Python：batch_get_workflows）

返回一系列给定工作流程名称的资源元数据列表。调用 ListWorkflows 操作后，您可以调用此操作来访问您有权访问的数据。此操作支持所有 IAM 权限，包括使用标签的权限条件。

请求

- Names – 必填：UTF-8 字符串数组，不少于 1 个或不超过 25 个字符串。

  触发器名称的列表，这些名称可能是通过 ListWorkflows 操作返回的名称。
- IncludeGraph – 布尔值。

  指定在返回工作流程资源元数据时是否包含图表。

响应

- Workflows – 工作流对象的数组，不少于 1 个或不超过 25 个结构。

  工作流程资源元数据的列表。
- MissingWorkflows – UTF-8 字符串数组，不少于 1 个或不超过 25 个字符串。

  未找到工作流程名称的列表。

错误

- InternalServiceException
- OperationTimeoutException
- InvalidInputException

GetWorkflowRun 操作（Python：get_workflow_run）

检索给定工作流程运行的元数据。您可以在 90 天内访问工作流和任务运行的任务运行历史记录。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要运行的工作流程的名称。
- RunId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  工作流程运行的 ID。
- IncludeGraph – 布尔值。

  指定是否在响应中包含工作流程图表。

响应

- Run – 一个 WorkflowRun 对象。

  请求的工作流程运行元数据。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetWorkflowRuns 操作（Python：get_workflow_runs）

检索给定工作流程的所有运行的元数据。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要返回其运行元数据的工作流程的名称。
- IncludeGraph – 布尔值。

  指定是否在响应中包含工作流程图表。
- NextToken – UTF-8 字符串。

  响应的最大大小。
- MaxResults – 数字（整数），不小于 1 或大于 1000。

  要包含在响应中的工作流程运行的最大数目。

响应

- Runs – WorkflowRun对象的数组，不少于 1 个或不超过 1000 个结构。

  工作流程运行元数据对象的列表。
- NextToken – UTF-8 字符串。

  延续令牌（如果尚未返回所有请求的工作流程运行）。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetWorkflowRunProperties 操作（Python：get_workflow_run_properties）

检索运行期间已测试的工作流程运行属性。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  已运行的工作流程的名称。
- RunId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  应返回其运行属性的工作流程运行的 ID。

响应

- RunProperties – 键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串。

  指定运行期间已测试的工作流程运行属性。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException

- OperationTimeoutException

PutWorkflowRunProperties 操作（Python：put_workflow_run_properties）

为给定工作流程运行放置指定的工作流程运行属性。如果指定的运行中已存在属性，则重写该值，否则将该属性添加到现有属性。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  已运行的工作流程的名称。
- RunId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  应更新其运行属性的工作流程运行的 ID。
- RunProperties – 必填：键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串。

  为指定的运行放置的属性。

响应

- 无响应参数。

错误

- AlreadyExistsException
- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

- ResourceNumberLimitExceededException
- ConcurrentModificationException

CreateBlueprint 操作（Python：create_blueprint）

在 AWS Glue 中注册蓝图。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  蓝图的名称。
- Description – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节。

  蓝图的描述。
- BlueprintLocation – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 8192 个字节，与 Custom string pattern #28 匹配。

  指定 Amazon S3 中发布蓝图的路径。
- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  应用于此蓝图的标签。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  返回已注册的蓝图的名称。

错误

- AlreadyExistsException

- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ResourceNumberLimitExceededException

UpdateBlueprint 操作（Python：update_blueprint）

更新已注册的蓝图。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  蓝图的名称。
- Description – UTF-8 字符串，长度不少于 1 个字节或超过 512 个字节。

  蓝图的描述。
- BlueprintLocation – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 8192 个字节，与 Custom string pattern #28 匹配。

  指定 Amazon S3 中发布蓝图的路径。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  返回已更新的蓝图的名称。

错误

- EntityNotFoundException
- ConcurrentModificationException
- InvalidInputException
- OperationTimeoutException

- InternalServiceException
- IllegalBlueprintStateException

DeleteBlueprint 操作（Python：delete_blueprint）

删除现有蓝图。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的蓝图的名称。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  返回已删除的蓝图的名称。

错误

- InvalidInputException
- OperationTimeoutException
- InternalServiceException

ListBlueprints 操作（Python：list_blueprint）

列出账户中的所有蓝图名称。

请求

- NextToken – UTF-8 字符串。

  延续令牌 (如果这是延续请求)。
- MaxResults – 数字（整数），不小于 1 或大于 25。

要返回的列表的最大大小。

- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  通过 AWS 资源标签筛选列表。

响应

- Blueprints – UTF-8 字符串数组。

  账户中蓝图的名称列表。
- NextToken – UTF-8 字符串。

  延续令牌（如果所有蓝图名称尚未返回）。

错误

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchGetBlueprints 操作（Python：batch_get_blueprints）

检索有关蓝图列表的信息。

请求

- Names – 必填：UTF-8 字符串数组，不少于 1 个或不超过 25 个字符串。

  蓝图名称的列表。
- IncludeBlueprint – 布尔值。

  指定是否在响应中包含蓝图。
- IncludeParameterSpec – 布尔值。

指定是否在响应中包含蓝图的参数（作为 JSON 字符串）。

响应

- `Blueprints` – Blueprint 对象的数组。

  返回作为 `Blueprints` 对象的蓝图列表。

- `MissingBlueprints` – UTF-8 字符串数组。

  返回未找到的 `BlueprintNames`。

错误

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

StartBlueprintRun 操作（Python：start_blueprint_run）

启动指定的蓝图的新运行。

请求

- `BlueprintName` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  蓝图的名称。

- `Parameters` – UTF-8 字符串，长度不少于 1 个字节或超过 131072 个字节。

  指定作为 `BlueprintParameters` 对象的参数。

- `RoleArn` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 1024 个字节，与 Custom string pattern #26 匹配。

  指定用于创建工作流的 IAM 角色。

响应

- RunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  此蓝图运行的运行 ID。

错误

- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ResourceNumberLimitExceededException
- EntityNotFoundException
- IllegalBlueprintStateException

GetBlueprintRun 操作（Python：get_blueprint_run）

检索蓝图运行的详细信息。

请求

- BlueprintName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #27 匹配。

  蓝图的名称。

- RunId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索的蓝图运行的运行 ID。

响应

- BlueprintRun – 一个 BlueprintRun 对象。

  返回 BlueprintRun 对象。

错误

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetBlueprintRuns 操作（Python：get_blueprint_runs）

检索指定蓝图的蓝图运行详细信息。

请求

- BlueprintName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  蓝图的名称。
- NextToken – UTF-8 字符串。

  延续令牌 (如果这是延续请求)。
- MaxResults – 数字（整数），不小于 1 或大于 1000。

  要返回的列表的最大大小。

响应

- BlueprintRuns – BlueprintRun 对象的数组。

  返回 BlueprintRun 对象的列表。
- NextToken – UTF-8 字符串。

  延续令牌（如果所有请求的蓝图运行尚未返回）。

错误

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

- InvalidInputException

StartWorkflowRun 操作（Python：start_workflow_run）

启动指定的工作流程的新运行。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要启动的工作流程的名称。
- RunProperties – 键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值是一个 UTF-8 字符串。

  新工作流运行的工作流运行属性。

响应

- RunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  新运行的 ID。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentRunsExceededException

StopWorkflowRun 操作（Python：stop_workflow_run）

停止执行指定的工作流程运行。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要停止的工作流程的名称。
- RunId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要停止的工作流程运行的 ID。

响应

- 无响应参数。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- IllegalWorkflowStateException

ResumeWorkflowRun 操作（Python：resume_workflow_run）

重新启动上一个部分完成的工作流运行的选定节点，并恢复工作流运行。然后，所选节点以及这些节点下游的所有节点就会运行。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索的工作流的名称。

- RunId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要停止的工作流运行的 ID。
- NodeIds – 必填：UTF-8 字符串数组。

  要重新启动的节点的节点 ID 的列表。要重新启动的节点必须在原始运行中尝试运行。

响应

- RunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  分配给恢复的工作流运行的新 ID。工作流运行的每个恢复都将有一个新的运行 ID。
- NodeIds – UTF-8 字符串数组。

  实际重新启动的节点的节点 ID 列表。

错误

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- ConcurrentRunsExceededException
- IllegalWorkflowStateException

使用情况配置文件

使用情况配置文件 API 描述了与在中创建、更新或查看使用情况配置文件相关的数据类型和 API AWS Glue。

数据类型

- ProfileConfiguration 结构
- ConfigurationObject 结构

- UsageProfileDefinition 结构

ProfileConfiguration 结构

指定管理员在 AWS Glue 使用情况配置文件中配置的作业和会话值。

字段

- `SessionConfiguration` – 键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值都是一个 ConfigurationObject 对象。

  会话配置参数的键值映射。 AWS Glue
- `JobConfiguration` – 键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值都是一个 ConfigurationObject 对象。

  作业配置参数的键值映射。 AWS Glue

ConfigurationObject 结构

指定管理员为 AWS Glue 使用情况配置文件中配置的每个作业或会话参数设置的值。

字段

- `DefaultValue` – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #31 匹配。

  参数的默认值。
- `AllowedValues` – UTF-8 字符串数组。

  参数允许值的列表。
- `MinValue` – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #31 匹配。

参数允许的最小值。

- `MaxValue` – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节，与 Custom string pattern #31 匹配。

  参数允许的最大值。

UsageProfileDefinition 结构

描述 AWS Glue 使用情况配置文件。

字段

- `Name` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  使用情况配置文件的名称。

- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  对使用情况配置文件的描述。

- `CreatedOn` – 时间戳。

  创建使用情况配置文件的日期和时间。

- `LastModifiedOn` – 时间戳。

  上次修改使用情况配置文件的日期和时间。

操作

- CreateUsageProfile 操作（Python：create_usage_profile）
- GetUsageProfile 动作（Python：get_usage_profile）
- UpdateUsageProfile 操作（Python：update_usage_profile）
- DeleteUsageProfile 操作（Python：delete_usage_profile）
- ListUsageProfiles 动作（Python：列表_usage_profiles）

CreateUsageProfile 操作（Python：create_usage_profile）

创建 AWS Glue 使用情况配置文件。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  使用情况配置文件的名称。

- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  对使用情况配置文件的描述。

- Configuration – 必填：一个 ProfileConfiguration 对象。

  一个ProfileConfiguration对象，用于指定配置文件的工作和会话值。

- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  应用于使用情况配置文件的标签列表。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  创建的使用情况配置文件的名称。

错误

- InvalidInputException
- InternalServiceException
- AlreadyExistsException
- OperationTimeoutException

- ResourceNumberLimitExceededException
- OperationNotSupportedException

GetUsageProfile 动作（Python：get_usage_profile）

检索有关指定 AWS Glue 使用情况配置文件的信息。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  要检索的使用情况配置文件的名称。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  使用情况配置文件的名称。
- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  对使用情况配置文件的描述。
- Configuration – 一个 ProfileConfiguration 对象。

  一个ProfileConfiguration对象，用于指定配置文件的工作和会话值。
- CreatedOn – 时间戳。

  创建使用情况配置文件的日期和时间。
- LastModifiedOn – 时间戳。

  上次修改使用情况配置文件的日期和时间。

错误

- InvalidInputException
- InternalServiceException

- EntityNotFoundException
- OperationTimeoutException
- OperationNotSupportedException

UpdateUsageProfile 操作（Python：update_usage_profile）

更新 AWS Glue 使用情况配置文件。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  使用情况配置文件的名称。
- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  对使用情况配置文件的描述。
- Configuration – 必填：一个 ProfileConfiguration 对象。

  一个ProfileConfiguration对象，用于指定配置文件的工作和会话值。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  已更新的使用情况配置文件的名称。

错误

- InvalidInputException
- InternalServiceException
- EntityNotFoundException
- OperationTimeoutException
- OperationNotSupportedException

- `ConcurrentModificationException`

DeleteUsageProfile 操作（Python：delete_usage_profile）

删除 AWS Glue 指定的使用情况配置文件。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的使用情况配置文件的名称。

响应

- 无响应参数。

错误

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `OperationNotSupportedException`

ListUsageProfiles 动作（Python：列表_usage_profiles）

列出所有 AWS Glue 使用情况配置文件。

请求

- `NextToken` – UTF-8 字符串，长度不超过 400000 个字节。

  延续标记 (如果这是延续调用，则包括)。
- `MaxResults` – 数字（整数），不小于 1 或大于 200。

  在单个响应中返回的最大使用情况配置文件数。

响应

- `Profiles` – UsageProfileDefinition 对象的数组。

  使用情况配置文件 (`UsageProfileDefinition`) 对象的列表。
- `NextToken` – UTF-8 字符串，长度不超过 400000 个字节。

  延续令牌 (如果当前列表片段不是最后一个，则呈现)。

错误

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `OperationNotSupportedException`

机器学习 API

机器学习 API 描述了机器学习数据类型，并包括用于创建、删除或更新转换或启动机器学习任务运行的 API。

数据类型

- TransformParameters 结构
- EvaluationMetrics 结构
- MLTransform 结构
- FindMatchesParameters 结构
- FindMatchesMetrics 结构
- ConfusionMatrix 结构
- GlueTable 结构
- TaskRun 结构
- TransformFilterCriteria 结构
- TransformSortCriteria 结构
- TaskRunFilterCriteria 结构
- TaskRunSortCriteria 结构

- TaskRunProperties 结构
- FindMatchesTaskRunProperties 结构
- ImportLabelsTaskRunProperties 结构
- ExportLabelsTaskRunProperties 结构
- LabelingSetGenerationTaskRunProperties 结构
- SchemaColumn 结构
- TransformEncryption 结构
- MLUserDataEncryption 结构
- ColumnImportance 结构

TransformParameters 结构

与机器学习转换关联的特定于算法的参数。

字段

- `TransformType` – 必填：UTF-8 字符串（有效值：`FIND_MATCHES`）。

  机器学习转换的类型

  有关机器学习转换的类型的信息，请参阅 创建 Machine Learning 转换。
- `FindMatchesParameters` – 一个 FindMatchesParameters 对象。

  查找匹配算法的参数。

EvaluationMetrics 结构

评估指标提供机器学习转换的质量估计值。

字段

- `TransformType` – 必填：UTF-8 字符串（有效值：`FIND_MATCHES`）。

  机器学习转换的类型
- `FindMatchesMetrics` – 一个 FindMatchesMetrics 对象。

  查找匹配算法的评估指标。

MLTransform 结构

机器学习转换的结构。

字段

- `TransformId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  为机器学习转换生成的唯一转换 ID。ID 保证唯一性，不会改变。
- `Name` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  机器学习转换的用户定义的名称。名称不保证唯一性，可随时更改。
- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  机器学习转换的用户定义的长格式描述文本。描述不保证唯一性，可随时更改。
- `Status` – UTF-8 字符串（有效值：`NOT_READY` | `READY` | `DELETING`）。

  机器学习转换的当前状态。
- `CreatedOn` – 时间戳。

  时间戳。此机器学习转换的创建时间和日期。
- `LastModifiedOn` – 时间戳。

  时间戳。此机器学习转换的最后一个修改时间点。
- `InputRecordTables` – GlueTable 对象的数组，不超过 10 个结构。

  转换使用的 AWS Glue 表定义的列表。
- `Parameters` – 一个 TransformParameters 对象。

  一个 `TransformParameters` 对象。您可以使用参数来优化（自定义）机器学习转换的行为，方法是指定它从哪些数据中学习，以及您对各种权衡的偏好（例如精确率与召回率，或准确度与成本）。
- `EvaluationMetrics` – 一个 EvaluationMetrics 对象。

  一个 `EvaluationMetrics` 对象。评估指标提供机器学习转换的质量估计值。
- `LabelCount` – 数字（整数）。

对于此转换由 AWS Glue 生成的标签文件的计数标识符。要创建更好的转换，您可以迭代方式下载、标注和上载标签文件。

- Schema – SchemaColumn 对象的数组，不超过 100 个结构。

  键值对的映射表示该转换可针对的列和数据类型。具有 100 列的上限。

- Role – UTF-8 字符串。

  具有所需权限的 IAM 角色的名称或 Amazon Resource Name（ARN）。所需权限包括 AWS Glue 资源的 AWS Glue 服务角色权限和转换所需的 Amazon S3 权限。

  - 此角色需要 AWS Glue 服务角色权限才能允许访问 AWS Glue 中的资源。请参阅将策略附加到访问 AWS Glue 的 IAM 用户。
  - 此角色需要对您的 Amazon Simple Storage Service (Amazon S3) 源、目标、临时目录、脚本以及此转换的任务运行所使用的任何库的权限。

- GlueVersion – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #20 匹配。

  此值确定与此机器学习转换兼容的 AWS Glue 版本。建议大多数用户使用 Glue 1.0。如果未设置此值，则 Glue 兼容性默认为 Glue 0.9。有关更多信息，请参阅开发人员指南中的 AWS Glue 版本。

- MaxCapacity – 数字（double）。

  分配给此转换的任务运行的 AWS Glue 数据处理单元（DPU）的数量。您可以分配 2 到 100 个 DPU；默认值为 10。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅 AWS Glue 价格页面。

  MaxCapacity 是具有 NumberOfWorkers 和 WorkerType 的互斥选项。

  - 如果已设置 NumberOfWorkers 或 WorkerType，则不能设置 MaxCapacity。
  - 如果已设置 MaxCapacity，则不能设置 NumberOfWorkers 或 WorkerType。
  - 如果已设置 WorkerType，则 NumberOfWorkers 为必填项（反之亦然）。
  - MaxCapacity 和 NumberOfWorkers 都必须至少为 1。

  在将 WorkerType 字段设置为 Standard 之外的值时，MaxCapacity 字段将自动设置并变为只读。

- WorkerType – UTF-8 字符串（有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""）。

  在此转换的任务运行时分配的预定义工作线程的类型。接受的值为 Standard、G.1X 或 G.2X。

- 对于 Standard 工作线程类型，每个工作线程提供 4 个 vCPU、16 GB 内存和 50GB 磁盘，并且每个工作线程提供 2 个执行器。
- 对于 G.1X 工作线程类型，每个工作线程提供 4 个 vCPU、16 GB 内存和 64GB 磁盘，并且每个工作线程提供 1 个执行器。
- 对于 G.2X 工作线程类型，每个工作线程提供 84 个 vCPU、32 GB 内存和 128GB 磁盘，并且每个工作线程提供 1 个执行器。

MaxCapacity 是具有 NumberOfWorkers 和 WorkerType 的互斥选项。

- 如果已设置 NumberOfWorkers 或 WorkerType，则不能设置 MaxCapacity。
- 如果已设置 MaxCapacity，则不能设置 NumberOfWorkers 或 WorkerType。
- 如果已设置 WorkerType，则 NumberOfWorkers 为必填项（反之亦然）。
- MaxCapacity 和 NumberOfWorkers 都必须至少为 1。

- NumberOfWorkers – 数字（整数）。

  在转换的任务运行时分配的已定义 workerType 的工作线程数。

  如果已设置 WorkerType，则 NumberOfWorkers 为必填项（反之亦然）。

- Timeout - 数字（整数），至少为 1。

  机器学习转换的超时（以分钟为单位）。

- MaxRetries – 数字（整数）。

  在机器学习转换的 MLTaskRun 失败后重试的最大次数。

- TransformEncryption – 一个 TransformEncryption 对象。

  应用于访问用户数据的转换的静态加密设置。机器学习转换可以访问在 Amazon S3 中使用 KMS 加密的用户数据。

FindMatchesParameters 结构

用于配置查找匹配项转换的参数。

字段

- PrimaryKeyColumnName – UTF-8 字符串，不少于 1 个字节或超过 1024 个字节，与 Single-line string pattern 匹配。

唯一标识源表中的行的列名称。用于帮助标识匹配的记录。

- PrecisionRecallTradeoff – 数字（双数），不超过 1.0。

  调整转换以在查准率与查全率之间取得平衡时选择的值。值 0.5 表示没有首选项；值 1.0 表示纯粹因精确率而产生的偏差，值 0.0 表示因召回率而产生的偏差。因为这是一种权衡，所以选择接近 1.0 的值表示非常低的召回率，选择接近 0.0 的值会导致非常低的精确率。

  精确率指标指示模型在预测匹配时正确的频率。

  查全率指标表示，对于实际匹配，您的模型预测匹配的频率。

- AccuracyCostTradeoff – 数字（双数），不超过 1.0。

  调整转换以在准确性与成本之间取得平衡时选择的值。值 0.5 表示系统平衡准确度和成本问题。值为 1.0 表示纯粹因准确度而产生的偏差，这通常会导致成本更高，有时会高得多。值为 0.0 表示纯粹因成本而产生的偏差，这会导致 FindMatches 转换，有时具有不可接受的准确度。

  准确性衡量转换发现真阳性和真阴性的程度。提高准确性需要更多的机器资源和成本。但这也会导致查全率提高。

  成本衡量运行转换所消耗的计算资源（从而产生成本）的数量。

- EnforceProvidedLabels – 布尔值。

  要启用或禁用的值，以强制输出与用户提供的标签相匹配。如果该值为 True，find matches 转换会强制输出来匹配提供的标注。结果将覆盖正常合并结果。如果值为 False，则 find matches 转换不能确保遵循提供的所有标签，并且结果依赖于训练后的模型。

  请注意，将此值设置为 true 可能会增加合并执行时间。

FindMatchesMetrics 结构

查找匹配算法的评估指标。通过转换预测一些匹配项并将结果与同一数据集中的已知匹配项进行比较来衡量机器学习转换的质量。质量指标基于数据的子集，因此它们并不精确。

字段

- AreaUnderPRCurve – 数字（双数），不超过 1.0。

  精确率与召回率曲线（AUPRC）下的区域是衡量转换整体质量的单一数字，这与精确率与召回率的选择无关。较高的值表明您具有更有吸引力的精确率与召回率权衡。

有关更多信息，请参阅 Wikipedia 中的查准率和查全率。

- `Precision` – 数字（双数），不超过 1.0。

  精确率指标指示转换在预测匹配时正确的频率。具体而言，精确率指标衡量转换从总真阳性可能中找到真阳性的程度。

  有关更多信息，请参阅 Wikipedia 中的查准率和查全率。

- `Recall` – 数字（双数），不超过 1.0。

  召回率指标表示，对于实际匹配，转换预测匹配的频率。具体而言，召回率指标衡量转换从源数据中的总记录中找到真阳性的程度。

  有关更多信息，请参阅 Wikipedia 中的查准率和查全率。

- F1 – 数字（双数），不超过 1.0。

  最大 F1 指标表示转换的准确度介于 0 和 1 之间，其中 1 是最大准确度。

  有关更多信息，请参阅 Wikipedia 中的 F1 分数。

- `ConfusionMatrix` – 一个 ConfusionMatrix 对象。

  混淆矩阵显示转换正在准确预测的内容以及它正在产生的错误类型。

  有关更多信息，请参阅 Wikipedia 中的混淆矩阵。

- `ColumnImportances` – ColumnImportance 对象的数组，不超过 100 个结构。

  `ColumnImportance` 结构列表，其中包含列重要性指标，按重要性降序排序。

ConfusionMatrix 结构

混淆矩阵显示转换正在准确预测的内容以及它正在产生的错误类型。

有关更多信息，请参阅 Wikipedia 中的混淆矩阵。

字段

- `NumTruePositives` – 数字（长型）。

  在转换的混淆矩阵中，转换正确找到的数据中的匹配项数量。

- `NumFalsePositives` – 数字（长型）。

在转换的混淆矩阵中，转换错误地将数据分类为匹配项的的非匹配项数量。

- NumTrueNegatives – 数字（长型）。

  在转换的混淆矩阵中，转换正确拒绝的数据中的非匹配项数量。

- NumFalseNegatives – 数字（长型）。

  在转换的混淆矩阵中，转换未找到的数据中的匹配项数量。

GlueTable 结构

用于输入或输出数据的 AWS Glue Data Catalog 中的数据库和表。

字段

- DatabaseName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  AWS Glue Data Catalog 中的数据库名称。

- TableName – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  AWS Glue Data Catalog 中的表名称。

- CatalogId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  AWS Glue Data Catalog 的唯一标识符。

- ConnectionName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  AWS Glue Data Catalog 中的连接名称。

- AdditionalOptions – 键值对的映射数组，不少于 1 对且不超过 10 对。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值都是一个描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

用于表的其他选项。目前支持两个键：

- pushDownPredicate：筛选分区，而不必列出并读取数据集中的所有文件。
- catalogPartitionPredicate：使用 AWS Glue Data Catalog 中的分区索引来使用服务器端分区修剪。

TaskRun 结构

与机器学习转换关联的采样参数。

字段

- TransformId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  转换的唯一标识符。

- TaskRunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  任务运行的唯一标识符。

- Status – UTF-8 字符串（有效值：STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT）。

  请求任务的当前状态。

- LogGroupName – UTF-8 字符串。

  用于进行安全日志记录的日志组的名称，与此任务运行关联。

- Properties – 一个 TaskRunProperties 对象。

  指定与此任务运行关联的配置属性。

- ErrorString – UTF-8 字符串。

  与此任务运行关联的错误字符串列表。

- StartedOn – 时间戳。

  此任务运行开始的日期和时间。

- LastModifiedOn – 时间戳。

请求任务运行的最后一个时间点已更新。

- CompletedOn – 时间戳。

  请求任务运行的最后一个时间点已完成。

- ExecutionTime – 数字（整数）。

  任务运行使用资源的时间长度（以秒为单位）。

TransformFilterCriteria 结构

筛选机器学习转换的条件。

字段

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  筛选机器学习转换的唯一转换名称。

- TransformType – UTF-8 字符串（有效值：FIND_MATCHES）。

  筛选机器学习转换的机器学习转换类型。

- Status – UTF-8 字符串（有效值：NOT_READY | READY | DELETING）。

  按转换的最后一个已知状态筛选机器学习转换列表（以指示是否可以使用转换）。“NOT_READY”、“READY”或“DELETING”之一。

- GlueVersion – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #20 匹配。

  此值确定与此机器学习转换兼容的 AWS Glue 版本。建议大多数用户使用 Glue 1.0。如果未设置此值，则 Glue 兼容性默认为 Glue 0.9。有关更多信息，请参阅开发人员指南中的 AWS Glue 版本。

- CreatedBefore – 时间戳。

  转换创建前的时间和日期。

- CreatedAfter – 时间戳。

  转换创建后的时间和日期。

- LastModifiedBefore – 时间戳。

筛选此日期之前最后一次修改的转换。

- LastModifiedAfter – 时间戳。

  筛选此日期之后最后一次修改的转换。

- Schema – SchemaColumn 对象的数组，不超过 100 个结构。

  筛选具有特定架构的数据集。Map<Column, Type> 对象是一个键值对数组，表示该转换接受的架构，其中 Column 是列的名称，Type 是数据的类型，如整数或字符串。具有 100 列的上限。

TransformSortCriteria 结构

与机器学习转换关联的排序标准。

字段

- Column – 必填：UTF-8 字符串（有效值：NAME | TRANSFORM_TYPE | STATUS | CREATED | LAST_MODIFIED）。

  在与机器学习转换关联的排序标准中所使用的列。

- SortDirection – 必填：UTF-8 字符串（有效值：DESCENDING | ASCENDING）。

  在与机器学习转换关联的排序标准中所使用的排序方向。

TaskRunFilterCriteria 结构

筛选用于机器学习转换的任务运行的条件。

字段

- TaskRunType – UTF-8 字符串（有效值：EVALUATION | LABELING_SET_GENERATION | IMPORT_LABELS | EXPORT_LABELS | FIND_MATCHES）。

  任务运行的类型。

- Status – UTF-8 字符串（有效值：STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT）。

  任务运行的当前状态。

- StartedBefore – 时间戳。

筛选在此日期之前启动的任务运行。

- StartedAfter – 时间戳。

  筛选在此日期之后启动的任务运行。

TaskRunSortCriteria 结构

对用于机器学习转换的任务运行的列表进行排序的排序条件。

字段

- Column – 必填： UTF-8 字符串（有效值：TASK_RUN_TYPE | STATUS | STARTED）。

  对用于机器学习转换的任务运行的列表进行排序的列。

- SortDirection – 必填：UTF-8 字符串（有效值：DESCENDING | ASCENDING）。

  对用于机器学习转换的任务运行的列表进行排序的排序方向。

TaskRunProperties 结构

任务运行的配置属性。

字段

- TaskType – UTF-8 字符串（有效值：EVALUATION | LABELING_SET_GENERATION | IMPORT_LABELS | EXPORT_LABELS | FIND_MATCHES）。

  任务运行的类型。

- ImportLabelsTaskRunProperties – 一个 ImportLabelsTaskRunProperties 对象。

  导入标签任务运行的配置属性。

- ExportLabelsTaskRunProperties – 一个 ExportLabelsTaskRunProperties 对象。

  导出标签任务运行的配置属性。

- LabelingSetGenerationTaskRunProperties – 一个 LabelingSetGenerationTaskRunProperties 对象。

  标签集生成任务运行的配置属性。

- `FindMatchesTaskRunProperties` – 一个 FindMatchesTaskRunProperties 对象。

  查找匹配项任务运行的配置属性。

FindMatchesTaskRunProperties 结构

指定“查找匹配项”任务运行的配置属性。

字段

- `JobId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  “查找匹配项”任务运行的任务 ID。
- `JobName` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  用于“查找匹配项”任务运行所分配给任务的名称。
- `JobRunId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  “查找匹配项”任务运行的任务运行 ID。

ImportLabelsTaskRunProperties 结构

指定导入标签任务运行的配置属性。

字段

- `InputS3Path` – UTF-8 字符串。

  您将从中导入标签的 Amazon Simple Storage Service（Amazon S3）路径。
- `Replace` – 布尔值。

  指示是否覆盖现有标签。

ExportLabelsTaskRunProperties 结构

指定导出标签任务运行的配置属性。

字段

- OutputS3Path – UTF-8 字符串。

  您将导出标签的 Amazon Simple Storage Service（Amazon S3）路径。

LabelingSetGenerationTaskRunProperties 结构

指定标签集生成任务运行的配置属性。

字段

- OutputS3Path – UTF-8 字符串。

  您将生成标签集的 Amazon Simple Storage Service（Amazon S3）路径。

SchemaColumn 结构

键值对表示该转换可针对的列和数据类型。MLTransform 的 Schema 参数可能包含多达 100 个这些结构。

字段

- Name – UTF-8 字符串，不少于 1 个字节或超过 1024 个字节，与 Single-line string pattern 匹配。

  列的名称。
- DataType – UTF-8 字符串，不超过 131072 个字节，与 Single-line string pattern 匹配。

  列的数据类型。

TransformEncryption 结构

应用于访问用户数据的转换的静态加密设置。机器学习转换可以访问在 Amazon S3 中使用 KMS 加密的用户数据。

此外，导入的标签和经过训练的转换现在可以使用客户提供的 KMS 密钥进行加密。

字段

- MlUserDataEncryption – 一个 MLUserDataEncryption 对象。

包含加密模式和客户提供的 KMS 密钥 ID 的 MLUserDataEncryption 对象。

- TaskRunSecurityConfigurationName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  安全配置的名称。

MLUserDataEncryption 结构

应用于访问用户数据的转换的静态加密设置。

字段

- MlUserDataEncryptionMode – 必填：UTF-8 字符串（有效值：DISABLED | SSE-KMS="SSEKMS"）。

  应用于用户数据的加密模式。有效值为：

  - DISABLED：加密已禁用
  - SSEKMS：将服务器端使用 AWS Key Management Service（SSE-KMS）进行的加密用于存储在 Amazon S3 中的用户数据。
- KmsKeyId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  客户提供的 KMS 密钥的 ID。

ColumnImportance 结构

包含列名称和列重要性分数的结构。

列重要性可识别您的记录中的哪些列比其他列更重要，帮助您了解列如何对模型产生影响。

字段

- ColumnName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  列的名称。
- Importance – 数字（双数），不超过 1.0。

列的列重要性分数（以小数形式）。

操作

- CreateMLTransform 操作（Python：create_ml_transform）
- UpdateMLTransform 操作（Python：update_ml_transform）
- DeleteMLTransform 操作（Python：delete_ml_transform）
- GetMLTransform 操作（Python：get_ml_transform）
- GetMLTransforms 操作（Python：get_ml_ransforms）
- ListMLTransforms 操作（Python：list_ml_transforms）
- StartMLEvaluationTaskRun 操作（Python：start_ml_evaluation_task_run）
- StartMLLabelingSetGenerationTaskRun 操作（Python：start_ml_labeling_set_generation_task_run）
- GetMLTaskRun 操作（Python：get_ml_task_run）
- GetMLTaskRuns 操作（Python：get_ml_task_runs）
- CancelMLTaskRun 操作（Python：cancel_ml_task_run）
- StartExportLabelsTaskRun 操作（Python：start_export_labels_task_run）
- StartImportLabelsTaskRun 操作（Python：start_import_labels_task_run）

CreateMLTransform 操作（Python：create_ml_transform）

创建 AWS Glue 机器学习转换。此操作将创建转换以及训练它所有所需的参数。

调用此操作作为使用机器学习转换过程中的第一步（例如 `FindMatches` 转换）来消除重复数据。您可以提供一个可选的 `Description`，以及要用于算法的参数。

您还必须为 AWS Glue 代表您运行的任务指定特定参数，作为从数据中学习和创建高质量机器学习转换的一部分。这些参数包括 `Role`，并且可以选择 `AllocatedCapacity`、`Timeout` 和 `MaxRetries`。有关更多信息，请参阅任务。

请求

- `Name` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

在创建转换时为其提供的唯一名称。

- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  正在定义的机器学习转换的描述。默认值是空字符串。

- `InputRecordTables` – 必填：GlueTable 对象的数组，不超过 10 个结构。

  转换使用的 AWS Glue 表定义的列表。

- `Parameters` – 必填：一个 TransformParameters 对象。

  特定于所使用转换类型的算法参数。有条件地依赖于转换类型。

- `Role` – 必填：UTF-8 字符串。

  具有所需权限的 IAM 角色的名称或 Amazon Resource Name（ARN）。所需权限包括 AWS Glue 资源的 AWS Glue 服务角色权限和转换所需的 Amazon S3 权限。

  - 此角色需要 AWS Glue 服务角色权限才能允许访问 AWS Glue 中的资源。请参阅将策略附加到访问 AWS Glue 的 IAM 用户。
  - 此角色需要对您的 Amazon Simple Storage Service (Amazon S3) 源、目标、临时目录、脚本以及此转换的任务运行所使用的任何库的权限。

- `GlueVersion` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #20 匹配。

  此值确定与此机器学习转换兼容的 AWS Glue 版本。建议大多数用户使用 Glue 1.0。如果未设置此值，则 Glue 兼容性默认为 Glue 0.9。有关更多信息，请参阅开发人员指南中的 AWS Glue 版本。

- `MaxCapacity` – 数字（double）。

  分配给此转换的任务运行的 AWS Glue 数据处理单元（DPU）的数量。您可以分配 2 到 100 个 DPU；默认值为 10。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅 AWS Glue 价格页面。

  `MaxCapacity` 是具有 `NumberOfWorkers` 和 `WorkerType` 的互斥选项。

  - 如果已设置 `NumberOfWorkers` 或 `WorkerType`，则不能设置 `MaxCapacity`。
  - 如果已设置 `MaxCapacity`，则不能设置 `NumberOfWorkers` 或 `WorkerType`。
  - 如果已设置 `WorkerType`，则 `NumberOfWorkers` 为必填项（反之亦然）。
  - `MaxCapacity` 和 `NumberOfWorkers` 都必须至少为 1。

在将 WorkerType 字段设置为 Standard 之外的值时，MaxCapacity 字段将自动设置并变为只读。

在将 WorkerType 字段设置为 Standard 之外的值时，MaxCapacity 字段将自动设置并变为只读。

- WorkerType – UTF-8 字符串（有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""）。

  在此任务运行时分配的预定义工作线程的类型。接受的值为 Standard、G.1X 或 G.2X。

  - 对于 Standard 工作线程类型，每个工作线程提供 4 个 vCPU、16 GB 内存和 50GB 磁盘，并且每个工作线程提供 2 个执行器。
  - 对于 G.1X 工作线程类型，每个工作线程提供 4 个 vCPU、16 GB 内存和 64GB 磁盘，并且每个工作线程提供 1 个执行器。
  - 对于 G.2X 工作线程类型，每个工作线程提供 84 个 vCPU、32 GB 内存和 128GB 磁盘，并且每个工作线程提供 1 个执行器。

  MaxCapacity 是具有 NumberOfWorkers 和 WorkerType 的互斥选项。

  - 如果已设置 NumberOfWorkers 或 WorkerType，则不能设置 MaxCapacity。
  - 如果已设置 MaxCapacity，则不能设置 NumberOfWorkers 或 WorkerType。
  - 如果已设置 WorkerType，则 NumberOfWorkers 为必填项（反之亦然）。
  - MaxCapacity 和 NumberOfWorkers 都必须至少为 1。
- NumberOfWorkers – 数字（整数）。

  任务运行时分配的已定义 workerType 的工作线程数。

  如果已设置 WorkerType，则 NumberOfWorkers 为必填项（反之亦然）。
- Timeout - 数字（整数），至少为 1。

  此转换的任务运行超时时间（以分钟为单位）。这是此转换的任务运行在终止并进入 TIMEOUT 状态前可以使用资源的最长时间。默认值为 2880 分钟（48 小时）。
- MaxRetries – 数字（整数）。

  在任务运行失败后重试此转换的任务的最大次数。
- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

每个值是一个 UTF-8 字符串，不超过 256 个字节。

用于此机器学习转换的标签。您可以使用标签来限制对机器学习转换的访问权限。有关 AWS Glue 中的标签的更多信息，请参阅开发人员指南中的 AWS Glue 中的 AWS 标签。

- `TransformEncryption` – 一个 TransformEncryption 对象。

  应用于访问用户数据的转换的静态加密设置。机器学习转换可以访问在 Amazon S3 中使用 KMS 加密的用户数据。

响应

- `TransformId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  为转换生成的唯一标识符。

错误

- `AlreadyExistsException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `AccessDeniedException`
- `ResourceNumberLimitExceededException`
- `IdempotentParameterMismatchException`

UpdateMLTransform 操作（Python：update_ml_transform）

更新现有的机器学习转换。调用此操作可优化算法参数来获得更好的结果。

调用此操作后，您可以调用 `StartMLEvaluationTaskRun` 操作来评估新参数实现目标的程度（例如提高机器学习转换的质量，或使其更具成本效益）。

请求

- `TransformId` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  在创建转换时为其生成的唯一标识符。

- `Name` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  在创建转换时为其提供的唯一名称。

- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  对转换的说明。默认值是空字符串。

- `Parameters` – 一个 TransformParameters 对象。

  特定于所使用转换类型（算法）的配置参数。有条件地依赖于转换类型。

- `Role` – UTF-8 字符串。

  具有所需权限的 IAM 角色的名称或 Amazon Resource Name（ARN）。

- `GlueVersion` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #20 匹配。

  此值确定与此机器学习转换兼容的 AWS Glue 版本。建议大多数用户使用 Glue 1.0。如果未设置此值，则 Glue 兼容性默认为 Glue 0.9。有关更多信息，请参阅开发人员指南中的 AWS Glue 版本。

- `MaxCapacity` – 数字（double）。

  分配给此转换的任务运行的 AWS Glue 数据处理单元（DPU）的数量。您可以分配 2 到 100 个 DPU；默认值为 10。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅 AWS Glue 价格页面。

  在将 `WorkerType` 字段设置为 `Standard` 之外的值时，`MaxCapacity` 字段将自动设置并变为只读。

- `WorkerType` – UTF-8 字符串（有效值：`Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""`）。

  在此任务运行时分配的预定义工作线程的类型。接受的值为 Standard、G.1X 或 G.2X。

  - 对于 Standard 工作线程类型，每个工作线程提供 4 个 vCPU、16 GB 内存和 50GB 磁盘，并且每个工作线程提供 2 个执行器。
  - 对于 G.1X 工作线程类型，每个工作线程提供 4 个 vCPU、16 GB 内存和 64GB 磁盘，并且每个工作线程提供 1 个执行器。
  - 对于 G.2X 工作线程类型，每个工作线程提供 84 个 vCPU、32 GB 内存和 128GB 磁盘，并且每个工作线程提供 1 个执行器。
- NumberOfWorkers – 数字（整数）。

  任务运行时分配的已定义 workerType 的工作线程数。
- Timeout - 数字（整数），至少为 1。

  转换的任务运行超时时间（以分钟为单位）。这是此转换的任务运行在终止并进入 TIMEOUT 状态前可以使用资源的最长时间。默认值为 2880 分钟（48 小时）。
- MaxRetries – 数字（整数）。

  在任务运行失败后重试此转换的任务的最大次数。

响应

- TransformId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  已更新的转换的唯一标识符。

错误

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- AccessDeniedException

DeleteMLTransform 操作（Python：delete_ml_transform）

删除 AWS Glue 机器学习转换。机器学习转换是一种特殊类型的转换，它通过从人类提供的示例中学习，使用机器学习来了解要执行转换的详细信息。然后，这些转换将通过 AWS Glue 保存。如果您不再需要转换，可以通过调用 `DeleteMLTransforms` 来删除它。但是，仍然引用已删除转换的任何 AWS Glue 任务将不再成功运行。

请求

- `TransformId` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除转换的唯一标识符。

响应

- `TransformId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  已删除转换的唯一标识符。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

GetMLTransform 操作（Python：get_ml_transform）

获取 AWS Glue 机器学习转换构件及其所有相应的元数据。机器学习转换是一种特殊类型的转换，它通过从人类提供的示例中学习，使用机器学习来了解要执行转换的详细信息。然后，这些转换将通过 AWS Glue 保存。您可以通过调用 `GetMLTransform` 来检索其元数据。

请求

- `TransformId` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

转换的唯一标识符，在创建转换时生成。

响应

- `TransformId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  转换的唯一标识符，在创建转换时生成。
- `Name` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  在创建转换时为其提供的唯一名称。
- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  对转换的说明。
- `Status` – UTF-8 字符串（有效值：`NOT_READY` | `READY` | `DELETING`）。

  转换的最后一个已知状态（指示是否可以使用）。“NOT_READY”、“READY”或“DELETING”之一。
- `CreatedOn` – 时间戳。

  转换的创建日期和时间。
- `LastModifiedOn` – 时间戳。

  转换的上次修改日期和时间。
- `InputRecordTables` – GlueTable 对象的数组，不超过 10 个结构。

  转换使用的 AWS Glue 表定义的列表。
- `Parameters` – 一个 TransformParameters 对象。

  特定于所使用算法的配置参数。
- `EvaluationMetrics` – 一个 EvaluationMetrics 对象。

  最新的评估指标。
- `LabelCount` – 数字（整数）。

  可用于此转换的标签数量。

- Schema – SchemaColumn 对象的数组，不超过 100 个结构。

  Map<Column, Type> 对象，该对象表示此转换接受的架构。具有 100 列的上限。

- Role – UTF-8 字符串。

  具有所需权限的 IAM 角色的名称或 Amazon Resource Name（ARN）。

- GlueVersion – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Custom string pattern #20 匹配。

  此值确定与此机器学习转换兼容的 AWS Glue 版本。建议大多数用户使用 Glue 1.0。如果未设置此值，则 Glue 兼容性默认为 Glue 0.9。有关更多信息，请参阅开发人员指南中的 AWS Glue 版本。

- MaxCapacity – 数字（double）。

  分配给此转换的任务运行的 AWS Glue 数据处理单元（DPU）的数量。您可以分配 2 到 100 个 DPU；默认值为 10。DPU 是对处理能力的相对度量，它由 4 个 vCPU 的计算容量和 16GB 内存组成。有关更多信息，请参阅 AWS Glue 价格页面。

  在将 WorkerType 字段设置为 Standard 之外的值时，MaxCapacity 字段将自动设置并变为只读。

- WorkerType – UTF-8 字符串（有效值：Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""）。

  在此任务运行时分配的预定义工作线程的类型。接受的值为 Standard、G.1X 或 G.2X。

  - 对于 Standard 工作线程类型，每个工作线程提供 4 个 vCPU、16 GB 内存和 50GB 磁盘，并且每个工作线程提供 2 个执行器。
  - 对于 G.1X 工作线程类型，每个工作线程提供 4 个 vCPU、16 GB 内存和 64GB 磁盘，并且每个工作线程提供 1 个执行器。
  - 对于 G.2X 工作线程类型，每个工作线程提供 84 个 vCPU、32 GB 内存和 128GB 磁盘，并且每个工作线程提供 1 个执行器。

- NumberOfWorkers – 数字（整数）。

  任务运行时分配的已定义 workerType 的工作线程数。

- Timeout - 数字（整数），至少为 1。

  转换的任务运行超时时间（以分钟为单位）。这是此转换的任务运行在终止并进入 TIMEOUT 状态前可以使用资源的最长时间。默认值为 2880 分钟（48 小时）。

- MaxRetries – 数字（整数）。

在任务运行失败后重试此转换的任务的最大次数。

- `TransformEncryption` – 一个 TransformEncryption 对象。

  应用于访问用户数据的转换的静态加密设置。机器学习转换可以访问在 Amazon S3 中使用 KMS 加密的用户数据。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

GetMLTransforms 操作（Python：get_ml_ransforms）

获取一个可排序、可筛选的现有 AWS Glue 机器学习转换列表。机器学习转换是一种特殊类型的转换，它通过从人类提供的示例中学习，使用机器学习来了解要执行转换的详细信息。然后，这些转换将通过 AWS Glue 保存，您还可以通过调用 `GetMLTransforms` 来检索其元数据。

请求

- `NextToken` – UTF-8 字符串。

  用于偏移结果的分页令牌。

- `MaxResults` – 数字（整数），不小于 1 或大于 1000。

  要返回的最大结果数量。

- `Filter` – 一个 TransformFilterCriteria 对象。

  转换筛选条件。

- `Sort` – 一个 TransformSortCriteria 对象。

  排序标准。

响应

- `Transforms` – 必填：MLTransform 对象的数组。

机器学习转换的列表。

- NextToken – UTF-8 字符串。

  分页令牌（如果有更多结果可用）。

错误

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

ListMLTransforms 操作（Python：list_ml_transforms）

在此 AWS 账户中检索可排序、可筛选的现有 AWS Glue 机器学习转换列表或带指定标签的资源。此操作采用可选的 Tags 字段，您可以将其用作响应的筛选条件，以便将标记的资源作为一个组进行检索。如果您选择使用标签筛选，则仅检索带标签的资源。

请求

- NextToken – UTF-8 字符串。

  延续令牌 (如果这是延续请求)。

- MaxResults – 数字（整数），不小于 1 或大于 1000。

  要返回的列表的最大大小。

- Filter – 一个 TransformFilterCriteria 对象。

  用于机器学习转换筛选的 TransformFilterCriteria。

- Sort – 一个 TransformSortCriteria 对象。

  用于机器学习转换排序的 TransformSortCriteria。

- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

指定仅返回这些已标记的资源。

响应

- TransformIds – 必填：UTF-8 字符串数组。

  所有机器学习的标识符都在账户中进行转换，或者机器学习将使用指定的标签进行转换。

- NextToken – UTF-8 字符串。

  延续令牌 (如果返回的列表不包含上一个可用的指标)。

错误

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

StartMLEvaluationTaskRun 操作（Python：start_ml_evaluation_task_run）

开始一项任务来估计转换的质量。

当您提供标签集作为真实示例时，AWS Glue 机器学习使用其中的一些例子来学习。其余的标签用作测试，以估计质量。

返回运行的唯一标识符。您可以调用 GetMLTaskRun 来获取更多有关 EvaluationTaskRun 的统计信息。

请求

- TransformId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  机器学习转换的唯一标识符。

响应

- TaskRunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  与此运行关联的唯一标识符。

错误

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ConcurrentRunsExceededException
- MLTransformNotReadyException

StartMLLabelingSetGenerationTaskRun 操作（Python：start_ml_labeling_set_generation_task_run）

启动机器学习转换的主动学习工作流，通过生成标签集和添加标签来提高转换的质量。

当 StartMLLabelingSetGenerationTaskRun 完成后，AWS Glue 将产生一个“标签集”或一组供人类回答的问题。

如果是 FindMatches 转换，这些问题的形式是“将这些行组合到完全由匹配记录组成的组中的正确方法是什么？”

标记过程完成后，您可以通过调用 StartImportLabelsTaskRun 来上载您的标签。StartImportLabelsTaskRun 完成后，机器学习转换的所有未来运行都将使用新的和改进的标签，并执行更高质量的转换。

请求

- TransformId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  机器学习转换的唯一标识符。

- `OutputS3Path` – 必填：UTF-8 字符串。

  生成标签集的 Amazon Simple Storage Service（Amazon S3）路径。

响应

- `TaskRunId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  与此任务运行关联的唯一运行标识符。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `ConcurrentRunsExceededException`

GetMLTaskRun 操作（Python：get_ml_task_run）

获取在机器学习转换上特定任务运行的详细信息。机器学习任务运行是 AWS Glue 作为各种机器学习工作流的一部分代表您运行的异步任务。您可以通过调用使用 `TaskRunID` 及其父级转换的 `TransformID` 的 `GetMLTaskRun` 来检查任何任务运行的统计信息。

请求

- `TransformId` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  机器学习转换的唯一标识符。

- `TaskRunId` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  任务运行的唯一标识符。

响应

- TransformId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  任务运行的唯一标识符。

- TaskRunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  与此运行关联的唯一运行标识符。

- Status – UTF-8 字符串（有效值：STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT）。

  此任务运行的运行状态。

- LogGroupName – UTF-8 字符串。

  与任务运行关联的日志组的名称。

- Properties – 一个 TaskRunProperties 对象。

  与任务运行关联的属性列表。

- ErrorString – UTF-8 字符串。

  与任务运行关联的错误字符串。

- StartedOn – 时间戳。

  此任务运行开始的日期和时间。

- LastModifiedOn – 时间戳。

  此任务运行的上次修改日期和时间。

- CompletedOn – 时间戳。

  此任务运行的完成日期和时间。

- ExecutionTime – 数字（整数）。

  任务运行使用资源的时间长度（以秒为单位）。

错误

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

GetMLTaskRuns 操作（Python：get_ml_task_runs）

获取机器学习转换的运行列表。机器学习任务运行是 AWS Glue 作为各种机器学习工作流的一部分代表您运行的异步任务。您可以通过调用使用与其父级转换的 TransformID 的 GetMLTaskRuns 和本节中介绍的其他可选参数来获取可排序、可筛选的机器学习任务运行列表。

此操作返回历史运行列表，必须进行分页。

请求

- TransformId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  机器学习转换的唯一标识符。
- NextToken – UTF-8 字符串。

  用于分页结果的令牌。默认值为空。
- MaxResults – 数字（整数），不小于 1 或大于 1000。

  要返回的最大结果数量。
- Filter – 一个 TaskRunFilterCriteria 对象。

  筛选标准，位于 TaskRunFilterCriteria 结构，用于任务运行。
- Sort – 一个 TaskRunSortCriteria 对象。

  排序标准，位于 TaskRunSortCriteria 结构，用于任务运行。

响应

- TaskRuns – TaskRun 对象的数组。

与转换相关联的任务运行列表。

- NextToken – UTF-8 字符串。

  分页令牌（如果有更多结果可用）。

错误

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

CancelMLTaskRun 操作（Python：cancel_ml_task_run）

取消（停止）任务运行。机器学习任务运行是 AWS Glue 作为各种机器学习工作流的一部分代表您运行的异步任务。您可以随时取消机器学习任务运行，方法是调用使用任务运行父级转换的 TransformID 和任务运行的 TaskRunId 的 CancelMLTaskRun。

请求

- TransformId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  机器学习转换的唯一标识符。

- TaskRunId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  任务运行的唯一标识符。

响应

- TransformId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  机器学习转换的唯一标识符。

- TaskRunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  任务运行的唯一标识符。

- Status – UTF-8 字符串（有效值：STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT）。

  此运行的运行状态。

错误

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

StartExportLabelsTaskRun 操作（Python：start_export_labels_task_run）

开始异步任务以导出特定转换的所有标记数据。此任务是唯一不属于典型主动学习工作流的标签相关的 API 调用。当您希望同时处理所有现有标签时，例如，当您想要删除或更改以前作为真实提交的标签时，您通常使用 StartExportLabelsTaskRun。此 API 操作接受要导出其标签的 TransformId，以及将标签导出的 Amazon Simple Storage Service（Amazon S3）路径。操作返回 TaskRunId。您可以通过调用 GetMLTaskRun API 来检查任务运行状态。

请求

- TransformId – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  机器学习转换的唯一标识符。

- OutputS3Path – 必填：UTF-8 字符串。

  您导出标签的 Amazon S3 路径。

响应

- TaskRunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  任务运行的唯一标识符。

错误

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

StartImportLabelsTaskRun 操作（Python：start_import_labels_task_run）

使您能够提供其他标签（真相示例），用于教授机器学习转换并提高其质量。此 API 操作通常用作主动学习工作流的一部分，其中该工作流于 StartMLLabelingSetGenerationTaskRun 调用，并最终提高您的机器学习转换的质量。

StartMLLabelingSetGenerationTaskRun 完成后，AWS Glue 机器学习将产生一系列供人类回答的问题。（回答这些问题通常称为机器学习工作流中的“标签”）。如果是 FindMatches 转换，这些问题的形式是“将这些行组合到完全由匹配记录组成的组中的正确方法是什么？”标签过程完成后，用户会通过调用 StartImportLabelsTaskRun 来上载他们的答案/标签。StartImportLabelsTaskRun 完成后，机器学习转换的所有未来运行都使用新的和改进的标签，并执行更高质量的转换。

默认情况下，StartMLLabelingSetGenerationTaskRun 会不断地从您上载的所有标签中学习和合并所有标签，除非您将 Replace 设置为 True。如果您将 Replace 设置为 true，StartImportLabelsTaskRun 会删除和忘记所有先前上载的标签，并仅从您上载的确切标签集中获取信息。如果您意识到以前上载了不正确的标签，并且您认为它们对转换质量产生负面影响，则替换标签会很有帮助。

您可以通过调用 GetMLTaskRun 操作来检查任务运行状态。

请求

- `TransformId` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  机器学习转换的唯一标识符。
- `InputS3Path` – 必填：UTF-8 字符串。

  导入标签的 Amazon Simple Storage Service（Amazon S3）路径。
- `ReplaceAllLabels` – 布尔值。

  指示是否覆盖现有标签。

响应

- `TaskRunId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  任务运行的唯一标识符。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`

数据质量 API

数据质量 API 描述了数据质量数据类型，并包括用于创建、删除或更新数据质量规则集、运行和评估的 API。

数据类型

- DataSource 结构
- DataQualityRulesetListDetails 结构

- DataQualityTargetTable 结构
- DataQualityRulesetEvaluationRunDescription 结构
- DataQualityRulesetEvaluationRunFilter 结构
- DataQualityEvaluationRunAdditionalRunOptions 结构
- DataQualityRuleRecommendationRunDescription 结构
- DataQualityRuleRecommendationRunFilter 结构
- DataQualityResult 结构
- DataQualityAnalyzerResult 结构
- DataQualityObservation 结构
- MetricBasedObservation 结构
- DataQualityMetricValues 结构
- DataQualityRuleResult 结构
- DataQualityResultDescription 结构
- DataQualityResultFilterCriteria 结构
- DataQualityRulesetFilterCriteria 结构

DataSource 结构

要获得数据质量结果的数据源（ AWS Glue 表）。

字段

- `GlueTable` – 必填：一个 GlueTable 对象。

  一张 AWS Glue 桌子。

DataQualityRulesetListDetails 结构

描述 `GetDataQualityRuleset` 返回的数据质量规则集。

字段

- `Name` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  数据质量规则集的名称。

- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  数据质量规则集的描述。

- `CreatedOn` – 时间戳。

  创建数据质量规则集的日期和时间。

- `LastModifiedOn` – 时间戳。

  上次修改数据质量规则集的日期和时间。

- `TargetTable` – 一个 DataQualityTargetTable 对象。

  一个表示 AWS Glue 表格的对象。

- `RecommendationRunId` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  当根据建议运行创建规则集时，会生成此运行 ID 以将两者联系在一起。

- `RuleCount` – 数字（整数）。

  规则集中的规则数量。

DataQualityTargetTable 结构

一个表示 AWS Glue 表格的对象。

字段

- `TableName` – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  AWS Glue 表的名称。

- `DatabaseName` – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  该 AWS Glue 表所在的数据库的名称。

- `CatalogId` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  表所在的 AWS Glue 目录 ID。

DataQualityRulesetEvaluationRunDescription 结构

描述数据质量规则集评估运行的结果。

字段

- `RunId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  与此运行关联的唯一运行标识符。

- `Status` – UTF-8 字符串（有效值：`STARTING` | `RUNNING` | `STOPPING` | `STOPPED` | `SUCCEEDED` | `FAILED` | `TIMEOUT`）。

  此运行的运行状态。

- `StartedOn` – 时间戳。

  启动运行的日期和时间。

- `DataSource` – 一个 DataSource 对象。

  与运行相关的数据源（ AWS Glue 表）。

DataQualityRulesetEvaluationRunFilter 结构

筛选条件。

字段

- `DataSource` – 必填：一个 DataSource 对象。

  根据与运行关联的数据源（ AWS Glue 表）进行筛选。

- `StartedBefore` – 时间戳。

  按在此时间之前开始的运行筛选结果。

- `StartedAfter` – 时间戳。

  按在此时间之后开始的运行筛选结果。

DataQualityEvaluationRunAdditionalRunOptions 结构

您可以为评估运行指定的其他运行选项。

字段

- `CloudWatchMetricsEnabled` – 布尔值。

  是否启用 CloudWatch 指标。

- `ResultsS3Prefix` – UTF-8 字符串。

  用于存储结果的 Amazon S3 的前缀。

- `CompositeRuleEvaluationMethod` – UTF-8 字符串（有效值：`COLUMN` | `ROW`）。

  将规则集中复合规则的评估方法设置为 ROW/COLUMN

DataQualityRuleRecommendationRunDescription 结构

描述数据质量规则建议运行的结果。

字段

- `RunId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  与此运行关联的唯一运行标识符。

- `Status` – UTF-8 字符串（有效值：`STARTING` | `RUNNING` | `STOPPING` | `STOPPED` | `SUCCEEDED` | `FAILED` | `TIMEOUT`）。

  此运行的运行状态。

- `StartedOn` – 时间戳。

  此运行开始的日期和时间。

- `DataSource` – 一个 DataSource 对象。

  与建议运行关联的数据源（AWS Glue 表）。

DataQualityRuleRecommendationRunFilter 结构

用于列出数据质量建议运行的筛选器。

字段

- DataSource – 必填：一个 DataSource 对象。

  根据指定的数据源（AWS Glue 表）进行筛选。

- StartedBefore – 时间戳。

  根据时间进行筛选，以查看在提供的时间之前开始的结果。

- StartedAfter – 时间戳。

  根据时间进行筛选，以查看在提供的时间之后开始的结果。

DataQualityResult 结构

描述数据质量结果。

字段

- ResultId – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  数据质量结果的唯一结果 ID。

- Score – 数字（双数），不超过 1.0。

  汇总的数据质量分数。表示规则与传递到规则总数的比率。

- DataSource – 一个 DataSource 对象。

  与数据质量结果相关的表（如果有）。

- RulesetName – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与数据质量结果相关的规则集的名称。

- EvaluationContext – UTF-8 字符串。

在 AWS Glue Studio 的作业环境中，通常会为画布中的每个节点分配某种名称，并且数据质量节点会有名称。如果有多个节点，则 evaluationContext 可以区分这些节点。

- StartedOn – 时间戳。

  此数据质量运行开始的日期和时间。

- CompletedOn – 时间戳。

  此数据质量运行完成的日期和时间。

- JobName – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与数据质量结果相关的作业名称（如果有）。

- JobRunId – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与数据质量结果相关的作业运行 ID（如果有）。

- RulesetEvaluationRunId – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  此数据质量结果的规则集评估的唯一运行 ID。

- RuleResults – 一组 DataQualityRuleResult 对象，不超过 2000 个结构。

  代表每条规则结果的 DataQualityRuleResult 对象列表。

- AnalyzerResults – 一组 DataQualityAnalyzerResult 对象，不超过 2000 个结构。

  代表每个分析器结果的 DataQualityAnalyzerResult 对象列表。

- Observations – DataQualityObservation 对象的数组，不超过 50 个结构。

  代表评估规则和分析器后生成的观测值的 DataQualityObservation 对象列表。

DataQualityAnalyzerResult 结构

描述数据质量分析器评估的结果。

字段

- Name – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  数据质量分析器的名称。

- Description – UTF-8 字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  数据质量分析器的描述。

- EvaluationMessage – UTF-8 字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  评估消息。

- EvaluatedMetrics – 键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值都是数字（双精度数）。

  与分析器评估相关的指标地图。

DataQualityObservation 结构

描述评估规则和分析器后生成的观测。

字段

- Description – UTF-8 字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  数据质量观测的描述。

- MetricBasedObservation – 一个 MetricBasedObservation 对象。

  表示基于所评估数据质量指标的观测的 MetricBasedObservation 类型对象。

MetricBasedObservation 结构

描述基于所评估数据质量指标生成的基于指标的观测。

字段

- `MetricName` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  用于生成观测的数据质量指标的名称。
- `MetricValues` – 一个 DataQualityMetricValues 对象。

  表示数据质量指标值分析的 `DataQualityMetricValues` 类型对象。
- `NewRules` – UTF-8 字符串数组。

  根据数据质量指标值，作为观测结果的一部分生成的新数据质量规则列表。

DataQualityMetricValues 结构

根据对历史数据的分析，描述数据质量指标值。

字段

- `ActualValue` – 数字（double）。

  数据质量指标的实际值。
- `ExpectedValue` – 数字（double）。

  根据对历史数据的分析，数据质量指标的预期值。
- `LowerLimit` – 数字（double）。

  根据对历史数据的分析，数据质量指标的下限值。
- `UpperLimit` – 数字（double）。

  根据对历史数据的分析，数据质量指标的上限值。

DataQualityRuleResult 结构

描述数据质量规则评估运行的结果。

字段

- Name – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  数据质量规则的名称。

- Description – UTF-8 字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  数据质量规则的描述。

- EvaluationMessage – UTF-8 字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  评估消息。

- Result – UTF-8 字符串（有效值：PASS | FAIL | ERROR）。

  规则的通过或失败状态。

- EvaluatedMetrics – 键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值都是数字（双精度数）。

  与规则评估相关的指标地图。

DataQualityResultDescription 结构

描述数据质量结果。

字段

- ResultId – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  此数据质量结果的唯一结果 ID。

- DataSource – 一个 DataSource 对象。

  与数据质量结果相关的表。

- JobName – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与数据质量结果相关的作业名称。

- JobRunId – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与数据质量结果相关的作业运行 ID。

- StartedOn – 时间戳。

  此数据质量结果的运行开始的时间。

DataQualityResultFilterCriteria 结构

用于返回数据质量结果的标准。

字段

- DataSource – 一个 DataSource 对象。

  按指定的数据源筛选结果。例如，检索 AWS Glue 表的所有结果。

- JobName – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  按指定的任务名称筛选结果。

- JobRunId – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  按指定的任务运行 ID 筛选结果。

- StartedAfter – 时间戳。

  按在此时间之后开始的运行筛选结果。

- StartedBefore – 时间戳。

  按在此时间之前开始的运行筛选结果。

DataQualityRulesetFilterCriteria 结构

用于筛选数据质量规则集的条件。

字段

- Name – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  规则集筛选条件的名称。

- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  规则集筛选条件的描述。

- CreatedBefore – 时间戳。

  筛选在此日期之前创建的规则集。

- CreatedAfter – 时间戳。

  筛选在此日期之后创建的规则集。

- LastModifiedBefore – 时间戳。

  筛选此日期之前最后一次修改的规则集。

- LastModifiedAfter – 时间戳。

  筛选此日期之后最后一次修改的规则集。

- TargetTable – 一个 DataQualityTargetTable 对象。

  目标表的名称和数据库名称。

操作

- StartDataQualityRulesetEvaluationRun 操作（Python：start_data_quality_ruleset_evaluation_run）
- CancelDataQualityRulesetEvaluationRun 操作（Python：取消数据质量规则集评估_运行）
- GetDataQualityRulesetEvaluationRun 操作（Python：get_data_quality_ruleset_requalition_run）
- ListDataQualityRulesetEvaluationRuns 操作（Python：list_data_quality_ruleset_requalition_runs）

- StartDataQualityRuleRecommendationRun 动作（Python：start_data_quality_rule_rule_remuldation_
- CancelDataQualityRuleRecommendationRun 操作（Python：取消数据质量规则_推荐_运行）
- GetDataQualityRuleRecommendationRun action（Python：获取数据质量规则_推荐_运行）
- ListDataQualityRuleRecommendationRuns action（Python：list_data_quality_rule_rule_runs）
- GetDataQualityResult 操作（Python：获取数据质量结果）
- BatchGetDataQualityResult 操作（Python：batch_get_data_quality_result）
- ListDataQualityResults 操作（Python：列表数据质量结果）
- CreateDataQualityRuleset 操作（Python：create_data_quality_ruleset）
- DeleteDataQualityRuleset 操作（Python：delete_data_quality_ruleset）
- GetDataQualityRuleset 操作（Python：get_data_quality_ruleset）
- ListDataQualityRulesets 操作（Python：列表数据质量规则集）
- UpdateDataQualityRuleset 操作（Python：更新数据质量规则集）

StartDataQualityRulesetEvaluationRun 操作（Python：start_data_quality_ruleset_evaluation_run）

有了规则集定义（建议定义或您自己的规则）后，就可以调用此操作来根据数据源（表）评估规则集。AWS Glue 评估会计算出您可以使用 `GetDataQualityResult` API 检索的结果。

请求

- `DataSource` – 必填：一个 DataSource 对象。

  与此次运行相关的数据源（AWS Glue 表）。
- `Role` – 必填：UTF-8 字符串。

  为加密运行结果而提供的 IAM 角色。
- `NumberOfWorkers` – 数字（整数）。

  运行所用的 `G.1X` 工作线程数量。默认值为 5。
- `Timeout` - 数字（整数），至少为 1。

  运行超时（分钟）。这是任务运行在终止并进入 `TIMEOUT` 状态前可以使用资源的最长时间。默认值为 2880 分钟（48 小时）。

- `ClientToken` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  用于幂等性，建议设置为随机 ID（例如，UUID），以避免创建或启动同一资源的多个实例。

- `AdditionalRunOptions` – 一个 DataQualityEvaluationRunAdditionalRunOptions 对象。

  您可以为评估运行指定的其他运行选项。

- `RulesetNames` — 必填：UTF-8 字符串数组，不少于 1 个或不超过 10 个字符串。

  规则集名称的列表。

- `AdditionalDataSources` – 键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值都是一个 DataSource 对象。

  您可以为评估运行指定的其他数据来源的引用字符串地图。

响应

- `RunId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  与此运行关联的唯一运行标识符。

错误

- `InvalidInputException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `InternalServiceException`
- `ConflictException`

CancelDataQualityRulesetEvaluationRun 操作（Python：取消数据质量规则集评估_运行）

取消正在根据数据源评估规则集的运行。

请求

- RunId – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与此运行关联的唯一运行标识符。

响应

- 无响应参数。

错误

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

GetDataQualityRulesetEvaluationRun 操作（Python：get_data_quality_ruleset_requalition_run）

检索正在根据数据源评估规则集的特定运行。

请求

- RunId – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与此运行关联的唯一运行标识符。

响应

- RunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  与此运行关联的唯一运行标识符。

- DataSource – 一个 DataSource 对象。

  与此评估运行相关的数据源（ AWS Glue 表）。

- Role – UTF-8 字符串。

  为加密运行结果而提供的 IAM 角色。

- NumberOfWorkers – 数字（整数）。

  运行所用的 G.1X 工作线程数量。默认值为 5。

- Timeout - 数字（整数），至少为 1。

  运行超时（分钟）。这是任务运行在终止并进入 TIMEOUT 状态前可以使用资源的最长时间。默认值为 2880 分钟（48 小时）。

- AdditionalRunOptions – 一个 DataQualityEvaluationRunAdditionalRunOptions 对象。

  您可以为评估运行指定的其他运行选项。

- Status – UTF-8 字符串（有效值：STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT）。

  此运行的运行状态。

- ErrorString – UTF-8 字符串。

  与运行关联的错误字符串。

- StartedOn – 时间戳。

  此运行开始的日期和时间。

- LastModifiedOn – 时间戳。

  时间戳。此数据质量规则建议运行最后一个修改时间点。

- CompletedOn – 时间戳。

  此运行的完成日期和时间。

- ExecutionTime – 数字（整数）。

  运行使用资源的时间长度（以秒为单位）。
- RulesetNames – UTF-8 字符串数组，不少于 1 个字符串，不超过 10 个字符串。

  运行的规则集名称列表。目前，此参数仅采用一个规则集名称。
- ResultIds — UTF-8 字符串数组，不少于 1 个字符串，不超过 10 个字符串。

  运行数据质量结果的结果 ID 列表。
- AdditionalDataSources – 键值对的映射数组。

  每个键是一个 UTF-8 字符串，不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  每个值都是一个 DataSource 对象。

  您可以为评估运行指定的其他数据来源的引用字符串地图。

错误

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

ListDataQualityRulesetEvaluationRuns 操作（Python：list_data_quality_ruleset_requalition_runs）

列出所有符合筛选条件的运行，其中根据数据源对规则集进行评估。

请求

- Filter – 一个 DataQualityRulesetEvaluationRunFilter 对象。

  筛选条件。
- NextToken – UTF-8 字符串。

  用于偏移结果的分页令牌。

- MaxResults – 数字（整数），不小于 1 或大于 1000。

  要返回的最大结果数量。

响应

- Runs – DataQualityRulesetEvaluationRunDescription 对象的数组。

  代表数据质量规则集规则的 DataQualityRulesetEvaluationRunDescription 对象列表。
- NextToken – UTF-8 字符串。

  分页令牌（如果有更多结果可用）。

错误

- InvalidInputException
- OperationTimeoutException
- InternalServiceException

StartDataQualityRuleRecommendationRun 动作（Python：start_data_quality_rule_rule_remuldation_

当你不知道要写什么规则时，启动推荐运行，用于生成规则。 AWS Glue 数据质量分析数据，并针对潜在的规则集提出建议。然后，您可以对规则集进行分类，并根据自己的喜好修改生成的规则集。

建议运行在 90 天后被自动删除。

请求

- DataSource – 必填：一个 DataSource 对象。

  与此次运行相关的数据源（AWS Glue 表）。
- Role – 必填：UTF-8 字符串。

  为加密运行结果而提供的 IAM 角色。
- NumberOfWorkers – 数字（整数）。

  运行所用的 G.1X 工作线程数量。默认值为 5。

- `Timeout` - 数字（整数），至少为 1。

  运行超时（分钟）。这是任务运行在终止并进入 `TIMEOUT` 状态前可以使用资源的最长时间。默认值为 2880 分钟（48 小时）。

- `CreatedRulesetName` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  规则集的名称。

- `ClientToken` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  用于幂等性，建议设置为随机 ID（例如，UUID），以避免创建或启动同一资源的多个实例。

响应

- `RunId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  与此运行关联的唯一运行标识符。

错误

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `ConflictException`

CancelDataQualityRuleRecommendationRun 操作（Python：取消数据质量规则_推荐_运行）

取消用于生成规则的指定建议运行。

请求

- `RunId` – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

与此运行关联的唯一运行标识符。

响应

- 无响应参数。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

GetDataQualityRuleRecommendationRun action（Python：获取数据质量规则_推荐_运行）

获取用于生成规则的指定建议运行。

请求

- `RunId` – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与此运行关联的唯一运行标识符。

响应

- `RunId` – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  与此运行关联的唯一运行标识符。
- `DataSource` – 一个 DataSource 对象。

  与此运行相关的数据源（ AWS Glue 表）。
- `Role` – UTF-8 字符串。

为加密运行结果而提供的 IAM 角色。

- NumberOfWorkers – 数字（整数）。

  运行所用的 G.1X 工作线程数量。默认值为 5。

- Timeout - 数字（整数），至少为 1。

  运行超时（分钟）。这是任务运行在终止并进入 TIMEOUT 状态前可以使用资源的最长时间。默认值为 2880 分钟（48 小时）。

- Status – UTF-8 字符串（有效值：STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT）。

  此运行的运行状态。

- ErrorString – UTF-8 字符串。

  与运行关联的错误字符串。

- StartedOn – 时间戳。

  此运行开始的日期和时间。

- LastModifiedOn – 时间戳。

  时间戳。此数据质量规则建议运行最后一个修改时间点。

- CompletedOn – 时间戳。

  此运行的完成日期和时间。

- ExecutionTime – 数字（整数）。

  运行使用资源的时间长度（以秒为单位）。

- RecommendedRuleset – UTF-8 字符串，长度不少于 1 个字节，不超过 65536 个字节。

  启动规则建议运行完成后，它会创建建议的规则集（一组规则）。该成员的规则采用数据质量定义语言（DQDL）格式。

- CreatedRulesetName – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  运行创建的规则集的名称。

错误

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

ListDataQualityRuleRecommendationRuns action（Python：list_data_quality_rule_rule_runs）

列出符合筛选条件的建议运行。

请求

- Filter – 一个 DataQualityRuleRecommendationRunFilter 对象。

  筛选条件。
- NextToken – UTF-8 字符串。

  用于偏移结果的分页令牌。
- MaxResults – 数字（整数），不小于 1 或大于 1000。

  要返回的最大结果数量。

响应

- Runs – DataQualityRuleRecommendationRunDescription 对象的数组。

  DataQualityRuleRecommendationRunDescription 对象的列表。
- NextToken – UTF-8 字符串。

  分页令牌（如果有更多结果可用）。

错误

- InvalidInputException
- OperationTimeoutException

- InternalServiceException

GetDataQualityResult 操作（Python：获取数据质量结果）

检索数据质量规则评估的结果。

请求

- ResultId – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  数据质量结果的唯一结果 ID。

响应

- ResultId – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  数据质量结果的唯一结果 ID。
- Score – 数字（双数），不超过 1.0。

  汇总的数据质量分数。表示规则与传递到规则总数的比率。
- DataSource – 一个 DataSource 对象。

  与数据质量结果相关的表（如果有）。
- RulesetName – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与数据质量结果相关的规则集的名称。
- EvaluationContext – UTF-8 字符串。

  在 AWS Glue Studio 的作业环境中，通常会为画布中的每个节点分配某种名称，并且数据质量节点会有名称。如果有多个节点，则 evaluationContext 可以区分这些节点。
- StartedOn – 时间戳。

  开始运行此数据质量结果的日期和时间。
- CompletedOn – 时间戳。

完成运行此数据质量结果的日期和时间。

- JobName – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与数据质量结果相关的作业名称（如果有）。

- JobRunId – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与数据质量结果相关的作业运行 ID（如果有）。

- RulesetEvaluationRunId – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  与规则集评估相关的唯一运行 ID。

- RuleResults – 一组 DataQualityRuleResult 对象，不超过 2000 个结构。

  代表每条规则结果的 DataQualityRuleResult 对象列表。

- AnalyzerResults – 一组 DataQualityAnalyzerResult 对象，不超过 2000 个结构。

  代表每个分析器结果的 DataQualityAnalyzerResult 对象列表。

- Observations – DataQualityObservation 对象的数组，不超过 50 个结构。

  代表评估规则和分析器后生成的观测值的 DataQualityObservation 对象列表。

错误

- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- EntityNotFoundException

BatchGetDataQualityResult 操作（Python：batch_get_data_quality_result）

检索指定结果 ID 的数据质量结果列表。

请求

- `ResultIds` – 必填：UTF-8 字符串数组，不少于 1 个字符串，不超过 100 个字符串。

  数据质量结果的唯一结果 ID 列表。

响应

- `Results` – 必填：DataQualityResult 对象的数组。

  代表数据质量规则集的 `DataQualityResult` 对象列表。
- `ResultsNotFound` – UTF-8 字符串数组，不少于 1 个字符串，不超过 100 个字符串。

  未找到结果的结果 ID 列表。

错误

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

ListDataQualityResults 操作（Python：列表数据质量结果）

返回您账户的所有数据质量执行结果。

请求

- `Filter` – 一个 DataQualityResultFilterCriteria 对象。

  筛选条件。
- `NextToken` – UTF-8 字符串。

  用于偏移结果的分页令牌。
- `MaxResults` – 数字（整数），不小于 1 或大于 1000。

  要返回的最大结果数量。

响应

- Results – 必填：DataQualityResultDescription 对象的数组。

  DataQualityResultDescription 对象的列表。
- NextToken – UTF-8 字符串。

  分页令牌（如果有更多结果可用）。

错误

- InvalidInputException
- OperationTimeoutException
- InternalServiceException

CreateDataQualityRuleset 操作（Python：create_data_quality_ruleset）

创建数据质量规则集，将 DQDL 规则应用于指定表。 AWS Glue

您可以使用数据质量定义语言 (DQDL) 创建规则集。有关更多信息，请参阅 AWS Glue 开发者指南。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  数据质量规则集的唯一名称。
- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  数据质量规则集的描述。
- Ruleset – 必填： UTF-8 字符串，长度不少于 1 个字节，不超过 65536 个字节。

  数据质量定义语言（DQDL）规则集。有关更多信息，请参阅 AWS Glue 开发者指南。
- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

应用于数据质量规则集的标签列表。

- `TargetTable` – 一个 DataQualityTargetTable 对象。

  与数据质量规则集关联的目标表。

- `RecommendationRunId` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  建议运行的唯一运行 ID。

- `ClientToken` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  用于幂等性，建议设置为随机 ID（例如，UUID），以避免创建或启动同一资源的多个实例。

响应

- `Name` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  数据质量规则集的唯一名称。

错误

- `InvalidInputException`
- `AlreadyExistsException`
- `OperationTimeoutException`
- `InternalServiceException`
- `ResourceNumberLimitExceededException`

DeleteDataQualityRuleset 操作（Python：delete_data_quality_ruleset）

删除数据质量规则集。

请求

- `Name` – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

数据质量规则集的名称。

响应

- 无响应参数。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

GetDataQualityRuleset 操作（Python：get_data_quality_ruleset）

按标识符或名称返回现有规则集。

请求

- `Name` – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  规则集的名称。

响应

- `Name` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  规则集的名称。
- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  规则集的描述。
- `Ruleset` – UTF-8 字符串，长度不少于 1 个字节，不超过 65536 个字节。

  数据质量定义语言（DQDL）规则集。有关更多信息，请参阅 AWS Glue 开发者指南。

- `TargetTable` – 一个 DataQualityTargetTable 对象。

  目标表的名称和数据库名称。

- `CreatedOn` – 时间戳。

  时间戳。此数据质量规则集的创建时间和日期。

- `LastModifiedOn` – 时间戳。

  时间戳。此数据质量规则最后一个修改时间点。

- `RecommendationRunId` – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  当根据建议运行创建规则集时，会生成此运行 ID 以将两者联系在一起。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

ListDataQualityRulesets 操作（Python：列表数据质量规则集）

返回指定表列表的规则集的分页列表。 AWS Glue

请求

- `NextToken` – UTF-8 字符串。

  用于偏移结果的分页令牌。

- `MaxResults` – 数字（整数），不小于 1 或大于 1000。

  要返回的最大结果数量。

- `Filter` – 一个 DataQualityRulesetFilterCriteria 对象。

  筛选条件。

- `Tags` – 键值对的映射数组，不超过 50 对。

每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

每个值是一个 UTF-8 字符串，不超过 256 个字节。

键值对标签的列表。

响应

- `Rulesets` – DataQualityRulesetListDetails 对象的数组。

  指定表列表的规则集的分页列表。 AWS Glue
- `NextToken` – UTF-8 字符串。

  分页令牌（如果有更多结果可用）。

错误

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

UpdateDataQualityRuleset 操作（Python：更新数据质量规则集）

更新指定的数据质量规则集。

请求

- `Name` – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  数据质量规则集的名称。
- `Description` – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  规则集的描述。
- `Ruleset` – UTF-8 字符串，长度不少于 1 个字节，不超过 65536 个字节。

数据质量定义语言（DQDL）规则集。有关更多信息，请参阅 AWS Glue 开发者指南。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  数据质量规则集的名称。

- Description – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  规则集的描述。

- Ruleset – UTF-8 字符串，长度不少于 1 个字节，不超过 65536 个字节。

  数据质量定义语言（DQDL）规则集。有关更多信息，请参阅 AWS Glue 开发者指南。

错误

- EntityNotFoundException
- AlreadyExistsException
- IdempotentParameterMismatchException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ResourceNumberLimitExceededException

敏感数据检测 API

敏感数据检测 API 描述了用于在结构化数据的列和行中检测敏感数据的 API。

数据类型

- CustomEntityType 结构

CustomEntityType 结构

表示用于在结构化数据的列和行中检测敏感数据的自定义模式的对象。

字段

- `Name` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  允许稍后检索或删除的自定义模式的名称。对于每个 AWS 账户来说，该名称必须是唯一的。

- `RegexString` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  用于检测自定义模式中敏感数据的正则表达式字符串。

- `ContextWords` – UTF-8 字符串数组，不少于 1 个字符串，不超过 20 个字符串。

  上下文字词列表。如果在正则表达式附近找不到这些上下文字词，则数据将不会被检测为敏感数据。

  如果没有传递上下文字词，则只检查正则表达式。

操作

- CreateCustomEntityType 操作（Python：create_custom_entity_type）
- DeleteCustomEntityType 操作（Python：delete_custom_entity_type）
- GetCustomEntityType 操作（Python：get_custom_entity_type）
- BatchGetCustomEntityTypes 操作（Python：batch_get_ustom_entity_type）
- ListCustomEntityTypes 操作（Python：list_custom_entity_type）

CreateCustomEntityType 操作（Python：create_custom_entity_type）

创建自定义模式，用于在结构化数据的列和行中检测敏感数据。

您创建的每个自定义模式都会指定一个正则表达式和上下文字词的可选列表。如果没有传递上下文字词，则只检查正则表达式。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  允许稍后检索或删除的自定义模式的名称。对于每个 AWS 账户来说，该名称必须是唯一的。
- RegexString – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  用于检测自定义模式中敏感数据的正则表达式字符串。
- ContextWords – UTF-8 字符串数组，不少于 1 个字符串，不超过 20 个字符串。

  上下文字词列表。如果在正则表达式附近找不到这些上下文字词，则数据将不会被检测为敏感数据。

  如果没有传递上下文字词，则只检查正则表达式。
- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  应用于自定义实体类型的标签列表。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  您创建的自定义模式的名称。

错误

- AccessDeniedException
- AlreadyExistsException
- IdempotentParameterMismatchException
- InternalServiceException
- InvalidInputException
- OperationTimeoutException

- ResourceNumberLimitExceededException

DeleteCustomEntityType 操作（Python：delete_custom_entity_type）

通过指定自定义模式名称来将其删除。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要删除的自定义模式的名称。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  已删除的自定义模式的名称。

错误

- EntityNotFoundException
- AccessDeniedException
- InternalServiceException
- InvalidInputException
- OperationTimeoutException

GetCustomEntityType 操作（Python：get_custom_entity_type）

通过指定自定义模式名称来检索其详细信息。

请求

- Name – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  要检索的自定义模式的名称。

响应

- Name – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  已检索的自定义模式的名称。

- RegexString – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  用于检测自定义模式中敏感数据的正则表达式字符串。

- ContextWords – UTF-8 字符串数组，不少于 1 个字符串，不超过 20 个字符串。

  上下文字词列表（如在创建自定义模式时指定）。如果在正则表达式附近找不到这些上下文字词，则数据将不会被检测为敏感数据。

错误

- EntityNotFoundException
- AccessDeniedException
- InternalServiceException
- InvalidInputException
- OperationTimeoutException

BatchGetCustomEntityTypes 操作（Python：batch_get_ustom_entity_type）

检索名称列表指定的自定义模式的详细信息。

请求

- Names – 必填：UTF-8 字符串数组，不少于 1 个或不超过 50 个字符串。

  要检索的自定义模式的名称列表。

响应

- CustomEntityTypes – CustomEntityType 对象的数组。

表示已创建的自定义模式的 CustomEntityType 对象列表。

- CustomEntityTypesNotFound – UTF-8 字符串数组，不少于 1 个字符串，不超过 50 个字符串。

  未找到的自定义模式名称列表。

错误

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

ListCustomEntityTypes 操作（Python：list_custom_entity_type）

列出所有已创建的自定义模式。

请求

- NextToken – UTF-8 字符串。

  用于偏移结果的分页令牌。
- MaxResults – 数字（整数），不小于 1 或大于 1000。

  要返回的最大结果数量。
- Tags – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  键值对标签的列表。

响应

- CustomEntityTypes – CustomEntityType 对象的数组。

  表示自定义模式的 CustomEntityType 对象列表。
- NextToken – UTF-8 字符串。

分页令牌（如果有更多结果可用）。

错误

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

AWS Glue 中的标记 API

数据类型

- Tag 结构

Tag 结构

Tag 对象表示用户可分配给 AWS 资源的标签。每个标签都包含定义的一个密钥和一个可选值。

有关标签以及如何控制对 AWS Glue 中资源的访问的更多信息，请参阅开发人员指南中的 AWS Glue 中的 AWS 标签和 指定 AWS Glue 资源 ARN。

字段

- `key` – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  标签键。在对象上创建标签时需要这个键。键区分大小写，并且不得包含前缀 aws。
- `value` – UTF-8 字符串，不超过 256 个字节。

  标签值。在对象上创建标签时，值是可选的。值区分大小写，并且不得包含前缀 aws。

操作

- TagResource 操作（Python：tag_resource）
- UntagResource 操作（Python：untag_resource）
- GetTags 操作（Python：get_tags）

TagResource 操作（Python：tag_resource）

为资源添加标签。标签是为 AWS 资源分配的标记。在 AWS Glue 中，您只能为特定资源添加标签。有关可添加标签的资源的信息，请参阅 AWS Glue 中的 AWS 标签。

除了调用标签相关的 API 的标记权限外，您还需要在连接上调用标记 API 的 `glue:GetConnection` 权限以及对数据库调用标记 API 的 `glue:GetDatabase` 权限。

请求

- `ResourceArn` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  要向其添加标签的 AWS Glue 资源的 ARN。有关 AWS Glue 资源 ARN 的更多信息，请参阅 AWS Glue ARN 字符串模式。
- `TagsToAdd` – 必填：键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  每个值是一个 UTF-8 字符串，不超过 256 个字节。

  要添加到此资源的标签。

响应

- 无响应参数。

错误

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `EntityNotFoundException`

UntagResource 操作（Python：untag_resource）

删除资源的标签。

请求

- `ResourceArn` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  要从中删除标签的资源的 Amazon Resource Name（ARN）。

- `TagsToRemove` – 必填：UTF-8 字符串数组，不超过 50 个字符串。

  要从该资源中删除的标签。

响应

- 无响应参数。

错误

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `EntityNotFoundException`

GetTags 操作（Python：get_tags）

检索与资源关联的标签的列表。

请求

- `ResourceArn` – 必填：UTF-8 字符串，长度不少于 1 个字节或超过 10240 个字节，与 Custom string pattern #22 匹配。

  要检索其标签的资源的 Amazon Resource Name（ARN）。

响应

- `Tags` – 键值对的映射数组，不超过 50 对。

  每个键都是一个 UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

每个值是一个 UTF-8 字符串，不超过 256 个字节。

请求的标签。

错误

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `EntityNotFoundException`

常见数据类型

常见数据类型介绍 AWS Glue中的各种常见的数据类型。

Tag 结构

该Tag对象表示可以分配给 AWS 资源的标签。每个标签都包含定义的一个密钥和一个可选值。

有关标签和控制资源访问权限的更多信息 AWS Glue，请参阅开发者指南中的AWS 标签 AWS Glue和指定 AWS Glue 资源 ARN。

字段

- `key` – UTF-8 字符串，长度不少于 1 个字节或超过 128 个字节。

  标签键。在对象上创建标签时需要这个键。键区分大小写，并且不得包含前缀 aws。
- `value` – UTF-8 字符串，不超过 256 个字节。

  标签值。在对象上创建标签时，值是可选的。值区分大小写，并且不得包含前缀 aws。

DecimalNumber 结构

包含以十进制格式表示的数字值。

字段

- `UnscaledValue` – 必填：Blob。

未标定的数字值。

- Scale – 必填：数字（整数）。

  确定小数点落在未标定的值中的位置的标定。

ErrorDetail 结构

包含有关错误的详细信息。

字段

- ErrorCode – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  与此错误关联的代码。

- ErrorMessage – 描述字符串，长度不超过 2048 个字节，与 URI address multi-line string pattern 匹配。

  描述错误的消息。

PropertyPredicate 结构

定义属性谓词。

字段

- Key – 值字符串，不超过 1024 个字节。

  属性的键。

- Value – 值字符串，不超过 1024 个字节。

  属性的值。

- Comparator – UTF-8 字符串（有效值：EQUALS | GREATER_THAN | LESS_THAN | GREATER_THAN_EQUALS | LESS_THAN_EQUALS）。

  用于将此属性与其他属性进行比较的比较运算符。

ResourceUri 结构

函数资源的 URI。

字段

- `ResourceType` – UTF-8 字符串（有效值：`JAR` | `FILE` | `ARCHIVE`）。

  资源的类型。

- `Uri` - 统一资源标识符 (uri)，不少于 1 个字节或超过 1024 个字节，与 URI address multi-line string pattern 匹配。

  用于访问资源的 URI。

ColumnStatistics 结构

表示表或分区生成的列级统计数据。

字段

- `ColumnName` – 必填：UTF-8 字符串，长度不少于 1 个字节，不超过 255 个字节，与 Single-line string pattern 匹配。

  统计数据所属列的名称。

- `ColumnType` – 必填：类型名称，长度不超过 20000 个字节，与 Single-line string pattern 匹配。

  列的数据类型。

- `AnalyzedTime` – 必填：时间戳。

  生成列统计数据的时间戳。

- `StatisticsData` – 必填：一个 ColumnStatisticsData 对象。

  `ColumnStatisticData` 对象，其中包含统计数据值。

ColumnStatisticsError 结构

封装失败的 `ColumnStatistics` 对象以及失败原因。

字段

- ColumnStatistics – 一个 ColumnStatistics 对象。

  列的 ColumnStatistics。
- Error – 一个 ErrorDetail 对象。

  包含操作失败原因的错误消息。

ColumnError 结构

封装失败的列名称以及失败原因。

字段

- ColumnName – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  封装失败的列名称。
- Error – 一个 ErrorDetail 对象。

  包含操作失败原因的错误消息。

ColumnStatisticsData 结构

包含单个类型的列统计数据。只应设置一个数据对象，并由 Type 属性指示。

字段

- Type – 必填：UTF-8 字符串（有效值：BOOLEAN | DATE | DECIMAL | DOUBLE | LONG | STRING | BINARY）。

  列统计数据的类型。
- BooleanColumnStatisticsData – 一个 BooleanColumnStatisticsData 对象。

  布尔值列统计数据。
- DateColumnStatisticsData – 一个 DateColumnStatisticsData 对象。

  日期列统计数据。

- `DecimalColumnStatisticsData` – 一个 DecimalColumnStatisticsData 对象。

  十进制列统计数据。 UnscaledValues 其中有 Base64 编码的二进制对象，存储十进制未缩放值的大端二进制补码表示形式。

- `DoubleColumnStatisticsData` – 一个 DoubleColumnStatisticsData 对象。

  双列统计数据。

- `LongColumnStatisticsData` – 一个 LongColumnStatisticsData 对象。

  长列统计数据。

- `StringColumnStatisticsData` – 一个 StringColumnStatisticsData 对象。

  字符串列统计数据。

- `BinaryColumnStatisticsData` – 一个 BinaryColumnStatisticsData 对象。

  二进制列统计数据。

BooleanColumnStatisticsData 结构

定义支持布尔值数据列的列统计数据。

字段

- `NumberOfTrues` – 必填：数字（长度），至多为“无”。

  列中的 True 值数量。

- `NumberOfFalses` – 必填：数字（长度），至多为“无”。

  列中的 False 数量。

- `NumberOfNulls` – 必填：数字（长度），至多为“无”。

  列中空值的数量。

DateColumnStatisticsData 结构

定义支持时间戳数据列的列统计数据。

字段

- MinimumValue – 时间戳。

  列中的最低值。

- MaximumValue – 时间戳。

  列中的最高值。

- NumberOfNulls – 必填：数字（长度），至多为“无”。

  列中空值的数量。

- NumberOfDistinctValues – 必填：数字（长度），至多为“无”。

  列中的独特值的数量。

DecimalColumnStatisticsData 结构

定义支持固定点数数据列的列统计数据。

字段

- MinimumValue – 一个 DecimalNumber 对象。

  列中的最低值。

- MaximumValue – 一个 DecimalNumber 对象。

  列中的最高值。

- NumberOfNulls – 必填：数字（长度），至多为“无”。

  列中空值的数量。

- NumberOfDistinctValues – 必填：数字（长度），至多为“无”。

  列中的独特值的数量。

DoubleColumnStatisticsData 结构

定义支持浮动点数数据列的列统计数据。

字段

- MinimumValue – 数字（double）。

  列中的最低值。

- MaximumValue – 数字（double）。

  列中的最高值。

- NumberOfNulls – 必填：数字（长度），至多为“无”。

  列中空值的数量。

- NumberOfDistinctValues – 必填：数字（长度），至多为“无”。

  列中的独特值的数量。

LongColumnStatisticsData 结构

定义支持整数数据列的列统计数据。

字段

- MinimumValue – 数字（长型）。

  列中的最低值。

- MaximumValue – 数字（长型）。

  列中的最高值。

- NumberOfNulls – 必填：数字（长度），至多为“无”。

  列中空值的数量。

- NumberOfDistinctValues – 必填：数字（长度），至多为“无”。

  列中的独特值的数量。

StringColumnStatisticsData 结构

定义支持字符序列数据值的列统计数据。

字段

- MaximumLength – 必填：数字（长度），至多为“无”。

  列中最长字符串的长度。

- AverageLength – 必填：数字（长度），至多为“无”。

  列中的平均字符串长度。

- NumberOfNulls – 必填：数字（长度），至多为“无”。

  列中空值的数量。

- NumberOfDistinctValues – 必填：数字（长度），至多为“无”。

  列中的独特值的数量。

BinaryColumnStatisticsData 结构

定义支持位序列数据值的列统计数据。

字段

- MaximumLength – 必填：数字（长度），至多为“无”。

  列中最长位序列的长度。

- AverageLength – 必填：数字（长度），至多为“无”。

  列中的平均位序列长度。

- NumberOfNulls – 必填：数字（长度），至多为“无”。

  列中空值的数量。

字符串模式

API 使用以下正则表达式来定义对于各种字符串参数和成员有效的内容：

- 单行字符串模式 -“[\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\t]*”
- URI 地址多行字符串模式 -“[\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\r\n\t]*”

- Logstash Grok 字符串模式 -“[\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\r\t]*”
- 标识符字符串模式 -“[A-Za-z_][A-Za-z0-9_]*”
- AWS IAM ARN 字符串模式 –“arn:aws:iam::\d{12}:role/.*”
- 版本字符串模式 -“^[a-zA-Z0-9-_]+$”
- 日志组字符串模式 -“[\.\-_/#A-Za-z0-9]+”
- 日志流字符串模式 -“[^:*]*”
- 自定义字符串模式 #10 –“[^\r\n]”
- 自定义字符串模式 #11 –“^arn:aws(-(cn|us-gov|iso(-[bef])?))?:secretsmanager:.*$”
- 自定义字符串模式 #12 –“^(https?)://[-a-zA-Z0-9+@#/%?=~_|!:,.;]*[-a-zA-Z0-9+@#/%=~_|]”
- 自定义字符串模式 #13 –“\S+”
- 自定义字符串模式 #14 –“^(https?):\/\/[^\s/$.?#].[^\s]*$”
- 自定义字符串模式 #15 –“^subnet-[a-z0-9]+$”
- 自定义字符串模式 #16 –“[\p{L}\p{N}\p{P}]*”
- 自定义字符串模式 #17 –“[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}”
- 自定义字符串模式 #18 –“[a-zA-Z0-9-_$#.]+”
- 自定义字符串模式 #19 –“^\w+\.\w+\.\w+$”
- 自定义字符串模式 #20 –“^\w+\.\w+$”
- 自定义字符串模式 #21 –“^([2-3]|3[.]9)$”
- 自定义字符串模式 #22 –“arn:(aws|aws-us-gov|aws-cn):glue:.*”
- 自定义字符串模式 #23 –“(^arn:aws:iam::\w{12}:root)”
- 自定义字符串模式 #24 –“^arn:aws(-(cn|us-gov|iso(-[bef])?))?:iam::[0-9]{12}:role/.+”
- 自定义字符串模式 #25 –“arn:aws:kms:.*”
- 自定义字符串模式 #26 –“arn:aws[^:]*:iam::[0-9]*:role/.+”
- 自定义字符串模式 #27 –“[\.\-_A-Za-z0-9]+”
- 自定义字符串模式 #28 –“^s3://([^/]+)/([^/]+/)*([^/]+)$”

- 自定义字符串模式 #29 –“.*”
- 自定义字符串模式 #30 –“^(Sun|Mon|Tue|Wed|Thu|Fri|Sat):([01]?[0-9]|2[0-3])$”
- 自定义字符串模式 #31 –“[a-zA-Z0-9_.-]+”
- 自定义字符串模式 #32 –“.*\S.*”
- 自定义字符串模式 #33 –“[a-zA-Z0-9-=._/@]+”
- 自定义字符串模式 #34 –“[1-9][0-9]*|[1-9][0-9]*-[1-9][0-9]*”
- 自定义字符串模式 #35 – "[\s\S]*"
- 自定义字符串模式 36 –“([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF]|[^\S\r\n"'= ;])*”
- 自定义字符串模式 37 –“[*A-Za-z0-9_-]*”
- 自定义字符串模式 #38 –“([\u0020-\u007E\r\s\n])*”
- 自定义字符串模式 #39 –“[A-Za-z0-9_-]*”
- 自定义字符串模式 #40 –“([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF]|[^\S\r\n"'])*”
- 自定义字符串模式 #41 –“([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF]|[^\S\r\n])*”
- 自定义字符串模式 #42 –“([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\s])*”
- 自定义字符串模式 #43 — “([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF]|[^\r\n])*”

异常

本节介绍可以用于查找问题根源并修复问题的 AWS Glue 例外情况。有关与机器学习相关的异常的 HTTP 错误代码和字符串的详细信息，请参阅 the section called “AWS Glue 机器学习异常”。

AccessDeniedException 结构

对资源的访问被拒绝。

字段

- Message – UTF-8 字符串。

描述问题的消息。

AlreadyExistsException 结构

要创建或添加的资源已存在。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

ConcurrentModificationException 结构

两个进程正在同时尝试修改某个资源。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

ConcurrentRunsExceededException 结构

并发运行的任务太多。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

CrawlerNotRunningException 结构

指定的爬网程序未在运行。

字段

- Message – UTF-8 字符串。

描述问题的消息。

CrawlerRunningException 结构

无法执行此操作，因为爬网程序已在运行。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

CrawlerStoppingException 结构

指定的爬网程序正在停止。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

EntityNotFoundException 结构

指定的实体不存在

字段

- Message – UTF-8 字符串。

  描述问题的消息。
- FromFederationSource – 布尔值。

  表示异常是否与联合来源有关。

FederationSourceException 结构

联合来源失败。

字段

- FederationSourceErrorCode - UTF-8 字符串（有效值：AccessDeniedException | EntityNotFoundException | InvalidCredentialsException | InvalidInputException | InvalidResponseException | OperationTimeoutException | OperationNotSupportedException | InternalServiceException | PartialFailureException | ThrottlingException）。

  问题的错误码。
- Message – UTF-8 字符串。

  描述问题的消息。

FederationSourceRetryableException 结构

联合来源失败，但可能会重试该操作。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

GlueEncryptionException 结构

加密操作失败。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

IdempotentParameterMismatchException 结构

两个不同记录关联了相同的唯一标识符。

字段

- Message – UTF-8 字符串。

描述问题的消息。

IllegalWorkflowStateException 结构

工作流程处于无效状态，无法执行请求的操作。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

InternalServiceException 结构

出现内部服务错误。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

InvalidExecutionEngineException 结构

指定的执行引擎未知或无效。

字段

- message – UTF-8 字符串。

  描述问题的消息。

InvalidInputException 结构

提供的输入无效。

字段

- Message – UTF-8 字符串。

描述问题的消息。

- FromFederationSource – 布尔值。

  表示异常是否与联合来源有关。

InvalidStateException 结构

指出您的数据处于无效状态的错误。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

InvalidTaskStatusTransitionException 结构

从一个任务正确过渡到下一个任务失败。

字段

- message – UTF-8 字符串。

  描述问题的消息。

JobDefinitionErrorException 结构

作业定义无效。

字段

- message – UTF-8 字符串。

  描述问题的消息。

JobRunInTerminalStateException 结构

作业运行的最终状态标志着失败。

字段

- message – UTF-8 字符串。

  描述问题的消息。

JobRunInvalidStateTransitionException 结构

作业运行遇到了从源状态到目标状态的无效转换。

字段

- jobRunId – UTF-8 字符串，长度不少于 1 个字节或超过 255 个字节，与 Single-line string pattern 匹配。

  相关作业运行的 ID。

- message – UTF-8 字符串。

  描述问题的消息。

- sourceState - UTF-8 字符串（有效值：STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT | ERROR | WAITING | EXPIRED）。

  源状态。

- targetState - UTF-8 字符串（有效值：STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT | ERROR | WAITING | EXPIRED）。

  目标状态。

JobRunNotInTerminalStateException 结构

作业运行未处于最终状态。

字段

- message – UTF-8 字符串。

  描述问题的消息。

LateRunnerException 结构

作业运行程序延迟。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

NoScheduleException 结构

没有适用的计划。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

OperationTimeoutException 结构

操作已超时。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

ResourceNotReadyException 结构

资源尚未为事务做好准备。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

ResourceNumberLimitExceededException 结构

超出了资源数字限制。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

SchedulerNotRunningException 结构

指定的计划程序未在运行。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

SchedulerRunningException 结构

指定的计划程序已在运行。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

SchedulerTransitioningException 结构

指定的计划程序正在转换。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

UnrecognizedRunnerException 结构

作业运行程序无法识别。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

ValidationException 结构

值无法验证。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

VersionMismatchException 结构

出现版本冲突。

字段

- Message – UTF-8 字符串。

  描述问题的消息。

AWS Glue 使用 AWS 软件开发工具包的 API 代码示例

以下代码示例说明如何 AWS Glue 使用 AWS 软件开发套件 (SDK)。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是展示如何通过在同一服务中调用多个函数来完成特定任务任务的代码示例。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

开始使用

你好 AWS Glue

以下代码示例展示了如何开始使用 AWS Glue。

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考ListJobs中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

C MakeLists .txt cMake 文件的代码。

```
# 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 源文件的代码。

```
#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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考[ListJobs](#)中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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);
        });
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListJobs中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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;
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ListJobs中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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)),
            }
        }
```

- 有关 API 的详细信息，请参阅适用ListJobs于 Rust 的AWS SDK API 参考。

代码示例

- AWS Glue 使用 AWS SDK 的操作
  - CreateCrawler与 AWS SDK 或 CLI 配合使用
  - CreateJob与 AWS SDK 或 CLI 配合使用
  - DeleteCrawler与 AWS SDK 或 CLI 配合使用
  - DeleteDatabase与 AWS SDK 或 CLI 配合使用
  - DeleteJob与 AWS SDK 或 CLI 配合使用
  - DeleteTable与 AWS SDK 或 CLI 配合使用
  - GetCrawler与 AWS SDK 或 CLI 配合使用
  - GetDatabase与 AWS SDK 或 CLI 配合使用
  - GetDatabases与 AWS SDK 或 CLI 配合使用
  - GetJob与 AWS SDK 或 CLI 配合使用
  - GetJobRun与 AWS SDK 或 CLI 配合使用
  - GetJobRuns与 AWS SDK 或 CLI 配合使用
  - GetTables与 AWS SDK 或 CLI 配合使用
  - ListJobs与 AWS SDK 或 CLI 配合使用
  - StartCrawler与 AWS SDK 或 CLI 配合使用
  - StartJobRun与 AWS SDK 或 CLI 配合使用
- AWS Glue 使用 AWS SDK 的场景
  - 开始使用 SD AWS Glue K 运行爬虫和作业 AWS

AWS Glue 使用 AWS SDK 的操作

以下代码示例演示如何使用 AWS 软件开发工具包执行单个 AWS Glue 操作。这些摘录调用 AWS Glue API，是大型程序的代码摘录，这些程序必须在上下文中运行。每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关设置和运行代码的说明。

以下示例仅包括最常用的操作。有关完整列表，请参阅 AWS Glue API 参考。

示例

- CreateCrawler与 AWS SDK 或 CLI 配合使用
- CreateJob与 AWS SDK 或 CLI 配合使用
- DeleteCrawler与 AWS SDK 或 CLI 配合使用
- DeleteDatabase与 AWS SDK 或 CLI 配合使用
- DeleteJob与 AWS SDK 或 CLI 配合使用
- DeleteTable与 AWS SDK 或 CLI 配合使用
- GetCrawler与 AWS SDK 或 CLI 配合使用
- GetDatabase与 AWS SDK 或 CLI 配合使用
- GetDatabases与 AWS SDK 或 CLI 配合使用
- GetJob与 AWS SDK 或 CLI 配合使用
- GetJobRun与 AWS SDK 或 CLI 配合使用
- GetJobRuns与 AWS SDK 或 CLI 配合使用
- GetTables与 AWS SDK 或 CLI 配合使用
- ListJobs与 AWS SDK 或 CLI 配合使用
- StartCrawler与 AWS SDK 或 CLI 配合使用
- StartJobRun与 AWS SDK 或 CLI 配合使用

`CreateCrawler`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `CreateCrawler`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考CreateCrawler中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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;
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考CreateCrawler中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考CreateCrawler中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考CreateCrawler中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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")
    }
}
```

- 有关 API 的详细信息，请参阅适用CreateCrawler于 K otlin 的AWS SDK API 参考。

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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,
                        ]]
                ],
            ]);
```

```
        });
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考CreateCrawler中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def 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
```

- 有关 API 的详细信息，请参阅适用CreateCrawler于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# 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
```

- 有关 API 的详细信息，请参阅 AWS SDK for Ruby API 参考CreateCrawler中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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(()),
        }?;
```

- 有关 API 的详细信息，请参阅适用CreateCrawler于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`CreateJob`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `CreateJob`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考CreateJob中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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;
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考CreateJob中的。

CLI

AWS CLI

创建用于转换数据的任务

以下 `create-job` 示例创建了一个运行存储在 S3 中的脚本的流式处理任务。

```
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 的内容：

```
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()
    }
}
```

输出：

```
{
    "Name": "my-testing-job"
}
```

有关更多信息，请参阅《Glue 开发者指南》中的 “在 AWS Glue 中AWS 创作作作业”。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考CreateJob中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考CreateJob中的。

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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,
       ]);
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考CreateJob中的。

PowerShell

用于 PowerShell

示例 1：此示例在 AWS Glue 中创建了一个新作业。命令名称的值始终为**glueetl**。AWS Glue 支持运行用 Python 或 Scala 编写的作业脚本。在此示例中，作业脚本 (MyTestGlueJob.py) 是用 Python 编写的。Python 参数在**$DefArgs**变量中指定，然后传递给**DefaultArguments**参数中的 PowerShell 命令，参数接受哈希表。**$JobParams**变量中的参数来自 CreateJob API，记录在 AWS Glue API 参考的 Jobs (https://docs.aws.amazon.com/glue/latest/dg/ aws-glue-api-jobs-job.html) 主题中。

```
$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
```

- 有关 API 的详细信息，请参阅 AWS Tools for PowerShell Cmdlet 参考[CreateJob](#)中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def 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
```

- 有关 API 的详细信息，请参阅适用CreateJob于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# 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
```

- 有关 API 的详细信息，请参阅 AWS SDK for Ruby API 参考CreateJob中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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())
        })?;
```

- 有关 API 的详细信息，请参阅适用CreateJob于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteCrawler`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DeleteCrawler`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考DeleteCrawler中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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;
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考DeleteCrawler中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });
```

```
  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考DeleteCrawler中的。

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        echo "Delete the crawler.\n";
        $glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);

    public function deleteCrawler($crawlerName)
    {
        return $this->glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考DeleteCrawler中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def 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
```

- 有关 API 的详细信息，请参阅适用DeleteCrawler于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# 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
```

- 有关 API 的详细信息，请参阅 AWS SDK for Ruby API 参考DeleteCrawler中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        glue.delete_crawler()
            .name(self.crawler())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
```

- 有关 API 的详细信息，请参阅适用DeleteCrawler于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteDatabase`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DeleteDatabase`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考[DeleteDatabase](#)中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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;
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考DeleteDatabase中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考DeleteDatabase中的。

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        echo "Delete the databases.\n";
        $glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);

    public function deleteDatabase($databaseName)
    {
        return $this->glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考DeleteDatabase中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
```

```
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def 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
```

- 有关 API 的详细信息，请参阅适用DeleteDatabase于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# 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
```

- 有关 API 的详细信息，请参阅 AWS SDK for Ruby API 参考DeleteDatabase中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        glue.delete_database()
            .name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
```

- 有关 API 的详细信息，请参阅适用DeleteDatabase于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteJob`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DeleteJob`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考DeleteJob中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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;
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考DeleteJob中的。

CLI

AWS CLI

删除任务

以下 delete-job 示例删除了不再需要的任务。

```
aws glue delete-job \
    --job-name my-testing-job
```

输出：

```
{
    "JobName": "my-testing-job"
}
```

有关更多信息，请参阅《Glue 开发者指南》中的 “在 AWS Glue 控制台AWS 上处理作业”。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteJob中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考DeleteJob中的。

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        echo "Delete the job.\n";
        $glueClient->deleteJob([
            'JobName' => $job['Name'],
        ]);

    public function deleteJob($jobName)
    {
        return $this->glueClient->deleteJob([
            'JobName' => $jobName,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考DeleteJob中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
```

```
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def 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
```

- 有关 API 的详细信息，请参阅适用DeleteJob于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# 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
```

- 有关 API 的详细信息，请参阅 AWS SDK for Ruby API 参考DeleteJob中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        glue.delete_job()
            .job_name(self.job())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
```

- 有关 API 的详细信息，请参阅适用DeleteJob于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteTable`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DeleteTable`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考DeleteTable中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考DeleteTable中的。

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考DeleteTable中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def 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
```

- 有关 API 的详细信息，请参阅适用DeleteTable于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# 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
```

- 有关 API 的详细信息，请参阅 AWS SDK for Ruby API 参考DeleteTable中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        for t in &self.tables {
            glue.delete_table()
                .name(t.name())
                .database_name(self.database())
                .send()
                .await
                .map_err(GlueMvpError::from_glue_sdk)?;
        }
```

- 有关 API 的详细信息，请参阅适用DeleteTable于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetCrawler`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `GetCrawler`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考GetCrawler中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
       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;
       }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考GetCrawler中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考GetCrawler中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考GetCrawler中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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")
    }
}
```

- 有关 API 的详细信息，请参阅适用GetCrawler于 K otlin 的AWS SDK API 参考。

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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,
            ]);
        });
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考GetCrawler中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def 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
```

- 有关 API 的详细信息，请参阅适用GetCrawler于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# 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
```

- 有关 API 的详细信息，请参阅 AWS SDK for Ruby API 参考GetCrawler中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
            let tmp_crawler = glue
                .get_crawler()
                .name(self.crawler())
                .send()
                .await
                .map_err(GlueMvpError::from_glue_sdk)?;
```

- 有关 API 的详细信息，请参阅适用GetCrawler于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetDatabase`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `GetDatabase`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考GetDatabase中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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;
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考GetDatabase中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考GetDatabase中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考GetDatabase中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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")
    }
}
```

- 有关 API 的详细信息，请参阅适用GetDatabase于 K otlin 的AWS SDK API 参考。

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $databaseName = "doc-example-database-$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,
            ]);
        });
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考GetDatabase中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
```

```
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def 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"]
```

- 有关 API 的详细信息，请参阅适用GetDatabase于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# 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
```

- 有关 API 的详细信息，请参阅 AWS SDK for Ruby API 参考GetDatabase中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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()))?;
```

- 有关 API 的详细信息，请参阅适用GetDatabase于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

GetDatabases与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 GetDatabases。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

CLI

AWS CLI

在 Glue 数据目录中列出部分或全部 AWS 数据库的定义

以下 get-databases 示例返回有关数据目录中数据库的信息。

```
aws glue get-databases
```

输出：

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

有关更多信息，请参阅《AWS Glue 开发人员指南》中的在数据目录中定义数据库。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetDatabases中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考GetDatabases中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

GetJob与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 GetJob。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

CLI

AWS CLI

检索有关任务的信息

以下 get-job 示例检索有关任务的信息。

```
aws glue get-job \
    --job-name my-testing-job
```

输出：

```
{
    "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"
    }
}
```

有关更多信息，请参阅《AWS Glue 开发人员指南》中的任务。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetJob中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getJob = (jobName) => {
  const client = new GlueClient({});

  const command = new GetJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考GetJob中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetJobRun`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `GetJobRun`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考GetJobRun中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
       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;
       }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考GetJobRun中的。

CLI

AWS CLI

获取有关任务运行的信息

以下 get-job-run 示例检索有关任务运行的信息。

```
aws glue get-job-run \
    --job-name "Combine legistators data" \
    --run-id jr_012e176506505074d94d761755e5c62538ee1aad6f17d39f527e9140cf0c9a5e
```

输出：

```
{
    "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"
    }
}
```

有关更多信息，请参阅《AWS Glue 开发人员指南》中的任务运行。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetJobRun中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考GetJobRun中的。

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考GetJobRun中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
```

```
        self.glue_client = glue_client


    def 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"]
```

- 有关 API 的详细信息，请参阅适用GetJobRun于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# 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
```

- 有关 API 的详细信息，请参阅 AWS SDK for Ruby API 参考GetJobRun中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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();
        }
```

- 有关 API 的详细信息，请参阅适用GetJobRun于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetJobRuns`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `GetJobRuns`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考[GetJobRuns](#)中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
       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());
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考[GetJobRuns](#)中的。

CLI

AWS CLI

获取有关任务的所有任务运行的信息

以下 `get-job-runs` 示例检索有关任务的任务运行的信息。

```
aws glue get-job-runs \
    --job-name "my-testing-job"
```

输出：

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

有关更多信息，请参阅《AWS Glue 开发人员指南》中的任务运行。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetJobRuns中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });
```

```
  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考GetJobRuns中的。

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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);
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考GetJobRuns中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def 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"]
```

- 有关 API 的详细信息，请参阅适用GetJobRuns于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# 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
```

- 有关 API 的详细信息，请参阅 AWS SDK for Ruby API 参考GetJobRuns中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

GetTables与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 GetTables。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/// <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;
```

```
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考[GetTables](#)中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
       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();
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考[GetTables](#)中的。

CLI

AWS CLI

列出指定数据库中的部分或全部表的定义

以下 `get-tables` 示例返回有关指定数据库中表的信息。

```
aws glue get-tables --database-name 'tempdb'
```

输出：

```
{
```

```
    "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"
        }
    ]
}
```

有关更多信息，请参阅《Glue 开发者指南》中的 AWS “在 Gl AWS u e 数据目录中定义表”。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetTables中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考[GetTables](#)中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考GetTables中的。

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $databaseName = "doc-example-database-$uniqid";

        $tables = $glueService->getTables($databaseName);

    public function getTables($databaseName): Result
    {
        return $this->glueClient->getTables([
```

```
            'DatabaseName' => $databaseName,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考GetTables中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def 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"]
```

- 有关 API 的详细信息，请参阅适用GetTables于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# 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
```

- 有关 API 的详细信息，请参阅 AWS SDK for Ruby API 参考GetTables中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        let tables = glue
            .get_tables()
            .database_name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        let tables = tables.table_list();
```

- 有关 API 的详细信息，请参阅适用GetTables于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListJobs`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListJobs`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考ListJobs中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
       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());
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考ListJobs中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考ListJobs中的。

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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);
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考ListJobs中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def 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"]
```

- 有关 API 的详细信息，请参阅适用ListJobs于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# 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
```

- 有关 API 的详细信息，请参阅 AWS SDK for Ruby API 参考ListJobs中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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)),
        }
    }
```

- 有关 API 的详细信息，请参阅适用ListJobs于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`StartCrawler`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `StartCrawler`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考StartCrawler中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
       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;
        }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考StartCrawler中的。

CLI

AWS CLI

启动爬网程序

以下 `start-crawler` 示例启动了一个爬网程序。

```
aws glue start-crawler --name my-crawler
```

输出：

```
None
```

有关更多信息，请参阅《AWS Glue 开发人员指南》中的定义爬网程序。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考StartCrawler中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考StartCrawler中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考StartCrawler中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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.")
    }
}
```

- 有关 API 的详细信息，请参阅适用StartCrawler于 K otlin 的AWS SDK API 参考。

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $crawlerName = "example-crawler-test-" . $uniqid;

        $databaseName = "doc-example-database-$uniqid";

        $glueService->startCrawler($crawlerName);

    public function startCrawler($crawlerName): Result
    {
        return $this->glueClient->startCrawler([
            'Name' => $crawlerName,
        ]);
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考StartCrawler中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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_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
```

- 有关 API 的详细信息，请参阅适用StartCrawler于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# 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
```

- 有关 API 的详细信息，请参阅 AWS SDK for Ruby API 参考StartCrawler中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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)),
                }
            }
        }?;
```

- 有关 API 的详细信息，请参阅适用StartCrawler于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`StartJobRun`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `StartJobRun`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 爬网程序和作业入门

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考StartJobRun中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
       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;
            }
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考StartJobRun中的。

CLI

AWS CLI

开始运行任务

以下 `start-job-run` 示例启动了一个任务。

```
aws glue start-job-run \
    --job-name my-job
```

输出：

```
{
    "JobRunId":
 "jr_22208b1f44eb5376a60569d4b21dd20fcb8621e1a366b4e7b2494af764b82ded"
}
```

有关更多信息，请参阅《AWS Glue 开发人员指南》中的编写任务。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考StartJobRun中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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);
};
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考StartJobRun中的。

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        $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,
            ],
        ]);
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考StartJobRun中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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_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"]
```

- 有关 API 的详细信息，请参阅适用StartJobRun于 Python 的AWS SDK (Boto3) API 参考。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# 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
```

- 有关 API 的详细信息，请参阅 AWS SDK for Ruby API 参考StartJobRun中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
        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();
```

- 有关 API 的详细信息，请参阅适用StartJobRun于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

AWS Glue 使用 AWS SDK 的场景

以下代码示例向您展示了如何 AWS Glue 使用 AWS 软件开发工具包实现常见场景。这些场景向您展示了如何通过在其中调用多个函数来完成特定任务 AWS Glue。每个场景都包含一个指向的链接 GitHub，您可以在其中找到有关如何设置和运行代码的说明。

示例

- 开始使用 SD AWS Glue K 运行爬虫和作业 AWS

开始使用 SD AWS Glue K 运行爬虫和作业 AWS

以下代码示例显示了如何：

- 创建爬网程序，爬取公有 Amazon S3 存储桶并生成包含 CSV 格式的元数据的数据库。
- 列出您的中的数据库和表的相关信息 AWS Glue Data Catalog。
- 创建任务，从 S3 存储桶提取 CSV 数据，转换数据，然后将 JSON 格式的输出加载到另一个 S3 存储桶中。
- 列出有关作业运行的信息，查看转换后的数据，并清除资源。

有关更多信息，请参阅教程： AWS Glue Studio 入门。

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建一个封装场景中使用的 AWS Glue 函数的类。

```
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;
    }

}
```

创建运行场景的类。

```
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);
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 详细信息，请参阅 AWS SDK for C++ API 参考中的以下主题。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

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

- 有关 API 详细信息，请参阅 AWS SDK for Java 2.x API 参考中的以下主题。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建并运行爬网程序，爬取公共 Amazon Simple Storage Service（Amazon S3）存储桶并生成一个描述其找到的 CSV 格式数据的元数据数据库。

```
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 };
  };
```

列出您的中的数据库和表的相关信息 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 };
  };
```

创建并运行任务，从源 Amazon S3 存储桶提取 CSV 数据，通过删除和重命名字段对其进行转换，然后将 JSON 格式的输出加载到另一个 Amazon S3 存储桶中。

```
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 };
  };
```

列出有关任务运行的信息，并查看一些转换后的数据。

```
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 };
  };
```

删除演示创建的所有资源。

```
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 };
};
```

- 有关 API 详细信息，请参阅《AWS SDK for JavaScript API 参考》中的以下主题。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases

- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的以下主题。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases

- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

PHP

适用于 PHP 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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,
        ]);
    }
}
```

- 有关 API 详细信息，请参阅 AWS SDK for PHP API 参考中的以下主题。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob

- DeleteTable
- GetCrawler
- GetDatabase
- GetDatabases
- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建一个封装场景中使用的 AWS Glue 函数的类。

```
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
```

创建运行场景的类。

```
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.")
```

创建一个 ETL 脚本，用于在作业运行期间 AWS Glue 提取、转换和加载数据。

```
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()
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的以下主题。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建一个封装场景中使用的 AWS Glue 函数的类。

```
# 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
```

创建运行场景的类。

```
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
```

创建一个 ETL 脚本，用于在作业运行期间 AWS Glue 提取、转换和加载数据。

```
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()
```

- 有关 API 详细信息，请参阅 AWS SDK for Ruby API 参考中的以下主题。
  - CreateCrawler
  - CreateJob

- DeleteCrawler
- DeleteDatabase
- DeleteJob
- DeleteTable
- GetCrawler
- GetDatabase
- GetDatabases
- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建并运行爬网程序，爬取公共 Amazon Simple Storage Service（Amazon S3）存储桶并生成一个描述其找到的 CSV 格式数据的元数据数据库。

```
        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)),
                }
            }
        }?;
```

列出您的中的数据库和表的相关信息 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();
```

创建并运行任务，从源 Amazon S3 存储桶提取 CSV 数据，通过删除和重命名字段对其进行转换，然后将 JSON 格式的输出加载到另一个 Amazon S3 存储桶中。

```
        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();
```

删除演示创建的所有资源。

```
        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)?;
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的以下主题。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK 配合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

AWS Glue 中的安全性

AWS 十分重视云安全性。作为 AWS 客户，您将从专为满足大多数安全敏感型企业的要求而打造的数据中心和网络架构中受益。

安全性是 AWS 和您的共同责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云的安全性 – AWS 负责保护在 AWS 云中运行 AWS 服务的基础设施。AWS 还向您提供可安全使用的服务。作为 AWS 合规性计划的一部分，第三方审核人员将定期测试和验证安全性的有效性。要了解适用于 AWS Glue 的合规性计划，请参阅合规性计划范围内的 AWS 服务。
- 云中的安全性：您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

此文档将帮助您了解如何在使用 AWS Glue 时应用责任共担模型。以下主题说明如何配置 AWS Glue 以实现您的安全性和合规性目标。您还会了解如何使用其他 AWS 服务以帮助您监控和保护 AWS Glue 资源。

主题

AWS Glue 中的数据保护

AWS Glue 提供多种功能，旨在帮助保护您的数据。

主题

- AWS Glue 对其他 AWS 服务的依赖
- 开发终端节点

静态加密

AWS Glue 支持 使用 AWS Glue Studio 构建可视化 ETL 作业 和 使用开发终端节点来开发脚本 的静态数据加密。您可以配置提取、转换和加载（ETL）任务和开发终端节点，以使用 AWS Key Management Service（AWS KMS）密钥写入加密的静态数据。您也可以使用您通过 AWS KMS 管理的密钥加密存储在 AWS Glue Data Catalog 中的元数据。此外，您可以使用 AWS KMS 密钥来加密作业书签以及爬网程序和 ETL 作业生成的日志。

您可以加密 AWS Glue Data Catalog 中的元数据对象，以及由任务、爬网程序和开发终端节点写入 Amazon Simple Storage Service（Amazon S3）和 Amazon CloudWatch Logs 的数据。当您在 AWS Glue 中创建任务、爬网程序和开发终端节点时，您可以通过附加安全配置来提供加密设置。安全配置包含 Amazon S3 托管的服务器端加密密钥（SSE-S3）或存储在 AWS KMS 中的客户主密钥（CMK）（SSE-KMS）。您可以使用 AWS Glue 控制台创建安全配置。

您也可以在账户中启用整个数据目录的加密。您可以通过指定存储在 AWS KMS 中的 CMK 来执行此操作。

> **Important**
>
> AWS Glue 仅支持对称客户管理型密钥。有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 Customer Managed Keys (CMKs)。

启用加密后，当您添加数据目录对象时，运行爬网程序、运行任务或启动开发终端节点时，SSE-S3 或 SSE-KMS 密钥用于写入静态数据。此外，您可以将 AWS Glue 配置为仅通过受信任的传输层安全性 (TLS) 协议访问 Java 数据库连接 (JDBC) 数据存储。

在 AWS Glue 中，您可以在以下位置控制加密设置：

- 数据目录的设置。
- 您创建的安全配置。
- 作为参数传递给 AWS Glue ETL（提取、转换和加载）任务的服务器端加密设置（SSE-S3 或 SSE-KMS）。

有关如何设置加密的更多信息，请参阅 在 AWS Glue 中设置加密。

主题

加密数据目录

AWS Glue Data Catalog 加密可提高敏感数据的安全性。AWS Glue与 AWS Key Management Service（AWS KMS）集成，以加密存储在数据目录中的元数据。您可以使用 AWS Glue 控制台或 AWS CLI，为数据目录中的资源启用或禁用加密设置。

为数据目录启用加密时，您创建的所有新对象都将被加密。禁用加密后，您创建的新对象将不会被加密，但现有的加密对象将保持加密状态。

您可以使用 AWS 托管加密密钥或客户管理型加密密钥对整个数据目录进行加密。有关密钥类型和状态的更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 AWS Key Management Service 概念。

AWS 托管密钥

AWS 托管密钥是由与 AWS KMS集成的 AWS 服务代表您在账户中创建、管理和使用的 KMS 密钥。可以在 AWS CloudTrail 日志中查看账户中的 AWS 托管密钥，查看其密钥策略以及审计其使用情况。但是，您无法管理这些 KMS 钥或更改其权限。

静态加密自动集成 AWS KMS，从而管理 AWS Glue 用于加密元数据的 AWS 托管密钥。如果启用元数据加密时 AWS 托管密钥尚不存在，AWS KMS 将自动为您创建新密钥。

有关更多信息，请参阅 AWS 托管式密钥。

客户管理密钥

客户托管密钥是在您的 AWS 账户 中创建、拥有和托管的 KMS 密钥。您对 KMS 密钥拥有全部控制权。您可以：

- 建立和维护密钥政策、IAM 策略和授权
- 启用和禁用密钥
- 轮换加密材料
- 添加标签
- 创建引用密钥的别名

- 计划密钥的删除

有关管理客户管理型密钥权限的更多信息，请参阅客户管理型密钥。

> **Important**
>
> AWS Glue 仅支持对称客户管理型密钥。KMS 密钥列表仅显示对称密钥。但是，如果选择了选择 KMS 密钥 ARN，则可通过控制台输入任何密钥类型的 ARN。确保仅为对称密钥输入 ARN。
> 要创建对称客户管理型密钥，请按照《AWS Key Management Service 开发人员指南》中的 creating symmetric customer managed keys 部分的步骤进行操作。

启用数据目录静态加密时，将使用 KMS 密钥对以下资源类型进行加密：

- 数据库
- 表
- 分区
- 表格版本
- 列统计数据
- 用户定义的函数
- 数据目录视图

AWS Glue 加密上下文

加密上下文是一组可选的键值对，包含有关数据的其他上下文信息。AWS KMS 会将加密上下文用作其他已经过验证的数据以支持经过身份验证的加密。在请求中包含加密上下文以加密数据时，AWS KMS 将加密上下文绑定到加密的数据。要解密数据，请在请求中包含相同的加密上下文。AWS Glue 在所有 AWS KMS 加密操作中使用相同的加密上下文，其中键为 `glue_catalog_id`，值为 `catalogId`。

```
"encryptionContext": {
    "glue_catalog_id": "111122223333"
}
```

使用 AWS 托管密钥或对称的户管理型密钥来加密数据目录时，还可以使用审计记录和日志中的加密上下文来识别密钥的使用情况。加密上下文还会显示在 AWS CloudTrail 生成的日志或 Amazon CloudWatch 日志中。

启用加密

您可以通过 AWS Glue 控制台中的数据目录设置，或者使用 AWS CLI 为 AWS Glue Data Catalog 对象启用加密。

Console

使用控制台启用加密

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格中，选择数据目录。
3. 在数据目录设置页面上，选中元数据加密复选框，然后选择一个 AWS KMS 密钥。

   启用加密时，如果您未指定客户管理型密钥，则加密设置将使用 AWS 托管 KMS 密钥。
4. （可选）如果您使用客户管理型密钥加密数据目录，Data Cat alog提供了一个用于注册 IAM 角色来加密和解密资源的选项。您需要向您的 IAM 角色授予 AWS Glue 可以代表代入的权限。这包括加密和解密数据的 AWS KMS 权限。

   当您在数据目录中创建新资源时，AWS Glue 将代入所提供的 IAM 角色来加密数据。同样，当使用者访问资源时，AWS Glue 也会代入该 IAM 角色来解密数据。如果您注册了具有所需权限的 IAM 角色，则调用主体不再需要访问密钥和解密数据的权限。

   > **Important**
   >
   > 只有当您使用客户管理型密钥加密数据目录资源时，您才能将 KMS 操作委派给 IAM 角色。KMS 角色委派功能目前不支持使用 AWS 托管密钥加密数据目录资源的情形。

   > **Warning**
   >
   > 当您启用 IAM 角色委派 KMS 操作的权限时，您将无法再访问以前使用 AWS 托管密钥加密的数据目录资源。

   a. 要启用 AWS Glue 可以代入以代表您加密和解密数据的 IAM 角色，请选择将 KMS 操作委派给 IAM 角色选项。
   b. 然后选择一个 IAM 角色。

要创建 IAM 角色，请参阅 为 AWS Glue 创建 IAM 角色。

AWS Glue 访问数据目录时将代入的 IAM 角色必须具有加密和解密数据目录中元数据的权限。您可以创建一个 IAM 角色并附加以下内联策略：

- 添加以下策略以包括加密和解密数据目录的 AWS KMS 权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:Encrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:<region>:<account-id>:key/<key-id>"
        }
    ]
}
```

- 然后将以下信任策略添加到该角色，以便 AWS Glue 服务代入该 IAM 角色。

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

- 然后向该 IAM 角色添加 `iam:PassRole` 权限。

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

启用加密后，如果您尚未指定 AWS Glue 要代入的 IAM 角色，则访问数据目录的主体必须具有执行以下 API 操作的权限：

- kms:Decrypt
- kms:Encrypt
- kms:GenerateDataKey

AWS CLI

使用开发工具包或 AWS CLI 启用加密

- 使用 PutDataCatalogEncryptionSettings API 操作。如果未指定密钥，AWS Glue 将使用客户账户的 AWS 托管加密密钥对数据目录进行加密。

```
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>"
    }

  }'
```

启用加密后，您在数据目录对象中创建的所有对象都将被加密。如果您清除此设置，您在数据目录中创建的对象将不再加密。您可以继续使用所需的 KMS 权限访问数据目录中现有的加密对象。

> **Important**
>
> 对于在数据目录中使用 AWS KMS 密钥加密的任何对象，此密钥必须在 AWS KMS 密钥存储中保持可用。如果删除密钥，则无法再对对象进行解密。在某些情况下，您可能希望阻止访问数据目录元数据。

监控用于 AWS Glue 的 KMS 密钥

当您将 KMS 密钥用于数据目录资源时，您可以使用 AWS CloudTrail 或 Amazon CloudWatch 日志来跟踪 AWS Glue 发送到 AWS KMS的请求。AWS CloudTrail 会监控并记录 AWS Glue 为了访问使用您的 KMS 密钥加密的数据而调用的 KMS 操作。

以下是 `Decrypt` 和 `GenerateDataKey` 操作的 AWS CloudTrail 事件示例。

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"
}
```

加密连接密码

您可以使用 `GetConnection` 和 `GetConnections` API 操作在 AWS Glue Data Catalog 中检索连接密码。这些密码存储在数据目录连接中，并在 AWS Glue 连接到 Java 数据库连接（JDBC）数据存储时使用。创建或更新连接时，数据目录设置中有一个选项确定密码是否已加密，如果已加密，则确定指定了哪个 AWS Key Management Service（AWS KMS）密钥。

在 AWS Glue 控制台中，您可以在 Data catalog settings (数据目录设置) 页面上启用该选项：

加密连接密码

1. 登录 AWS Management Console，然后打开 AWS Glue 控制台，网址为：https://console.aws.amazon.com/glue/。
2. 在导航窗格中，选择 Settings (设置)。
3. 在 Data catalog settings (数据目录设置) 页面上，选择 Encrypt connection passwords (加密连接密码)，然后选择一个 AWS KMS 密钥。

> **Important**
>
> AWS Glue 只支持对称客户主密钥（CMK）。AWS KMS key (Amazon KMS 密钥) 列表仅显示对称密钥。但是，如果选择 Choose a AWS KMS key ARN (选择 Amazon KMS 密钥 ARN)，控制台允许您为任何密钥类型输入 ARN。确保仅为对称密钥输入 ARN。

有关更多信息，请参阅 Data Catalog 设置 。

加密 AWS Glue 写入的数据

安全配置是 AWS Glue 可以使用的一组安全属性。您可以使用安全配置加密静态数据。以下场景显示了使用安全配置的一些方法。

- 将安全配置附加到 AWS Glue 爬网程序以写入加密的 Amazon CloudWatch Logs。有关将安全配置附加到爬网程序的更多信息，请参阅 the section called “步骤 3：配置安全设置”。
- 将安全配置附加到提取、转换和加载（ETL）任务以写入加密的 Amazon Simple Storage Service（Amazon S3）目标和加密的 CloudWatch Logs。
- 将安全配置附加到 ETL 任务以将其任务书签作为加密 Amazon S3 数据写入。
- 将安全配置附加到开发终端节点以写入加密的 Amazon S3 目标。

> **Important**
>
> 目前，安全配置会覆盖作为 ETL 任务参数传递的任何服务器端加密（SSE-S3）设置。因此，如果安全配置和 SSE-S3 参数都与作业关联，则忽略 SSE-S3 参数。

有关安全配置的更多信息，请参阅 在 AWS Glue 控制台上处理安全配置。

主题

- 将 AWS Glue 设置为使用安全配置
- 为 VPC 作业和爬网程序创建到 AWS KMS 的路由
- 在 AWS Glue 控制台上处理安全配置

将 AWS Glue 设置为使用安全配置

请按照以下步骤设置您的 AWS Glue 环境以使用安全配置。

1. 创建或更新您的 AWS Key Management Service（AWS KMS）密钥，向传递到 AWS Glue 爬网程序和任务的 IAM 角色授予 AWS KMS 权限以加密 CloudWatch Logs。有关更多信息，请参阅 Amazon CloudWatch Logs 用户指南中的使用 AWS KMS 更改 CloudWatch Logs 中的日志数据。

在下面的示例中，*"role1"*、*"role2"* 和 *"role3"* 是传递到爬网程序和任务的 IAM 角色。

```
{
       "Effect": "Allow",
       "Principal": { "Service": "logs.region.amazonaws.com",
       "AWS": [
                "role1",
                "role2",
                "role3"
             ] },
                    "Action": [
                           "kms:Encrypt*",
                           "kms:Decrypt*",
                           "kms:ReEncrypt*",
                           "kms:GenerateDataKey*",
                           "kms:Describe*"
                    ],
                    "Resource": "*"
}
```

如果您使用此密钥加密 CloudWatch Logs，则需要 `Service` 语句（显示为 `"Service": "logs.region.amazonaws.com"`）。

2. 确保 AWS KMS 密钥为 `ENABLED`，然后才可使用。

> **Note**
>
> 如果您使用 Iceberg 作为数据湖框架，则 Iceberg 表有自己的机制来启用服务器端加密。除了 AWS Glue 的安全配置外，您还应该启用这些配置。要在 Iceberg 表上启用服务器端加密，请查看 Iceberg 文档中的指南。

为 VPC 作业和爬网程序创建到 AWS KMS 的路由

您可以通过 Virtual Private Cloud (VPC) 中的私有终端节点直接连接到 AWS KMS，而不是通过互联网连接。当您使用 VPC 终端节点时，您的 VPC 和 AWS KMS 之间的通信完全在 AWS 网络内进行。

您可以在 VPC 中创建 AWS KMS VPC 终端节点。如果没有此步骤，您的任务或爬网程序可能会失败，任务上具有 `kms timeout` 或爬网程序上具有 `internal service exception`。有关详细

说明，请参阅 AWS Key Management Service 开发人员指南中的通过 VPC 终端节点连接到 AWS KMS。

当您按照以下说明操作时，在 VPC 控制台上，您必须执行以下操作：

- 选择 Enable Private DNS name (启用私有 DNS 名称)。
- 选择 Security group (安全组)（带自引用规则），此安全组将用于访问 Java 数据库连接 (JDBC) 的任务或爬网程序。有关 AWS Glue 连接的更多信息，请参阅 连接到数据。

当您将安全配置添加到访问 JDBC 数据存储的爬网程序或任务时，AWS Glue 必须具有到 AWS KMS 终端节点的路由。您可以使用网络地址转换 (NAT) 网关或 AWS KMS VPC 终端节点提供路由。要创建 NAT 网关，请参阅 Amazon VPC 用户指南中的 NAT 网关。

在 AWS Glue 控制台上处理安全配置

> **Warning**
>
> Ray 作业目前不支持 AWS Glue 安全配置。

AWS Glue 中的安全配置包含当您写入加密数据时所需的属性。您在 AWS Glue 控制台上创建安全配置，以提供由爬网程序、作业和开发终端节点使用的加密属性。

要查看您创建的所有安全配置的列表，请点击 https://console.aws.amazon.com/glue/ 打开 AWS Glue 控制台，然后在导航窗格中选择 Security configurations (安全配置)。

Security configurations (安全配置) 列表显示有关每个配置的以下属性：

名称

在创建配置时提供的唯一名称。名称可以包含字母（A-Z）、数字（0-9）、连字符（-）、或下划线（_），且长度最多为 255 个字符。

启用 Amazon S3 加密

如果开启，对于数据目录中的元数据存储会启用 Amazon Simple Storage Service（Amazon S3）加密模式（如 SSE-KMS 或 SSE-S3）。

启用 Amazon CloudWatch 日志加密

如果启用，在向 Amazon CloudWatch 写入日志时会使用 Amazon S3 加密模式（如 SSE-KMS）。

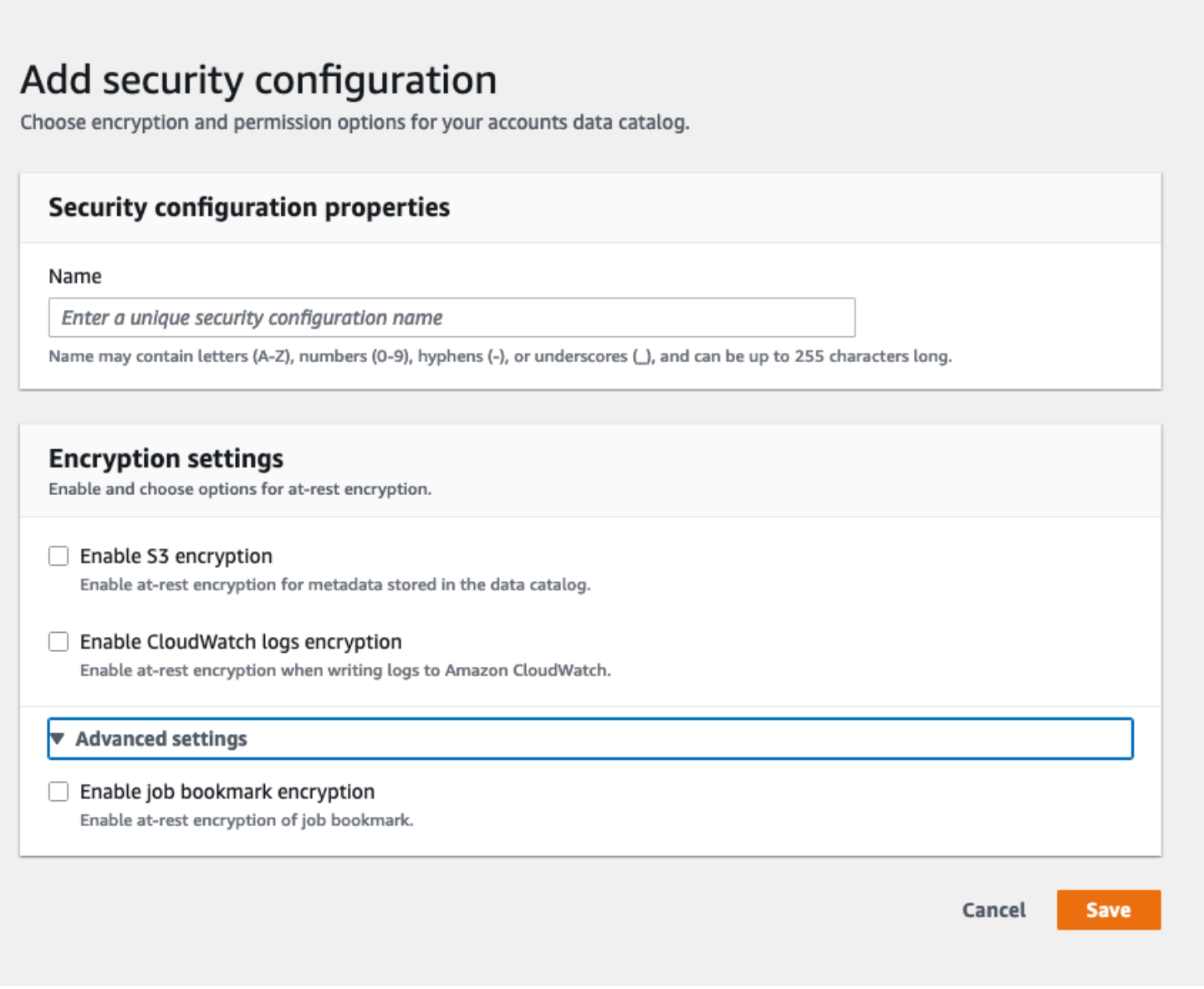
高级设置：启用作业书签加密

如果启用，在将作业添加到书签时会使用 Amazon S3 加密模式（如 CSE-KMS）。

您可以在控制台上的 Security configurations (安全配置) 部分中添加或删除配置。要查看配置的详细信息，请在列表中选择配置名称。详细信息包括您在创建配置时定义的信息。

添加安全配置

要使用 AWS Glue 控制台添加安全配置，请在 Security configurations (安全配置) 页面上选择 Add security configuration (添加安全配置)。



安全配置属性

输入唯一的安全配置名称。名称可以包含字母（A-Z）、数字（0-9）、连字符（-）、或下划线（_），且长度最多为 255 个字符。

加密设置

您可以对存储在 Amazon S3 的 Data Catalog 和 Amazon CloudWatch 的日志中的元数据启用静态加密。要在 AWS Glue 控制台上使用 AWS Key Management Service (AWS KMS) 密钥对数据和元数据加密，请向控制台用户添加一个策略。此策略必须将允许的资源指定为用于对 Amazon S3 数据存储进行加密的密钥 Amazon Resource Names（ARN），如以下示例所示。

```
{
"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**
>
> 当安全配置附加到爬网程序或任务时，传递的 IAM 角色必须具有 AWS KMS 权限。有关更多信息，请参阅 加密 AWS Glue 写入的数据。

在定义配置时，您可以为以下属性提供值：

启用 S3 加密

当您写入 Amazon S3 数据时，您可以将服务器端加密与 Amazon S3 托管密钥（SSE-S3）结合使用，也可以将服务器端加密（SSE-KMS）与 AWS KMS 托管密钥结合使用。该字段是可选的。要允许访问 Amazon S3，请选择 AWS KMS 密钥，或选择 Enter a key ARN (输入密钥 ARN) 并提供密钥的 ARN。输入 `arn:aws:kms:region:account-id:key/key-id` 格式的 ARN。您也可以提供密钥别名形式的 ARN，例如 `arn:aws:kms:region:account-id:alias/alias-name`。

如果您为作业启用 Spark 用户界面，则上传到 Amazon S3 的 Spark 用户界面日志文件将采用相同的加密方式。

> **Important**
>
> AWS Glue 只支持对称客户主密钥（CMK）。AWS KMS key (Amazon KMS 密钥) 列表仅显示对称密钥。但是，如果选择 Choose a AWS KMS key ARN (选择 Amazon KMS 密钥 ARN)，控制台允许您为任何密钥类型输入 ARN。确保仅为对称密钥输入 ARN。

启用 CloudWatch 日志加密

服务器端加密（SSE-KMS）用于加密 CloudWatch Logs。该字段是可选的。要启用它，请选择 AWS KMS 密钥，或选择 Enter a key ARN (输入密钥 ARN) 并提供密钥的 ARN。输入 `arn:aws:kms:region:account-id:key/key-id` 格式的 ARN。您也可以提供密钥别名形式的 ARN，例如 `arn:aws:kms:region:account-id:alias/alias-name`。

高级设置：作业书签加密

客户端 (CSE-KMS) 加密用于加密作业书签。该字段是可选的。书签数据先进行加密，然后再发送到 Amazon S3 进行存储。要启用它，请选择 AWS KMS 密钥，或选择 Enter a key ARN (输入密钥 ARN) 并提供密钥的 ARN。输入 `arn:aws:kms:region:account-id:key/key-id` 格式的 ARN。您也可以提供密钥别名形式的 ARN，例如 `arn:aws:kms:region:account-id:alias/alias-name`。

有关更多信息，请参阅 Amazon Simple Storage Service 用户指南中的以下主题：

- 有关 `SSE-S3` 的更多信息，请参阅使用具有 Amazon S3 托管加密密钥 (SSE-S3) 的服务器端加密 (SSE-S3) 保护数据。
- 有关 `SSE-KMS` 的更多信息，请参阅 Protecting Data Using Server-Side Encryption with AWS KMS keys。
- 有关 `CSE-KMS` 的信息，请参阅 Using a KMS key stored in AWS KMS。

传输中加密

AWS 为传输中数据提供传输层安全性 (TLS) 加密。您可以使用 AWS Glue 中的安全配置为爬网程序、ETL 任务和开发端点配置加密设置。您可以通过数据目录的设置启用 AWS Glue Data Catalog 加密。

截至 2018 年 9 月 4 日，支持用于 AWS Glue ETL 和 AWS Glue Data Catalog 的 AWS KMS（自带密钥和服务器端加密）。

FIPS 合规性

如果在通过命令行界面或 API 访问 AWS 时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-2 版》。

密钥管理

您可以使用具有 AWS Glue 的 AWS Identity and Access Management（IAM）来定义用户、AWS 资源、组、角色和有关访问、拒绝等的精细策略。

您可以根据组织的需要，同时使用基于资源和基于身份的策略来定义对元数据的访问。基于资源的策略列出了允许或拒绝访问您的资源的委托人，允许您设置跨账户访问等策略。身份策略专门附加到 IAM 中的用户、组和角色。

有关分步示例，请参阅 AWS 大数据博客中的 Restrict access to your AWS Glue Data Catalog with resource-level IAM permissions and resource-based policies。

策略的精细访问部分在 `Resource` 条款中进行了定义。此部分定义了可执行此操作的 AWS Glue Data Catalog 对象，以及该操作将返回哪些对象。

开发终端节点 是可用于开发和测试您的 AWS Glue 脚本的环境。您可以添加、删除或轮换开发终端节点的 SSH 密钥。

截至 2018 年 9 月 4 日，支持用于 AWS Glue ETL 和 AWS Glue Data Catalog 的 AWS KMS（自带密钥和服务器端加密）。

AWS Glue 对其他 AWS 服务的依赖

某个用户要能够使用 AWS Glue 控制台，必须拥有一组允许其使用其 AWS 账户的 AWS Glue 资源的最低权限。除这些 AWS Glue 权限以外，控制台还需要来自以下服务的权限：

- 用于显示日志的 Amazon CloudWatch Logs 权限。
- 用于列出并传递角色的 AWS Identity and Access Management（IAM）权限。
- 用于处理堆栈的 AWS CloudFormation 权限。
- 用于列出Virtual Private Cloud（VPC）、子网、安全组、实例和其他对象（用于在运行任务、爬虫程序和创建开发终端节点时设置 Amazon EC2 项目，如 VPC）的 Amazon Elastic Compute Cloud（Amazon EC2）权限。
- 用于列出存储桶和对象以及检索和保存脚本的 Amazon Simple Storage Service（Amazon S3）权限。

- 用于使用集群的 Amazon Redshift 权限。
- 用于列出实例的 Amazon Relational Database Service（Amazon RDS）权限。

开发终端节点

开发终端节点是可用于开发和测试您的 AWS Glue 脚本的环境。您可以使用 AWS Glue 创建、编辑和删除开发终端节点。您可以列出所有已创建的开发端点。您可以添加、删除或轮换开发终端节点的 SSH 密钥。您还可以创建使用开发终端节点的笔记本。

您提供配置值以预置开发环境。这些值告知 AWS Glue 如何设置网络，以便您可以安全地访问开发终端节点，并且您的端点可以访问您的数据存储。然后，您可以创建连接到开发终端节点的笔记本。您可以使用笔记本编写和测试 ETL 脚本。

选择一个 AWS Identity and Access Management（IAM）角色，该角色与您用于运行 AWS Glue ETL 任务的 IAM 角色具有相似的权限。使用 Virtual Private Cloud（VPC）、子网和安全组创建可以安全地连接到您的数据资源的开发终端节点。您可以生成一个 SSH 密钥对，以使用 SSH 连接到开发环境。

您可以使用 JDBC 在用于访问数据集的 VPC 中为 Amazon S3 数据创建开发终端节点。

您可以在本地计算机上安装 Jupyter notebook，并使用它在开发端点上调试和测试 ETL 脚本。或者，您可以使用 Sagemaker 笔记本在 AWS 上的 JupyterLab 中编写 ETL 脚本。请参阅将 SageMaker 笔记本与您的开发端点结合使用。

AWS Glue 使用前缀为 `aws-glue-dev-endpoint` 的名称标记 Amazon EC2 实例。

您可以在开发端点上设置笔记本服务器，以运行具有 AWS Glue 扩展的 PySpark。

AWS Glue 的身份和访问管理

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以进行身份验证（登录）和授权（有权限）使用 AWS Glue 资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

> **Note**
>
> 您可以使用AWS Glue方法或 AWS Lake Formation 授权来授予对 AWS Glue 数据目录中数据的访问权限。您可以使用 AWS Identity and Access Management (IAM) 策略通过方法设置精

细的AWS Glue访问控制。Lake Formation 使用更简单的 GRANT/REVOKE 权限模型，类似于关系数据库系统中的 GRANT/REVOKE 命令。
本节包括有关如何使用 AWS Glue 方法的信息。有关使用 Lake Formation 授权的信息，请参阅 AWS Lake Formation 开发人员指南中的授予 Lake Formation 权限。

主题

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您在 Glue 中所做 AWS 的工作。

服务用户-如果您使用 AWS Glue 服务完成工作，则管理员会为您提供所需的凭证和权限。当你使用更多的 AWS Glue 功能来完成工作时，你可能需要额外的权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 AWS Glue 中的功能，请参阅 对 AWS Glue 身份和访问进行故障排除。

服务管理员 — 如果您负责公司的 AWS Glue 资源，则可能拥有对 Glue 的完全访问 AWS 权限。您的工作是确定您的服务用户应该 AWS 访问哪些 Glue 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要详细了解贵公司如何将 IAM 与 AWS Glue 结合使用，请参阅Glue AWS 如何与 IAM 配合使用。

IAM 管理员 — 如果您是 IAM 管理员，则可能需要详细了解如何编写策略来管理 Glue 的访问 AWS 权限。要查看您可以在 IAM 中使用的基于身份的 AWS Glue 策略示例，请参阅。适用于 AWS Glue 的基于身份的策略示例

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的签署 AWS API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的 多重身份验证和《IAM 用户指南》中的 在 AWS中使用多重身份验证（MFA）。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建帐户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求您以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）使用与身份提供商的联合身份验证 AWS 服务 通过临时证书进行访问。

联合身份是指您的企业用户目录、Web 身份提供商、Identity C enter 目录中的用户，或者任何使用 AWS 服务 通过身份源提供的凭据进行访问的用户。 AWS Directory Service当联合身份访问时 AWS 账户，他们将扮演角色，角色提供临时证书。

要集中管理访问权限，建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和群组，也可以连接并同步到您自己的身份源中的一组用户和群组，以便在您的所有 AWS 账户 和

应用程序中使用。有关 IAM Identity Center 的信息，请参阅《AWS IAM Identity Center 用户指南》中的什么是 IAM Identity Center？

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的 对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 AWS Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的 权限集。
- 临时 IAM 用户权限 – IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 – 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。
- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Simple Storage Service (Amazon S3) 中

存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。

- 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
- 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
- 服务相关角色-服务相关角色是一种链接到的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

- 在 A@@ mazon EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅IAM 用户指南中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM policy，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM policy。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅 IAM 用户指南中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

访问控制列表 (ACL)

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持 ACL 的服务示例。 AWS WAF要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的访问控制列表（ACL）概览。

其他策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界：权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边

界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM 用户指南》中的 IAM 实体的权限边界。

- 服务控制策略 (SCP)-SCP 是 JSON 策略，用于指定组织或组织单位 (OU) 的最大权限。 AWS Organizations AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果在组织内启用了所有功能，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organizations 和 SCP 的更多信息，请参阅《AWS Organizations 用户指南》中的 SCP 的工作原理。
- 会话策略 – 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

Glue AWS 如何与 IAM 配合使用

在使用 IAM 管理对 Glue 的访问 AWS 权限之前，请先了解有哪些 IAM 功能可用于 AWS Glue。

你可以在 AWS Glue 中使用的 IAM 功能

IAM 功能	AWS Glue 支持
基于身份的策略	是
基于资源的策略	部分
策略操作	是
策略资源	是
策略条件键（特定于服务）	是
ACL	否

IAM 功能	AWS Glue 支持
ABAC（策略中的标签）	部分
临时凭证	是
主体权限	否
服务角色	是
服务相关角色	否

要全面了解 G AWS lue 和其他 AWS 服务如何与大多数 IAM 功能配合使用，请参阅 IAM 用户指南中与 IAM 配合使用的AWS 服务。

Glue 基于身份的策略 AWS

支持基于身份的策略	是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM policy。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

AWS Glue 对所有 AWS Glue 操作均支持基于身份的策略 (IAM policy)。通过附加策略，可向其授予创建、访问或修改 AWS Glue 资源（例如 AWS Glue Data Catalog 中的表）的权限。

Glue 基于身份的策略示例 AWS

要查看 Glue AWS 基于身份的策略示例，请参阅。适用于 AWS Glue 的基于身份的策略示例

Glue 中 AWS 基于资源的策略

支持基于资源的策略	部分

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户存取，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当委托人和资源处于不同位置时 AWS 账户，可信账户中的 IAM 管理员还必须向委托人实体（用户或角色）授予访问资源的权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅 IAM 用户指南中的跨账户访问 IAM 中的资源。

> **Note**
>
> 只能使用 AWS Glue 资源策略来管理 Data Catalog 资源的权限。不能将其附加到任何其他 AWS Glue 资源，如任务、触发器、开发终端节点、爬网程序或分类器。
> 每个目录只允许使用一个资源策略，大小限制为 10KB。

在 AWS Glue 中，资源策略附加到目录，目录是前面提到的所有类型的数据目录资源的虚拟容器。每个 AWS 账户在一个 AWS 区域中拥有一个目录，其目录 ID 与 AWS 账户 ID 相同。您无法删除或修改目录。

将对目录的所有 API 调用评估资源策略，其中调用方委托人包含在策略文档的 `"Principal"` 块中。

要查看基于 AWS Glue 资源的策略的示例，请参阅AWS Glue 基于资源的策略示例。

AWS Glue 的政策行动

支持策略操作	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

要查看 Glue 操作 AWS 列表，请参阅《服务授权参考 AWS 》中的 Glue 定义的操作。

Glue AWS 中的策略操作在操作前使用以下前缀：

```
glue
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "glue:action1",
      "glue:action2"
         ]
```

您也可以使用通配符（*）指定多个操作。例如，要指定以单词 Get 开头的所有操作，包括以下操作：

```
"Action": "glue:Get*"
```

要查看示例策略，请参阅 AWS Glue 访问控制策略示例。

AWS Glue 的政策资源

支持策略资源	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称 (ARN) 指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

有关如何使用 ARN 控制 Glue 资源 AWS 访问权限的更多信息，请参阅指定 AWS Glue 资源 ARN。

要查看 Glue 资源 AWS 类型及其 ARN 的列表，请参阅《服务授权参考》中的 AWS Glue 定义的资源。要了解您可以使用哪些操作来指定每种资源的 ARN，请参阅 G AWS lue 定义的操作。

AWS Glue 的策略条件密钥

支持特定于服务的策略条件键	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM policy 元素：变量和标签。

AWS 支持全局条件密钥和特定于服务的条件密钥。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

要查看 Glue AWS 条件键列表，请参阅《服务授权参考》AWS 中的 Glue 条件密钥。要了解可以将条件键与哪些操作和资源一起使用，请参阅 Glue 定义的 AWS 操作。

要查看示例策略，请参阅 使用条件键或上下文键控制设置。

Glue 中的 AWS ACL

支持 ACL	否

访问控制列表 (ACL) 控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

带有 Glue 的 ABA AWS C

支持 ABAC（策略中的标签）	部分

基于属性的访问权限控制（ABAC）是一种授权策略，该策略基于属性来定义权限。在中 AWS，这些属性称为标签。您可以向 IAM 实体（用户或角色）和许多 AWS 资源附加标签。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，您需要使用 `aws:ResourceTag/`*`key-name`*、`aws:RequestTag/`*`key-name`* 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息,请参阅《IAM 用户指南》中的什么是 ABAC？。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

> **Important**
>
> 条件上下文键仅应用于针对爬网程序、作业、触发器和开发端点的 AWS Glue API 操作。有关哪些 API 操作受到影响的更多信息，请参阅 Glue 的条件密 AWS 钥。
> AWS Glue Data Catalog API 操作目前不支持 `aws:referer` 和 `aws:UserAgent` 全局条件上下文键。

要查看基于身份的策略（用于根据资源上的标签来限制对该资源的访问）的示例，请参阅使用标签授权。

在 AWS Glue 中使用临时证书

支持临时凭证	是

当你使用临时证书登录时，有些 AWS 服务 不起作用。有关更多信息，包括哪些 AWS 服务 适用于临时证书，请参阅 IAM 用户指南中的AWS 服务 与 IA M 配合使用的信息。

如果您使用除用户名和密码之外的任何方法登录，则 AWS Management Console 使用的是临时证书。例如，当您 AWS 使用公司的单点登录 (SSO) 链接进行访问时，该过程会自动创建临时证书。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的 切换到角色（控制台）。

您可以使用 AWS CLI 或 AWS API 手动创建临时证书。然后，您可以使用这些临时证书进行访问 AWS。 AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅 IAM 中的临时安全凭证。

Glue 的跨服务主体 AWS 权限

支持转发访问会话（FAS）	否

当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

AWS Glue 的服务角色

支持服务角色	是

服务角色是由一项服务担任、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

> **Warning**
>
> 更改服务角色的权限可能会中断 AWS Glue 的功能。只有当 AWS Glue 提供相关指导时，才能编辑服务角色。

有关为 Glue 创建服务角色的 AWS 详细说明，请参阅步骤 1：为 AWS Glue 服务创建 IAM policy和步骤 2：为 AWS Glue 创建 IAM 角色。

Glue 的 AWS 服务相关角色

支持服务相关角色	否

服务相关角色是一种链接到的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

有关创建或管理服务相关角色的详细信息，请参阅能够与 IAM 搭配使用的AWS 服务。在表中查找服务相关角色列中包含 Yes 的表。选择是链接以查看该服务的服务相关角色文档。

为 AWS Glue 配置 IAM 权限

您可使用 AWS Identity and Access Management（IAM）定义访问由 AWS Glue 使用的资源所需的策略和角色。以下步骤将引导您完成为 AWS Glue 设置权限的各种选项。根据您的业务需求，您可能必须添加或减少对您的资源的访问权限。

> **Note**
>
> 要改为开始为 AWS Glue 使用基本 IAM 权限，请参阅 为 AWS Glue 设置 IAM 权限。

1. 为 AWS Glue 服务创建 IAM policy：创建允许访问 AWS Glue 资源的服务策略。
2. 为 AWS Glue 创建 IAM 角色：创建 IAM 角色并附加 AWS Glue 服务策略和用于您的 Amazon Simple Storage Service（Amazon S3）资源的策略，这些资源供 AWS Glue 使用。
3. 将策略附加到访问 AWS Glue 的用户或组：将策略附加到登录 AWS Glue 控制台的任何用户或组。
4. 为笔记本创建 IAM policy：创建一个笔记本服务器策略，该策略用于在开发端点上创建笔记本服务器。
5. 为笔记本创建 IAM 角色：创建 IAM 角色并附加笔记本服务器策略。
6. 创建用于 Amazon SageMaker 笔记本的 IAM policy：创建在开发端点上创建 Amazon SageMaker 笔记本时使用的 IAM policy。
7. 为 Amazon SageMaker 笔记本创建 IAM 角色：创建 IAM 角色并附加策略，以便在开发端点上创建 Amazon SageMaker 笔记本时授予权限。

步骤 1：为 AWS Glue 服务创建 IAM policy

对于访问其他AWS资源上的数据的任何操作（例如访问您在 Amazon S3 中的对象），AWS Glue 需要代表您访问该资源的权限。您通过使用 AWS Identity and Access Management（IAM）提供这些权限。

> **Note**
>
> 如果您使用了AWS托管式策略 **`AWSGlueServiceRole`**，则可跳过此步骤。

在此步骤中，您将创建一个类似于 `AWSGlueServiceRole` 的策略。您可以在 IAM 控制台中找到最新版本的 `AWSGlueServiceRole`。

为 AWS Glue 创建 IAM policy

此策略为某些 Amazon S3 操作授予管理您账户中某些资源的权限，这些资源是 AWS Glue 代入使用此策略的角色时需要的资源。此策略中指定的某些资源引用了 AWS Glue 对 Amazon S3 存储桶、Amazon S3 ETL 脚本、CloudWatch Logs 和 Amazon EC2 资源使用的默认名称。为简便起见，默认情况下，AWS Glue 会将某些 Amazon S3 对象写入到您的账户中带有前缀 `aws-glue-*` 的存储桶。

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧导航窗格中，选择 Policies（策略）。
3. 请选择创建策略。
4. 在 Create Policy (创建策略) 屏幕上，导航到用于编辑 JSON 的选项卡。使用以下 JSON 语句创建策略文档，然后选择 Review policy (查看策略)。

   > **Note**
   >
   > 添加 Amazon S3 资源所需的任何权限。您可能需要将您的访问策略的资源部分限定为那些需要的资源。

```
{
    "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/*"
            ]
        }
```

```
    ]
}
```

下表描述了此策略授予的权限。

操作	资源	描述
"glue:*"	"*"	授予运行所有 AWS Glue API 操作的权限。
"s3:GetBucketLocation", "s3:ListBucket", "s3:ListAllMyBuckets", "s3:GetBucketAcl",	"*"	允许从爬网程序、任务、开发终端节点和笔记本服务器列示 Amazon S3 存储桶。
"ec2:DescribeVpcEndpoints", "ec2:DescribeRouteTables", "ec2:CreateNetworkInterface", "ec2:DeleteNetworkInterface", "ec2:DescribeNetworkInterfaces", "ec2:DescribeSecurityGroups", "ec2:DescribeSubnets", "ec2:DescribeVpcAttribute",	"*"	允许在运行任务、爬网程序和开发终端节点时设置 Amazon EC2 网络项，如 Virtual Private Cloud（VPC）。
"iam:ListRolePolicies", "iam:GetRole", "iam:GetRolePolicy"	"*"	允许从爬网程序、任务、开发终端节点和笔记本服务器列示 IAM 角色。
"cloudwatch:PutMetricData"	"*"	允许为任务编写 CloudWatch 指标。

操作	资源	描述
"s3:CreateBucket", "s3:PutBucketPublicAccessBlock"	"arn:aws:s3:::aws-glue-*"	允许从任务和笔记本服务器在您的账户中创建 Amazon S3 存储桶。 命名约定：使用名为 aws-glue- 的 Amazon S3 文件夹。 使 AWS Glue 创建阻止公有访问的存储桶。
"s3:GetObject", "s3:PutObject", "s3:DeleteObject"	"arn:aws:s3:::aws-glue-*/*", "arn:aws:s3:::*/*aws-glue-*/*"	允许在存储 ETL 脚本和笔记本服务器位置等对象时在您的账户中获取、放置和删除 Amazon S3 对象。 命名约定：向名称以 aws-glue- 为前缀的 Amazon S3 存储桶或文件夹授予权限。
"s3:GetObject"	"arn:aws:s3:::crawler-public*", "arn:aws:s3:::aws-glue-*"	允许从爬网程序和任务获取示例和教程所使用的 Amazon S3 对象。 命名约定：Amazon S3 存储桶名称以 crawler-public 和 aws-glue- 开头。
"logs:CreateLogGroup", "logs:CreateLogStream", "logs:PutLogEvents"	"arn:aws:logs:*:*:log-group:/aws-glue/*"	允许将日志写入 CloudWatch Logs。 命名约定：AWS Glue 将日志写入到名称以 aws-glue 开头的日志组。

操作	资源	描述
`"ec2:CreateTags", "ec2:DeleteTags"`	`"arn:aws:ec2:*:*:network-interface/*", "arn:aws:ec2:*:*:security-group/*", "arn:aws:ec2:*:*:instance/*"`	允许标记为开发终端节点创建的 Amazon EC2 资源。 命名约定：AWS Glue 使用 aws-glue-service-resource 标记 Amazon EC2 网络接口、安全组和实例。

5. 在 Review Policy (查看策略) 屏幕上，输入您的 Policy Name (策略名称)，例如 GlueServiceRolePolicy。输入可选描述，然后在您对该策略满意时选择 Create policy (创建策略)。

步骤 2：为 AWS Glue 创建 IAM 角色

您需要向您的 IAM 角色授予在 AWS Glue 代表您调用其他服务时可使用的权限。这包括对 Amazon S3 的访问权限，针对用于 AWS Glue 的任何源、目标、脚本和临时目录。爬网程序、任务和开发终端节点需要权限。

您通过使用 AWS Identity and Access Management（IAM）提供这些权限。请向您传递到 AWS Glue 的 IAM 角色添加一个策略。

为 AWS Glue 创建 IAM 角色

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧导航窗格中，选择 Roles (角色)。
3. 选择创建角色。
4. 对于角色类型，选择 AWS Service (亚马逊云科技服务)，找到并选择 Glue，然后选择 Next: Permissions (下一步: 权限)。
5. 在 Attach permissions policy (附加权限策略) 页面上，选择包含所需权限的策略；例如，适用于一般 AWS Glue 权限的 AWS 托管式策略 `AWSGlueServiceRole` 和适用于对 Amazon S3 资源的访问权限的 AWS 托管式策略 AmazonS3FullAccess。然后选择 Next: Review。

> **Note**
>
> 请确保此角色中的策略之一授予针对您的 Amazon S3 源和目标的权限。您可能想要提供您自己的策略来访问特定 Amazon S3 资源。数据源需要 `s3:ListBucket` 和 `s3:GetObject` 权限。数据目标需要`s3:ListBucket`、`s3:PutObject` 和 `s3:DeleteObject` 权限。有关为您的资源创建 Amazon S3 策略的更多信息，请参阅在策略中指定资源。有关示例 Amazon S3 策略，请参阅编写 IAM policy：如何授予对 Amazon S3 存储桶的访问权限。
>
> 如果您计划访问使用 SSE-KMS 加密的 Amazon S3 资源和目标，请附加一个允许 AWS Glue 爬网程序、任务和开发终端节点解密数据的策略。有关更多信息，请参阅使用具有 AWS KMS 托管式密钥的服务器端加密（SSE-KMS）保护数据。
>
> 以下是示例。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "kms:Decrypt"
         ],
         "Resource":[
            "arn:aws:kms:*:account-id-without-hyphens:key/key-id"
         ]
      }
   ]
}
```

6. 对于 Role name (角色名称)，输入您角色的名称，例如 `AWSGlueServiceRoleDefault`。使用以字符串 `AWSGlueServiceRole` 为前缀的名称创建角色,以允许角色从控制台用户传递到服务。AWS Glue 提供了要求 IAM 服务角色以 `AWSGlueServiceRole` 开头的策略。否则，您必须添加一个为您的用户授予对 IAM 角色的 `iam:PassRole` 权限的策略，以匹配您的命名约定。请选择 Create Role(创建角色)。

> **Note**
>
> 当您创建具有角色的笔记本时，该角色将传递至交互式会话，以便同一角色可以在两个位置使用。因此，`iam:PassRole` 权限需要成为角色策略的一部分。

使用以下示例为您的角色创建新策略。将账号替换为您的账号，并将角色名称替换为您的角色名称。

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

步骤 3：将策略附加到访问 AWS Glue 的用户或组

管理员必须使用 AWS Glue 控制台或 AWS Command Line Interface（AWS CLI）为任何用户、组或角色分配权限。您可使用 AWS Identity and Access Management（IAM）通过策略提供这些权限。该步骤介绍向用户或组分配权限。

完成此步骤后，您的用户或组将附加以下策略：

- AWS托管式策略 `AWSGlueConsoleFullAccess` 或自定义策略 GlueConsoleAccessPolicy
- **`AWSGlueConsoleSageMakerNotebookFullAccess`**
- **`CloudWatchLogsReadOnlyAccess`**
- **`AWSCloudFormationReadOnlyAccess`**
- **`AmazonAthenaFullAccess`**

附加内联策略并将其嵌入到用户或组中

您可以将 AWS 托管式策略或内联策略附加到用户或组以便访问 AWS Glue 控制台。在此策略中指定的某些资源引用了 AWS Glue 对 Amazon S3 存储桶、Amazon S3 ETL 脚本、CloudWatch Logs、AWS CloudFormation 和 Amazon EC2 资源使用的默认名称。为简便起见，默认情况下，AWS Glue 会将某些 Amazon S3 对象写入到您的账户中带有前缀 `aws-glue-*` 的存储桶。

> **Note**
>
> 如果您使用了AWS托管式策略 **AWSGlueConsoleFullAccess**，则可跳过此步骤。

> **Important**
>
> AWS Glue 需要代入用于代表您执行工作的角色的权限。要达到此目的，您要将 **iam:PassRole** 权限添加到您的 AWS Glue 用户或组。此策略向以 AWSGlueServiceRole 开头的角色授予对 AWS Glue 服务角色的权限，并向以 AWSGlueServiceNotebookRole 开头的角色授予对您创建笔记本服务器时所需的角色的权限。您还可以针对 iam:PassRole 权限创建您自己的遵循您的命名约定的策略。
>
> 根据安全性最佳实践，建议通过收紧策略来限制访问，从而进一步限制对 Amazon S3 存储桶和 Amazon CloudWatch 日志组的访问。有关示例 Amazon S3 策略，请参阅编写 IAM policy：如何授予对 Amazon S3 存储桶的访问权限。

在此步骤中，您将创建一个类似于 AWSGlueConsoleFullAccess 的策略。您可以在 IAM 控制台中找到最新版本的 AWSGlueConsoleFullAccess。

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择用户或用户组。
3. 在列表中，请选择要在其中嵌入策略的用户或组的名称。
4. 选择 Permissions (权限) 选项卡，然后展开 Permissions policies (权限策略) 部分（如有必要）。
5. 选择 Add Inline policy (添加内联策略) 链接。
6. 在 Create Policy (创建策略) 屏幕上，导航到用于编辑 JSON 的选项卡。使用以下 JSON 语句创建策略文档，然后选择 Review policy (查看策略)。

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

下表描述了此策略授予的权限。

操作	资源	描述
"glue:*"	"*"	授予运行所有 AWS Glue API 操作的权限。 如果您之前已创建策略而未使用 "glue:*" 操作，则必须将以下各个权限添加到策略： • "glue:ListCrawlers" • "glue:BatchGetCrawlers" • "glue:ListTriggers" • "glue:BatchGetTriggers" • "glue:ListDevEndpoints" • "glue:BatchGetDevEndpoints" • "glue:ListJobs" • "glue:BatchGetJobs"

操作	资源	描述
"redshift:DescribeClusters", "redshift:DescribeClusterSubnetGroups"	"*"	允许创建与 Amazon Redshift 的连接。
"iam:ListRoles", "iam:ListRolePolicies", "iam:GetRole", "iam:GetRolePolicy", "iam:ListAttachedRolePolicies"	"*"	允许在使用爬网程序、任务、开发终端节点和笔记本服务器时列示 IAM 角色。
"ec2:DescribeSecurityGroups", "ec2:DescribeSubnets", "ec2:DescribeVpcs", "ec2:DescribeVpcEndpoints", "ec2:DescribeRouteTables", "ec2:DescribeVpcAttribute", "ec2:DescribeKeyPairs", "ec2:DescribeInstances"	"*"	允许在运行任务、爬网程序和开发终端节点时设置 Amazon EC2 网络项，如 VPC。
"rds:DescribeDBInstances"	"*"	允许创建与 Amazon RDS 的连接。
"s3:ListAllMyBuckets", "s3:ListBucket", "s3:GetBucketAcl", "s3:GetBucketLocation"	"*"	允许在使用爬网程序、任务、开发终端节点和笔记本服务器时列示 Amazon S3 存储桶。
"dynamodb:ListTables"	"*"	允许列出 DynamoDB 表。
"kms:ListAliases", "kms:DescribeKey"	"*"	允许使用 KMS 密钥。

操作	资源	描述
"cloudwatch:GetMetricData", "cloudwatch:ListDashboards"	"*"	允许使用 CloudWatch 指标。
"s3:GetObject", "s3:PutObject"	"arn:aws:s3::: aws-glue-*/*", "arn:aws:s3::: */*aws-glue-*/*", "arn:aws:s3::: aws-glue-*"	允许在存储 ETL 脚本和笔记本位置等对象时在您的账户中获取和放置 Amazon S3 对象。 命名约定：向名称以 aws-glue- 为前缀的 Amazon S3 存储桶或文件夹授予权限。
"tag:GetResources"	"*"	允许检索AWS标签。
"s3:CreateBucket", "s3:PutBucketPublicAccessBlock"	"arn:aws:s3::: aws-glue-*"	允许在存储 ETL 脚本和笔记本服务器位置等对象时在您的账户中创建 Amazon S3 存储桶。 命名约定：向名称以 aws-glue- 为前缀的 Amazon S3 存储桶授予权限。 使 AWS Glue 创建阻止公有访问的存储桶。
"logs:GetLogEvents"	"arn:aws:logs:*:*: /aws-glue/*"	允许检索 CloudWatch Logs。 命名约定：AWS Glue 将日志写入到名称以 aws-glue- 开头的日志组。

操作	资源	描述
"cloudformation:CreateStack", "cloudformation:DeleteStack"	"arn:aws:cloudformation:*:*:stack/aws-glue*/*"	允许在使用笔记本服务器时管理 AWS CloudFormation 堆栈。 命名约定：AWS Glue 创建名称以 aws-glue 开头的堆栈。
"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/*"	允许运行开发终端节点和笔记本服务器。
"iam:PassRole"	"arn:aws:iam::*:role/ AWSGlueServiceRole*"	允许 AWS Glue 使用针对以 AWSGlueServiceRole 开头的角色的 PassRole 权限。
"iam:PassRole"	"arn:aws:iam::*:role/ AWSGlueServiceNotebookRole*"	允许 Amazon EC2 使用针对以 AWSGlueServiceNotebookRole 开头的角色的 PassRole 权限。

操作	资源	描述
"iam:PassRole"	"arn:aws:iam::*:role/service-role/AWSGlueServiceRole*"	允许 AWS Glue 使用针对以 service-role/AWSGlueServiceRole 开头的角色的 PassRole 权限。

7. 在 Review policy (查看策略) 屏幕上，输入您的策略的名称，例如 GlueConsoleAccessPolicy。如果您对该策略感到满意，请选择 Create policy (创建策略)。确保屏幕顶部的红框中没有显示错误。更正报告的任何错误。

> **Note**
>
> 如果选择了 Use autoformatting，则每当您打开策略或选择 Validate Policy 时，都会重新设置策略的格式。

附加 AWSGlueConsoleFullAccess 托管策略

您可以附加 AWSGlueConsoleFullAccess 策略以提供 AWS Glue 控制台用户所需的权限。

> **Note**
>
> 如果您已针对 AWS Glue 控制台访问创建自己的策略，则可跳过此步骤。

1. 登录到 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择策略。
3. 在策略列表中，选中 AWSGlueConsoleFullAccess 旁边的复选框。您可以使用 Filter 菜单和搜索框来筛选策略列表。
4. 选择 Policy actions（策略操作），然后选择 Attach（附加）。
5. 选择要将策略附加到的用户。您可以使用 Filter（筛选条件）菜单和搜索框来筛选委托人实体列表。在选择要将策略附加到的用户后，选择 Attach policy (附加策略)。

附加 **AWSGlueConsoleSageMakerNotebookFullAccess** 托管式策略

可将 AWSGlueConsoleSageMakerNotebookFullAccess 策略附加到用户，以管理在 AWS Glue 控制台中创建的 SageMaker 笔记本。除了其他必要的 AWS Glue 控制台权限外，该策略还授予管理 SageMaker 笔记本所需资源的访问权限。

1. 登录到 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择策略。
3. 在策略列表中，选中 AWSGlueConsoleSageMakerNotebookFullAccess 旁边的复选框。您可以使用 Filter 菜单和搜索框来筛选策略列表。
4. 选择 Policy actions（策略操作），然后选择 Attach（附加）。
5. 选择要将策略附加到的用户。您可以使用 Filter（筛选条件）菜单和搜索框来筛选委托人实体列表。在选择要将策略附加到的用户后，选择 Attach policy (附加策略)。

附加 CloudWatchLogsReadOnlyAccess 托管策略

您可以将 CloudWatchLogsReadOnlyAccess 策略附加到用户以查看由 AWS Glue 在 CloudWatch Logs 控制台上创建的日志。

1. 登录到 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择策略。
3. 在策略列表中，选中 CloudWatchLogsReadOnlyAccess 旁边的复选框。您可以使用 Filter 菜单和搜索框来筛选策略列表。
4. 选择 Policy actions（策略操作），然后选择 Attach（附加）。
5. 选择要将策略附加到的用户。您可以使用 Filter（筛选条件）菜单和搜索框来筛选委托人实体列表。在选择要将策略附加到的用户后，选择 Attach policy (附加策略)。

附加 AWSCloudFormationReadOnlyAccess 托管策略

您可以将 AWSCloudFormationReadOnlyAccess 策略附加到用户以查看 AWS Glue 在 AWS CloudFormation 控制台上使用的 AWS CloudFormation 堆栈。

1. 登录到 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。

2. 在导航窗格中，选择策略。
3. 在策略列表中，选中 AWSCloudFormationReadOnlyAccess 旁边的复选框。您可以使用 Filter 菜单和搜索框来筛选策略列表。
4. 选择 Policy actions（策略操作），然后选择 Attach（附加）。
5. 选择要将策略附加到的用户。您可以使用 Filter（筛选条件）菜单和搜索框来筛选委托人实体列表。在选择要将策略附加到的用户后，选择 Attach policy (附加策略)。

附加 AmazonAthenaFullAccess 托管策略

您可以将 AmazonAthenaFullAccess 策略附加到用户，以在 Athena 控制台中查看 Amazon S3 数据。

1. 登录到 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择策略。
3. 在策略列表中，选中 AmazonAthenaFullAccess 旁边的复选框。您可以使用 Filter 菜单和搜索框来筛选策略列表。
4. 选择 Policy actions（策略操作），然后选择 Attach（附加）。
5. 选择要将策略附加到的用户。您可以使用 Filter（筛选条件）菜单和搜索框来筛选委托人实体列表。在选择要将策略附加到的用户后，选择 Attach policy (附加策略)。

步骤 4：创建用于笔记本服务器的 IAM policy

如果您计划将笔记本与开发终端节点结合使用，则必须在创建笔记本服务器时指定权限。您通过使用 AWS Identity and Access Management（IAM）提供这些权限。

此策略为某些 Amazon S3 操作授予管理您账户中某些资源的权限，这些资源是 AWS Glue 代入使用此策略的角色时需要的资源。此策略中指定的某些资源引用了 AWS Glue 对 Amazon S3 存储桶、Amazon S3 ETL 脚本和 Amazon EC2 资源使用的默认名称。为简便起见，默认情况下，AWS Glue 会将某些 Amazon S3 对象写入到您账户中带有前缀 `aws-glue-*` 的存储桶。

> **Note**
>
> 如果您使用了AWS托管式策略 **`AWSGlueServiceNotebookRole`**，则可跳过此步骤。

在此步骤中，您将创建一个类似于 `AWSGlueServiceNotebookRole` 的策略。您可以在 IAM 控制台中找到最新版本的 `AWSGlueServiceNotebookRole`。

为笔记本创建 IAM policy

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧导航窗格中，选择 Policies（策略）。
3. 请选择创建策略。
4. 在 Create Policy (创建策略) 屏幕上，导航到用于编辑 JSON 的选项卡。使用以下 JSON 语句创建策略文档，然后选择 Review policy (查看策略)。

```
{
   "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/*"
         ]
      }
   ]
}
```

下表描述了此策略授予的权限。

操作	资源	描述
"glue:*"	"*"	授予运行所有 AWS Glue API 操作的权限。
"s3:GetBucketLocation", "s3:ListBucket", "s3:ListAllMyBuckets", "s3:GetBucketAcl"	"*"	允许通过笔记本服务器列示 Amazon S3 存储桶。

操作	资源	描述
"s3:GetObject"	"arn:aws:s3:::crawler-public*", "arn:aws:s3:::aws-glue-*"	允许通过笔记本获取示例和教程所使用的 Amazon S3 对象。 命名约定：Amazon S3 存储桶名称以 crawler-public 和 aws-glue- 开头。
"s3:PutObject", "s3:DeleteObject"	"arn:aws:s3:::aws-glue*"	允许通过笔记本在您的账户中放置和删除 Amazon S3 对象。 命名约定：使用名为 aws-glue- 的 Amazon S3 文件夹。
"ec2:CreateTags", "ec2:DeleteTags"	"arn:aws:ec2:*:*:network-interface/*", "arn:aws:ec2:*:*:security-group/*", "arn:aws:ec2:*:*:instance/*"	允许标记为笔记本服务器创建的 Amazon EC2 资源。 命名约定：AWS Glue 使用 aws-glue-service-resource 标记 Amazon EC2 实例。

5. 在 Review Policy (查看策略) 屏幕上，输入您的 Policy Name (策略名称)，例如 GlueServiceNotebookPolicyDefault。输入可选描述，然后在您对该策略满意时选择 Create policy (创建策略)。

步骤 5：创建用于笔记本服务器的 IAM 角色

如果您计划将笔记本与开发终端节点结合使用，则需要为 IAM 角色授予权限。您可使用 AWS Identity and Access Management（IAM）通过 IAM 角色提供这些权限。

> **Note**
>
> 使用 IAM 控制台创建 IAM 角色时，控制台自动创建实例配置文件，按相应的角色为文件命名。

为笔记本创建 IAM 角色

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧导航窗格中，选择 Roles (角色)。
3. 选择创建角色。
4. 对于角色类型，选择 AWS Service（服务），找到并选择 EC2，然后选择 EC2 使用案例，再选择 Next: Permissions（下一步：权限）。
5. 在 Attach permissions policy (附加权限策略) 页面上，选择包含所需权限的策略；例如，适用于一般 AWS Glue 权限的 AWSGlueServiceNotebookRole 和适用于对 Amazon S3 资源的访问权限的 AWS 托管式策略 AmazonS3FullAccess。然后选择 Next: Review。

> **Note**
>
> 请确保此角色中的策略之一授予针对您的 Amazon S3 源和目标的权限。此外，确认您的策略支持对您在创建笔记本服务器时存储笔记本的位置的完全访问。您可能想要提供您自己的策略来访问特定 Amazon S3 资源。有关为您的资源创建 Amazon S3 策略的更多信息，请参阅在策略中指定资源。
> 如果您计划访问使用 SSE-KMS 加密的 Amazon S3 源和目标，请附加一个允许笔记本解密数据的策略。有关更多信息，请参阅使用具有 AWS KMS 托管式密钥的服务器端加密（SSE-KMS）保护数据。
> 以下是示例。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "kms:Decrypt"
         ],
         "Resource":[
            "arn:aws:kms:*:account-id-without-hyphens:key/key-id"
         ]
      }
   ]
}
```

6. 对于角色名称，请为您的角色输入一个名称。使用以字符串 AWSGlueServiceNotebookRole 为前缀的名称创建角色，以允许角色从控制台用户传递到笔记本服务器。AWS Glue 提供了要求 IAM 服务角色以 AWSGlueServiceNotebookRole 开头的策略。否则，您必须向您的用户添加一个策略以允许适用于 IAM 角色的 iam:PassRole 权限与您的命名约定匹配。例如，输入 AWSGlueServiceNotebookRoleDefault。然后选择创建角色。

步骤 6：为 SageMaker 笔记本创建 IAM policy

如果您计划将 SageMaker 笔记本与开发终端节点结合使用，则必须在创建笔记本时指定权限。您通过使用 AWS Identity and Access Management（IAM）提供这些权限。

为 SageMaker 笔记本创建 IAM policy

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧导航窗格中，选择 Policies（策略）。
3. 请选择创建策略。
4. 在 Create Policy (创建策略) 页面上，导航到用于编辑 JSON 的选项卡。使用以下 JSON 语句创建一个策略文档。编辑环境的 *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/*"
            ]
         }
    ]
}
```

然后，选择查看策略。

下表描述了此策略授予的权限。

操作	资源	描述
"s3:ListBucket*"	"arn:aws:s3::: *bucket-name* "	授予对 Amazon S3 存储桶的列出权限
"s3:GetObject"	"arn:aws:s3::: *bucket-name* *"	授予权限以获取 SageMaker 笔记本使用的 Amazon S3 对象。
"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-*"	授予从笔记本向 Amazon CloudWatch logs 写入日志的权限。 命名约定：向名称以 aws-glue 开头的日志组写入内容。
"glue:UpdateDevEndpoint", "glue:GetDevEndpoint", "glue:GetDevEndpoints"	"arn:aws:glue: *region-code* :*account-id* :devEndpoint/*"	授予权限以使用 SageMaker 笔记本中的开发终端节点。
"sagemaker:ListTags"	"arn:aws:sagemaker : *region-code* :*account-id* :notebook-instance /*"	授予返回 SageMaker 资源标签所需的权限。SageMaker 笔记本需要 aws-glue-dev-endpoint 标签才能连接到开发终端节点。

5. 在 Review Policy (查看策略) 屏幕上，输入您的 Policy Name (策略名称)，例如 AWSGlueSageMakerNotebook。输入可选描述，然后在您对该策略满意时选择 Create policy (创建策略)。

步骤 7：为 SageMaker 笔记本创建 IAM 角色

如果您计划将 SageMaker 笔记本与开发终端节点结合使用，则需要为 IAM 角色授予权限。您可使用 AWS Identity and Access Management（IAM）通过 IAM 角色提供这些权限。

为 SageMaker 笔记本创建 IAM 角色

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧导航窗格中，选择 Roles (角色)。
3. 选择创建角色。
4. 对于角色类型，选择 AWS Service (亚马逊云科技服务)，找到并选择 SageMaker，然后选择 SageMaker - Execution (SageMaker - 执行) 使用案例。然后选择下一步：权限。
5. 在 Attach permissions policy (附加权限策略) 页面上，选择包含所需权限的策略；例如，AmazonSageMakerFullAccess。选择 下一步: 审核。

   如果您计划访问使用 SSE-KMS 加密的 Amazon S3 源和目标，则附加一个允许笔记本解密数据的策略，如以下示例所示。有关更多信息，请参阅使用具有 AWS KMS 托管式密钥的服务器端加密（SSE-KMS）保护数据。

   ```
   {
      "Version":"2012-10-17",
      "Statement":[
         {
            "Effect":"Allow",
            "Action":[
               "kms:Decrypt"
            ],
            "Resource":[
               "arn:aws:kms:*:account-id-without-hyphens:key/key-id"
            ]
         }
      ]
   }
   ```

6. 对于角色名称，请为您的角色输入一个名称。要允许角色从控制台用户传递到 SageMaker，请使用以字符串`AWSGlueServiceSageMakerNotebookRole` 为前缀的名称。AWS Glue 提供了要求 IAM 角色以 `AWSGlueServiceSageMakerNotebookRole` 开头的策略。否则，您必须向您的用户添加一个策略以允许适用于 IAM 角色的 `iam:PassRole` 权限与您的命名约定匹配。

例如，输入 AWSGlueServiceSageMakerNotebookRole-Default，然后选择 Create role (创建角色)。

7. 创建角色之后，附加策略以允许从 AWS Glue 创建 SageMaker 笔记本所需的其他权限。

   选择您刚刚创建的角色 AWSGlueServiceSageMakerNotebookRole-Default，然后选择 Attach policy (附加策略)。将您创建的名为 AWSGlueSageMakerNotebook 的策略附加到角色。

AWS Glue 访问控制策略示例

本部分包含基于身份的（IAM）访问控制策略和 AWS Glue 资源策略的示例。

目录

适用于 AWS Glue 的基于身份的策略示例

默认情况下，用户和角色没有创建或修改 AWS Glue 资源的权限。他们也无法使用 AWS Management Console、AWS Command Line Interface（AWS CLI）或 AWS API 执行任务。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM policy。管理员随后可以向角色添加 IAM policy，用户可以代入角色。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅 IAM 用户指南中的 创建 IAM policy。

有关 AWS Glue 定义的操作和资源类型的详细信息，包括每种资源类型的 ARN 格式，请参阅服务授权参考中的 AWS Glue 的操作、资源和条件键。

> **Note**
>
> 本节中提供的示例均使用 `us-west-2` 区域。您可以将其替换为要使用的 AWS 区域。

主题

策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 AWS Glue 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下准则和建议：

- AWS 托管策略及转向最低权限许可入门 - 要开始向用户和工作负载授予权限，请使用 AWS 托管策略来为许多常见使用场景授予权限。您可以在 AWS 账户 中找到这些策略。我们建议通过定义特定于您的使用场景的 AWS 客户管理型策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略 或 工作职能的 AWS 托管式策略。
- 应用最低权限 – 在使用 IAM policy 设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM policy 中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果通过特定（AWS 服务例如 AWS CloudFormation）使用服务操作，您还可以使用条件来授予对服务操作的访问权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM policy，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM policy 语言 (JSON) 和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Acess Analyzer 策略验证。
- Require multi-factor authentication（MFA）[需要多重身份验证（MFA）] – 如果您所处的场景要求您的 AWS 账户 中有 IAM 用户或根用户，请启用 MFA 来提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的 配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实操。

资源级权限仅应用于特定的 AWS Glue 对象

您只能定义 AWS Glue 中特定对象的精细控制。因此，您必须编写客户端的 IAM policy，以便允许 `Resource` 语句的 Amazon Resource Name（ARN）的 API 操作与不允许使用 API 的操作不混合。

例如，以下 IAM policy 允许 `GetClassifier` 和 `GetJobRun` 的 API 操作。它将 `Resource` 定义为 `*`，因为 AWS Glue 不允许分类符和作业运行的 ARN。由于允许将 ARN 用于 `GetDatabase` 和 `GetTable` 等特定 API 操作，因此，可以在策略的第二部分中指定 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"
            ]
        }
    ]
}
```

有关允许 ARN 的 AWS Glue 对象的列表，请参阅资源 ARN。

使用 AWS Glue 控制台

要访问 AWS Glue 控制台，您必须拥有一组最低的权限。这些权限必须允许您列出和查看有关您的 AWS 账户 中的 AWS Glue 资源的详细信息。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于只需要调用 AWS CLI 或 AWS API 的用户，无需为其提供最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

为确保用户和角色仍可使用 AWS Glue 控制台，请同时将 AWS Glue *ConsoleAccess* 或 *ReadOnly* AWS 托管策略添加到实体。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

某个用户要能够使用 AWS Glue 控制台，必须拥有一组允许其使用其 AWS 账户的 AWS Glue 资源的最低权限。除这些 AWS Glue 权限以外，控制台还需要来自以下服务的权限：

- 用于显示日志的 Amazon CloudWatch Logs 权限。
- 用于列出并传递角色的 AWS Identity and Access Management（IAM）权限。

- 用于处理堆栈的 AWS CloudFormation 权限。
- 用于列出 VPC、子网、安全组、实例和其他对象的 Amazon Elastic Compute Cloud（Amazon EC2）权限。
- 用于列出存储桶和对象以及检索和保存脚本的 Amazon Simple Storage Service（Amazon S3）权限。
- 用于使用集群的 Amazon Redshift 权限。
- 用于列出实例的 Amazon Relational Database Service（Amazon RDS）权限。

有关用户查看和使用 AWS Glue 控制台所需的权限的更多信息，请参阅 步骤 3：将策略附加到访问 AWS Glue 的用户或组。

如果创建比必需的最低权限更为严格的 IAM policy，对于附加了该 IAM policy 的用户，控制台将无法按预期正常运行。为确保这些用户仍可使用 AWS Glue 控制台，同时如 AWS 的托管（预定义）策略 AWS Glue 中所述附加 `AWSGlueConsoleFullAccess` 托管策略。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上完成此操作或者以编程方式使用 AWS CLI 或 AWS API 所需的权限。

```
{
    "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": "*"
        }
    ]
}
```

授予对表的只读权限

以下策略授予对数据库 db1 中的 books 表的只读权限。有关资源的 Amazon Resource Name（ARN）的更多信息，请参阅 数据目录 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"
      ]
    }
  ]
}
```

此策略授予对名为 db1 的数据库中的名为 books 的表的只读权限。要授予对表的 Get 权限，同时需要对目录和数据库资源授予这一权限。

以下策略授予在数据库 db1 中创建表 tb1 所需的最低权限：

```
{
```

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

通过 GetTables 权限筛选表

假设在数据库 db1 中有三个表，即 customers、stores 和 store_sales。以下策略向 stores 和 store_sales 授予 GetTables 权限，但不向 customers 授予此权限。当您使用此策略调用 GetTables 时，结果将只包含两个授权表（不返回 customers 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"
            ]
        }
    ]
}
```

通过使用 store* 匹配以 store 开头的任何表名称，可简化上面的策略。

```
{
    "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*"
            ]
        }
    ]
}
```

同样，使用 /db1/* 匹配 db1 中的所有表，以下策略将授予对 db1 中所有表的 GetTables 访问权限。

```
{
    "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/*"
            ]
        }
    ]
}
```

如果未提供表 ARN，调用 GetTables 将成功，但会返回空列表。

```
{
```

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

如果策略中缺少数据库 ARN，则调用 GetTables 将失败，并出现 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/*"
            ]
        }
    ]
}
```

授予对表和所有分区的完整访问权限

以下策略授予对数据库 db1 中名为 books 的表的所有权限。这包括对表本身、对其存档版本及其所有分区的读取和写入权限。

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

在实践中，可以简化上述策略。

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

请注意，精细访问控制的最小粒度在表级别。这意味着，您不能向用户授予对表中某些分区的访问权限，而不授予对表中其他分区的访问权限，或授予对表中部分列的访问权限，而不授予对其他列的访问权限。用户要么可以访问表的所有内容，要么不能访问表的任何内容。

通过名称前缀和显式拒绝控制访问权限

在本示例中，假设您的 AWS Glue 数据目录中的数据库和表使用名称前缀来组织。发展阶段的数据库具有名称前缀 dev-，生产阶段的数据库具有名称前缀 prod-。您可以使用以下策略向开发人员授予对具有 dev- 前缀的所有数据库、表、UDF 等的完整访问权限。但是，您会对带有 prod- 前缀的所有内容授予只读访问权限。

```
{
    "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-*"
            ]
        }
    ]
}
```

上述策略中的第二条语句使用显式 deny。您可以使用显式 deny 覆盖被授予委托人的任何 allow 权限。这样，您就可以锁定关键资源的访问权限，并阻止其他策略意外授予这些资源的访问权限。

在上述示例中，尽管第一条语句授予对 prod- 资源的完整访问权限，第二条语句则显式撤消这些资源的写入访问权限，只保留 prod- 资源的读取访问权限。

使用标签授权

例如，假设您想将对触发器 t2 的访问限制为您账户中名为 Tom 的特定用户。所有其他用户（包括 Sam）都有权访问触发器 t1。触发器 t1 和 t2 具有以下属性。

```
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 管理员已将标签值 Tom (aws:ResourceTag/Name": "Tom") 附加到触发器 t2。AWS Glue 管理员还为 Tom 提供了一个 IAM policy，其中包含基于标签的条件语句。因此，Tom 只能使用对带标签值 Tom 的资源有效的 AWS Glue 操作。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
         "Effect": "Allow",
         "Action": "glue:*",
         "Resource": "*",
         "Condition": {
             "StringEquals": {
                "aws:ResourceTag/Name": "Tom"
              }
          }
         }
    ]
}
```

当 Tom 尝试访问触发器 t1 时，他收到一条指示访问被拒绝的消息。同时，他可以成功检索触发器 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 无法使用复数 GetTriggers API 操作列出触发器，因为此操作不支持对标签进行筛选。

为了向 Tom 提供对 GetTriggers 的访问权限，AWS Glue 管理员创建了一个将权限拆分为两部分的策略。一个部分允许 Tom 使用 GetTriggers API 操作访问所有触发器。另一个部分允许 Tom 访问使用值 Tom 标记的 API 操作。利用此策略，Tom 可以使用 GetTriggers 和 GetTrigger 来访问触发器 t2。

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

```
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Name": "Tom"
                }
            }
        }
    ]
}
```

使用标签拒绝访问

另一种资源策略方法是明确拒绝对资源的访问。

> **Important**
>
> 显式拒绝策略不适用于复数 API 操作（如 `GetTriggers`）。

在以下示例策略中，允许所有 AWS Glue 任务操作。但是，第二条 `Effect` 语句明确拒绝访问带有 `Team` 键和 `Special` 值标记的任务。

当管理员将以下策略附加到身份时，该身份可以访问除标记为 `Team` 键和 `Special` 值之外的所有任务。

```
{
  "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"
        }
      }
```

```
    }
  ]
}
```

将标签与 List 和 Batch API 操作结合使用

编写资源策略的第三种方法是允许使用 List API 操作访问资源以列出标签值的资源。然后，使用相应的 Batch API 操作以允许访问特定资源的详细信息。利用此方法，管理员无需允许访问复数 GetCrawlers、GetDevEndpoints、GetJobs 或 GetTriggers API 操作。相反，您可以允许使用以下 API 操作列出资源：

- ListCrawlers
- ListDevEndpoints
- ListJobs
- ListTriggers

此外，您可以允许使用以下 API 操作获取有关各个资源的详细信息：

- BatchGetCrawlers
- BatchGetDevEndpoints
- BatchGetJobs
- BatchGetTriggers

作为管理员，要使用此方法，您可以执行以下操作：

1. 将标签添加到您的爬网程序、开发终端节点、作业和触发器。
2. 拒绝用户对 Get API 操作（例如 GetCrawlers、GetDevEndponts、GetJobs 和 GetTriggers）的访问。
3. 要使用户能够查明他们有权访问的标记资源，请允许用户访问 List API 操作，例如 ListCrawlers、ListDevEndponts、ListJobs 和 ListTriggers。
4. 拒绝用户对 AWS Glue 标记 API（例如 TagResource 和 UntagResource）的访问。
5. 允许用户使用 BatchGet API 操作（例如 BatchGetCrawlers、BatchGetDevEndponts、BatchGetJobs 和 BatchGetTriggers）访问资源详细信息。

例如，在调用 ListCrawlers 操作时，请提供标签值以匹配用户名。然后，结果是与提供的标签值匹配的爬网程序的列表。向 BatchGetCrawlers 提供名称列表以获取有关每个带给定标签的爬网程序的详细信息。

例如，如果 Tom 应只能检索标记了 Tom 的触发器的详细信息，管理员可将标签添加到 Tom 的触发器，拒绝所有用户对 GetTriggers API 操作的访问，并允许所有用户访问 ListTriggers 和 BatchGetTriggers。

以下是 AWS Glue 管理员向 Tom 授予的资源策略。在此策略的第一个部分中，GetTriggers 拒绝 AWS Glue API 操作。在此策略的第二个部分中，所有资源允许 ListTriggers。但在第三个部分中，允许使用 BatchGetTriggers 访问标记了 Tom 的那些资源。

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

```
    ]
}
```

通过使用上一个示例中的相同触发器，Tom 可以访问触发器 t2 而不是触发器 t1。以下示例显示了 Tom 尝试使用 BatchGetTriggers 访问 t1 和 t2 时的结果。

```
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.
```

以下示例显示了 Tom 尝试通过同一 BatchGetTriggers 调用访问触发器 t2 和触发器 t3（不存在）时的结果。请注意，由于 Tom 有权访问触发器 t2 且该触发器存在，因此，仅返回 t2。虽然 Tom 有权访问触发器 t3，但触发器 t3 不存在，因此在 "TriggersNotFound": [] 列表的响应中将返回 t3。

```
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 * * ? *)"
    }
}
```

使用条件键或上下文键控制设置

授予创建和更新任务的权限时，可以使用条件键或上下文键。以下部分讨论了这些键：

- 使用条件键控制设置的策略
- 控制使用上下文键控制设置的策略

使用条件键控制设置的策略

AWS Glue 提供了三个 IAM 条件键，分别是 `glue:VpcIds`、`glue:SubnetIds` 和 `glue:SecurityGroupIds`。授予创建和更新任务的权限时，可以使用 IAM policy 中的条件键。您可以使用此设置确保创建（或更新）的作业或会话不会在所需 VPC 环境之外运行。VPC 设置信息不是 `CreateJob` 请求中的直接输入，而是从指向 AWS Glue 连接的任务“连接”字段推断得出。

示例用法

使用所需的 VpcId "vpc-id1234"、SubnetIds 和 SecurityGroupIds 创建名为 "traffic-monitored-connection"（流量监控连接）的 AWS Glue 网络类型连接。

为 IAM policy 中的 `CreateJob` 和 `UpdateJob` 操作指定条件键条件。

```
{
  "Effect": "Allow",
  "Action": [
    "glue:CreateJob",
    "glue:UpdateJob"
  ],
  "Resource": [
    "*"
  ],
  "Condition": {
    "ForAnyValue:StringLike": {
      "glue:VpcIds": [
        "vpc-id1234"
      ]
    }
```

```
  }
}
```

您可以创建类似的 IAM policy，以禁止在不指定连接信息的情况下创建 AWS Glue 任务。

限制 VPC 上的会话

要强制创建的会话在指定的 VPC 内运行，您可以通过对 `glue:CreateSession` 操作添加 `Deny` 效果来限制角色权限，条件是 glue:vpc-id 不等于 vpc-<123>。例如：

```
"Effect": "Deny",
"Action": [
    "glue:CreateSession"
 ],
"Condition": {
    "StringNotEquals" : {"glue:VpcIds" : ["vpc-123"]}
}
```

您还可以将创建的会话强制在 VPC 内运行，方法是为 `Deny` 是空的 `glue:CreateSession` 操作添加 `glue:vpc-id` 效果。例如：

```
{
    "Effect": "Deny",
    "Action": [
        "glue:CreateSession"
    ],
      "Condition": {
        "Null": {"glue:VpcIds": true}
    }
},
{
    "Effect": "Allow",
    "Action": [
        "glue:CreateSession"
    ],
    "Resource": ["*"]
}
```

控制使用上下文键控制设置的策略

AWS Glue 为 AWS Glue 提供给任务和开发人员端点的每个角色会话提供了上下文键 (glue:CredentialIssuingService= glue.amazonaws.com)。这允许您对 AWS Glue 脚本执行的操作实施安全控制。AWS Glue 还为每个角色会话提供了另一个上下文键（glue:RoleAssumedBy=glue.amazonaws.com），其中 AWS Glue 会代表客户调用另一个 AWS 服务（不是通过任务/开发端点，而是直接通过 AWS Glue 服务）。

示例用法

在 IAM policy 中指定条件权限，并将其附上 AWS Glue 任务要使用的角色。这确保了可以基于角色会话是否用于 AWS Glue 任务运行时环境而允许/拒绝某些操作。

```
{
    "Effect": "Allow",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::confidential-bucket/*",
    "Condition": {
        "StringEquals": {
            "glue:CredentialIssuingService": "glue.amazonaws.com"
        }
    }
}
```

拒绝某个身份创建数据预览会话

该部分包含一个 IAM policy 示例，用于拒绝某个身份创建数据预览会话。将此策略附加到身份，该身份与数据预览会话在运行期间使用的角色是分开的。

```
{
    "Sid": "DatapreviewDeny",
    "Effect": "Deny",
     "Action": [
           "glue:CreateSession"
     ],
     "Resource": [
          "arn:aws:glue:*:*:session/glue-studio-datapreview*"
      ]
 }
```

AWS Glue 基于资源的策略示例

本部分包含基于资源的策略示例，其中包括用于授予跨账户存取的策略。

此示例使用 AWS Command Line Interface（AWS CLI）与 AWS Glue 服务 API 操作进行交互。您可以使用 AWS Glue 控制台或使用 AWS SDK 之一执行相同操作。

> **Important**
>
> 通过更改 AWS Glue 资源策略，您可能意外撤消您账户中现有 AWS Glue 用户的权限并导致意外中断。仅在开发或测试账户中尝试这些示例，并确保它们不会中断任何现有的工作流，然后再进行更改。

主题

将基于资源的策略用于 AWS Glue 的注意事项

> **Note**
>
> IAM policy 和 AWS Glue 资源策略都需要几秒钟进行传播。附加新策略后，您可能会注意到，旧策略仍有效，直至新策略传播到整个系统。

可使用以 JSON 格式编写的策略文档来创建或修改资源策略。策略语法与基于身份的 IAM policy 相同（请参阅 IAM JSON 策略参考），但存在以下例外：

- 每条策略语句需要一个 `"Principal"` 或 `"NotPrincipal"` 块。
- `"Principal"` 或 `"NotPrincipal"` 必须识别有效的现有主体。不允许使用通配符模式（如 `arn:aws:iam::`*`account-id`*`:user/*`）。
- 策略中的 `"Resource"` 块要求所有资源 ARN 均与以下正则表达式语法匹配（其中第一个 `%s` 为 *`region`*，第二个 `%s` 为 *`account-id`*）：

```
*arn:aws:glue:%s:%s:(\*|[a-zA-Z\*]+\/?.*)
```

例如，arn:aws:glue:us-west-2:*account-id*:* 和 arn:aws:glue:us-west-2:*account-id*:database/default 都允许使用，但不允许 *。

- 与基于身份的策略不同，AWS Glue 资源策略只能包含属于该策略附加到的目录的资源的 Amazon 资源名称（ARN）。此类 ARN 总是以 arn:aws:glue: 开头。
- 策略不能导致创建它的身份被锁定，无法进一步创建或修改策略。
- 资源策略 JSON 文档的大小不能超过 10 KB。

使用资源策略控制同一账户中的访问

在此示例中，账户 A 中的管理员用户创建一个资源策略，该策略向账户 A 中的 IAM 用户 Alice 授予对目录的完整访问权限。Alice 未附加 IAM policy。

为执行此操作，管理员用户需运行以下 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:*"
      ]
    }
  ]
}'
```

不是在 AWS CLI 命令中输入 JSON 策略文档，您可以将策略文档保存在一个文件中，然后在 AWS CLI 命令中引用该文件路径，前缀为 file://。下面是具体操作示例。

```
$ 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
```

在此资源策略传播后，Alice 可以访问账户 A 中的所有 AWS Glue 资源，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"}
```

为了响应 Alice 的 `get-table` 调用，AWS Glue 服务将返回以下内容。

```
{
  "Table": {
    "Name": "tbl1",
    "PartitionKeys": [],
    "StorageDescriptor": {
        ......
    },
```

```
    ......
  }
}
```

AWSAWS Glue 的托管策略

AWS 托管策略是由创建和管理的独立策略 AWS。 AWS 托管策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住， AWS 托管策略可能不会为您的特定用例授予最低权限权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于您的使用场景的客户管理型策略来进一步减少权限。

您无法更改 AWS 托管策略中定义的权限。如果 AWS 更新 AWS 托管策略中定义的权限，则更新会影响该策略所关联的所有委托人身份（用户、组和角色）。 AWS 最有可能在启动新的 API 或现有服务可以使用新 AWS 服务 的 API 操作时更新 AWS 托管策略。

有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

AWS 的托管（预定义）策略 AWS Glue

AWS 通过提供由创建和管理的独立 IAM 策略来解决许多常见用例 AWS。这些 AWS 托管策略为常见用例授予必要的权限，这样您就可以不必调查需要哪些权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

您可以将以下 AWS 托管策略附加到账户中的身份，这些策略特定于用例场景，AWS Glue并按用例场景进行分组：

- AWSGlueConsoleFullAccess— 当策略所关联的身份使用时，授予对AWS Glue资源的完全访问权限 AWS Management Console。如果遵循此策略中指定的资源的命名约定，则用户具有完全控制台功能。此策略通常附加到 AWS Glue 控制台的用户。
- AWSGlueServiceRole— 授予访问代表您运行各种AWS Glue进程所需的资源的权限。这些资源包括 AWS Glue Amazon S3、IAM、 CloudWatch 日志和亚马逊 EC2。如果您遵循此策略中指定的资源的命名约定，则 AWS Glue 进程具有所需的权限。此策略通常附加到在定义爬网程序、作业和开发终端节点时指定的角色。
- AwsGlueSessionUserRestrictedServiceRole— 提供对除会话之外的所有AWS Glue资源的完全访问权限。允许用户仅创建和使用与用户关联的交互式会话。此策略包括管理其他 AWS 服务中的AWS Glue资源所需的其他权限。AWS Glue该策略还允许为其他 AWS 服务中的AWS Glue资源添加标签。

> **Note**
>
> 若要完全实现安全益处，请勿将此策略授予分配到 `AWSGlueServiceRole`、`AWSGlueConsoleFullAccess` 或 `AWSGlueConsoleSageMakerNotebookFullAccess` 策略的用户。

- AwsGlueSessionUserRestrictedPolicy— 仅当提供了与工作负责人的 AWS 用户 ID 匹配的标签密钥 “owner” 和值时，才允许使用 `CreateSession` API 操作创建AWS Glue交互式会话。此身份策略已附上调用 `CreateSession` API 操作的 IAM用户。此政策还允许受让人与使用与其 AWS 用户 ID 匹配的 “所有者” 标签和值创建的AWS Glue交互式会话资源进行交互。此策略拒绝在创建会话后从 AWS Glue 会话资源中更改或删除“拥有者”标签的权限。

  > **Note**
  >
  > 若要完全实现安全益处，请勿将此策略授予分配到 `AWSGlueServiceRole`、`AWSGlueConsoleFullAccess` 或 `AWSGlueConsoleSageMakerNotebookFullAccess` 策略的用户。

- AwsGlueSessionUserRestrictedNotebookServiceRole— 提供对AWS Glue Studio笔记本会话的足够访问权限，以便与特定的交AWS Glue互式会话资源进行交互。这些资源是使用 “own AWS er” 标签值创建的，该值与创建笔记本的委托人（IAM 用户或角色）的用户 ID 相匹配。有关这些标签的更多信息，请参阅 IAM 用户指南中的主体键值图表。

  此服务角色策略将附上使用笔记本中的魔术命令指定的角色，或作为角色传递给 `CreateSession` API 操作。只有标签键 “所有者” 和值与委托人的 AWS 用户 ID 相匹配时，该策略还允许委托人从 AWS Glue Studio笔记本界面创建AWS Glue交互式会话。此策略拒绝在创建会话后从 AWS Glue 会话资源中更改或删除“拥有者”标签的权限。该策略还包括写入和读取 Amazon S3 存储桶、写入 CloudWatch 日志，以及为所AWS Glue使用的 Amazon EC2 资源创建和删除标签的权限。

  > **Note**
  >
  > 若要完全实现安全益处，请勿将此策略授予分配到 `AWSGlueServiceRole`、`AWSGlueConsoleFullAccess` 或 `AWSGlueConsoleSageMakerNotebookFullAccess` 策略的角色。

- AwsGlueSessionUserRestrictedNotebookPolicy— 只有当标签键 “所有者” 和值与创建AWS Glue Studio笔记本的委托人（IAM 用户或角色）的 AWS 用户 ID 相匹配时，才提供从笔记本界面创建 AWS Glue交互式会话的权限。有关这些标签的更多信息，请参阅 IAM 用户指南中的主体键值图表。

  此策略已附上从 AWS Glue Studio 笔记本界面创建会话的主体（IAM 用户或角色）。此策略还允许对 AWS Glue Studio 笔记本的充分访问权限，以便与特定的 AWS Glue 交互式会话资源进行交互。这些资源是使用与委托人的 AWS 用户 ID 匹配的 “owner” 标签值创建的。此策略拒绝在创建会话后从 AWS Glue 会话资源中更改或删除“拥有者”标签的权限。

- AWSGlueServiceNotebookRole— 授予访问在AWS Glue Studio笔记本中启动的AWS Glue会话的权限。此政策允许列出和获取所有会话的会话信息，但仅允许用户创建和使用标有其 AWS 用户 ID 的会话。此政策拒绝向标有其 AWS ID 的AWS Glue会话资源中更改或删除 “所有者” 标签的权限。

  将此策略分配给使用中的笔记本界面创建作业的 AWS 用户AWS Glue Studio。

- AWSGlueConsoleSageMakerNotebookFullAccess— 当策略所关联的 AWS 身份使用时，授予对 Glue 和 SageMaker 资源的完全访问权限 AWS Management Console。如果遵循此策略中指定的资源的命名约定，则用户具有完全控制台功能。此政策通常附加到管理 SageMaker 笔记本AWS Glue的主机用户。

- AWSGlueSchemaRegistryFullAccess— 当策略所关联的身份使用 AWS Management Console 或时，授予对AWS Glue架构注册表资源的完全访问权限 AWS CLI。如果遵循此策略中指定的资源的命名约定，则用户具有完全控制台功能。此策略通常附加到AWS Glue控制台用户或管理AWS Glue架构注册表的 AWS CLI 用户。

- AWSGlueSchemaRegistryReadonlyAccess— 当策略所关联的身份使用 AWS Management Console 或时，授予对AWS Glue架构注册表资源的只读访问权限 AWS CLI。如果遵循此策略中指定的资源的命名约定，则用户具有完全控制台功能。此策略通常附加到AWS Glue控制台用户或使用AWS Glue架构注册表的 AWS CLI 用户。

> **Note**
>
> 您可以通过登录到 IAM 控制台并在该控制台中搜索特定策略来查看这些权限策略。

此外，您还可以创建您自己的自定义 IAM 策略，以授予 AWS Glue 操作和资源的相关权限。您可以将这些自定义策略附加到需要这些权限的 IAM 用户或组。

AWS Glue 对 AWS 托管策略的更新

查看自该服务开始跟踪这些更改以来对 AWS Glue 的 AWS 托管策略更新的详细信息。要获得有关此页面更改的自动提醒，请在 AWS Glue 文档历史记录页面上订阅 RSS feed。

更改	描述	日期
AwsGlueSessionUserRestrictedPolicy — 对现有政策的次要更新。	将 `glue:StartCompletion` 和 `glue:GetCompletion` 添加到策略。在 Glue 中集成亚马逊 Q 数据所必 AWS 需的。	2024年4月30日
AwsGlueSessionUserRestrictedNotebookServiceRole — 对现有政策的次要更新。	将 `glue:StartCompletion` 和 `glue:GetCompletion` 添加到策略。在 Glue 中集成亚马逊 Q 数据所必 AWS 需的。	2024年4月30日
AwsGlueSessionUserRestrictedServiceRole — 对现有政策的次要更新。	将 `glue:StartCompletion` 和 `glue:GetCompletion` 添加到策略。在 Glue 中集成亚马逊 Q 数据所必 AWS 需的。	2024年4月30日
AWSGlueServiceNotebookRole — 对现有政策的次要更新。	将 `glue:StartCompletion` 和 `glue:GetCompletion` 添加到策略。在 Glue 中集成亚马逊 Q 数据所必 AWS 需的。	2024年1月30日
AwsGlueSessionUserRestrictedNotebookPolicy — 对现有政策的次要更新。	将 `glue:StartCompletion` 和 `glue:GetCompletion` 添加到策略。在 Glue 中集成亚马逊 Q 数据所必 AWS 需的。	2023 年 11 月 29 日

更改	描述	日期
AWSGlueServiceNotebookRole — 对现有政策的次要更新。	为策略添加 `codewhisperer:GenerateRecommendations` 。Glue 生成 CodeWhisperer 推荐的新功能 AWS 所必需的。	2023 年 10 月 9 日
AWSGlueServiceRole — 对现有政策的次要更新。	收紧 CloudWatch 权限范围以更好地反映 AWS Glue 日志。	2023 年 8 月 4 日
AWSGlueConsoleFullAccess — 对现有政策的次要更新。	向策略添加 `databrew` 配方列表和描述权限。需要为 AWS Glue 可以访问配方的新功能提供完全的管理权限。	2023 年 5 月 9 日
AWSGlueConsoleFullAccess — 对现有政策的次要更新。	为策略添加 `cloudformation:ListStacks` 。更改 AWS CloudFormation 授权要求后保留现有功能。	2023 年 3 月 28 日
为交互式会话功能添加了新的托管策略 • AwsGlueSessionUserRestrictedServiceRole • AwsGlueSessionUserRestrictedPolicy • AwsGlueSessionUserRestrictedNotebookServiceRole • AwsGlueSessionUserRestrictedNotebookPolicy	这些策略旨在为 AWS Glue Studio 中的交互式会话和笔记本提供额外的安全性。这些策略会限制对 `CreateSession` API 操作的访问，使得只有拥有者有权访问。	2021 年 11 月 30 日

更改	描述	日期
AWSGlueConsoleSageMakerNotebookFullAccess — 更新现有政策。	为以下操作删除了冗余资源 ARN（`arn:aws:s3:::aws-glue-*/*`）：为 AWS Glue 用于存储脚本和临时文件的 Amazon S3 存储桶授予读取/写入权限。 通过将 `"StringEquals"` 更改为 `"ForAnyValue:StringLike"` 修复了语法问题，并且在行乱序的每个位置将 `"Effect": "Allow"` 行移到 `"Action":` 行之前。	2021 年 7 月 15 日
AWSGlueConsoleFullAccess — 更新现有政策。	为以下操作删除了冗余资源 ARN（`arn:aws:s3:::aws-glue-*/*`）：为 AWS Glue 用于存储脚本和临时文件的 Amazon S3 存储桶授予读取/写入权限。	2021 年 7 月 15 日
AWS Glue 已开启跟踪更改	AWS Glue开始跟踪其 AWS 托管策略的更改。	2021 年 6 月 10 日

指定 AWS Glue 资源 ARN

在 AWS Glue 中，您可以使用 AWS Identity and Access Management（IAM）policy 控制对资源的访问。在策略中，您可以使用 Amazon 资源名称（ARN）标识策略应用到的资源。并非 AWS Glue 中的所有资源都支持 ARN。

主题

- 数据目录 ARN
- AWS Glue 中非目录对象的 ARN
- AWS Glue 非目录单数 API 操作的访问控制

- 检索多个项目的 AWS Glue 非目录 API 操作的访问控制
- AWS Glue 非目录 BatchGet API 操作的访问控制

数据目录 ARN

数据目录资源具有层次结构，其中以 catalog 作为根。

```
arn:aws:glue:region:account-id:catalog
```

每个 AWS 账户在 AWS 区域中有单个数据目录，以 12 位的账户 ID 作为目录 ID。资源具有与其关联的唯一 ARN，如下表所示。

资源类型	ARN 格式
目录	arn:aws:glue: *region*:*account-id* :catalog 例如：arn:aws:glue:us-east-1:123456789012:catalog
数据库	arn:aws:glue: *region*:*account-id* :database/ *database name* 例如：arn:aws:glue:us-east-1:123456789012:database/db1 。
表	arn:aws:glue: *region*:*account-id* :table/*database name*/*table name* 例如：arn:aws:glue:us-east-1:123456789012:table/db1/tbl1 。
用户定义的函数	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*

资源类型	ARN 格式
	例如：arn:aws:glue:us-east-1:123456789012:connection/connection1 。
交互式会话	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 。

要启用精细访问控制，您可以在 IAM policy 和资源策略中使用这些 ARN 来授予或拒绝对特定资源的访问权限。允许在策略中使用通配符。例如，以下 ARN 与数据库 default 中的所有表匹配。

```
arn:aws:glue:us-east-1:123456789012:table/default/*
```

Important

对数据目录资源执行所有操作都需要具有对该资源及其所有原级的权限。例如，要为一个表创建分区，需要具有该表的权限以及该表所在数据库和目录的权限。以下示例显示对数据目录中的数据库 PrivateDatabase 中的表 PrivateTable 创建分区所需的权限。

```
{
   "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"
   ]
}
```

除了对资源及其所有原级的权限外，所有删除操作还需要具有该资源的所有子级的权限。例如，要删除一个数据库，您需要具有该数据库中的所有表和用户定义函数的权限，以及该数据库及其所在目录的权限。以下示例显示删除数据目录中的数据库 PrivateDatabase 所需的权限。

```
{
   "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"
   ]
}
```

概括来说，对数据目录资源的操作遵循以下权限规则：

- 针对目录的操作只需要具有目录的权限。
- 针对数据库的操作需要具有数据库和目录的权限。
- 针对数据库的删除操作需要具有数据库和目录的权限，以及数据库中所有表和用户定义函数的权限。
- 针对表、分区或表版本的操作需要具有表、数据库和目录的权限。
- 针对用户定义的函数的操作需要具有用户定义的函数、数据库和目录的权限。
- 针对连接的操作需要具有连接和目录的权限。

AWS Glue 中非目录对象的 ARN

有些 AWS Glue 资源允许资源级权限使用 ARN 来控制访问。您可以在 IAM policy 中使用这些 ARN 来启用精细访问控制。下表列出了可以包含资源 ARN 的资源。

资源类型	ARN 格式
爬网程序	arn:aws:glue: *region*:*account-id* :crawler/ *crawler-name* 例如：arn:aws:glue:us-east-1:123456789012:crawler/mycrawler 。

资源类型	ARN 格式
作业	arn:aws:glue: *region*:*account-id* :job/*job-name* 例如：arn:aws:glue:us-east-1:123456789012:job/testjob 。
触发器	arn:aws:glue: *region*:*account-id* :trigger/ *trigger-name* 例如：arn:aws:glue:us-east-1:123456789012:trigger/sampletrigger 。
开发终端节点	arn:aws:glue: *region*:*account-id* :devEndpoint/ *development-endpoint-name* 例如：arn:aws:glue:us-east-1:123456789012:devEndpoint/temporarydevendpoint 。
机器学习转换	arn:aws:glue: *region*:*account-id* :mlTransform/ *transform-id* 例如：arn:aws:glue:us-east-1:123456789012:mlTransform/tfm-1234567890 。

AWS Glue 非目录单数 API 操作的访问控制

AWS Glue 非目录单数 API 操作对单个项目（开发终端节点）执行。示例包括 GetDevEndpoint、CreateUpdateDevEndpoint 和 UpdateDevEndpoint。对于这些操作，策略必须将 API 名称放在 "action" 块中，将资源 ARN 放在 "resource" 块中。

假设您要允许用户调用 GetDevEndpoint 操作。以下策略将所需的最低权限授予名为 myDevEndpoint-1 的终端节点。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "MinimumPermissions",
            "Effect": "Allow",
```

```
            "Action": "glue:GetDevEndpoint",
            "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/
myDevEndpoint-1"
        }
    ]
}
```

以下策略允许 UpdateDevEndpoint 访问用通配符 (*) 与 myDevEndpoint- 匹配的资源。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PermissionWithWildcard",
            "Effect": "Allow",
            "Action": "glue:UpdateDevEndpoint",
            "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/myDevEndpoint-
*"
        }
    ]
}
```

您可以合并两个策略，如以下示例所示。您可能会看到名称以 A 开头的任何开发终端节点的 EntityNotFoundException。不过，当您尝试访问其他开发终端节点时，将返回一个访问被拒绝错误。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CombinedPermissions",
            "Effect": "Allow",
            "Action": [
                "glue:UpdateDevEndpoint",
                "glue:GetDevEndpoint"
            ],
            "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/A*"
        }
    ]
}
```

检索多个项目的 AWS Glue 非目录 API 操作的访问控制

有些 AWS Glue API 操作检索多个项目（如多个开发终端节点）；例如，GetDevEndpoints。对于此操作，您可以仅指定通配符 (*) 资源，而不是特定的 ARN。

例如，要在策略中包含 GetDevEndpoints，资源的范围必须限定为通配符 (*)。单数操作（GetDevEndpoint、CreateDevEndpoint 和 DeleteDevendpoint）的范围也确定为示例中的所有 (*) 资源。

```
{
            "Sid": "PluralAPIIncluded",
            "Effect": "Allow",
            "Action": [
                "glue:GetDevEndpoints",
                "glue:GetDevEndpoint",
                "glue:CreateDevEndpoint",
                "glue:UpdateDevEndpoint"
            ],
            "Resource": [
                "*"
            ]
}
```

AWS Glue 非目录 BatchGet API 操作的访问控制

有些 AWS Glue API 操作检索多个项目（如多个开发终端节点）；例如，BatchGetDevEndpoints。对于此操作，您可以指定一个 ARN 来限制可访问的资源的范围。

例如，要允许访问特定的开发终端节点，请将 BatchGetDevEndpoints 及其资源 ARN 包含在策略中。

```
{
            "Sid": "BatchGetAPIIncluded",
            "Effect": "Allow",
            "Action": [
                "glue:BatchGetDevEndpoints"
            ],
            "Resource": [
                "arn:aws:glue:us-east-1:123456789012:devEndpoint/de1"
            ]
}
```

利用此策略，您可以成功访问名为 de1 的开发终端节点。但是，如果您尝试访问名为 de2 的开发终端节点，则会返回错误。

```
An error occurred (AccessDeniedException) when calling the BatchGetDevEndpoints
 operation: No access to any requested resource.
```

> **Important**
>
> 有关设置 IAM policy 的替代方法（例如，使用 List 和 BatchGet API 操作），请参阅适用于 AWS Glue 的基于身份的策略示例。

授予跨账户访问权限

授予跨账户的数据目录资源访问权限，支持您提取、转换和加载（ETL）任务，以便从不同账户查询和联接数据。

主题

- 用于在 AWS Glue 中授予跨账户访问权限的方法
- 添加或更新数据目录资源策略
- 执行跨账户 API 调用
- 执行跨账户 ETL 调用
- 跨账户 CloudTrail 日志记录
- 跨账户资源所有权和账单
- 跨账户访问限制

用于在 AWS Glue 中授予跨账户访问权限的方法

您可以使用 AWS Glue 方法或 AWS Lake Formation 跨账户授权，为外部 AWS 账户授予数据访问权限。AWS Glue 方法使用 AWS Identity and Access Management（IAM）policy，实现精细访问控制。Lake Formation 使用更简单的 GRANT/REVOKE 权限模型，类似于关系数据库系统中的 GRANT/REVOKE 命令。

本部分介绍如何使用 AWS Glue 方法。有关使用 Lake Formation 跨账户授权的信息，请参阅《AWS Lake Formation 开发人员指南》中的授予 Lake Formation 权限。

有两种 AWS Glue 方法可用于授予跨账户资源访问权限：

- 使用数据目录资源策略
- 使用 IAM 角色

使用资源策略授予跨账户访问权限

以下是使用数据目录资源策略授予跨账户访问权限的一般步骤：

1. 账户 A 中的管理员（或其他授权身份）将资源策略附加到账户 A 中的数据目录。此策略将向账户 B 授予对账户 A 的目录中的资源执行操作的具体跨账户权限。
2. 账户 B 中的管理员将一个 IAM policy 附加到账户 B 中的 IAM 身份，用于委托从账户 A 收到的权限。

   现在，账户 B 中的身份有权访问账户 A 中的指定资源。

   用户需要同时具有资源拥有者（账户 A）及其父账户（账户 B）授予的权限，才能访问资源。

使用 IAM 角色授予跨账户访问权限

以下是使用 IAM 角色授予跨账户访问权限的一般步骤：

1. 账户中拥有资源（账户 A）的管理员（或其他授权身份）可创建一个 IAM 角色。
2. 账户 A 中的管理员会向该角色中附加一个策略，以授予用于访问所涉及资源的跨账户访问权限。
3. 账户 A 中的管理员将向该角色中附加一个信任策略，用于将其他账户（账户 B）中的 IAM 身份标识为可以担任该角色的委托人。

   如果您需要向 AWS 服务授予担任该角色的权限，则信任策略中的委托人也可以是 AWS 服务委托人。
4. 现在，账户 B 中的管理员向账户 B 中的一个或多个 IAM 身份委托权限，使其可以担任该角色。这样，账户 B 中的身份便可访问账户 A 中的资源。

有关使用 IAM 委派权限的更多信息，请参阅 IAM 用户指南中的访问权限管理。有关用户、组、角色和权限的更多信息，请参阅 IAM 用户指南中的身份（用户、组和角色）。

有关这两种方法的对比，请参阅 IAM 用户指南中的 IAM 角色与基于资源的策略有何不同。AWS Glue 支持这两个选项，但存在限制，即资源策略只能授予对 Data Catalog 资源的访问权限。

例如，要向账户 B 中的 Dev 角色授予对账户 A 中的数据库 db1 的访问权限，请将以下资源策略附加到账户 A 中的目录。

```
{
  "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，账户 B 还必须将以下 IAM policy 附加至 Dev 角色，然后才能实际访问账户 A 中的 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"
      ]
    }
  ]
}
```

添加或更新数据目录资源策略

您可以使用控制台、API 或 AWS Command Line Interface（AWS CLI）添加或更新 AWS Glue 数据目录资源策略。

> **Important**
>
> 如果您已经使用 AWS Lake Formation 从您的账户授予跨账户权限，则添加或更新数据目录资源策略需要额外步骤。有关更多信息，请参阅 AWS Lake Formation 开发人员指南中的 同时使用 AWS Glue 和 Lake Formation 管理跨账户权限。
>
> 要确定是否存在 Lake Formation 跨账户授权，请使用 `glue:GetResourcePolicies` API 操作或 AWS CLI。如果 `glue:GetResourcePolicies` 返回现有 Data Catalog 策略以外的任何策略，则存在 Lake Formation 授权。有关更多信息，请参阅 AWS Lake Formation 开发人员指南中的使用 GetResourcePolicies API 操作查看所有跨账户授权。

添加或更新数据目录资源策略（控制台）

1. 通过 https://console.aws.amazon.com/glue/ 打开 AWS Glue 控制台。

   作为具有 `glue:PutResourcePolicy` 权限的 AWS Identity and Access Management（IAM）管理用户登录。
2. 在导航窗格中，选择 Settings (设置)。
3. 在 Data catalog settings (数据目录设置) 页面上的 Permissions (权限) 下面，将资源策略粘贴到文本区域。然后选择保存。

   如果控制台显示提示，指明策略中的权限不属于使用 Lake Formation 授予的权限，请选择 Proceed (继续)。

添加或更新数据目录资源策略（AWS CLI）

- 提交 `aws glue put-resource-policy` 命令。如果 Lake Formation 授权已存在，请确保包含值为 `'TRUE'` 的 `--enable-hybrid` 选项。

  有关使用此命令的示例，请参阅 AWS Glue 基于资源的策略示例。

执行跨账户 API 调用

所有 AWS Glue Data Catalog 操作都有一个 `CatalogId` 字段。如果已授予所需的权限来启用跨账户访问，则发起人可以跨账户进行数据目录 API 调用。调用方通过传递 `CatalogId` 中的目标 AWS 账户 ID，以访问目标账户中的资源。

如果没有提供 `CatalogId` 值，AWS Glue 将默认使用调用方自己的账户 ID，但该调用不是跨账户调用。

执行跨账户 ETL 调用

有些 AWS Glue PySpark 和 Scala API 具有目录 ID 字段。如果已授予启用跨账户访问所需的全部权限，ETL 任务可通过在目录 ID 字段中传递目标 AWS 账户 ID 来对 API 操作进行跨账户的 PySpark 和 Scala 调用，以访问目标账户中的数据目录资源。

如果没有提供目录 ID 值，AWS Glue 将默认使用调用方自己的账户 ID，但该调用不是跨账户调用。

对于支持 `catalog_id` 的 PySpark API，请参阅 GlueContext 班级。对于支持 `catalogId` 的 Scala API，请参阅 AWS GlueScala API GlueContext 。

以下示例显示被授权者运行 ETL 任务所需的权限。在此示例中，*`grantee-account-id`* 是运行作业的客户端的 `catalog-id`，*`grantor-account-id`* 是资源的拥有者。此示例用于授予对授予者账户中所有目录资源的权限。为限制所授予资源的范围，您可以提供目录、数据库、表和连接的特定 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**
>
> 如果授予者账户中的某个表指向同在该账户中的一个 Amazon S3 位置，则授予者账户中用于运行 ETL 任务的 IAM 角色必须有权列出和获取授予者账户中的对象。

如果账户 A 中的客户端已经获得创建和运行 ETL 作业的权限，则设置用于跨账户访问的 ETL 作业的基本步骤如下：

1. 允许跨账户数据访问（如果已设置 Amazon S3 跨账户访问权限，则可跳过此步骤）。
   a. 更新账户 B 中的 Amazon S3 存储桶策略，以允许来自账户 A 的跨账户访问
   b. 更新账户 A 中的 IAM policy，以允许访问账户 B 中的存储桶。
2. 允许跨账户数据目录访问。
   a. 创建或更新账户 B 中附加到数据目录的资源策略，以允许来自账户 A 的访问。
   b. 更新账户 A 中的 IAM policy，以允许访问账户 B 中的数据目录。

跨账户 CloudTrail 日志记录

当 AWS Glue 提取、转换和加载（ETL）任务访问通过 AWS Lake Formation 跨账户授权分享的数据目录表的底层数据时，存在额外 AWS CloudTrail 日志记录行为。

为了本次讨论，共享表的 AWS 账户是拥有者账户，与该表的共享目标账户是收件人账户。当收件人账户中的 ETL 任务访问拥有者账户中表的数据时，添加到收件人账户日志中的数据访问 CloudTrail 事件将复制到拥有者账户的 CloudTrail 日志。这样，拥有者账户可以跟踪不同收件人账户的数据访问情况。默认情况下，CloudTrail 事件不包括人类可读的委托人标识符（委托人 ARN）。收件人账户中的管理员可以选择在日志中包含委托人 ARN。

有关更多信息，请参阅 AWS Lake Formation 开发人员指南中的跨账户 CloudTrail 日志记录。

> **另请参阅**
>
> - the section called “日志记录和监控”

跨账户资源所有权和账单

当一个 AWS 账户（账户 A）中的用户在其他账户（账户 B）中创建新资源（如数据库）时，该资源的所有者将为账户 B，即创建资源所在的账户。账户 B 中的管理员将自动获得访问这一新资源的完整权限，包括读取、写入和向第三个账户授予访问权限。账户 A 中的用户只有获得账户 B 授予的相应权限后，才能访问他们刚刚创建的资源。

与新资源直接关联的存储成本和其他成本将计入账户 B，即资源所有者。创建资源的用户的请求成本将计入请求者的账户，即账户 A。

有关 AWS Glue 计费和定价的更多信息，请参阅 AWS 定价原理。

跨账户访问限制

AWS Glue 跨账户访问具有以下限制：

- 如果您在区域支持 AWS Glue 之前已使用 Amazon Athena 或 Amazon Redshift Spectrum 创建数据库和表，并且资源拥有者账户尚未将 Amazon Athena 数据目录迁移至 AWS Glue，则不允许跨账户访问 AWS Glue。可以使用GetCatalogImportStatus (get_catalog_import_status)查找当前的迁移状态。有关如何将 Athena 目录迁移到 AWS Glue 的更多信息，请参阅 Amazon Athena 用户指南中的按步骤升级到 AWS Glue Data Catalog。
- 仅数据目录资源（包括数据库、表、用户定义的函数和连接）支持跨账户访问。
- 从 Athena 跨账户访问数据目录需要将目录注册为 Athena `DataCatalog` 资源。有关说明，请参阅 Amazon Athena 用户指南中的从其他账户注册 AWS Glue Data Catalog。

对 AWS Glue 身份和访问进行故障排除

使用以下信息来帮助您诊断和修复在使用 Glue 和 IAM 时可能遇到 AWS 的常见问题。

主题

- 我无权在 AWS Glue 中执行操作
- 我无权执行 iam：PassRole
- 我想允许我以外的人访问我的 AWS 账户AWS Glue 资源

我无权在 AWS Glue 中执行操作

如果您收到错误提示，表明您无权执行某个操作，则您必须更新策略以允许执行该操作。

当 mateojackson IAM 用户尝试使用控制台查看有关虚构 *my-example-widget* 资源的详细信息，但不拥有虚构 glue:*GetWidget* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 glue:GetWidget on resource: my-example-widget
```

在此情况下，必须更新 mateojackson 用户的策略，以允许使用 glue:*GetWidget* 操作访问 *my-example-widget* 资源。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam：PassRole

如果您收到错误消息，提示您无权执行iam:PassRole操作，则必须更新您的策略以允许您将角色传递给 AWS Glue。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 marymajor 的 IAM 用户尝试使用控制台在 AWS Glue 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 iam:PassRole 操作。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人访问我的 AWS 账户AWS Glue 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以担任角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 AWS Glue 是否支持这些功能，请参阅Glue AWS 如何与 IAM 配合使用。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。

- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过联合身份验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（联合身份验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问的区别，请参阅 IA M 用户指南中的跨账户资源访问。

AWS Glue 中的日志记录和监控

您可以自动运行您的 ETL（提取、转换和加载）作业。AWS Glue 提供了有关爬网程序和作业的指标，您可以监控这些指标。使用所需元数据设置 AWS Glue Data Catalog后，AWS Glue 会提供有关环境运行状况的统计数据。您可以基于 cron 使用基于时间的计划自动调用爬网程序和作业。您也可以在基于事件的触发器触发时触发作业。

AWS Glue 与 AWS CloudTrail 集成，后者是在 AWS Glue 中记录用户、角色或 AWS 服务所执行操作的服务。如果您创建跟踪，则可以使 CloudTrail 事件持续传送到 Amazon Simple Storage Service（Amazon S3）存储桶、Amazon CloudWatch Logs 和 Amazon CloudWatch Events。每个事件或日记账条目都包含有关生成请求的人员信息。

使用 Amazon CloudWatch Events 自动化您的 AWS 服务，以自动响应系统事件，例如应用程序可用性问题或资源更改。AWS 服务中的事件近乎实时地传输到 CloudWatch Events 。您可以编写简单规则来指示所关注的事件，并指示要在事件匹配规则时执行的自动化操作。

> 另请参阅
>
> - 使用 CloudWatch Events 实现 AWS Glue 自动化
> - 跨账户 CloudTrail 日志记录

云端安全的一个重要方面是日志记录。您必须以按以下方式配置日志记录：既不捕获密钥和机密材料，同时又能捕获调试和保护云基础结构所需的信息。务必要熟悉正在记录的内容。

合规性验证 AWS Glue

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- 安全与合规性快速入门指南 — 这些部署指南讨论了架构注意事项，并提供了部署以安全性和合规性为重点 AWS 的基准环境的步骤。
- 在 A@@ mazon Web Services 上构建 HIPAA 安全与合规性 — 本白皮书描述了各公司如何使用 AWS 来创建符合 HIPAA 资格的应用程序。

> **Note**
>
> 并非所有 AWS 服务 人都符合 HIPAA 资格。有关更多信息，请参阅符合 HIPAA 要求的服务参考。

- AWS 合AWS 规资源 — 此工作簿和指南集可能适用于您的行业和所在地区。
- AWS 客户合规指南 — 从合规角度了解责任共担模式。这些指南总结了保护的最佳实践， AWS 服务并将指南映射到跨多个框架（包括美国国家标准与技术研究院 (NIST)、支付卡行业安全标准委员会 (PCI) 和国际标准化组织 (ISO)）的安全控制。
- 使用AWS Config 开发人员指南中的规则评估资源 — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- AWS Security Hub— 这 AWS 服务 可以全面了解您的安全状态 AWS。Security Hub 通过安全控件评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控件的列表，请参阅 Security Hub 控件参考。
- Amazon GuardDuty — 它通过监控您的 AWS 账户环境中是否存在可疑和恶意活动，来 AWS 服务检测您的工作负载、容器和数据面临的潜在威胁。 GuardDuty 通过满足某些合规性框架规定的入侵检测要求，可以帮助您满足各种合规性要求，例如 PCI DSS。
- AWS Audit Manager— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

韧性在 AWS Glue

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。 AWS 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。利用可用区，您可以设计和操作在可用

区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

有关 AWS Glue 作业弹性的更多信息，请参阅中的错误：VPC 之间的故障转移行为。 AWS Glue

AWS Glue 中的基础设施安全性

作为一项托管式服务，AWS Glue 由 Amazon Web Services：安全流程概览白皮书中所述的 AWS 全球网络安全程序提供保护。

您可以使用 AWS 发布的 API 调用通过网络访问 AWS Glue。客户端必须支持传输层安全性协议 (TLS) 1.0 或更高版本。建议使用 TLS 1.2 或更高版本。客户端还必须支持具有完全向前保密（PFS）的密码套件，例如 Ephemeral Diffie-Hellman（DHE）或 Elliptic Curve Ephemeral Diffie-Hellman（ECDHE）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 委托人关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

主题

- AWS Glue 和接口 VPC 端点 (AWS PrivateLink)
- 共享的 Amazon VPC

AWS Glue 和接口 VPC 端点 (AWS PrivateLink)

您可以通过创建接口 VPC 端点在 VPC 和 AWS Glue 之间建立私有连接。接口端点由 AWS PrivateLink 提供支持，该技术支持您通过私密方式访问 AWS Glue API，而无需互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。VPC 中的实例即使没有公有 IP 地址也可与 AWS Glue API 进行通信。VPC 和 AWS Glue 之间的流量不会脱离 Amazon 网络。

每个接口终端节点均由子网中的一个或多个弹性网络接口表示。

有关更多信息，请参阅《Amazon VPC 用户指南》中的接口 VPC 端点 (AWS PrivateLink)。

AWS Glue VPC 端点注意事项

请务必先查看 Amazon VPC 用户指南中的接口端点属性和限制，然后再为 AWS Glue 设置接口 VPC 终端节点。

AWS Glue 支持从 VPC 调用它的所有 API 操作。

为 AWS Glue 创建接口 VPC 端点

您可以使用 Amazon VPC 控制台或 AWS Command Line Interface (AWS CLI) 为 AWS Glue 服务创建 VPC 端点。有关更多信息，请参阅《Amazon VPC 用户指南》中的创建接口端点

使用以下服务名称为 AWS Glue 创建 VPC 端点：

- com.amazonaws.*region*.glue

如果为端点启用私有 DNS，则可以使用其默认 DNS 名称作为区域，向 AWS Glue 发送 API 请求，例如 `glue.us-east-1.amazonaws.com`。

有关更多信息，请参阅《Amazon VPC 用户指南》中的通过接口端点访问服务。

为 AWS Glue 创建 VPC 端点策略

您可以为 VPC 端点附加控制对 AWS Glue 的访问的端点策略。该策略指定以下信息：

- 可执行操作的主体。
- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅《Amazon VPC 用户指南》中的使用 VPC 端点控制对服务的访问。

示例：适用于 AWS Glue 允许创建和更新任务的 VPC 终端节点策略

下面是用于 AWS Glue 的端点策略示例。当附加到端点时，此策略会向所有资源上的所有主体授予对列出的 AWS Glue 操作的访问权限。

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
             "glue:CreateJob",
             "glue:UpdateJob",
             "iam:PassRole"
         ],
```

```
         "Resource":"*"
      }
   ]
}
```

示例：允许只读数据目录访问的 VPC 终端节点策略

下面是用于 AWS Glue 的端点策略示例。当附加到端点时，此策略会向所有资源上的所有主体授予对列出的 AWS Glue 操作的访问权限。

```
{
    "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": "*"
        }
    ]
}
```

共享的 Amazon VPC

AWS Glue 支持 Amazon Virtual Private Cloud 中的共享 Virtual Private Cloud（VPC）。Amazon VPC 共享允许多个 AWS 账户将其应用程序资源（例如 Amazon EC2 实例和 Amazon Relational Database Service（Amazon RDS）数据库）创建到共享的集中管理式 Amazon VPC 中。在此模型中，拥有 VPC 的账户（拥有者）与属于 AWS Organizations 中同一企业的其他账户（参与者）共享一个或多个子网。共享子网之后，参与者可以查看、创建、修改和删除与他们共享的子网中的应用程序资源。

在 AWS Glue 中，要创建与共享子网的连接，您必须在账户中创建一个安全组，并将安全组附加到共享子网。

有关更多信息，请参阅以下主题：

- 《Amazon VPC 用户指南》中的使用共享 VPC
- 《AWS Organizations 用户指南》中的什么是 AWS Organizations？

故障排除 AWS Glue

主题

收集 AWS Glue 故障诊断信息

如果在 AWS Glue 中遇到错误或意外行为，并且需要与 AWS Support 联系，则应首先收集与失败操作关联的名称、ID 和日志的信息。有了这些信息，AWS Support 可以帮助您解决您遇到的问题。

除了您的 account ID，还收集以下每种故障类型的信息：

当爬网程序失败时，请收集以下信息：

- 爬网程序名称

  爬网程序运行的日志位于 CloudWatch Logs 中的 `/aws-glue/crawlers` 下面。

当测试连接失败时，请收集以下信息：

- 连接名称
- 连接 ID
- 采用 `jdbc:protocol://host:port/database-name` 形式的 JDBC 连接字符串。

  测试连接的日志位于 CloudWatch Logs 中的 `/aws-glue/testconnection` 下面。

当作业失败时，请收集以下信息：

- 任务名称
- 采用 `jr_xxxxx` 形式的作业运行 ID。

  任务运行的日志位于 CloudWatch Logs 中的 `/aws-glue/jobs` 下面。

对 Spark 中的错误 AWS Glue 进行故障排除

如果您在中遇到错误 AWS Glue，请使用以下解决方案来帮助您找到问题的根源并进行修复。

> **Note**
>
> 该 AWS Glue GitHub 存储库包含AWS Glue 常见问题解答中的其他疑难解答指南。

主题

- 错误：运行爬网程序失败
- 错误：分区未更新
- 错误：由于版本不匹配，作业书签更新失败
- 错误：启用作业书签后，作业正在重新处理数据
- 错误：中 VPC 之间的故障转移行为 AWS Glue
- 解决爬网程序使用 Lake Formation 凭证时出现的爬网程序错误

错误：资源不可用

如果 AWS Glue 返回资源不可用消息，则可以查看错误消息或日志，以帮助您详细了解该问题。以下作业描述了故障排除的通用方法。

- 对于您使用的任何连接和开发终端节点，请检查您的集群是否未用尽弹性网络接口。

错误：在 VPC 中找不到 subnetId 的 S3 终端节点或 NAT 网关

检查消息中的子网 ID 和 VPC ID 可帮助您诊断问题。

- 检查您是否设置了 AWS Glue需要的 Amazon S3 VPC 终端节点。此外，请检查您的 NAT 网关 (如果它是您的配置的一部分)。有关更多信息，请参阅 适用于 Amazon S3 的 Amazon VPC 终端节点。

错误：需要安全组中的入站规则

至少有一个安全组必须打开所有入口端口。为限制流量，入站规则中的源安全组可限制为同一安全组。

- 对于您使用的任何连接，请检查安全组中是否有属于自引用的入站规则。有关更多信息，请参阅 设置对数据存储的网络访问。
- 如果您使用的是开发终端节点，请检查安全组中是否有属于自引用的入站规则。有关更多信息，请参阅 设置对数据存储的网络访问。

错误：需要安全组中的出站规则

至少有一个安全组必须打开所有出口端口。为限制流量，出站规则中的源安全组可限制为同一安全组。

- 对于您使用的任何连接，请检查安全组中是否有属于自引用的出站规则。有关更多信息，请参阅 设置对数据存储的网络访问。

- 如果您使用的是开发终端节点，请检查安全组中是否有属于自引用的出站规则。有关更多信息，请参阅 设置对数据存储的网络访问。

错误：Job 运行失败，因为应向传递的角色授予该 AWS Glue 服务的代入角色权限

定义作业的用户必须具有针对适用于 AWS Glue的 `iam:PassRole` 的权限。

- 当用户创建 AWS Glue 任务时，请确认该用户的角色包含包含`iam:PassRole`的策略 AWS Glue。有关更多信息，请参阅 步骤 3：将策略附加到访问 AWS Glue 的用户或组。

错误： DescribeVpcEndpoints 操作未授权。无法验证 VPC ID vpc-id

- 检查传递给的策略以 AWS Glue 获取`ec2:DescribeVpcEndpoints`权限。

错误： DescribeRouteTables 操作未授权。无法验证子网 ID：VPC 中的子网 ID：vpc-id：vpc-id

- 检查传递给的策略以 AWS Glue 获取`ec2:DescribeRouteTables`权限。

错误：调用 ec2 失败：DescribeSubnets

- 检查传递给的策略以 AWS Glue 获取`ec2:DescribeSubnets`权限。

错误：调用 ec2 失败：DescribeSecurityGroups

- 检查传递给的策略以 AWS Glue 获取 `ec2:DescribeSecurityGroups`权限。

错误：找不到可用区的子网

- 可用区可能不可用 AWS Glue。请在不同于消息中指定的可用区的可用区中创建和使用新子网。

错误：对 JDBC 目标进行写入时发生作业运行异常

当您运行某个对 JDBC 目标进行写入的作业时，该作业在以下情况下可能遇到错误：

- 如果您的作业对 Microsoft SQL Server 表进行写入，而该表具有定义为 `Boolean` 类型的列，则必须在 SQL Server 数据库中预定义该表。在 AWS Glue 控制台上使用带有 “在数据目标中创建表” 选项的 SQL Server 目标定义作业时，不要将任何源列映射到具有数据类型的目标列`Boolean`。您可能在作业运行时遇到错误。

  您可以通过执行以下操作来避免错误：

  - 选择包含 Boolean 列的现有表。
  - 编辑 `ApplyMapping` 转换并将源中的 Boolean 列映射到目标中的数字或字符串。
  - 编辑 `ApplyMapping` 转换以从源中删除 Boolean 列。
- 如果您的作业对 Oracle 表进行写入，您可能需要调整 Oracle 对象的名称长度。在某些版本的 Oracle 中，最大标识符长度限制为 30 个字节或 128 个字节。此限制影响 Oracle 目标数据存储的表名称和列名称。

  您可以通过执行以下操作来避免错误：

  - 在您的版本的限制内为 Oracle 目标表命名。
  - 默认列名称从数据中的字段名称生成。要应对列名称长度超过限制的情况，请使用 `ApplyMapping` 或 `RenameField` 转换将列的名称更改得不超过限制。

错误：Amazon S3：此操作对该对象的存储类无效

如果 AWS Glue 返回此错误，则您的 AWS Glue 任务可能是在读取分区跨越 Amazon S3 存储类层的表中的数据。

- 通过使用存储类排除项，您可以确保您的 AWS Glue 作业可以在这些存储类层之间有分区的表上运行。如果没有排除项，则从这些层读取数据的作业将会失败，并显示以下错误：`AmazonS3Exception: The operation is not valid for the object's storage class`。

  有关更多信息，请参阅 排除 Amazon S3 存储类。

错误：Amazon S3 超时

如果 AWS Glue 返回连接超时错误，则可能是因为它正在尝试访问其他 AWS 区域的 Amazon S3 存储桶。

- Amazon S3 VPC 终端节点只能将流量路由到 AWS 区域内的存储桶。如果您需要连接到其他区域中的存储桶，一种可能的解决方法是使用 NAT 网关。有关更多信息，请参阅 NAT 网关。

错误：Amazon S3 访问被拒绝

如果对 Amazon S3 存储桶或对象 AWS Glue 返回访问被拒绝的错误，则可能是因为所提供的 IAM 角色没有权限访问您的数据存储的策略。

- ETL 任务必须有权访问用作源或目标的 Amazon S3 数据存储。爬网程序必须有权访问其爬取的 Amazon S3 数据存储。有关更多信息，请参阅 步骤 2：为 AWS Glue 创建 IAM 角色。

错误：Amazon S3 访问密钥 ID 不存在

如果在运行作业时 AWS Glue 返回访问密钥 ID 不存在错误，则可能是由于以下原因之一：

- ETL 任务使用 IAM 角色来访问数据存储和确认您的任务的 IAM 角色在任务开始前未被删除。
- IAM 角色包含以下权限：访问您的数据存储和确认包含 `s3:ListBucket` 的所有已附加的 Amazon S3 策略正确无误。

错误：作业在访问带有 **`s3a://`** URI 的 Amazon S3 时运行失败

如果作业运行时返回一个类似无法分析使用处理程序类的 XML 文档的错误，可能是由于尝试列出数百个使用 `s3a://` URI 的文件时失败。改用 `s3://` URI 访问您的数据存储。下面的异常跟踪突出显示了要查找的错误：

```
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)
```

错误：Amazon S3 服务令牌已过期

在将数据移入和移出 Amazon Redshift 时，使用了将在 1 小时后过期的临时 Amazon S3 凭证。如果您有长时间运行的作业，它可能会失败。有关如何设置您的长时间运行的任务以将数据移入和移出 Amazon Redshift 的信息，请参阅aws-glue-programming-etl-connect-redshift-home。

错误：找不到网络接口的私有 DNS

如果作业失败或开发终端节点无法预置，则可能是因为网络设置有问题。

- 如果您使用的是 Amazon 提供的 DNS，则值 `enableDnsHostnames` 必须设置为 true。有关更多信息，请参阅 DNS。

错误：开发终端节点预置失败

如果 AWS Glue 未能成功配置开发端点，则可能是因为网络设置存在问题。

- 当您定义开发终端节点时，系统将验证 VPC、子网和安全组以确认它们是否满足特定要求。
- 如果您提供了可选的 SSH 公有密钥，请检查它是否为有效的 SSH 公有密钥。
- 在 VPC 控制台中检查您的 VPC 是否使用了有效的 DHCP 选项集。有关更多信息，请参阅 DHCP 选项集。
- 如果集群仍处于 PROVISIONING 状态，请联系 AWS Support。

错误：笔记本服务器 CREATE_FAILED

如果 AWS Glue 无法为开发端点创建笔记本服务器，则可能是由于以下问题之一：

- AWS Glue 在设置笔记本服务器时，将 IAM 角色传递给 Amazon EC2。该 IAM 角色必须与 Amazon EC2 具有信任关系。
- 该 IAM 角色必须具有同名的实例配置文件。当您使用 IAM 控制台为 Amazon EC2 创建角色时，将自动创建与该角色同名的实例配置文件。请在日志中检查与无效的实例配置文件名称 `iamInstanceProfile.name` 有关的错误。有关更多信息，请参阅使用实例配置文件。
- 检查您的角色是否有权访问您传递的用来创建笔记本服务器的策略中的 `aws-glue*` 存储桶。

错误：本地笔记本无法启动

如果您的本地笔记本无法启动并报告“找不到某个目录或文件夹”错误，则可能由以下问题之一导致：

- 如果您是在 Microsoft Windows 上运行，请确保 `JAVA_HOME` 环境变量指向正确的 Java 目录。您可以在不更新此变量的情况下更新 Java，而如果它指向的文件夹不再存在，Jupyter Notebook 将无法启动。

错误：运行爬网程序失败

如果 AWS Glue 未能成功运行爬网程序来对您的数据进行分类，则可能是由于以下原因之一。首先检查 AWS Glue 控制台爬网程序列表中是否列出了错误。检查爬网程序名称旁边是否有 感叹号图标，并将鼠标指针悬停在该图标上以查看任何相关消息。

- 在“日志”下`/aws-glue/crawlers`查看搜寻器运行的 CloudWatch 日志。

错误：分区未更新

如果您在运行 ETL 作业时没有在数据目录中更新分区，那么日志中该DataSink类的这些 CloudWatch 日志语句可能会有所帮助：

- “Attempting to fast-forward updates to the Catalog - nameSpace:”— 显示该作业试图修改哪个数据库、表和 catalogId。如果此处没有此语句，请检查 enableUpdateCatalog 是否设置为 true 并作为 getSink() 参数正确地传递到 additional_options 中。
- “Schema change policy behavior:”— 显示您传入的架构 updateBehavior 值。
- “Schemas qualify (schema compare):”– 将为 true 或 false。
- “Schemas qualify (case-insensitive compare):”– 将为 true 或 false。
- 如果两者均为 false 且您的updateBehavior未设置为 UPDATE_IN_DATABASE，则您的 DynamicFrame 架构必须相同或包含数据目录表架构中显示的列的子集。

有关更新分区的更多信息，请参阅 使用 AWS Glue ETL 任务在 Data Catalog 中更新架构并添加新分区。

错误：由于版本不匹配，作业书签更新失败

您可能正在尝试对 AWS Glue 任务进行参数化，以便在 Amazon S3 中的不同数据集上应用相同的转换/逻辑。您想在提供的位置跟踪处理过的文件。当您在同一源存储桶上运行同一作业并同时写入相同/不同目标（并发 > 1）时，该作业将失败并显示此错误：

```
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
```

解决方案：将并发设置为 1，或不同时运行该作业。

目前， AWS Glue 书签不支持并发作业运行，提交将失败。

错误：启用作业书签后，作业正在重新处理数据

在某些情况下，您可能启用了 AWS Glue 作业书签，但是您的 ETL 作业正在重新处理之前运行中已经处理过的数据。检查此错误的以下常见原因：

最大并发数

如果将作业的最大并发运行次数设置为大于默认值 1 的值，可能会干扰作业书签。作业书签检查对象的上次修改时间以验证哪些对象需要重新处理时可能出现这种情况。有关更多信息，请参阅在中配置 Spark 作业的作业属性 AWS Glue中的最大并发数讨论。

缺少作业对象

确保您的作业运行脚本以下面的提交结束：

```
job.commit()
```

包含此对象时，会 AWS Glue 记录作业运行的时间戳和路径。如果您使用相同的路径再次运行作业，则仅 AWS Glue 处理新文件。如果您未包含此对象并且已启用作业书签，该作业将重新处理已处理的文件和新文件，并在作业的目标数据存储中创建冗余。

缺少转换上下文参数

转换上下文是 `GlueContext` 类中的一个可选参数，但如果不包含该参数，作业书签将不起作用。要解决此错误，请在创建时添加转换上下文参数 DynamicFrame，如下所示：

```
sample_dynF=create_dynamic_frame_from_catalog(database,
 table_name,transformation_ctx="sample_dynF")
```

输入源

如果将关系数据库（JDBC 连接）用作输入源，则仅当表的主键按顺序排列时，作业书签才起作用。作业书签适用于新行，但不适用于更新的行。这是因为作业书签查找已存在的主键。如果您的输入源为 Amazon Simple Storage Service（Amazon S3），则这不适用。

上次修改时间

对于 Amazon S3 输入源，任务书签将通过检查对象的上次修改时间而不是文件名来验证哪些对象需要重新处理。如果您的输入源数据在上次作业运行后已修改，则再次运行作业时将重新处理这些文件。

错误：中 VPC 之间的故障转移行为 AWS Glue

在 AWS Glue 4.0 及之前版本中，以下过程用于任务的故障转移。

摘要：在提交作业运行时选择 AWS Glue 连接。如果作业运行遇到一些问题（缺少 IP 地址、与源的连接、路由问题），则作业运行将失败。如果配置了重试，则 AWS Glue 将使用相同的连接重试。

1. 对于每次运行尝试，AWS Glue 都会按照任务配置中列出的顺序检查连接的运行状况，直到找到可以使用的顺序为止。如果可用区 (AZ) 出现故障，来自该可用区的连接将无法通过检查并被跳过。
2. AWS Glue 使用以下内容验证连接：
   - 检查 Amazon VPC ID 和子网是否有效。
   - 检查 NAT 网关或 Amazon VPC 端点是否存在。
   - 检查子网分配的 IP 地址是否超过 0 个。
   - 检查可用区是否运行正常。

   AWS Glue 提交作业运行时无法验证连接。
3. 对于使用 Amazon VPC 的作业，所有驱动程序和执行程序都将在同一个可用区中创建，并在提交作业运行时选择连接。
4. 如果配置了重试，则 AWS Glue 将使用相同的连接重试。这是因为我们不能保证此连接的问题是长期存在的。如果某个可用区出现故障，则该可用区中运行的现有作业（取决于作业运行的阶段）可能会失败。重试应该会检测到可用区故障，然后为新的运行选择另一个可用区。

解决爬网程序使用 Lake Formation 凭证时出现的爬网程序错误

使用 Lake Formation 凭证配置爬网程序时，通过以下信息诊断和修复各种问题。

错误：S3 位置：s3://examplepath 未注册

要让爬网程序使用 Lake Formation 凭证运行，您需要先设置 Lake Formation 权限。要解决此错误，请向 Lake Formation 注册目标 Amazon S3 位置。有关更多信息，请参阅 Registering an Amazon S3 location（注册 Amazon S3 位置）。

错误：用户/角色未授权执行：资源上的 lakeformation:GetDataAccess

请使用 IAM 控制台或 AWS CLI 将 `lakeformation:GetDataAccess` 权限添加至爬网程序。获得此权限后，Lake Formation 将授权访问数据的临时凭证请求。请参见以下策略：

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

```
  }
}
```

错误：其上的 Lake Formation 权限不足（数据库名称：exampleDatabase，表名：exampleTable）

在 Lake Formation 控制台 (https://console.aws.amazon.com/lakeformation/) 中，授予爬网程序针对数据库的角色访问权限（ `Create`、`Describe`、`Alter`），该数据库指定为输出数据库。您也可以授予对表的权限。有关更多信息，请参阅 Granting database permissions using the named resource method（使用 指定的资源方法授予数据库权限）。

错误：s3://examplepath 上的 Lake Formation 权限不足

1. 跨账户爬取
   a. 使用注册 Amazon S3 存储桶的账户（账户 B）登录 Lake Formation 控制台 (https://console.aws.amazon.com/lakeformation/)。向将运行爬网程序的账户（账户 A）授予数据位置权限。这将允许爬网程序从目标 Amazon S3 位置读取数据。
   b. 在创建爬网程序的账户（账户 A）中，授予用于爬网程序运行的 IAM 角色目标 Amazon S3 位置上的数据位置权限，以使爬网程序能够从 Lake Formation 中的目标读取数据。有关更多信息，请参阅 Granting data location permissions (external account)（授予数据位置权限（外部账户））。
2. 账户内（爬网程序与注册的 Amazon S3 位置位于同一账户）爬取 - 向用于爬网程序在 Amazon S3 位置上运行的 IAM 角色授予数据位置权限，以便爬网程序可以从 Lake Formation 中的目标读取数据。有关更多信息，请参阅 Granting data location permissions (same account)（授予数据位置权限（同一账户））。

有关使用 Lake Formation 凭证配置爬网程序的常见问题

1. 如何使用 AWS 控制台将爬网程序配置为使用 Lake Formation 凭证运行？

   在 AWS Glue 控制台（https://console.aws.amazon.com/glue/）中配置爬网程序时，选择选项 Use Lake Formation credentials for crawling Amazon S3 data source（使用 Lake Formation 凭证爬取 Amazon S3 数据来源）。对于跨账户爬取，请指定向 Lake Formation 注册的目标 Amazon S3 位置的 AWS 账户 ID。对于账户内爬取，accountId 字段是可选的。
2. 如何使用 AWS CLI 将爬网程序配置为使用 Lake Formation 凭证运行？

   在 `CreateCrawler` API 调用期间，添加 `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 凭证的爬网程序支持哪些目标？

   使用 Lake Formation 凭证的爬网程序仅支持 Amazon S3（账户内和跨账户爬取）和账户内数据目录目标（其底层位置为 Amazon S3）和 Apache Iceberg 目标。

4. 作为使用 Lake Formation 凭证的单个爬网程序的一部分，我是否可以爬取多个 Amazon S3 桶？

   不可以，对于使用 Lake Formation 凭证售卖的爬取目标，基础 Amazon S3 位置必须属于同一个桶。例如，如果它们在同一个桶 (bucket1) 下，则客户可以使用多个目标位置（`s3://bucket1/folder1, s3://bucket1/folder2`）。不支持指定不同的桶 (s3://bucket1/folder1, s3://bucket2/folder2)。

对日志中的 AWS Glue for Ray 错误进行故障排除

AWS Glue 提供对作业运行期间由 Ray 进程发出的日志的访问权限。如果您在 Ray 作业中遇到错误或意外行为，请先从日志中收集信息以确定失败原因。我们还为交互式会话提供类似的日志。会话日志以 `/aws-glue/ray/sessions` 前缀提供。

当您的作业运行时，日志行会实时发送到 CloudWatch。运行完成后，打印语句会附加到 CloudWatch 日志中。作业运行后，日志将保留两周。

检查 Ray 作业日志

当作业失败时，收集您的作业名称和作业运行 ID。您可以在 AWS Glue 控制台中找到这些内容。导航到作业页面，然后导航到 Runs（运行）选项卡。Ray 作业日志存储在以下专用 CloudWatch 日志组中。

- `/aws-glue/ray/jobs/script-log/` — 存储您的主 Ray 脚本发出的日志。
- `/aws-glue/ray/jobs/ray-monitor-log/` — 存储 Ray 自动缩放器进程发出的日志。这些日志是针对头节点生成的，而不是为其他 Worker 节点生成的。
- `/aws-glue/ray/jobs/ray-gcs-logs/` - 存储 GCS（全局控制存储）进程发出的日志。这些日志是针对头节点生成的，而不是为其他 Worker 节点生成的。

- `/aws-glue/ray/jobs/ray-process-logs/` - 存储在头节点上运行的其他 Ray 进程（主要是控制面板代理）发出的日志。这些日志是针对头节点生成的，而不是为其他 Worker 节点生成的。
- `/aws-glue/ray/jobs/ray-raylet-logs/` - 存储每个 raylet 进程发出的日志。这些日志是在每个 Worker 节点的单个数据流中收集的。
- `/aws-glue/ray/jobs/ray-worker-out-logs/` - 存储集群中每个工作线程的 `stdout` 日志。这些日志是为每个 Worker 节点生成的，包括头节点。
- `/aws-glue/ray/jobs/ray-worker-err-logs/` - 存储集群中每个工作线程的 `stderr` 日志。这些日志是为每个 Worker 节点生成的，包括头节点。
- `/aws-glue/ray/jobs/ray-runtime-env-log/` - 存储有关 Ray 设置过程的日志。这些日志是为每个 Worker 节点生成的，包括头节点。

Ray 作业错误故障排除

要了解 Ray 日志组的组织结构，并找到可以帮助您解决错误的日志组，需要了解有关 Ray 架构的背景信息。

在 AWS Glue ETL 中，工作线程对应于一个实例。当您为某项 AWS Glue 作业配置工作线程时，您需要设置专用于作业的实例的类型和数量。Ray 以不同的方式使用工件一词。

Ray 使用头节点和 Worker 节点来区分 Ray 集群中实例的职责。Ray Worker 节点可以托管多个操作者进程，这些进程执行计算以实现分布式计算的结果。运行函数副本的操作者称为副本。副本操作者也可以称为工作线程进程。副本也可以在头节点上运行，该节点被称为头节点，因为它运行其他进程来协调集群。

参与计算的每个操作者都会生成自己的日志流。这为我们提供了一些见解：

- 发出日志的进程数量，可能大于分配给作业的工作线程数量。通常，每个实例上的每个核心都有一个操作者。
- Ray 头节点发出集群管理和启动日志。相比之下，Ray Worker 节点只会发出在其上执行的工作的日志。

有关 Ray 架构的更多信息，请参阅 Ray 文档中的架构白皮书。

问题领域：Amazon S3 访问权限

检查作业运行的失败消息。如果这不能提供足够的信息，请检查 `/aws-glue/ray/jobs/script-log/`。

问题领域：PIP 依赖关系管理

检查 /aws-glue/ray/jobs/ray-runtime-env-log/。

问题领域：检查主进程中的中间值

向主脚本写入 stderr 或从主脚本写出 stdout，然后从 /aws-glue/ray/jobs/script-log/ 中检索日志。

问题领域：检查子进程中的中间值

从 remote 函数写入 stderr 或写出 stdout。然后，从 /aws-glue/ray/jobs/ray-worker-out-logs/ 或 /aws-glue/ray/jobs/ray-worker-err-logs/ 检索日志。您的函数可能已在任何副本上运行，因此您可能需要检查多个日志才能找到预期的输出。

问题领域：解释错误消息中的 IP 地址

在某些错误情况下，您的作业可能会发出包含 IP 地址的错误消息。这些 IP 地址是临时性的，供集群用来识别节点并在节点之间进行通信。根据 IP 地址，节点的日志将发布到带有唯一后缀的日志流。

在 CloudWatch 中，您可以通过识别此后缀来筛选日志，以检查特定于此 IP 地址的日志。例如，给定 *FAILED_IP* 和 *JOB_RUN_ID*，您可以用以下方式标识后缀：

```
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 机器学习异常

本主题介绍与机器学习相关的 AWS Glue 异常的 HTTP 错误代码和字符串。为执行操作时可能发生的每个机器学习活动提供错误代码和错误字符串。此外，您还可以看到是否可以重试导致错误的操作。

CancelMLTaskRunActivity

此活动具有以下异常：

- EntityNotFoundException (400)

- “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”
- “在转换 [transformName] 的账户 [accountId] 中找不到 [taskRunId] 的 ML 任务运行。”

是否确定重试：否。

CreateMLTaskRunActivity

此活动具有以下异常：

- InvalidInputException (400)
  - “由于意外输入而导致内部服务失败。”
  - “应在转换中指定 AWS Glue 表输入源。”
  - “输入源列 [columnName] 具有在目录中定义的无效数据类型。”
  - “必须只提供一个输入记录表。”
  - “应指定数据库名称。”
  - “应指定表名称。”
  - “架构未在转换上定义。”
  - “架构应包含给定的主键：[primaryKey]。”
  - “获取数据目录架构时出现问题：[message]。”
  - “无法同时设置最大容量和工作线程数量/类型。”
  - “应同时设置 WorkerType 和 NumberOfWorkers。”
  - “MaxCapacity 应大于或等于 [maxCapacity]。”
  - “NumberOfWorkers 应大于或等于 [maxCapacity]。”
  - “最大重试次数应为非负值。”
  - “查找匹配参数尚未设置。”
  - “必须在查找匹配项参数中指定主键。”

  是否确定重试：否。
- AlreadyExistsException (400)
  - “名为 [transformName] 的转换已存在。”

  是否确定重试：否。
- IdempotentParameterMismatchException (400)

- “转换 [transformName] 的幂等创建请求的参数不匹配。”

是否确定重试：否。

- InternalServiceException (500)
  - “依赖关系失败。”

是否确定重试：是。

- ResourceNumberLimitExceededException (400)
  - “ML 转换计数 ([count]) 已超过 [limit] 次转换的限制。”

是否确定重试：是（一旦您删除了一个转换以便为这个新转换腾出空间）。

DeleteMLTransformActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform”

是否确定重试：否。

GetMLTaskRunActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”
  - “在转换 [transformName] 的账户 [accountId] 中找不到 [taskRunId] 的 ML 任务运行。”

是否确定重试：否。

GetMLTaskRunsActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”

- “在转换 [transformName] 的账户 [accountId] 中找不到 [taskRunId] 的 ML 任务运行。”

是否确定重试：否。

GetMLTransformActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”

  是否确定重试：否。

GetMLTransformsActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”

  是否确定重试：否。
- InvalidInputException (400)
  - “账户 ID 不能为空。”
  - “列 [column] 不支持排序。”
  - “[column] 不能为空。”
  - “由于意外输入而导致内部服务失败。”

  是否确定重试：否。

GetSaveLocationForTransformArtifactActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”

  是否确定重试：否。

- InvalidInputException (400)
  - “不支持的构件类型 [artifactType]。”
  - “由于意外输入而导致内部服务失败。”

  是否确定重试：否。

GetTaskRunArtifactActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”
  - “在转换 [transformName] 的账户 [accountId] 中找不到 [taskRunId] 的 ML 任务运行。”

  是否确定重试：否。
- InvalidInputException (400)
  - “文件名‘[fileName]’无效，无法发布。”
  - “无法检索 [taskType] 任务类型的构件。”
  - “无法检索 [artifactType] 的构件。”
  - “由于意外输入而导致内部服务失败。”

  是否确定重试：否。

PublishMLTransformModelActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”
  - “找不到账户 ID - [accountId] - 和转换 ID - [transformId] 的版本为 [version] 的现有模型。”

  是否确定重试：否。
- InvalidInputException (400)
  - “文件名‘[fileName]’无效，无法发布。”
  - “元符号字符串 [string] 上的前导减号无效。”

- “[string] 结尾处的数字错误。”
- “字符串值 [string] 超过无符号长型值的范围。”
- “由于意外输入而导致内部服务失败。”

是否确定重试：否。

PullLatestMLTransformModelActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”

  是否确定重试：否。
- InvalidInputException (400)
  - “由于意外输入而导致内部服务失败。”

  是否确定重试：否。
- ConcurrentModificationException (400)
  - “由于竞争性插件的参数不匹配，无法创建要训练的模型版本。”
  - “转换 ID [transformId] 的 ML 转换模型过时或正在由另一个进程更新； 请重试。”

  是否确定重试：是。

PutJobMetadataForMLTransformActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”
  - “在转换 [transformName] 的账户 [accountId] 中找不到 [taskRunId] 的 ML 任务运行。”

  是否确定重试：否。
- InvalidInputException (400)
  - “由于意外输入而导致内部服务失败。”
  - “未知的作业元数据类型 [jobType]。”

- “必须提供任务运行 ID 才能更新。”

是否确定重试：否。

StartExportLabelsTaskRunActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”
  - “账户 ID [accountId] 中不存在 transformId [transformId] 的标签集。”

  是否确定重试：否。
- InvalidInputException (400)
  - “[message]。”
  - “提供的 S3 路径与转换不在同一个区域中。预期的区域为 [region]，但获得的区域为 [region]。”

  是否确定重试：否。

StartImportLabelsTaskRunActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”

  是否确定重试：否。
- InvalidInputException (400)
  - “[message]。”
  - “标签文件路径无效。”
  - “无法访问 [labelPath] 中的标签文件。[message]。”
  - “无法使用转换中提供的 IAM 角色。角色：[role]。”
  - “大小为 0 的标签文件无效。”
  - “提供的 S3 路径与转换不在同一个区域中。预期的区域为 [region]，但获得的区域为 [region]。”

  是否确定重试：否。

- ResourceNumberLimitExceededException (400)
  - “标签文件已超过 [limit] MB 的限制。”

  是否确定重试：否。考虑将标签文件拆分为几个较小的文件。

StartMLEvaluationTaskRunActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”

  是否确定重试：否。
- InvalidInputException (400)
  - “必须只提供一个输入记录表。”
  - “应指定数据库名称。”
  - “应指定表名称。”
  - “查找匹配参数尚未设置。”
  - “必须在查找匹配项参数中指定主键。”

  是否确定重试：否。
- MLTransformNotReadyException (400)
  - “此操作只能应用于处于 READY 状态的转换。”

  是否确定重试：否。
- InternalServiceException (500)
  - “依赖关系失败。”

  是否确定重试：是。
- ConcurrentRunsExceededException (400)
  - “ML 任务运行计数 [count] 已超过 [limit] 个任务运行的转换限制。”
  - “ML 任务运行计数 [count] 已超过 [limit] 个任务运行的限制。”

  是否确定重试：是，等待任务运行完成后。

StartMLLabelingSetGenerationTaskRunActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”

  是否确定重试：否。
- InvalidInputException (400)
  - “必须只提供一个输入记录表。”
  - “应指定数据库名称。”
  - “应指定表名称。”
  - “查找匹配参数尚未设置。”
  - “必须在查找匹配项参数中指定主键。”

  是否确定重试：否。
- InternalServiceException (500)
  - “依赖关系失败。”

  是否确定重试：是。
- ConcurrentRunsExceededException (400)
  - “ML 任务运行计数 [count] 已超过 [limit] 个任务运行的转换限制。”

  是否确定重试：是，在任务运行完成后。

UpdateMLTransformActivity

此活动具有以下异常：

- EntityNotFoundException (400)
  - “在具有句柄 [transformName] 的账户 [accountId] 中找不到 MLTransform。”

  是否确定重试：否。
- InvalidInputException (400)
  - “另一个名为 [transformName] 的转换已经存在。”
  - “[message]。”

- “转换名称不能为空。”
- “无法同时设置最大容量和工作线程数量/类型。”
- “应同时设置 WorkerType 和 NumberOfWorkers。”
- “MaxCapacity 应大于或等于 [minMaxCapacity]。”
- “NumberOfWorkers 应大于或等于 [minNumWorkers].”
- “最大重试次数应为非负值。”
- “由于意外输入而导致内部服务失败。”
- “查找匹配参数尚未设置。”
- “必须在查找匹配项参数中指定主键。”

  是否确定重试：否。

- AlreadyExistsException (400)
  - “名为 [transformName] 的转换已存在。”

  是否确定重试：否。

- IdempotentParameterMismatchException (400)
  - “转换 [transformName] 的幂等创建请求的参数不匹配。”

  是否确定重试：否。

AWS Glue 限额

您可以联系 AWS Support，请求增加《AWS 一般参考》中列出的服务限额。除非另有说明，否则，每个配额都特定于区域。有关更多信息，请参阅 AWS Glue 终端节点和配额。

提高 AWS Glue 性能

性能优化的基准策略

为了提高 AWS Glue 性能，您可以考虑更新某些与性能相关的 AWS Glue 参数。准备优化参数时，请遵循以下最佳实践：

- 应首先确定性能目标，然后再开始确定性能问题。
- 应首先使用指标来确定问题，然后再尝试更改优化参数。

为确保在优化作业时获得稳定一致的结果，应为优化工作制定基线策略。

性能优化的工作流通常如下：

1. 确定性能目标。
2. 衡量指标。
3. 识别瓶颈。
4. 减少瓶颈的影响。
5. 重复第 2-4 步，直到达到预期目标为止。

适合作业类型的微调策略

Spark 作业 — 按照 AWS 规范性指导上的 Apache Spark 作业性能调整 AWS Glue 最佳实践中的指导进行操作。

其他作业 — 您可以通过调整 AWS Glue 其他运行时环境中可用的策略来调整 Ray 和 AWS Glue Python shell 作业。

提高 AWS Glue for Apache Spark 作业的性能

要提高 AWS Glue for Spark 的性能，建议更新某些与性能相关的 AWS Glue 和 Spark 参数。

要详细了解如何通过指标识别瓶颈以及减少其影响的具体策略，请参阅《AWS 规范性指南》中的 优化 AWS Glue for Apache Spark 作业性能的最佳实践。本指南介绍了在所有运行时环境中适用于 Apache Spark 的关键主题，例如 Spark 架构和弹性分布式数据集。使用这些主题，本指南将可指导您实施针对性的性能微调策略，例如优化随机排序和并行化任务。

您可以通过配置 AWS Glue 来显示 Spark UI，从而识别瓶颈。有关更多信息，请参阅 the section called “使用 Spark UI 进行监控”。

此外，AWS Glue 还提供了可能适用于作业所连接的特定数据存储类型的性能功能。有关数据存储性能参数的参考信息请参阅 the section called “连接参数”。

在 AWS Glue ETL 中通过下推优化读取

下推是一种优化技术，它可以将检索数据的逻辑推向离数据源更近的地方。源可以是数据库或文件系统，例如 Amazon S3。直接在源端执行某些操作时，无需将所有数据通过网络传送到由 AWS Glue 管理的 Spark 引擎，从而节省时间和处理能力。

换言之，下推可以减少数据扫描量。要详细了解确定何时适合使用这种技术的过程，请参阅《AWS 规范性指南》中“优化 AWS Glue for Apache Spark 作业性能的最佳实践”指南中的 减少数据扫描量。

对存储在 Amazon S3 上的文件的谓词下推

在 Amazon S3 上处理按前缀组织的文件时，可以通过定义下推谓词来筛选目标 Amazon S3 路径。可以直接将筛选器应用于 AWS Glue Data Catalog 中存储的分区元数据，而不必读取完整的数据集并在 `DynamicFrame` 中应用筛选器。这种方法允许您有选择地列出和只读必要的数据。有关此过程的更多信息，包括按分区写入存储桶，请参阅 the section called “管理分区”。

通过使用 `push_down_predicate` 参数，可以在 Amazon S3 中实现谓词下推。假设按年、月和日分区的 Amazon S3 中的一个存储桶。如果您想检索 2022 年 6 月的客户数据，可以指示 AWS Glue 仅读取相关的 Amazon S3 路径。在本例中，`push_down_predicate` 为 `year='2022' and month='06'`。综上所述，可以实现读取操作，如下所示：

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()
```

在前面的场景中，push_down_predicate 从 AWS Glue Data Catalog 中检索所有分区的列表，并在读取底层 Amazon S3 文件之前对其进行筛选。尽管这在大多数情况下都有帮助，但在处理具有数百万个分区的数据集时，列出分区的过程可能很耗时。为了解决这个问题，可以使用服务器端的分区修剪来提高性能。这通过在 AWS Glue Data Catalog 中为数据建立分区索引来完成。有关分区索引的更多信息，请参阅 the section called “使用分区索引”。然后，您可以使用 catalogPartitionPredicate 选项来引用索引。有关使用 catalogPartitionPredicate 检索分区的示例，请参阅 the section called “目录分区谓词”。

使用 JDBC 源时下推

GlueContext 中使用的 AWS Glue JDBC 读取器通过提供可以直接在源上运行的自定义 SQL 查询，支持对支持的数据库进行下推。这可以通过设置 sampleQuery 参数来实现。您的示例查询可以指定要选择的列，还可以提供下推谓词来限制传输到 Spark 引擎的数据。

默认情况下，示例查询在单个节点上运行，这可能会在处理大量数据时导致作业失败。要使用此功能大规模查询数据，您应该通过设置 enablePartitioningForSampleQuery 为 true 来配置查询分区，这将通过您选择的键将查询分发到多个节点。查询分区还需要一些其他必要的配置参数。有关查询分区的更多信息，请参阅 the section called “从 JDBC 并行读取”。

设置 enablePartitioningForSampleQuery 时，AWS Glue 会在查询数据库时将您的下推谓词与分区谓词组合在一起。sampleQuery 必须以 AND for AWS Glue 结尾才能附加分区条件。（如果您未提供下推谓词，则 sampleQuery 必须以 WHERE 结尾）。请参阅下面的示例，其中我们下推一个谓词以仅检索 id 大于 1000 的行。此 sampleQuery 将仅返回 id 大于指定值的行的名称和位置列：

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 在数据目录和底层数据存储中的名称不同，则必须在 sample_query 中提供底层表的名称，因为查询将传递到底层数据存储。

您也可以在不与 AWS Glue Data Catalog 集成的情况下对 JDBC 表进行查询。您可以通过提供 useConnectionProperties 和 connectionName 来重用来自先前存在连接的凭证，而不必提供用户名和密码作为该方法的参数。在本例中，我们从名为 my_postgre_connection 的连接检索凭证。

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 中下推的注意事项和限制

作为一个概念，“下推”适用于从非串流源读取数据。AWSGlue 支持多种信号源，下推的能力取决于源和连接器。

- 连接到 Snowflake 时，您可以使用 `query` 选项。AWS Glue 4.0 及更高版本的 Redshift 连接器中也有类似的功能。有关使用 `query` 从 Snowflake 读取内容的更多信息，请参阅 the section called “从 Snowflake 读取”。
- DynamoDB ETL 读取器不支持筛选条件或下推谓词。MongoDB 和 DocumentDB 也不支持这种功能。
- 从以开放表格式存储在 Amazon S3 中的数据中读取数据时，Amazon S3 中文件的分区方法已不再足够。要使用开放表格式从分区读取和写入，请查阅格式文档。
- DynamicFrame 方法不执行 Amazon S3 投影下推。所有列都将从传递谓词筛选器的文件中读取。
- 在 AWS Glue 中使用 `custom.jdbc` 连接器时，下推的能力取决于源和连接器。请查看相应的连接器文档，以确认它是否以及如何支持 AWS Glue 中的下推。

将 Auto Scaling 用于 AWS Glue

弹性伸缩现在可用于使用 AWS Glue 版本 3.0 或更高版本的 AWS Glue ETL 和串流任务。

启用 Auto Scaling 后，您将获得以下优势：

- AWS Glue 将根据各个阶段的并行性或任务运行的微批处理，向集群中自动添加工件，以及从集群中自动删除工件。
- 它将使您无需试验和决定要为您的 AWS Glue ETL 任务分配的工件数量。
- 如果您选择最大工件数量，AWS Glue 将为工作负载选择适当大小的资源。
- 您可以通过查看 AWS Glue Studio 中任务运行详细信息页面上的 CloudWatch 指标，了解集群大小在任务运行过程中如何变化。

AWS Glue ETL 的 Auto Scaling 和串流任务可使您的 AWS Glue 任务的计算资源实现按需纵向扩展和缩减。按需纵向扩展可帮助您最初只在任务运行启动时分配所需的计算资源，还可以在任务期间根据需求预配所需的资源。

Auto Scaling 还支持在任务过程中动态缩减 AWS Glue 任务资源。在任务运行过程中，当 Spark 应用程序请求更多执行程序时，将向集群添加更多工件。当执行程序在没有活动计算任务的情况下处于空闲状态时，则将删除该执行程序和相应的工件。

Auto Scaling 帮助 Spark 应用程序优化成本和利用率的常见场景包括：Spark 驱动程序列出 Amazon S3 中的大量文件，或在执行程序处于非活动状态时执行负载，Spark 阶段由于过度配置而仅运行数个执行程序，以及在 Spark 各个阶段的数据偏差或不均衡的计算需求。

要求

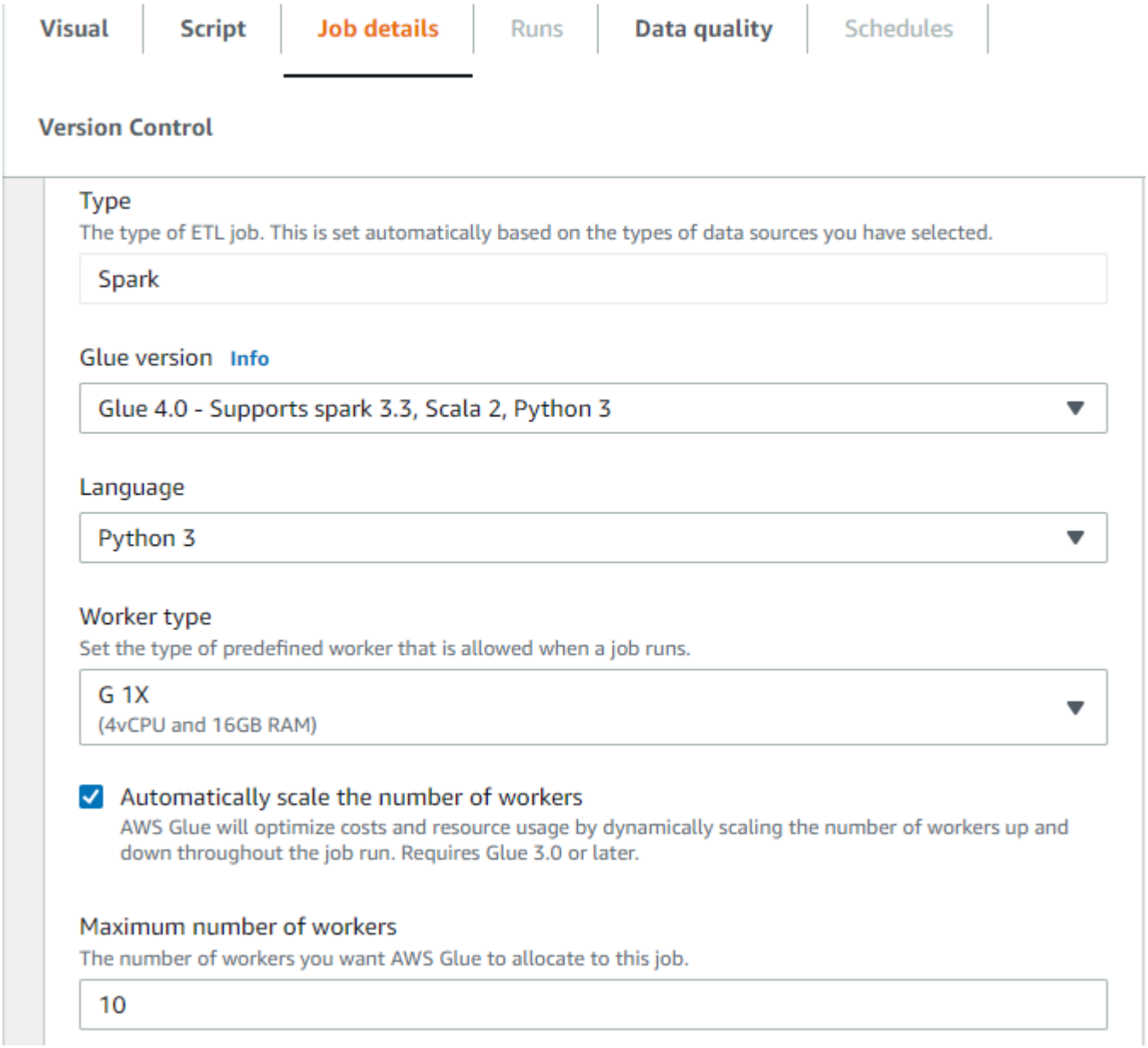
Auto Scaling 仅适用于 AWS Glue 版本 3.0 或更高版本。要使用 Auto Scaling，您可以按照迁移指南将现有任务迁移到 AWS Glue 版本 3.0 或更高版本，或者使用 AWS Glue 版本 3.0 或更高版本创建新作业。

Auto Scaling 可用于兼有 `G.1X`、`G.2X`、`G.4X`、`G.8X` 或 `G.025X` 或（仅用于流式处理作业）工件类型的 AWS Glue 作业。不支持标准 DPU。

在 AWS Glue Studio 中启用自动扩缩

在 AWS Glue Studio 中的 Job details（任务详细信息）选项卡上，将类型选择为 Spark 或 Spark Streaming，并将 Glue version（Glue 版本）选择为 **`Glue 3.0`** 或 **`Glue 4.0`**。随后将在 Worker type（工件类型）下方出现一个复选框。

- 选择 Automatically scale the number of workers（自动扩展工件数量）选项。
- 设置 Maximum number of workers（最大工件数量）以定义可提供给任务运行的最大工件数量。



使用 AWS CLI 或开发工具包启用自动扩缩

要通过 AWS CLI 为任务运行启用自动扩缩，使用以下配置运行 `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 任务运行完成后，您还可以调用 `get-job-run` 以检查 DPU-seconds 内任务运行的实际资源使用情况。请注意：为 AWS Glue 3.0 或更高版本上运行的批处理任务启用自动扩缩时，才会显示新字段 DPUSeconds。流式处理任务不支持此字段。

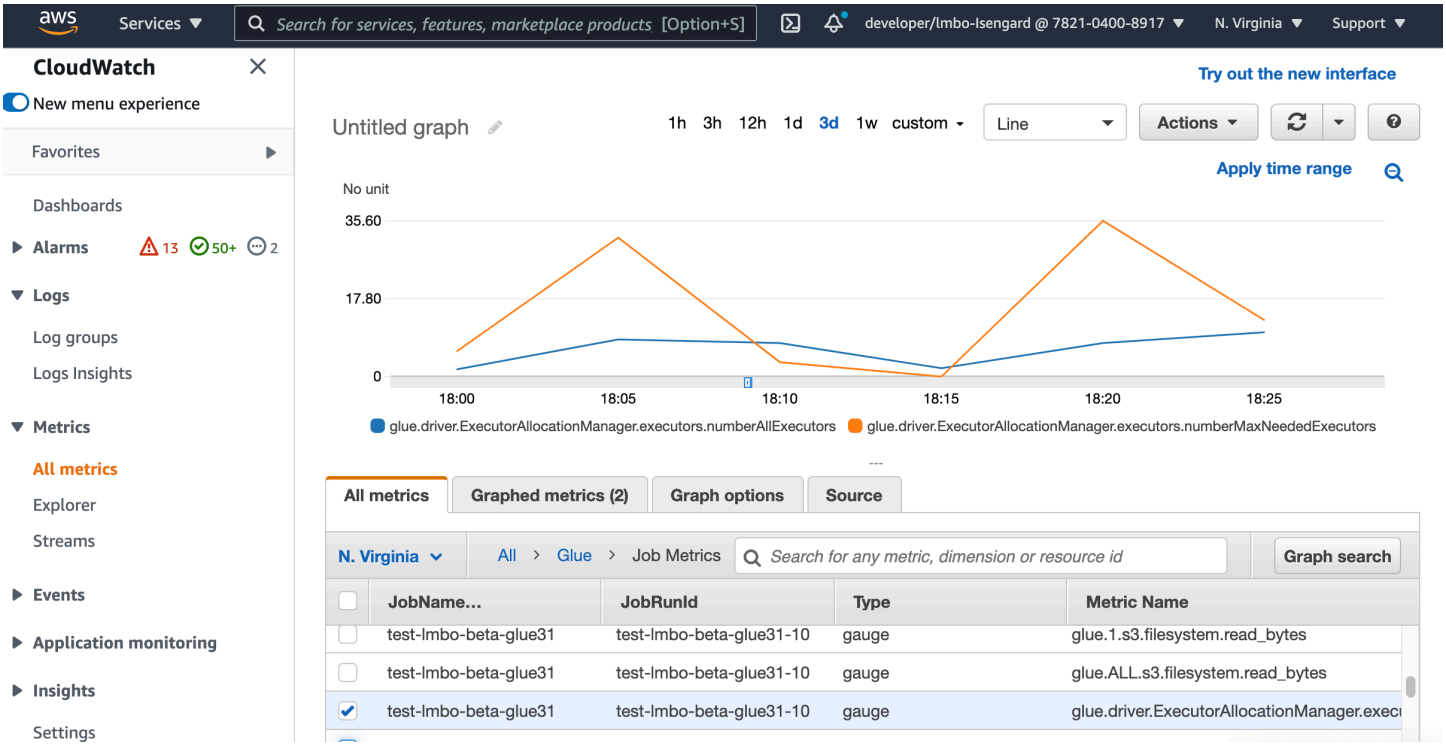
```
$ aws glue get-job-run --job-name your-job-name --run-id jr_xx --endpoint https://
glue.us-east-1.amazonaws.com --region us-east-1
{
    "JobRun": {
        ...
        "GlueVersion": "3.0",
        "DPUSeconds": 386.0
    }
}
```

您还可以使用具有相同配置的 AWS Glue 开发工具包配置启用弹性伸缩的任务运行。

使用 Amazon CloudWatch 指标监控 Auto Scaling

如果您启用了 Auto Scaling，则 CloudWatch 执行程序指标可用于您的 AWS Glue 3.0 或更高版本任务。这些指标可以用于监控使用 Auto Scaling 启用的 Spark 应用程序中执行程序的需求和优化使用情况。有关更多信息，请参阅 使用 Amazon CloudWatch 指标监控 AWS Glue。

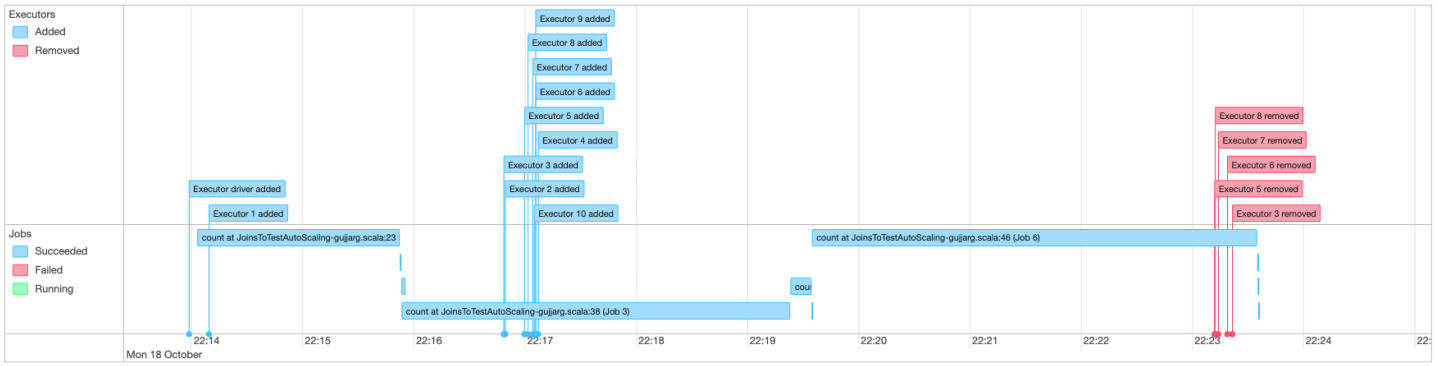
- glue.driver.ExecutorAllocationManager.executors.numberAllExecutors
- glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors



有关这些指标的更多详细信息，请参阅 监控 DPU 容量规划。

使用 Spark UI 监控 Auto Scaling

启用 Auto Scaling 后，您还可以使用 Glue Spark UI 监控根据 AWS Glue 任务中的需求借助动态纵向扩展及缩减添加和删除的执行程序。有关更多信息，请参阅 为 AWS Glue 作业启用 Apache Spark Web UI。



监控弹性伸缩任务运行的 DPU 使用情况

您可以使用 AWS Glue Studio 作业运行视图以检查自动扩缩作业的 DPU 使用情况。

1. 从 AWS Glue Studio 导航窗格中选择监控。此时将显示 Monitoring（监控）页面。

2. 向下滚动到任务运行图表。

3. 导航到您感兴趣的任务运行，然后滚动到 DPU 小时列以检查特定任务运行的使用情况。

限制

AWS Glue 串流 Auto Scaling 目前不支持串流 DataFrame 与在 `ForEachBatch` 外部创建的静态 DataFrame 联接。在 `ForEachBatch` 内部创建的静态 DataFrame 将按预期方式工作。

具有有界执行的工作负载分区

Spark 应用程序中的错误通常是由于效率低下的 Spark 脚本、大规模转换的分布式内存中执行，以及数据集异常引起的。有很多原因可能会导致驱动程序或执行程序内存不足问题，例如数据偏斜、列出太多的对象或大量数据随机排序。当您使用 Spark 处理大量积压数据时，通常会出现这些问题。

AWS Glue 可让您解决 OOM 问题，并通过工作负载分区使您的 ETL 处理工作更加轻松。启用工作负载分区后，每个 ETL 任务运行仅拾取未处理的数据，数据集大小或此任务运行要处理的文件数有一个上限。以后的任务运行将处理剩余的数据。例如，如果需要处理 1000 个文件，则可以将文件数设置为 500，并将它们分成两次任务运行。

仅 Amazon S3 数据源支持工作负载分区。

启用工作负载分区

您可以通过手动设置脚本中的选项或添加目录表属性来启用有界执行。

要在脚本中启用具有有界执行的工作负载分区，请执行以下操作：

1. 要避免重新处理数据，请在新任务或现有任务中启用任务书签。有关更多信息，请参阅使用任务书签来跟踪已处理的数据。
2. 修改脚本并在 AWS Glue `getSource` API 中设置其他选项中的有界限制。您还应该为任务书签设置转换上下文，以存储 `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", "")
```

要在数据目录表中启用具有有界执行的工作负载分区，请执行以下操作：

1. 在数据目录中的表结构的 parameters 字段中设置键值对。有关更多信息，请参阅查看和编辑表详细信息。
2. 设置数据集大小或处理的文件数量的上限：
   - 将 boundedSize 设置为数据集的目标大小（以字节为单位）。任务运行将在达到表中的目标大小后停止。
   - 将 boundedFiles 设置为目标文件数。处理目标数量的文件后，任务运行将停止。

> **Note**
>
> 您应该只设置一个 boundedSize 或 boundedFiles，因为仅支持单个界限。

设置一个 AWS Glue 触发器以自动运行作业

启用有界执行后，您可以设置 AWS Glue 触发器以自动运行任务，并在顺序运行中以增量方式加载数据。转至 AWS Glue 控制台并创建触发器，设置计划时间，然后附加到您的任务。然后，它将自动触发下一个任务运行并处理新一批数据。

您还可以使用 AWS Glue 工作流来编排多个任务，以并行处理来自不同分区的数据。有关更多信息，请参阅 AWS Glue 触发器和 AWS Glue 工作流。

有关使用案例和选项的更多信息，请参阅博客 Optimizing Spark applications with workload partitioning in AWS Glue。

AWS Glue 的已知问题

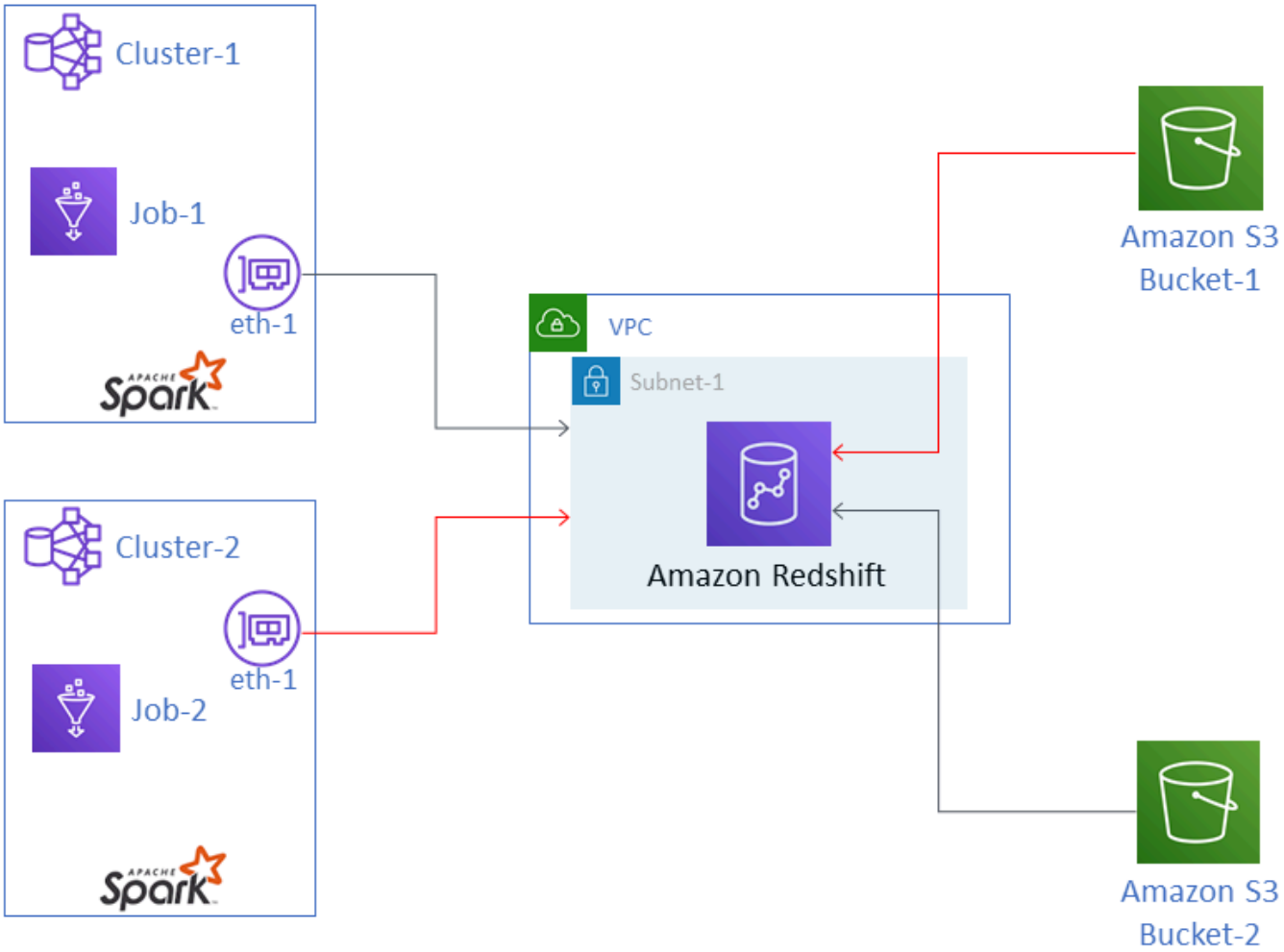
请注意 AWS Glue 的以下已知问题。

主题

防止跨作业数据访问

考虑在单个 AWS 账户中有两个 AWS Glue Spark 任务的情况，每个任务都在一个单独的 AWS Glue Spark 集群中运行。这些作业使用 AWS Glue 连接来访问同一 Virtual Private Cloud（VPC）中的资源。在这种情况下，在一个集群中运行的作业可能能够访问在另一个集群中运行的作业的数据。

下图说明了这种情况的一个示例。



在图中，AWS Glue Job-1 在 Cluster-1 中运行，Job-2 在 Cluster-2 中运行。这两个任务都使用位于 VPC 的 Subnet-1 中的同一个 Amazon Redshift 实例。Subnet-1 可以是公有子网或私有子网。

Job-1 将 Amazon Simple Storage Service（Amazon S3）Bucket-1 中的数据传输出来，并写入 Amazon Redshift 中。Job-2 对 Bucket-2 中的数据执行相同的操作。Job-1 使用 AWS Identity and Access Management (IAM) 角色 Role-1（未显示），该角色授予对 Bucket-1 的访问权限。Job-2 使用 Role-2（未显示），该角色授予对 Bucket-2 的访问权限。

这两个作业均具有使其能够与对方的集群进行通信的网络路径，从而能够访问对方的数据。例如，Job-2 可以访问 Bucket-1 中的数据。在图中，这显示为红色的路径。

为防止这种情况，我们建议您将不同的安全配置附加到 Job-1 和 Job-2。通过附加安全配置，可以凭借 AWS Glue 创建的证书阻止对数据的跨作业访问。安全配置可以是虚拟配置。也就是说，您可以创

建安全配置，而无需启用 Amazon S3 数据、Amazon CloudWatch 数据或任务书签的加密。所有三个加密选项都可以被禁用。

有关安全配置的信息，请参阅the section called “加密 AWS Glue 写入的数据”。

将安全配置附加到作业

1. 通过 https://console.aws.amazon.com/glue/ 打开 AWS Glue 控制台。
2. 在作业的 Configure the job properties (配置作业属性) 页面上，展开 Security configuration, script libraries, and job parameters (安全配置、脚本库和作业参数) 部分。
3. 在列表中选择一个安全配置。

AWS Glue 的文档历史记录

变更	说明	日期
对 AWS Glue 使用情况配置文件的 Support	管理员可以为账户内不同类别的用户（例如开发人员、测试人员和产品团队）创建 AWS Glue 使用情况配置文件。这种灵活性使管理员能够对每类用户应用不同的使用和成本控制。有关更多信息，请参阅设置 AWS Glue 使用情况配置文件。	2024年6月18日
支持 Spark 版 Salesforce AWS Glue 连接器	添加了有关 Salesforce 新 AWS Glue 连接器的信息。此功能允许你使用 AWS Glue Spark 在 AWS Glue 4.0 及更高版本中读取和写入到 Salesforce。有关更多信息，请参阅连接到 Salesforce。	2024年5月22日
AWS Glue (GA) 中的 Amazon Q 数据集成	Amazon Q 数据集成 AWS Glue 是一种新的生成式 AI 功能 AWS Glue ，它使数据工程师和 ETL 开发人员能够使用自然语言构建数据集成作业。工程师和开发人员可以让 Q 来创作作业、解决问题并回答有关 AWS Glue 数据集成的问题。有关更多信息，请参阅 AWS Glue中的 Amazon Q 数据集成。此功能包括对AwsGlueSessionUserRestrictedPolicy AwsGlueSessionUserRestricte	2024 年 4 月 30 日

	dNotebookServiceRole 、和AwsGlueSessionUserRestrictedServiceRole AWS 托管策略的更新。有关更多信息，请参阅AWS 托管策略的AWS Glue 更新。	
Amazon Q 数据集成中 AWS Glue （预览版）	Amazon Q 数据集成 AWS Glue 是一种新的生成式 AI 功能 AWS Glue ，它使数据工程师和 ETL 开发人员能够使用自然语言构建数据集成作业。工程师和开发人员可以让 Q 来创作作业、解决问题并回答有关 AWS Glue 数据集成的问题。有关更多信息，请参阅 AWS Glue中的 Amazon Q 数据集成。此功能包括对AwsGlueSessionUserRestrictedNotebookPolicy AWS 托管策略的更新。有关更多信息，请参阅AWS 托管策略的 AWS Glue 更新。	2024 年 1 月 30 日
更新 AWS Glue 直播文档	为 AWS Glue 直播添加了新的章节，其中包含新的和经过重新整理的内容。本内容描述了流媒体的工作原理 AWS Glue、实时数据处理的特征以及如何监控您的流媒体任务。有关更多信息，请参阅 AWS Glue 流式处理。	2023 年 12 月 27 日

支持使用精细敏感数据检测	借助检测敏感数据转换功能，可以检测、遮蔽或移除您定义的或由 AWS Glue 预定义的实体。您还可以借助精细操作对每个实体应用特定的操作。有关更多信息，请参阅使用精细敏感数据检测。	2023 年 11 月 26 日
Support 支持使用 AWS Glue 可观察性指标监控作业	使用 AWS Glue 可观测性指标可深入了解 AWS Glue 内部发生的情况，以便 Apache Spark 作业可以改进对问题的分类和分析。有关更多信息，请参阅使用 AWS Glue 可观测性指标进行监控。	2023 年 11 月 26 日
Support 支持 AWS Glue 数据质量中的异常检测	AWS Glue 数据质量异常检测将机器学习（ML）算法应用于一段时间内的数据统计信息，以检测难以通过规则检测到的异常模式，和隐藏的数据质量问题。有关更多信息，请参阅 AWS Glue 数据质量自动监测功能中的异常检测。	2023 年 11 月 26 日
更新为默认的 Spark 用户界面日志记录行为	生成 Spark 用户界面日志的 Spark 作业现在将使用不同的文件名模式进行写入，以支持 AWS Glue 控制台中的 Spark 用户界面。这不会改变 CloudWatch 日志行为。您可以通过更新任务配置恢复原来的行为。有关更多信息，请参阅使用 Apache Spark Web UI 监控作业。	2023 年 11 月 17 日

支持 Spark 中的 AWS Glue 新数据源	现在，内部原生支持与亚马逊 OpenSearch 服务、Azure SQL、适用于 NoSQL 的 Azure Cosmos、SAP HANA Teradata Vantage 和 Vertica 的连接。 AWS Glue此外，现在可以在 Studio 可视化编辑器中 AWS Glue 使用与这些数据源的连接以及 MongoDB。有关如何支持 Spark 的信息，请参阅 Spark 中的 AWS Glue ETL 连接类型和选项； AWS Glue 有关在 AWS Glue Studio 可视化编辑器中使用的信息，请参阅 “添加 AWS Glue 连接”。	2023 年 11 月 17 日
支持生成列统计信息	无需设置其他数据管道，即可计算 Parquet、ORC、JSON、ION、CSV 和 XML 等数据格式的 AWS Glue Data Catalog 表的列级统计数据。有关更多信息，请参阅使用列统计信息。	2023 年 11 月 16 日
支持 Iceberg 表的数据压缩	为了提高诸如Amazon Athena和Amazon EMR之类的 AWS 分析服务以及ETL任务的读取性能 AWS Glue ，数据目录为数据目录中的冰山表提供了托管压缩（一种将小型Amazon S3对象压缩成较大对象的过程）。有关更多信息，请参阅优化 Iceberg 表。	2023 年 11 月 13 日

更新了作业运行等待行为	在某些情况下，标准 Spark 和 Python Shell 作业运行现在将过渡到 `WAITING`，而不是立即变为 `FAILED`。有关更多信息，请参阅 AWS Glue 作业运行状态。	2023 年 11 月 8 日
AWS Glue Studio 用户指南已合并到 AWS Glue 开发者指南中	AWS Glue Studio 用户指南已移至开发者指南，用于为 AWS Glue Studio、AWS Glue 控制台和 AWS Glue Studio 编程访问创建单一的统一用户指南。	2023 年 10 月 25 日
AWSGlueServiceNotebookRole AWS 托管策略的更新	添加了有关 AWSGlueServiceNotebookRole AWS 托管策略小更新的信息。有关更多信息，请参阅AWS 托管策略的AWS Glue更新。	2023 年 10 月 9 日
AWS Glue Studio 支持五种新的内置转换	AWS Glue Studio 支持以下五种新的内置转换：记录匹配、移除空行、解析 JSON 列、提取 JSON 路径和正则表达式提取器。有关更多信息，请参阅编辑 AWS Glue 托管数据转换节点。	2023 年 8 月 11 日
AWSGlueServiceRole AWS 托管策略的更新	添加了有关 AWSGlueServiceRole AWS 托管策略小更新的信息。有关更多信息，请参阅AWS 托管策略的AWS Glue更新。	2023 年 8 月 4 日

支持爬取 Apache Hudi 表	添加了有关使用 AWS Glue 在 Amazon S3 存储桶中抓取 Hudi 表以及将 Hudi 表注册到的信息。 AWS Glue Data Catalog有关更多信息，请参阅 Which data stores can I crawl? 和 Crawler properties。	2023 年 7 月 21 日
AWSGlueConsoleFullAccess AWS 托管策略的更新	添加了有关 AWSGlueConsoleFullAccess AWS 托管策略小更新的信息。有关更多信息，请参阅AWS 托管策略的 AWS Glue更新。	2023 年 7 月 14 日
支持爬取 Apache Iceberg 表	添加了有关使用 AWS Glue 在 Amazon S3 存储桶中抓取 Iceberg 表以及将 Iceberg 表注册到的信息。 AWS Glue Data Catalog有关更多信息，请参阅 Which data stores can I crawl? 和 Crawler properties。	2023 年 7 月 7 日
Support w AWS Glue ith Ray	添加了有关 Wit AWS Glue h Ray 的信息，这是一款可以支持 AWS Glue 任务的新引擎。 AWS Glue 使用 Spark 内容重组现有内容以消除歧义。	2023 年 5 月 30 日

支持 AWS Glue 数据质量 (GA)	AWS Glue 数据质量现已正式发布。 AWS Glue 数据质量可帮助您评估和监控数据质量。有关如何在数据目录中使用 AWS Glue 数据质量的信息，请参阅AWS Glue 数据质量。要了解有关 AWS Glue 数据质量的信息 AWS Glue Studio，请参阅使用评估数据质量 AWS Glue Studio。	2023 年 5 月 24 日
为 Apache Spark 作业支持更大的工作线程类型	现在支持使用适用于 Apache Spark 作业的 G.4X 和 G.8X 工作线程类型。对于工作线程类型适合包含要求最高的转换、聚合、联接和查询的作业。有关更多信息，请参阅在 AWS Glue 中添加任务。	2023 年 5 月 8 日
支持在爬取表时创建分区索引	添加了有关爬网程序如何支持为爬网程序检测到的表创建分区索引的信息。有关更多信息，请参阅 Setting the partition index crawler configuration option。	2023 年 4 月 24 日
支持资源使用量指标	添加了有关在 Amazon 中查看服务资源使用情况和配置警报的信息 CloudWatch。有关更多信息，请参阅 AWS Glue resource monitoring。	2023 年 4 月 7 日
AWSGlueConsoleFullAccess AWS 托管策略的更新	添加了有关 AWSGlueConsoleFullAccess AWS 托管策略小更新的信息。有关更多信息，请参阅AWS 托管策略的 AWS Glue更新。	2023 年 3 月 28 日

添加了与 AWS SDK AWS Glue 配合使用的指南，并附有示例	《AWS Glue开发者指南》新增了两个章节，提供了帮助您AWS Glue使用 S AWS DK 的信息。有关更多信息，请参阅与 AWS SDK AWS Glue 配合使用和AWS Glue 使用 S AWS DK 的代码示例。	2023 年 2 月 23 日
更新了 IAM 的文档 AWS Glue	重新整理并添加了有关将 IAM 与配合 AWS Glue使用的信息。有关更多信息，请参阅 AWS Glue的身份和访问权限管理。	2023 年 2 月 15 日
AWS Glue 版本 4.0 支持运行流式处理 ETL 作业	添加了有关在 Glue 版本 4.0 中支持运行流式处理 ETL 作业的信息，以及连接到 Kafka 集群或适用于 Apache Kafka 集群的 Amazon Managed Streaming 和 Amazon Kinesis Data Streams 的新选项。有关更多信息，请参阅在 AWS Glue 中添加流式处理 ETL 作业和 AWS Glue 中的 ETL 连接类型和选项。	2023 年 2 月 8 日
支持爬取 MongoDB Atlas 数据来源	添加了有关使用 AWS Glue 抓取 MongoDB Atlas 数据源的信息。有关更多信息，请参阅我可以爬取哪些数据存储？、MongoDB 和 MongoDB Atlas 连接属性，以及使用 MongoDB 或 MongoDB Atlas 连接。	2023 年 2 月 6 日

支持使用本地 Delta Lake 连接器爬取 Delta Lake 表	添加了有关使用 AWS Glue 原生 Delta Lake 连接器爬取 Delta Lake 表的信息。此功能允许您使用 AWS 查询引擎直接查询 Delta 事务日志并使用诸如时空旅行和 ACID 保证之类的功能，还可以将来自 Amazon S3 事务文件的 Delta Lake 元数据同步到数据目录中，以便在 Lake Formation 中对您的查询启用列权限。有关更多信息，请参阅如何为 Delta Lake 数据存储指定配置选项和查询 Delta Lake 表。	2022 年 12 月 15 日
Support 对 AWS Glue 数据质量的支持（预览版）	Support 现已为 AWS Glue 数据质量（预览版）提供。 AWS Glue 使用 AWS Glue 3.0 时，数据质量可帮助您评估和监控数据质量。有关如何在数据目录中使用 AWS Glue 数据质量的信息，请参阅AWS Glue 数据质量（预览）。要了解有关 AWS Glue 数据质量的信息 AWS Glue Studio，请参阅使用评估数据质量 AWS Glue Studio。	2022 年 11 月 30 日

支持具有新功能和性能改进的全新 Amazon Redshift Spark 连接器	现在支持带有新的 JDBC 驱动程序的新 Amazon Redshift Spark 连接器，该连接器可用于 AWS Glue ETL 任务，用于构建 Apache Spark 应用程序，在 AmazonRedshift 中读取和写入数据，作为数据摄取和转换管道的一部分。有关更多信息，请参阅将数据移入和移出 Amazon Redshift。	2022 年 11 月 29 日
支持 AWS Glue 版本 4.0。	添加了有关 AWS Glue 版本 4.0 的支持的信息。功能包括对带有 Apache Hudi、Delta Lake 和 Apache Iceberg 的开放数据湖框架的原生支持，以及对基于 Amazon S3 的 Cloud Shuffle 存储插件（Apache Spark 插件）的原生支持，该插件可使用 Amazon S3 进行随机排序和弹性存储容量。有关更多信息，请参阅 AWS Glue 发布说明和将 AWS Glue 任务迁移到 AWS Glue 版本 4.0。	2022 年 11 月 28 日
AWS Glue Studio 现在提供自定义视觉转换	自定义视觉转换允许客户在团队之间定义、重用和共享特定于业务的 ETL 逻辑。有关更多信息，请参阅自定义视觉转换。	2022 年 11 月 28 日

支持使用 AWS Glue 爬网程序发布 JDBC 数据存储的元数据	现在支持使用 AWS Glue 爬网程序将注释和原始类型等元数据发布到 JDBC 数据存储的数据目录。有关更多信息，请参阅 Crawler 在数据目录表上设置的参数、C rawler 属性和JdbcTarget 结构。	2022 年 11 月 18 日
支持爬取 Snowflake 数据存储	现在支持 AWS Glue 用于爬取 Snowflake 表和视图，以及将元数据作为表条目发布到数据目录。对于 Amazon S3 中的 Snowflake 外部表，爬网程序还会爬取外部表的 Amazon S3 位置和文件格式类型，并填充为表参数。有关更多信息，请参阅我可以爬取哪些数据存储？、AWS Glue 连接属性和由爬网程序在数据目录表上设置的参数。	2022 年 11 月 18 日
支持改进 Spark 应用程序的随机排序管理	现在支持适用于 Apache Spark 的新 Cloud Shuffle 存储插件。有关更多信息，请参阅带有 Amazon S3 的 AWS Glue Spark shuffle 插件和适用于 Apache Spark 的 Cloud Shuffle 存储插件。	2022 年 11 月 15 日
添加了在加速爬取 Amazon S3 事件通知时对 Data Catalog 目标的支持	除了对 Amazon S3 目标的现有支持外，现在还支持使用 Amazon S3 事件通知加速对 Data Catalog 目标的爬取。有关更多信息，请参阅使用 Amazon S3 事件通知加速网络爬取。	2022 年 10 月 13 日

支持指定爬网程序可以创建的最大表数	现已支持指定允许爬网程序创建的最大表数。有关更多信息，请参阅如何指定允许爬网程序创建的最大表数。	2022 年 9 月 6 日
在 AWS Glue 中的 Python shell 任务中支持 Python 3.9	现在支持在 AWS Glue 的 Python shell 任务中运行与 Python 3.9 兼容的脚本，以及选择使用预打包的库集。有关更多信息，请参阅 AWS Glue 中的 Python shell 任务。	2022 年 8 月 11 日
Support 支持在空闲容量上运行非紧急或非时间敏感型 AWS Glue 作业	现在支持为非紧急任务（例如生产前任务、测试和一次性数据加载）配置灵活的任务运行。有关更多信息，请参阅在 AWS Glue 中添加任务。	2022 年 8 月 9 日
支持适用于流式处理任务的新 Worker 类型	现在提供适用于低容量流式处理任务的 `G.025X` 类型。有关更多信息，请参阅在 AWS Glue 中添加任务。	2022 年 7 月 14 日
支持在 AWS Glue 连接中使用 Kafka SASL	现在支持在 AWS Glue 连接中使用 Kafka SASL。有关更多信息，请参阅适用于客户端身份认证的 AWS Glue Kafka 连接属性。	2022 年 7 月 5 日
针对 Protobuf 架构的 Apache Kafka Connector 支持	现已提供针对 Protobuf 架构的 Apache Kafka Connector 支持有关更多信息，请参阅 AWS Glue 架构注册表。	2022 年 6 月 9 日

支持 AWS Glue 任务的自动扩缩 (GA)	添加了 AWS Glue 版本 3.0 中的任务使用自动扩缩的信息，从而动态扩缩计算资源。有关更多信息，请参阅为 AWS Glue 使用自动扩缩。	2022 年 4 月 14 日
更新了有关 AWS Glue 开发和测试 AWS Glue 任务脚本的文档	重组并添加了有关 AWS Glue 的可用开发和测试方法的信息，包括使用 Docker 进行开发的说明。有关更多信息，请参阅开发和测试 AWS Glue 任务脚本。	2022 年 3 月 14 日
将协议缓冲区作为受支持的数据格式添加到 AWS Glue 架构注册表	添加了有关 Protobuf 作为受支持的数据格式（除 AVRO 和 JSON 之外）的信息。有关更多信息，请参阅 AWS Glue 架构注册表。	2022 年 2 月 25 日
支持爬取 Delta Lake 表	添加了有关使用 AWS Glue 爬取 Delta Lake 表格的信息。有关更多信息，请参阅如何为 Delta Lake 数据存储指定配置选项。	2022 年 2 月 24 日
Support 对 AWS Glue 工作见解的支持	添加了有关使用 AWS Glue 作业见解来简化作业调试和 AWS Glue 作业优化的信息。有关更多信息，请参阅通过 AWS Glue 任务洞察进行监控。	2022 年 2 月 8 日

使用 VPC 终端节点支持网络爬取 Amazon S3 支持的数据目录表	除了 Amazon S3 数据商店之外，出于安全、审计或控制目的，您可以配置您的 Amazon S3 支持的数据目录表，使之仅由 Amazon Virtual Private Cloud 环境 (Amazon VPC) 访问。有关更多信息，请参阅使用 VPC 终端节点网络爬取 Amazon S3 数据存储或 Amazon S3 支持的数据目录表。	2022 年 2 月 3 日
支持 Lake Formation 受管表	添加了关于 AWS Glue 对 Lake Formation 受管表的支持的信息，这些表支持 ACID 事务、自动数据压缩和时间旅行查询。有关更多信息，请参阅 AWS Glue API 和 AWS Lake Formation 开发人员指南。	2021 年 11 月 30 日
为交互式会话和笔记本添加了新的 AWS 托管策略	新的 IAM 托管策略增强了用于交互 AWS Glue 式会话和笔记本的安全性。有关更多信息，请参阅适用于 AWS Glue的 AWS 托管策略。	2021 年 11 月 30 日
流式处理任务现在支持 Glue 架构注册表	您可以创建能够访问属于 Glue 架构注册表的表的流式处理任务。有关更多信息，请参阅 AWS Glue 结构注册表和在 AWS Glue中添加流式处理 ETL 任务。	2021 年 11 月 15 日

支持新的机器学习功能	添加了关于查找匹配项机器学习转换（包括递增匹配项和匹配项评分）的新功能的信息。有关更多信息，请参阅 查找递增匹配项和使用匹配项置信度分数估算匹配项质量。	2021 年 10 月 31 日
（私有预览版）支持 AWS Glue 灵活任务	增加了关于配置具有灵活执行类的 AWS Glue Spark 任务的信息，适用于启动和完成时间可能不同的时间不敏感型任务。有关更多信息，请参阅在 AWS Glue 中添加任务。	2021 年 10 月 29 日
支持使用 Amazon S3 事件通知加速网络爬取	添加了关于使用 Amazon S3 事件通知加速网络爬取的信息。有关更多信息，请参阅使用 Amazon S3 事件通知加速网络爬取。	2021 年 10 月 15 日
与访问控制和 VPC 相关的其他安全配置选项	添加了关于您如何在 AWS Glue 上配置新的访问控制权限以及 VPC 配置信息。有关更多信息，请参阅中的AWSAWS Glue标签、使用条件键或上下文密钥控制设置的基于身份的策略（IAM 策略）以及将所有 AWS 调用配置为通过您的 VPC。	2021 年 10 月 13 日
支持 VPC 终端节点策略	添加了关于支持 AWS Glue 中的 Virtual Private Cloud (VPC) 端点策略的信息。有关更多信息，请参阅 AWS Glue 和接口 VPC 终端节点 (AWS PrivateLink)。	2021 年 10 月 11 日

Glue Studio 现已在中国可用	AWS Glue Studio 现已在中国北京和宁夏区域可用。	2021 年 10 月 11 日
AWS Glue Studio 提供笔记本创作，用于交互式作业编辑	笔记本可以帮助您编写和执行代码、可视化结果和共享见解。通常，数据科学家使用笔记本来执行实验和数据探索任务。有关更多信息，请参阅使用 notebook。	2021 年 10 月 1 日
现在可以直接访问流式处理源	在可视化编辑器中将数据源添加到 ETL 任务时，您可以提供信息来访问数据流，而不必使用数据目录数据库和表。	2021 年 9 月 30 日
记录了 AWS Glue 版本支持策略	添加了关于 AWS Glue 版本支持策略和某些 AWS Glue 版本的使用寿命结束阶段的信息。有关更多信息，请参阅 AWS Glue 版本支持策略。	2021 年 9 月 24 日
自定义连接器现在可以与数据预览一起使用	使用自定义连接器编辑数据源节点时，可以选择 Dat preview（数据集预览）选项卡预览数据集。有关更多信息，请参阅自定义连接器。	2021 年 9 月 24 日

Support 支持 AWS Glue 交互式会话（私人预览）	（私有预览）添加了有关使用 AWS Glue 交互式会话从任何 Jupyter 笔记本在云中运行 Spark 工作负载的信息。使用 AWS Glue 2.0 或更高版本时，交互式会话是开发 AWS Glue 提取、转换和加载 (ETL) 代码的首选方法。有关更多信息，请参阅为 Jupyter Notebook 设置和运行 AWS Glue 交互式会话。	2021 年 8 月 24 日
支持通过蓝图创建工作流 (GA)	添加了有关在蓝图中对常见提取、转换和加载（ETL）使用案例进行编码和通过蓝图创建工作流的信息。使数据分析人员能够轻松创建和运行复杂的 ETL 进程。有关更多信息，请参阅使用 AWS Glue 中的蓝图和工作流执行复杂的 ETL 活动。	2021 年 8 月 23 日

支持 AWS Glue 版本 3.0。	添加了有关支持 AWS Glue 版本 3.0 的信息，该版本支持用于运行 Apache Spark ETL 任务的 Apache Spark 3.0 引擎升级，以及其他优化和升级信息。有关更多信息，请参阅 AWS Glue 发布说明和将 AWS Glue 任务迁移到 AWS Glue 版本 3.0。此版本中的其他功能包括 AWS Glue 随机播放管理器、SIMD 矢量化 CSV 读取器和目录分区谓词。有关更多信息，请参阅使用 Amazon S3 的 AWS Glue Spark 随机播放管理器、AWS Glue 中的 ETL 输入和输出的格式选项和使用目录分区谓词的服务器端筛选。	2021 年 8 月 18 日
AWS GovCloud (US) Region	AWS Glue Studio现已在 AWS GovCloud (US) Region	2021 年 8 月 18 日
Python Shell 编写适用于 AWS Glue Studio	创建新任务时，您现在可以选择创建 Python Shell 任务。有关更多信息，请参阅 开启任务创建流程和在 AWS Glue Studio 中编辑 Python Shell 任务。	2021 年 8 月 13 日
支持通过 Amazon EventBridge 活动启动工作流程	添加了有关 AWS Glue 如何可以成为事件驱动架构中的事件使用者的信息。有关更多信息，请参阅使用 Amazon EventBridge 事件启动AWS Glue工作流程和查看启动工作流程 EventBridge 的事件。	2021 年 7 月 14 日

将 JSON 作为受支持的数据格式添加到 AWS Glue 架构注册表	添加了有关 JSON 作为受支持的数据格式（除 AVRO 之外）的信息。有关更多信息，请参阅 AWS Glue 架构注册表。	2021 年 6 月 30 日
不使用数据目录表创建 AWS Glue 流式处理任务	`create_data_frame_from_options` Python 函数或 `getSource` 支持创建直接引用数据流的流式处理 ETL 任务的 Scala 脚本，无需数据目录表。	2021 年 6 月 15 日
AWS Glue机器学习变换现在支持按键 AWS Key Management Service	在使用控制台、CLI 或 AWS Glue API 配置 Mac AWS Glue hine Learning 转换时，您可以指定安全配置或 AWS KMS 密钥。有关更多信息，请参阅 将数据加密与 Machine Learning 转换结合使用和 AWS Glue Machine Learning API。	2021 年 6 月 15 日
AWSGlueConsoleFullAccess AWS 托管策略的更新	添加了有关 AWSGlueConsoleFullAccess AWS 托管策略小更新的信息。有关更多信息，请参阅AWS 托管策略的 AWS Glue更新。	2021 年 6 月 10 日
在创建和编辑任务时查看任务的数据集	您可以为任务图中的节点使用新的 Data Preview (数据预览) 选项卡，以查看由该节点处理的数据示例。有关更多信息，请参阅在可视化任务编辑器中使用数据预览。	2021 年 6 月 7 日

支持指定用于指示爬网程序输出表位置的值。	添加了有关在配置爬网程序输出时指定表位置的值的信息。有关更多信息，请参阅如何指定表位置。	2021 年 6 月 4 日
支持网络爬取 Simple Storage Service (Amazon S3) 数据存储时网络爬取数据集中的示例文件	添加了有关在网络爬取 Amazon S3 时如何网络爬取示例文件的信息。有关更多信息，请参阅爬网程序属性。	2021 年 5 月 10 日
支持 AWS Glue 优化的 Parquet 写入器	添加了有关使用AWS Glue 优化的 parquet Writer DynamicFrames 来创建或更新`parquet`分类表的信息。有关更多信息，请参阅通过 AWS Glue ETL 任务在数据目录中创建表、更新架构和添加新分区和在 AWS Glue 中的 ETL 输入和输出的格式选项。	2021 年 5 月 4 日
支持 Kafka 客户端身份验证密码	添加了有关 AWS Glue 中的流式处理 ETL 任务如何支持 Apache Kafka 串流创建器的 SSL 客户端证书身份验证的信息。现在，您可以提供自定义证书，同时定义 Apache Kafka 集群的 AWS Glue 连接，AWS Glue 将在该证书进行身份验证时使用该连接。有关更多信息，请参阅 AWS Glue 连接属性和连接 API。	2021 年 4 月 28 日

支持在流式处理 ETL 任务的其他账户中使用来自 Amazon Kinesis Data Streams 的数据	添加了有关创建流式处理 ETL 任务以在其他账户中使用来自 Amazon Kinesis Data Streams 的数据的信息。有关更多信息，请参阅在 AWS Glue 中添加流式处理 ETL 任务。	2021 年 3 月 30 日
提供 SQL 转换	您可以使用 SQL 转换节点以 SQL 查询形式编写您自己的转换。有关更多信息，请参阅使用 SQL 查询转换数据。	2021 年 3 月 23 日
支持从蓝图创建工作流（公开预览版）	（公开预览版）添加了有关在蓝图中对常见提取、转换和加载（ETL）使用案例进行编码和通过蓝图创建工作流的信息。使数据分析人员能够轻松创建和运行复杂的 ETL 进程。有关更多信息，请参阅使用 AWS Glue 中的蓝图和工作流执行复杂的 ETL 活动。	2021 年 3 月 22 日
连接器可用于数据目标	现在支持为数据目标使用自定义 AWS Marketplace 连接器或连接器。有关更多信息，请参阅使用自定义连接器编写任务。	2021 年 3 月 15 日
支持 AWS Glue 机器学习转换的列重要指标	添加了有关使用 AWS Glue 机器学习转换时查看列重要性指标的信息。有关更多信息，请参阅在 AWS Glue 控制台上使用 Machine Learning 转换	2021 年 2 月 5 日

任务计划现在适用于 AWS Glue Studio	您可以在 AWS Glue Studio 中为任务运行定义基于时间的计划。您可以使用控制台创建基本计划，或者使用类似 Unix 的 cron 语法定义更复杂的计划。有关更多信息，请参阅计划任务运行。	2020 年 12 月 21 日
发布了 AWS Glue 自定义连接器	AWS Glue 自定义连接器允许您在 AWS Marketplace中发现和订阅连接器。我们还发布了 AWS Glue Spark 运行时接口，用于插入为 Apache Spark Datasource、Athena 联合查询和 JDBC API 构建的连接器。有关更多信息，请参阅 在 AWS Glue Studio 中使用连接器和连接。	2020 年 12 月 21 日
AWS Glue 版本 2.0 支持运行流式处理 ETL 任务	添加了有关支持在 Glue 版本 2.0 中运行流式处理 ETL 任务的信息 有关更多信息，请参阅在 AWS Glue 中添加流式处理 ETL 任务。	2020 年 12 月 18 日
支持具有有界执行的工作负载分区	添加了有关启用工作负载分区来配置数据集大小或 ETL 任务运行时处理的文件数的上限的信息。有关更多信息，请参阅具有有界执行的工作负载分区。	2020 年 11 月 23 日
支持增强型分区管理	添加了有关如何使用新 API 将分区索引添加到现有表/从现有表删除的信息。有关更多信息，请参阅使用分区索引。	2020 年 11 月 23 日

支持 AWS Glue 架构注册表	添加了有关使用 AWS Glue 架构注册表来集中发现、控制和演变架构的信息。有关更多信息，请参阅 AWS Glue 架构注册表。	2020 年 11 月 19 日
支持流式处理 ETL 作业中的 grok 输入格式	添加了有关将 Grok 模式应用于流式处理源（如日志文件）的信息。有关更多信息，请参阅将 Grok 模式应用于流式处理源。	2020 年 11 月 17 日
支持在 AWS Glue 控制台将标签添加到工作流	添加了有关使用 AWS Glue 控制台创建工作流时添加标签的信息。有关更多信息，请参阅使用 AWS Glue 控制台创建和构建工作流。	2020 年 10 月 27 日
支持递增爬网程序运行	添加了有关支持仅爬取自上次运行以来添加的 Amazon S3 文件夹的增量爬网程序运行的信息。有关更多信息，请参阅增量爬网。	2020 年 10 月 21 日
支持流式处理 ETL 数据源的架构检测。支持 Avro 流式处理 ETL 数据源和自行管理 kafka	在 AWS Glue 中的流式处理提取、转换和加载（ETL）任务现在可以自动检测传入记录的架构，并在每条记录的基础上处理架构更改。现已支持自行管理的 Kafka 数据源。流式处理 ETL 任务现支持数据源中的 Avro 格式。有关更多信息，请参阅 AWS Glue 中的流式处理 ETL、定义流式处理 ETL 任务的任务属性和 Avro 流式处理源的注释和限制。	2020 年 10 月 7 日

支持网络爬取 MongoDB 和 DocumentDB 数据源	添加了有关支持网络爬取 MongoDB 和 Amazon DocumentDB (with MongoDB compatibility) 数据源的信息。有关更多信息，请参阅定义爬网程序。	2020 年 10 月 5 日
支持 FIPS 合规性	添加了有关客户在使用 AWS Glue 访问数据时需要 FIPS 140-2 验证的加密模块的 FIPS 终端节点的信息。有关更多信息，请参阅 FIPS 合规性。	2020 年 9 月 23 日
AWS Glue Studio 提供了一个易于使用的可视化界面，用于创建和监控任务	现在，您可以使用简单的基于图形的界面来编写移动和转换数据的任务，并可以在 AWS Glue 中运行。然后，您可以在 AWS Glue Studio 中使用任务运行控制面板来监控 ETL 执行情况，并确保您的任务按预期运行。有关更多信息，请参阅《AWS Glue Studio 用户指南》。	2020 年 9 月 23 日
支持创建表索引来提高查询性能	添加了有关创建表索引来允许您从表中检索分区子集的信息。有关更多信息，请参阅使用分区索引。	2020 年 9 月 9 日

支持在 AWS Glue 版本 2.0 中运行 Apache Spark ETL 任务时减少启动时间。	添加了有关支持 AWS Glue 版本 2.0 的信息，该版本提供了升级的基础设施，用于运行 Apache Spark ETL 任务，减少了启动时间、更改了日志记录，并支持在任务级别指定其他 Python 模块。有关更多信息，请参阅 AWS Glue 发布说明和运行 Spark ETL 任务，减少启动时间。	2020 年 8 月 10 日
支持限制并发工作流运行次数。	添加了有关如何限制特定工作流的并发工作流运行次数的信息。有关更多信息，请参阅使用 AWS Glue 控制台创建和构建工作流。	2020 年 8 月 10 日
支持使用 VPC 终端节点网络爬取 Simple Storage Service (Amazon S3) 数据存储	添加了有关配置仅通过 Amazon Virtual Private Cloud 环境（Amazon VPC）且出于安全、审计或控制目的，才可以访问 Amazon S3 数据存储的信息。有关更多信息，请参阅使用 VPC 终端节点网络爬取 Amazon S3 数据存储。	2020 年 8 月 7 日
支持恢复工作流运行	添加了以下相关信息：如何恢复因一个或多个节点（任务或爬网程序）未成功完成而导致斤部分完成的工作流运行。有关更多信息，请参阅修复和恢复工作流运行。	2020 年 7 月 27 日

支持在 AWS Glue Kafka 连接中启用私有 CA 证书。	添加了有关支持在 AWS Glue Kafka 连接中启用私有 CA 证书的新连接选项的信息。有关更多信息，请参阅 AWS Glue 中的 ETL 的连接类型和选项和 AWS Glue 使用的特殊参数。	2020 年 7 月 20 日
支持从其他账户中读取 DynamoDB 数据	添加了有关 AWS Glue 支持从另一个 AWS 账户的 DynamoDB 表中读取数据的信息。有关更多信息，请参阅从另一个账户中读取 DynamoDB 数据。	2020 年 7 月 17 日
支持 AWS Glue 版本 1.0 或更高版本中的 DynamoDB 写入器连接	添加了有关支持 DynamoDB 写入器，以及用于 DynamoDB 读取或写入的新连接选项或更新的连接选项的信息。有关更多信息，请参阅 AWS Glue 中的 ETL 的连接类型和选项。	2020 年 7 月 17 日
支持同时使用 AWS Glue 和 Lake Formation 的资源链接和跨账户访问权限控制	添加了有关称为资源链接的新数据目录对象，以及如何管理借助 AWS Glue 和 AWS Lake Formation跨账户的共享数据目录资源的内容。有关更多信息，请参阅授予跨账户访问权限和表资源链接。	2020 年 7 月 7 日
在对 DynamoDB 数据存储进行网络爬取时，支持对记录进行采样	添加了有关在对 DynamoDB 数据存储进行网络爬取时可配置的新属性的信息。有关更多信息，请参阅爬网程序属性。	2020 年 6 月 12 日

支持停止工作流程运行。	添加了有关如何停止特定工作流程的工作流程运行的信息。有关更多信息，请参阅停止工作流程运行。	2020 年 5 月 14 日
支持 Spark 流式处理 ETL 任务	增加了有关使用流式处理数据源创建提取、转换和加载 (ETL) 任务的信息。有关更多信息，请参阅在 AWS Glue 中添加流式处理 ETL 任务。	2020 年 4 月 27 日
支持在运行 ETL 任务后在数据目录中创建表、更新架构和添加新分区	添加了有关如何启用创建表、更新架构和添加新分区以在数据目录中查看 ETL 作业结果的信息。有关更多信息，请参阅通过 AWS Glue ETL 任务在数据目录中创建表、更新架构和添加新分区。	2020 年 4 月 2 日
支持在 AWS Glue 中将 Apache Avro 数据格式的版本指定为 ETL 输入和输出	添加了有关在 AWS Glue 中将 Apache Avro 数据格式的版本指定为 ETL 输入和输出的信息。默认版本 1.7。您可以使用 `version` 格式选项指定 Avro 版本 1.8 以启用逻辑读取/写入。有关更多信息，请参阅 AWS Glue 中 ETL 输入和输出的格式选项。	2020 年 3 月 31 日
支持 EMRFS 经 S3 优化的提交程序，用于将 Parquet 数据写入 Simple Storage Service (Amazon S3)	添加了有关如何设置新标志以启用经 EMRFR S3 优化的提交程序在创建或更新 AWS Glue 任务时将 Parquet 数据写入 Amazon S3 的信息。有关更多信息，请参阅 AWS Glue 所使用的特殊参数。	2020 年 3 月 30 日

对机器学习的支持转变为由资源标签管理的 AWS 资源	在中添加了有关使用 AWS 资源标签管理和控制对机器学习转换的AWS Glue访问权限的信息。您可以为任务、触发器、端点、爬虫和机器学习转换分配 AWS 资源标签。AWS Glue 有关更多信息，请参阅 AWS Glue 中的AWS 标签。	2020 年 3 月 2 日
支持不可覆盖的任务参数	添加了有关对于特殊作业参数的支持的信息，您无法在触发器中或运行作业时覆盖这些参数。有关更多信息，请参阅在 AWS Glue 中添加作业。	2020 年 2 月 12 日
支持新的转换以在 Simple Storage Service (Amazon S3) 中使用数据集	添加了有关新转换（合并、清除和过渡）和 Amazon S3 存储类排除的信息，以便 Apache Spark 应用程序在 Amazon S3 中使用数据集。有关支持 Python 的这些转换的更多信息，请参阅mergeDynamicFrame和在 Amazon S3 中使用数据集。有关 Scala 的信息，请参阅mergeDynamicFrames和 AWS GlueScala API。 GlueContext	2020 年 1 月 16 日
支持使用 ETL 任务中的新分区信息更新数据目录	添加了有关如何编写提取、转换和加载 (ETL) 脚本以 AWS Glue Data Catalog 使用新的分区信息进行更新的信息。使用此功能，您不再需要在作业完成后重新运行爬网程序来查看新分区。有关更多信息，请参阅使用新分区更新数据目录。	2020 年 1 月 15 日

新教程：使用 SageMaker 笔记本	添加了一个教程，演示如何使用 Amazon SageMaker 笔记本来帮助开发 ETL 和机器学习脚本。参见教程：将 Amazon SageMaker 笔记本与您的开发终端节点搭配使用。	2020 年 1 月 3 日
支持从 MongoDB 和 Amazon DocumentDB（与 MongoDB 兼容）读取数据	添加了有关用于读取和写入 MongoDB 和 Amazon DocumentDB (with MongoDB compatibility) 的新连接类型和连接选项的信息。有关更多信息，请参阅 AWS Glue 中的 ETL 的连接类型和选项。	2019 年 12 月 17 日
各种更正和说明	全文中添加了更正和说明。从“已知问题”一章中删除了条目。添加了指示 AWS Glue 仅在指定数据目录加密设置和创建安全配置时支持对称客户主密钥 (CMK) 的警告。添加了指明 AWS Glue 不支持写入 Amazon DynamoDB 的注释。	2019 年 12 月 9 日
支持自定义 JDBC 驱动程序	添加了有关使用 AWS Glue 本身不支持的 JDBC 驱动程序（例如 MySQL 版本 8 和 Oracle 数据库版本 18）连接到数据源和目标的信息。有关更多信息，请参阅 JDBC 连接类型值。	2019 年 11 月 25 日

Support 支持将 SageMaker 笔记本连接到不同的开发端点	添加了有关如何将 SageMaker 笔记本连接到不同开发端点的信息。更新描述了用于切换到新开发终端节点的新控制台操作以及新的 SageMaker IAM 策略。有关更多信息，请参阅在AWS Glue控制台上使用笔记本和为亚马逊 SageMaker 笔记本创建 IAM 政策。	2019 年 11 月 21 日
支持机器学习转换的 AWS Glue 版本	添加了有关在机器学习转换中定义 AWS Glue 版本的信息，以指示机器学习转换与哪个版本的 AWS Glue 兼容。有关更多信息，请参阅在 AWS Glue 控制台上使用 Machine Learning 转换。	2019 年 11 月 21 日
支持倒回任务书签	添加了有关将您的作业书签倒回之前的任何作业运行，从而使后续作业运行只再处理已做上标签的作业的信息。描述了允许您在两个书签之间运行作业的 `job-bookmark-pause` 选项的两个新子选项。有关更多信息，请参阅使用任务书签来跟踪已处理的数据和由 AWS Glue 使用的特殊参数。	2019 年 10 月 22 日
支持自定义 JDBC 证书以连接到数据存储	添加了有关 AWS Glue 支持自定义 JDBC 证书以实现与 AWS Glue 数据源或目标的 SSL 连接的信息。有关更多信息，请参阅使用 AWS Glue 控制台上的连接。	2019 年 10 月 10 日

支持 Python Wheel	添加了有关 AWS Glue 支持 wheel 文件（以及 egg 文件）作为 Python shell 任务的依赖项的信息。有关更多信息，请参阅提供您自己的 Python 库。	2019 年 9 月 26 日
支持 AWS Glue 中的开发端点的版本控制	添加了有关在开发终端节点中定义 `Glue version` 的信息。`Glue version` 确定 AWS Glue 支持的 Apache Spark 和 Python 的版本。有关更多信息，请参阅添加开发终端节点。	2019 年 9 月 19 日
支持使用 Spark UI 监控 AWS Glue	添加了有关使用 Apache Spark UI 监控和调试在 AWS Glue 作业系统上运行的 AWS Glue ETL 任务以及在 AWS Glue 开发终端节点上运行的 Spark 应用程序的信息。有关更多信息，请参阅使用 Spark UI 监控 AWS Glue。	2019 年 9 月 19 日
使用公有 AWS Glue ETL 库，增强了进行本地 ETL 脚本开发的支持	更新了 AWS Glue ETL 库内容以反映现在支持 AWS Glue 版本 1.0。有关更多信息，请参阅使用 AWS Glue ETL 库在本地开发和测试 ETL 脚本。	2019 年 9 月 18 日
支持在运行任务时排除 Simple Storage Service (Amazon S3) 存储类	添加了有关在运行从 Amazon S3 中读取文件或分区的 AWS Glue ETL 任务时排除 Amazon S3 存储类的信息。有关更多信息，请参阅排除 Amazon S3 存储类。	2019 年 8 月 29 日

使用公有 AWS Glue ETL 库，支持进行本地 ETL 脚本开发	添加了有关如何在本地开发和测试 Python 和 Scala ETL 脚本，而无需网络连接的信息。有关更多信息，请参阅使用 AWS Glue ETL 库在本地开发和测试 ETL 脚本。	2019 年 8 月 28 日
已知问题	添加了有关 AWS Glue 中已知问题的信息。有关更多信息，请参阅 AWS Glue 的已知问题。	2019 年 8 月 28 日
支持 AWS Glue 中的机器学习转换	添加了有关 AWS Glue 提供的用以创建自定义转换的机器学习功能的信息。您可以在创建作业时创建这些转换。有关更多信息，请参阅 AWS Glue 中的 Machine Learning 转换。	2019 年 8 月 8 日
支持共享 Amazon Virtual Private Cloud	添加了有关 AWS Glue 支持共享 Amazon Virtual Private Cloud 的信息。有关更多信息，请参阅共享的 Amazon VPC。	2019 年 8 月 6 日
支持 AWS Glue 中的版本控制	添加了有关在任务属性中定义 `Glue version` 的信息。AWS Glue 版本确定 AWS Glue 支持的 Apache Spark 和 Python 的版本。有关更多信息，请参阅在 AWS Glue 中添加任务。	2019 年 7 月 24 日

支持开发终端节点的其他配置选项	针对具有内存密集型工作负载的开发终端节点，添加了有关配置选项的信息。您现在可以从两个新配置中进行选择，这两个新配置可为每个执行程序提供更多内存。有关更多信息，请参阅在 AWS Glue 控制台上使用开发终端节点。	2019 年 7 月 24 日
支持使用工作流执行提取、传输和加载 (ETL) 活动	添加了以下相关信息：使用称为工作流的新构造来设计可由 AWS Glue 作为单个实体运行和跟踪的复杂的多任务提取、转换和加载（ETL）活动。有关更多信息，请参阅使用 AWS Glue 中的工作流执行复杂的 ETL 活动。	2019 年 6 月 20 日
支持 Python Shell 任务中的 Python 3.6	增加了在 Python Shell 作业中对 Python 3.6 的支持的相关信息。您可以指定 Python 2.7 或 Python 3.6 作为作业属性。有关更多信息，请参阅在 AWS Glue 中添加 Python Shell 任务。	2019 年 6 月 5 日
支持 virtual private cloud (VPC) 终端节点	添加了有关在 VPC 中通过接口终端节点直接连接到 AWS Glue 的信息。当您使用 VPC 接口端点时，VPC 与 AWS Glue 之间的通信完全在 AWS 网络内安全进行。有关更多信息，请参阅将 AWS Glue 与接口 VPC 终端节点一起使用。	2019 年 6 月 4 日

支持对 AWS Glue 任务进行实时的连续日志记录。	添加了有关启用和查看实时 Apache Spark 作业日志的信息，CloudWatch 包括驱动程序日志、每个执行者日志和 Spark 作业进度条。有关更多信息，请参阅 AWS Glue 任务的连续日志记录。	2019 年 5 月 28 日
支持将现有数据目录表作为爬网程序源	添加了有关将现有数据目录表的列表指定为爬网程序源的信息。然后，当新数据变为可用时，爬网程序可以检测对表架构的更改、更新表定义并注册新分区。有关更多信息，请参阅爬网程序属性。	2019 年 5 月 10 日
对于内存密集型任务支持额外的配置选项	针对具有内存密集型工作负载的 Apache Spark 任务，添加了有关配置选项的信息。您现在可以从两个新配置中进行选择，这两个新配置可为每个执行程序提供更多内存。有关更多信息，请参阅在 AWS Glue 中添加任务。	2019 年 4 月 5 日
支持 CSV 自定义分类器	添加了有关使用自定义 CSV 分类器来推断各种类型的 CSV 数据的架构的信息。有关更多信息，请参阅编写自定义分类器。	2019 年 3 月 26 日

对 AWS 资源标签的支持	添加了有关使用 AWS 资源标签来帮助您管理和控制对AWS Glue资源的访问的信息。您可以在中为作业、触发器、终端节点和爬虫分配 AWS 资源标签。AWS Glue有关更多信息，请参阅 AWS Glue 中的AWS 标签。	2019 年 3 月 20 日
支持 Spark SQL 任务的数据目录	添加了有关配置AWS Glue任务和开发端点以 AWS Glue Data Catalog 用作外部 Apache Hive Metastore 的信息。这允许任务和开发终端节点直接对存储在 AWS Glue Data Catalog中的表运行 Apache Spark SQL 查询。有关更多信息，请参阅 Spark SQL 任务的AWS Glue Data Catalog 支持。	2019 年 3 月 14 日
支持 Python Shell 任务	添加了有关 Python shell 任务和新字段 Maximum capacity (最大容量) 的信息。有关更多信息，请参阅在 AWS Glue 中添加 Python Shell 任务。	2019 年 1 月 18 日
支持数据库和表发生更改时的通知	添加了有关为数据库、表和分区 API 调用的更改生成的事件的信息。您可以在 “CloudWatch 事件” 中配置操作以响应这些事件。有关更多信息，请参阅自动AWS Glue处理 CloudWatch 事件。	2019 年 1 月 16 日
支持加密连接密码	添加了有关加密连接对象中所用密码的信息。有关更多信息，请参阅加密连接密码。	2018 年 12 月 11 日

对于资源级权限和基于资源的策略的支持	添加了有关将资源级权限和基于资源的策略用于 AWS Glue 的信息。有关更多信息，请参阅 AWS Glue 中的安全性中的主题。	2018 年 10 月 15 日
对 SageMaker 笔记本的 Support	添加了有关使用带有AWS Glue开发端点的 SageMaker 笔记本的信息。有关更多信息，请参阅管理笔记本。	2018 年 10 月 5 日
支持加密	添加了有关对 AWS Glue 使用加密的信息。有关更多信息，请参阅静态加密、传输中加密和在 AWS Glue 中设置加密。	2018 年 8 月 24 日
Apache Spark 任务指标支持	添加了有关使用 Apache Spark 指标的信息，以便更好地调试和分析 ETL 作业。您可以轻松跟踪运行时指标，例如读取和写入的字节数，驱动程序和执行程序的内存使用率和 CPU 负载，以及来自 AWS Glue 控制台的执行程序之间的数据随机排序。有关更多信息，请参阅AWS Glue使用 CloudWatch 指标进行监控、作业监控和调试以及在AWS Glue控制台上使用作业。	2018 年 7 月 13 日
支持将 DynamoDB 作为数据源	添加了有关网络爬取 DynamoDB 以及将其用作 ETL 任务的数据源的信息。有关更多信息，请参阅使用爬网程序编录表和连接参数。	2018 年 7 月 10 日

更新以创建笔记本服务器过程	更新了有关如何在与开发终端节点相关联的 Amazon EC2 实例上创建笔记本服务器的信息。有关更多信息，请参阅创建与开发终端节点相关联的笔记本服务器。	2018 年 7 月 9 日
现在可通过 RSS 更新	您现在可以订阅 RSS 源来接收有关 AWS Glue 开发人员指南更新的通知。	2018 年 25 月 6 日
支持任务的延迟通知	添加了有关在任务运行时配置延迟阈值的信息。有关更多信息，请参阅在 AWS Glue 中添加任务。	2018 年 5 月 25 日
配置爬网程序以追加新列	添加了有关爬虫新配置选项的信息。 MergeNewColumns有关更多信息，请参阅配置爬网程序。	2018 年 5 月 7 日
支持任务超时	添加了有关在任务运行时设置超时阈值的信息。有关更多信息，请参阅在 AWS Glue 中添加任务。	2018 年 4 月 10 日
支持 Scala ETL 脚本并基于其他运行状态触发任务	添加了有关使用 Scala 作为 ETL 编程语言的信息。此外，触发器 API 现在还支持在满足任意条件时触发 (除所有条件之外)。此外，还可以基于“失败”或“已停止”的任务运行触发任务 (除“已成功”的任务运行之外)。	2018 年 1 月 12 日

早期更新

下表描述 2018 年 1 月之前发布的每个 AWS Glue 开发人员指南中的重要变化。

更改	描述	日期
支持 XML 数据源和新的爬网程序配置选项	添加了有关为分区更改分类 XML 数据源和新爬网程序选项的信息。	2017 年 11 月 16 日
新转换、对其他 Amazon RDS 数据库引擎的支持以及开发终端节点增强功能	添加了有关映射和筛选转换、对 Amazon RDS Microsoft SQL Server 和 Amazon RDS Oracle 的支持以及开发终端节点的新功能的信息。	2017 年 9 月 29 日
AWS Glue 初始版本	这是 AWS Glue 开发人员指南的初始版本。	2017 年 8 月 14 日

AWS 词汇表

有关最新 AWS 术语，请参阅《AWS 词汇表 参考资料》中的AWS 词汇表。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。